defined graph explicit. This recursive procedure is straightforward, and the search graph (tree) can be constructed without the extra bookkeeping. However, this process would only generate a search tree where the same node might be generated as a part of several possible paths.

For example, node E is being generated in four different paths. If we are interested in finding an optimal path to travel from S to G, it is more efficient to merge those different paths that lead to the same node E. We can pick the shortest path up to C, since everything following E is the same for the rest of the paths. This is consistent with the dynamic programming principle—when looking for the best path from S to G, all partial paths from S to any node E, other than the best path from S to E, should be discarded. The dynamic programming merge also eliminates cyclic paths implicitly, since a cyclic path cannot be the shortest path. Performing this extra bookkeeping (merging different paths leading into the same node) generates a search graph rather than a search tree.

Although a graph search has the potential advantage over a tree search of being more efficient, it does require extra bookkeeping. Whether this effort is justified depends on the individual problem one has to address.

Most search strategies search in a forward direction, i.e., build the search graph (or tree) by starting with the initial configuration (the starting state S) from the root. In the general AI literature, this is referred to as *forward reasoning* [43], because it performs rule-base reasoning by matching the left side of rules first. However, for some specific problem domains, it might be more efficient to use *backward reasoning* [43], where the search graph is built from the bottom up (the goal state G). Possible scenarios include:

- *There are more initial states than goal states.* Obviously it is easy to start with a small set of states and search for paths leading to one of the bigger sets of states. For example, suppose the initial state S is the hometown for John in the city-traveling problem in Figure 12.1 and the goal state G is an unfamiliar city for him. In the absence of a map, there are certainly more locations (neighboring cities) that John can identify as being close[1] to his home city S than those he can identify as being close to an unfamiliar location. In a sense, all of those locations being identified as close to John's home city S are equivalent to the initial state S. This means John might want to consider reasoning backward from the unfamiliar goal city G for the trip planning.
- *The branching factor for backward reasoning is smaller than that for forward reasoning.* In this case it makes sense to search in the direction with lower branching factor.

It is in principle possible to search from both ends simultaneously, until two partial paths meet somewhere in the middle. This strategy is called *bi-directional search* [43]. Bi-directional search seems particularly appealing if the number of nodes at each step grows

[1] *Being close* means that, once John reaches one of those neighboring cities, he can easily remember the best path to return home. It is similar to the killer book for chess play. Once the player reaches a particular board configuration, he can follow the killer book for moves that can guarantee a victory.

exponentially with the depth that needs to be explored. However, sometimes bi-directional search can be devastating. The two searches may cross each other, as illustrated in Figure 12.3.

The process of explicitly generating part of an implicitly defined graph forms the essence of our general graph search procedure. The procedure is summarized in Algorithm 12.1. It maintains two lists: *OPEN*, which stores the nodes waiting for expansion, and *CLOSE*, which stores the already expanded nodes. Steps 6a and 6b are basically the bookkeeping process to merge different paths going into the same node by picking the one that has the minimum distance. Step 6a handles the case where v is in the *OPEN* list and thus is not expanded. The merging process is straightforward, with a single comparison and change of traceback pointer if necessary. However, when v is in the *CLOSE* list and thus is already expanded in Step 6b, the merging requires additional forward propagation of the new score if the current path is found to be better than the best subpath already in the *CLOSE* list. This forward propagation could be very expensive. Fortunately, most of the search strategy can avoid such a procedure if we know that the already expanded node must belong in the best path leading to it. We discuss this in Section 12.5.

As described earlier, it may not be worthwhile to perform bookkeeping for a graph search, so Steps 6a and 6b are optional. If both steps are omitted, the graph search algorithm described above becomes a tree search algorithm. To illustrate different search strategies, tree search is used as the basic graph search algorithm in the sections that follows. However, you should note that all the search methods described here could be easily extended to graph search with the extra bookkeeping (merging) process as illustrated in Steps 6a and 6b of Algorithm 12.1.

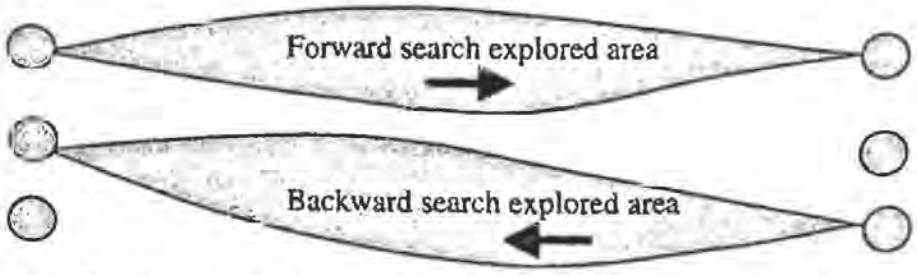


Figure 12.3 A bad case for bi-directional search, where the forward search and the backward search crossed each other [42].

ALGORITHM 12.1: *THE GRAPH-SEARCH ALGORITHM*

Step 1: Initialization: Put S in the $OPEN$ list and create an initially empty $CLOSE$ list.
Step 2: If the $OPEN$ list is empty, exit and declare failure.
Step 3: Pop up the first node N in the $OPEN$ list, remove it from the $OPEN$ list and put it into the $CLOSE$ list.
Step 4: If node N is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from N to S.
Step 5: Expand node N by applying the successor operator to generate the successor set $SS(N)$ of node N. Be sure to eliminate the ancestors of N from $SS(N)$.
Step 6: $\forall v \in SS(N)$ do

6a. (optional) If $v \in OPEN$ and the accumulated distance of the new path is smaller than that for the one in the $OPEN$ list, do

(i) change the traceback (parent) pointer of v to N and adjust the accumulated distance for v.

(ii) go to Step 7.

6b. (optional) If $v \in CLOSE$ and the accumulated distance of the new path is smaller than the partial path ending at v in the $CLOSE$ list, do

(i) change the traceback (parent) pointer of v to N and adjust the accumulated distance for all paths that contain v.

(ii) go to Step 7.

6c. Create a pointer pointing to N and push it into the $OPEN$ list.
Step 7: Reorder the $OPEN$ list according to search strategy or some heuristic measurement.
Step 8: Go to Step 2.

12.1.2. Blind Graph Search Algorithms

If the aim of the search problem is to find an acceptable path instead of the best path, blind search is often used. *Blind search* treats every node in the $OPEN$ list the same and blindly decides the order to be expanded without using any domain knowledge. Since blind search treats every node equally, it is often referred to as *uniform search* or *exhaustive search*, because it exhaustively tries out all possible paths. In AI, people are typically not interested in blind search. However, it does provide a lot of insight into many sophisticated heuristic search algorithms. You should note that blind search does not expand nodes randomly. Instead, it follows some systematic way to explore the search graph. Two popular types of blind search are depth-first search and breadth-first search.

12.1.2.1. Depth-First Search

When we are in a maze, the most natural way to find a way out is to mark the branch we take whenever we reach a branching point. The marks allow us to go back to a choice point with an unexplored alternative, withdraw the most recently made choice and undo all consequences of the withdrawn choice whenever a dead-end is reached. Once the alternative choice is selected and marked, we go forward based on the same procedure. This intuitive search strategy is called *backtracking*. The famous N-queens puzzle [32] can be handily solved by the backtracking strategy.

Depth-first search picks an arbitrary alternative at every node visited. The search sticks with this partial path and works forward from the partial path. Other alternatives at the same level are ignored completely (for the time being) in the hope of finding a solution based on the current choice. This strategy is equivalent to ordering the nodes in the *OPEN* list by their depth in the search graph (tree). The deepest nodes are expanded first and nodes of equal depth are ordered arbitrarily.

Although depth-first search hopes the current choice leads to a solution, sometimes the current choice could lead to a dead-end (a node which is neither a goal node nor can be expanded further). In fact, it is desirable to have many short dead-ends. Otherwise, the algorithm may search for a very long time before it reaches a dead-end, or it might not ever reach a solution if the search space is infinite. When the search reaches a dead-end, it goes back to the last decision point and proceeds with another alternative.

Figure 12.4 shows all the nodes being expanded under the depth-first search algorithm for the city-traveling problem illustrated in Figure 12.1. The only differences between the graph search and the depth-first search algorithms are:

1. The graph search algorithm generates all successors at a time (although all except one are ignored first), while depth-first search generates only one successor at a time.
2. The graph search, when successfully finding a path, saves only one path from the starting node to the goal node, while depth-first search in general saves the entire record of the search graph.

Depth-first search could be dangerous because it might search an impossible path that is actually an infinite dead-end. To prevent exploring of paths that are too long, a depth bound can be placed to constrain the nodes to be expanded, and any node reaching that depth limit is treated as a terminal node (as if it had no successor).

The general graph search algorithm can be modified into a depth-first search algorithm as illustrated in Algorithm 12.2.

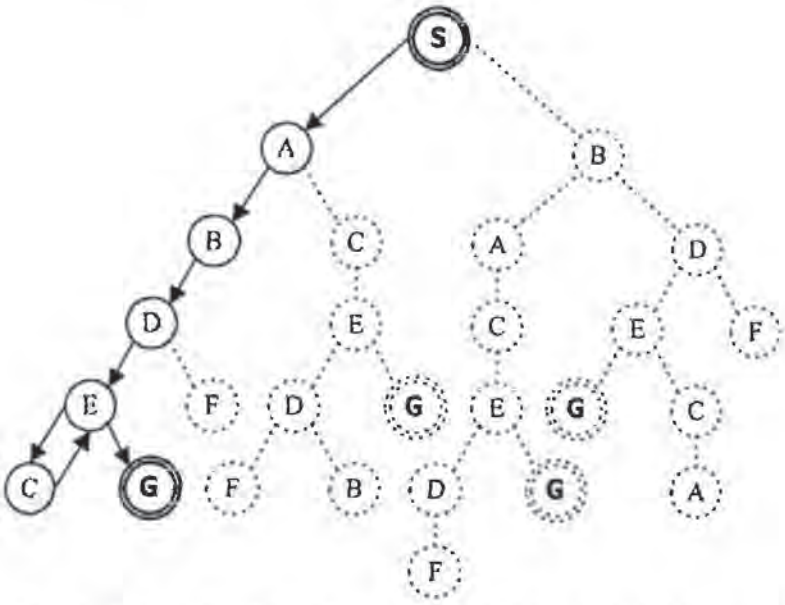


Figure 12.4 The node-expanding procedure of the depth-first search for the path search problem in Figure 12.1. When it fails to find the goal city in node *C*, it backtracks to the parent and continues the search until it finds the goal city. The gray nodes are those that are explored. The dotted nodes are not visited during the search [42].

ALGORITHM 12.2: *THE DEPTH-FIRST SEARCH ALGORITHM*

Step 1: Initialization: Put S in the *OPEN* list and create an initially empty the CLOSE list.
Step 2: If the *OPEN* list is empty, exit and declare failure.
Step 3: Pop up the first node N in the *OPEN* list, remove it from the *OPEN* list and put it into the *CLOSE* list.
Step 4: If node N is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from N to S.

4a. If the depth of node N is equal to the depth bound, go to Step 2.

Step 5: Expand node N by applying the successor operator to generate the successor set SS(N) of node N. Be sure to eliminate the ancestors of N from SS(N).
Step 6: $\forall v \in SS(N)$ do

6c. Create a pointer pointing to N and push it into the *OPEN* list.

Step 7: Reorder the the *OPEN* list in descending order of the depth of the nodes.
Step 8: Go to Step 2.

12.1.2.2. Breadth-First Search

One natural alternative to the depth-first search strategy is breadth-first search. *Breadth-first search* examines all the nodes on one level before considering any of the nodes on the next level (depth). As shown in Figure 12.5, node *B* would be examined just after node *A*. The search moves on level-by-level, finally discovering G on the fourth level.

Breadth-first search is guaranteed to find a solution if one exists, assuming that a finite number of successors (branches) always follow any node. The proof is straightforward. If there is a solution, its path length must be finite. Let's assume the length of the solution is *M*. Breadth-first search explores all paths of the same length increasingly. Since the number of paths of fixed length *N* is always finite, it eventually explores all paths of length *M*. By that time it should find the solution.

It is also easy to show that a breadth-first search can work on a search tree (graph) with infinite depth on which an unconstrained depth-first search will fail. Although a breadth-first might not find a shortest-distance path for the city-travel problem, it is guaranteed to find the one with fewest cities visited (minimum-length path). In some cases, it is a very desirable solution. On the other hand, a breadth-first search may be highly inefficient when all solutions leading to the goal node are at approximately the same depth. The breadth-first search algorithm is summarized in Algorithm 12.3.

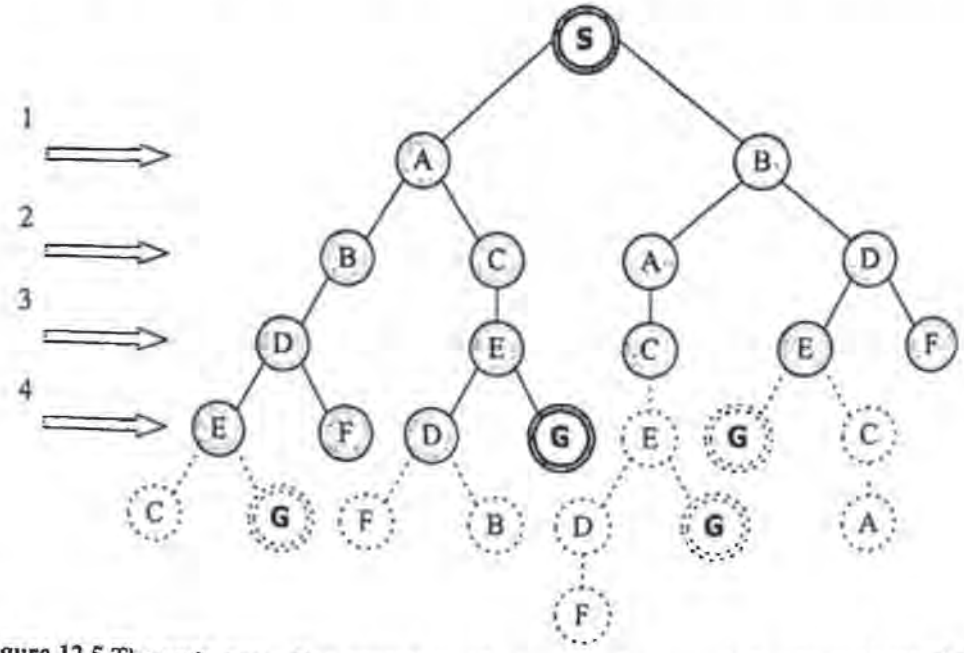


Figure 12.5 The node-expanding procedure of a breadth-first search for the path search problem in Figure 12.1. It searches through each level until the goal is identified. The gray nodes are those that are explored. The dotted nodes are not visited during the search [42].

ALGORITHM 12.3: *THE BREADTH-FIRST SEARCH ALGORITHM*

Step 1: Initialization: Put S in the *OPEN* list and create an initially empty the *CLOSE* list.
Step 2: If the *OPEN* list is empty, exit and declare failure.
Step 3: Pop up the first node N in the *OPEN* list, remove it from the *OPEN* list and put it into the *CLOSE* list.
Step 4: If node N is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from N to S.
Step 5: Expand node N by applying the successor operator to generate the successor set $SS(N)$ of node N. Be sure to eliminate the ancestors of N, from $SS(N)$.
Step 6: $\forall v \in SS(N)$ do

6c. Create a pointer pointing to N and push it into the *OPEN* list.

Step 7: Reorder the *OPEN* list in increasing order of the depth of the nodes.
Step 8. Go to Step 2.

12.1.3. Heuristic Graph Search

Blind search methods, like depth-first search and breadth-first search, have no sense (or guidance) of where the goal node lies ahead. Consequently, they often spend a lot of time searching in hopeless directions. If there is guidance, the search can move in the direction that is more likely to lead to the goal. For example, you may want to find a driving route to the World Trade Center in New York. Without a map at hand, you can still use a straight-line distance estimated by eye as a hint to see if you are closer to the goal (World Trade Center). This *hill-climbing* style of guidance can help you to find the destination much more efficiently.

Blind search finds only one arbitrary solution instead of the optimal solution. To find the optimal solution with depth-first or breadth-first search, you must not stop searching when the first solution is discovered. Instead, the search needs to continue until it reaches all the solutions, so you can compare them to pick the best. This strategy for finding the optimal solution is called *British Museum search* or *brute-force search*. Obviously, it is unfeasible when the search space is large. Again, to conduct selective search and yet still be able to find the optimal solution, some guidance on the search graph is necessary.

The guidance obviously comes from domain-specific knowledge. Such knowledge is usually referred to as *heuristic* information, and search methods taking advantage of it are called *heuristic search* methods. There is usually a wide variety of different heuristics for the problem domain. Some heuristics can reduce search effort without sacrificing optimality, while other can greatly reduce search effort but provide only sub-optimal solutions. In most practical problems, the choice of different heuristics is usually a tradeoff between the quality of the solution and the cost of finding the solution.

Heuristic information works like an evaluation function $h(N)$ that maps each node N to a real number, and which serves to indicate the relative goodness (or cost) of continuing the search path from that node. Since in our city-travel problem, straight-line distance is a natural way of measuring the goodness of a path, we can use the heuristic function $h(N)$ for the distance evaluation as:

$h(N)$=Heuristic estimate of the remaining distance from node N to goal G (12.1)

Since $g(N)$, the distance of the partial path to the current node N, is generally known, we have:

$g(N)$=The distance of the partial path already traveled from root S to node N (12.2)

We can define a new heuristic function, $f(N)$, which estimates the total distance for the path (not yet finished) going through node N.

$$f(N) = g(N) + h(N) \tag{12.3}$$

A heuristic search method basically uses the heuristic function $f(N)$ to re-order the *OPEN* list in the Step 7 of Algorithm 12.1. The node with the best heuristic value is explored first (expanded first). Some heuristic search strategies also prune some unpromising partial paths forever to save search space. This is why heuristic search is often referred to as heuristic pruning.

The choice of the heuristic function is critical to the search results. If we use one that overestimates the distance of some nodes, the search results may be suboptimal. Therefore, heuristic functions that do not overestimate the distance are often used in search methods aiming to find the optimal solution.

To close this section, we describe two of the most popular heuristic search methods: best-first (or A* Search) [32, 43] and beam search [43]. They are widely used in many components of spoken language systems.

12.1.3.1. Best-First (A* Search)

Once we have a reasonable heuristic function to evaluate the goodness of each node in the *OPEN* list, we can explore the best node (the node with smallest $f(N)$ value) first, since it offers the best hope of leading to the best path. This natural search strategy is called *best-first search*. To implement best-first search based on the Algorithm 12.1, we need to first evaluate $f(N)$ for each successor before putting the successors in the *OPEN* list in Step 6. We also need to sort the elements in the *OPEN* list based on $f(N)$ in Step 7, so that the best node is in the front-most position waiting to be expanded in Step 3. The modified procedure for performing best-first search is illustrated in Algorithm 12.4. To avoid duplicating nodes in the *OPEN* list, we include Steps 6a and 6b to take advantage of the dynamic programming principle. They perform the needed bookkeeping process to merge different paths leading into the same node.

ALGORITHM 12.4: *THE BEST-FIRST SEARCH ALGORITHM*

Step 1: Initialization: Put S in the *OPEN* list and create an initially empty the *CLOSE* list.
Step 2: If the *OPEN* list is empty, exit and declare failure.
Step 3. Pop up the first node N in the *OPEN* list, remove it from the *OPEN* list and put it into the *CLOSE* list.
Step 4: If node N is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from N to S.
Step 5: Expand node N by applying the successor operator to generate the successor set $SS(N)$ of node N. Be sure to eliminate the ancestors of N, from $SS(N)$.
Step 6: $\forall v \in SS(N)$ do

6a. (optional) If $v \in OPEN$ and the accumulated distance of the new path is smaller than that for the one in the the *OPEN* list, do

(i) Change the traceback (parent) pointer of v to N and adjust the accumulated distance for v.

(ii) Evaluate heuristic function $f(v)$ for v and go to Step 7.

6b. (optional) If $v \in CLOSE$ and the accumulated distance of the new path is small than the partial path ending at v in the the *CLOSE* list,

(i) Change the traceback (parent) pointer of v to N and adjust the accumulated distance and heuristic function f for all the paths containing v.

(ii) go to Step 7.

6c. Create a pointer pointing to N and push it into the *OPEN* list.
Step 7: Reorder the the *OPEN* list in the increasing order of the heuristic function $f(N)$.
Step 8: Go to Step 2.

A search algorithm is said to be *admissible* if it can guarantee to find an optimal solution, if one exists. Now we show that if the heuristic function $h(N)$ of estimating the remaining distance from N to goal node G is an underestimate[2] of the true distance from N to goal node G, the best-first search illustrated in Algorithm 12.4 is admissible. In fact, when $h(N)$ satisfies the above criterion, the best-first algorithm is called A* (pronounced as /eh/-star) Search.

The proof can be carried out informally as follows. When the frontmost node in the *OPEN* list is the goal node G in Step 4, it immediately implies that

$$\forall v \in OPEN \quad f(v) \geq f(G) = g(G) + h(G) = g(G) \tag{12.4}$$

[2] For admissibility, we actually require only that the heuristic function not overestimate the distance from N to G. Since it is very rare to have an exact estimate, we use underestimate throughout this chapter without loss of generality. Sometimes we refer to an underestimate function as a lower-bound estimate of the true value.

Equation (12.4) says that the distance estimate of any incomplete path is no shorter than the first found complete path. Since the distance estimate for any incomplete path is underestimated, the first found complete path in Step 4 must be the optimal path. A similar argument can also be used to prove that the Step 6b is actually not necessary for admissible heuristic functions; that is, there cannot be another path with a shorter distance from the starting node to a node that has been expanded. This is a very important feature since Step 6b is, in general, very expensive and it requires significant updates of many already expanded paths.

The A* search method is actually a family of search algorithms. When $h(N) = 0$ for all N, the search degenerates into an uninformed search[3] [40]. In fact, this type of uninformed search is the famous *branch-and-bound search* algorithm that is often used in many *operations research* problems. Branch-and-bound search always expands the shortest path leading into an open node until there is a path reaching the goal that is of a length no longer than all incomplete paths terminating at open nodes. When $g(N)$ is defined as the depth of the node N, the use of heuristic function $f(N)$ makes the search method identical to breadth-first search. In Section 12.1.2.2, we mention that breadth-first search is guaranteed to find a minimum length path. This can certainly be derived from the admissibility of the A* search method.

When the heuristic function is close to the true remaining distance, the search can usually find the optimal solution without too much effort. In fact, when the true remaining distances for all nodes are known, the search can be done in a totally greedy fashion without any search at all, i.e., the only path explored is the solution. Any non-zero heuristic function is then called an informed heuristic function, and the search using such a function is called informed search. A heuristic function h_1 is said to be more informed than a heuristic function h_2 if the estimate h_1 is everywhere larger than h_2 and yet still admissible (underestimate). Finding an informed admissible heuristic function (guaranteed to underestimate for all nodes) is, in general, a difficult task. The heuristic often requires extensive analysis of the domain-specific knowledge and knowledge representation.

Let's look at a simple example—the 8-puzzle problem. The 8-puzzle consists of eight numbered, movable tiles set in a 3×3 frame. One cell of this frame is always empty, so it is possible to move an adjacent numbered tile into the empty cell. A solution for the 8-puzzle is to find a sequence of moves to change the initial configuration into a given goal configuration as shown in Figure 12.6. One choice for an informed admissible heuristic function h_1 is the number of misplaced tiles associated with the current configuration. Since each misplaced tile needs to move at least once to be in the right position, this heuristic function is clearly a lower bound of the true movements remaining. Based on this heuristic function, the value for the initial configuration will be 7 in Figure 12.7. If we examine this problem further, a more informed heuristic function h_2 can be defined as the sum of all row and column distances of all misplaced tiles and their goal positions. For example, the row and column distance between the tile 8 in the initial configuration and the goal position is 2 + 1= 3,

[3] In some literature an uninformed search is referred to as uniform-cost search.



Figure 12.6 Initial and goal configurations for the 8-puzzle problem.

which indicates that one must move tile 8 at least 3 times in order for it to be in the right position. Based on the heuristic function h_2, the value for the initial configuration will be 16 in Figure 12.6. h_2 is again admissible.

In our city-travel problem, one natural choice for the underestimating heuristic function of the remaining distance between node N and goal G is the straight-line distance since the true distance must be no shorter than the straight-line distance.

Figure 12.7 shows an augmented city-distance map with straight-line distance to goal node attached to each node. Accordingly, the heuristic search tree can be easily constructed for improved efficiency. Figure 12.8 shows the search progress of applying the A* search algorithm for the city-traveling problem by using the straight-line distance heuristic function to estimate the remaining distances.

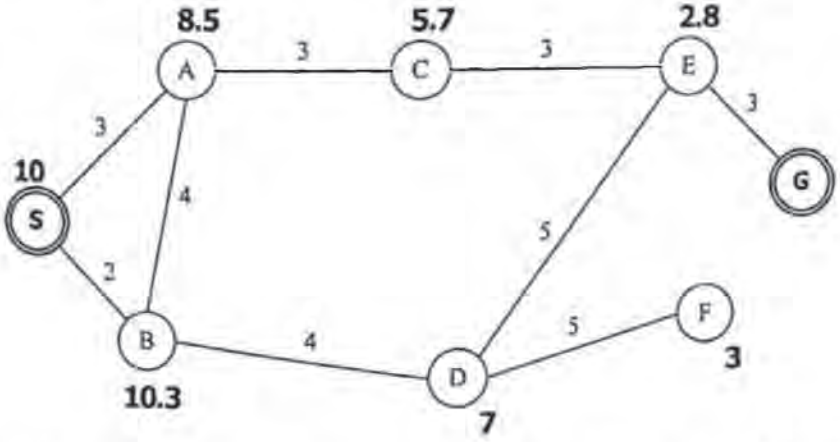


Figure 12.7 The city-travel problem augmented with heuristic information. The numbers beside each node indicate the straight-line distance to the goal node G [42].

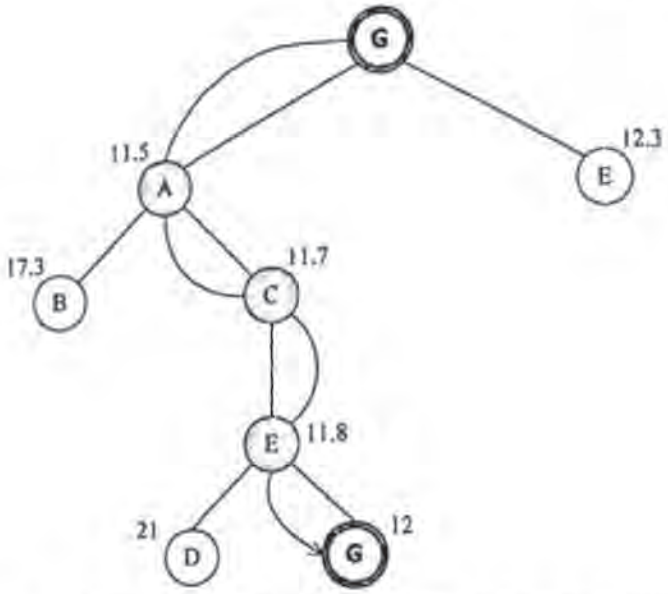


Figure 12.8 The search progress of applying A* search for the city-travel problem. The search determines that path S-A-C-E-G is the optimal one. The number beside the node is f values on which the sorting of the *OPEN* list is based [42].

12.1.3.2. Beam Search

Sometimes, it is impossible to find any effective heuristic estimate, as required in A* search, particularly when there is very little (or no) information about the remaining paths. For example, in real-time speech recognition, there is little information about what the speaker will utter for the remaining speech. Therefore, an efficient uninformed search strategy is very important to tackle this type of problem.

Breadth-first style search is an important strategy for heuristic search. A breadth-first search virtually explores all the paths with the same depth before exploring deeper paths. In practice, paths of the same depth are often easier to compare. It requires fewer heuristics to rank the goodness of each path. Even with uninformed heuristic function ($h(N)=0$), the direct comparison of g (distance so far) of the paths with the same length should be a reasonable choice.

Beam search is a widely used search technique for speech recognition systems [26, 31, 37]. It is a breadth-first style search and progresses along with the depth. Unlike traditional breadth-first search, however, beam search only expands nodes that are likely to succeed at each level. Only these nodes are kept in the beam, and the rest are ignored (pruned) for improved efficiency.

In general, a beam search only keeps up to w best paths at each stage (level), and the rest of the paths are discarded. The number w is often referred to as beam width. The number of nodes explored remains manageable in beam search even if the whole search space is gigantic. If a beam width w is used in a beam search with an average branching factor b, only $w \times b$ nodes need to be explored at any depth, instead of the exponential number

needed for breadth-first search. Suppose that a beam width of 2 is used for the city-travel problem. Figure 12.9 illustrates how beam search progresses to find the path. We can also see that the beam search saved a large number of unneeded nodes, as shown by the dotted nodes.

The beam search algorithm can be easily modified from the breadth-first search algorithm and is illustrated in Algorithm 12.5. For simplicity, we do not include the merging step here. In Algorithm 12.5, Step 4 obviously requires sorting, which is time-consuming if the number $w \times b$ is huge. In practice, the beam is usually implemented as a flexible list where nodes are expanded if their heuristic functions $f(N)$ are within some threshold (a.k.a., beam threshold) of the best node (the smallest value) at the same level. Thus, we only need to identify the best node and then prune away nodes that are outside of the threshold. Although this makes the beam size change dynamically, it significantly reduces the effort for sorting of the *Beam-Candidate* list. In fact, by adjusting the beam threshold, the beam size can be controlled indirectly and yet kept manageable.

Unlike A′ search, beam search is an approximate heuristic search method that is not admissible. However, it has a number of unique merits. Because of its simplicity in both its search strategy and its requirement of domain-specific heuristic information, it has become one of the most popular methods for complicated speech recognition problems. It is particularly attractive when integration of different knowledge sources is required in a time-synchronous fashion. It has the advantages of providing a consistent way of exploring nodes level by level and of offering minimally needed communication between different paths. It is also very suitable for parallel implementation because of its breadth-first search nature.

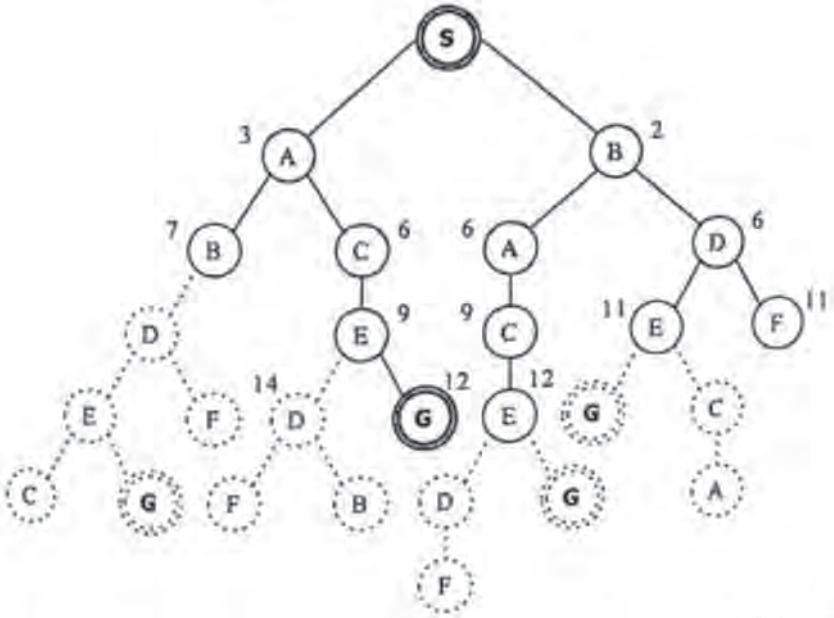


Figure 12.9 Beam search for the city-travel problem. The nodes with gray color are the ones kept in the beam. The transparent nodes were explored but pruned because of higher cost. The dotted nodes indicate all the savings because of pruning [42].

ALGORITHM 12.5: *THE BEAM SEARCH ALGORITHM*

Step 1: Initialization: Put S in the *OPEN* list and create an initially empty *CLOSE* list.
Step 2: If the *OPEN* list is empty, exit and declare failure.
Step 3: $\forall N \in OPEN$ do
3a. Pop up node N in the *OPEN* list, remove it from the *OPEN* list and put it into the *CLOSE* list.
3b. If node N is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from N to S.
3c. Expand node N by applying a successor operator to generate the successor set $SS(N)$ of node N. Be sure to eliminate the successors, which are ancestors of N, from $SS(N)$.
3d. $\forall v \in SS(N)$ Create a pointer pointing to N and push it into *Beam-Candidate* list.
Step 4: Sort the *Beam-Candidate* list according to the heuristic function $f(N)$ so that the best w nodes can be pushed into the the *OPEN* list. Prune the rest of nodes in the *Beam-Candidate* list.
Step 5: Go to Step 2.

12.2. SEARCH ALGORITHMS FOR SPEECH RECOGNITION

As described in Chapter 9, the decoder is basically a search process to uncover the word sequence $\hat{\mathbf{W}} = w_1 w_2 ... w_m$ that has the maximum posterior probability $P(\mathbf{W}|\mathbf{X})$ for the given acoustic observation $\mathbf{X} = X_1 X_2 ... X_n$. That is,

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{X}) = \arg\max_{\mathbf{W}} \frac{P(\mathbf{W})P(\mathbf{X}|\mathbf{W})}{P(\mathbf{X})} = \arg\max_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X}|\mathbf{W}) \qquad (12.5)$$

One obvious way is to search all possible word sequences and select the one with the best posterior probability score.

The unit of acoustic model $P(\mathbf{X}|\mathbf{W})$ is not necessary a word model. For large-vocabulary speech recognition systems, subword models, which include phonemes, demisyllables, and syllables are often used. When subword models are used, the word model $P(\mathbf{X}|\mathbf{W})$ is then obtained by concatenating the subword models according to the pronunciation transcription of the words in a lexicon or dictionary.

When word models are available, speech recognition becomes a search problem. The goal for speech recognition is thus to find a sequence of word models that best describes the input waveform against the word models. As neither the number of words nor the boundary of each word or phoneme in the input waveform is known, appropriate search strategies to deal with these variable-length nonstationary patterns are extremely important.

When HMMs are used for speech recognition systems, the states in the HMM can be expanded to form the state-search space in the search. In this chapter, we use HMMs as our speech models. Although the HMM framework is used to describe the search algorithms, all

techniques mentioned in this and the following chapter can be used for systems based on other modeling techniques, including template matching and neural networks. In fact, many search techniques had been invented before HMMs were applied to speech recognition. Moreover, the HMMs state transition network is actually general enough to represent the general search framework for all modeling approaches.

12.2.1. Decoder Basics

The lessons learned from dynamic programming or the Viterbi algorithm introduced in Chapter 8 tell us that the exponential blind search can be avoided if we can store some intermediate optimal paths (results). Those intermediate paths are used for other paths without being recomputed each time. Moreover, the beam search described in the previous section shows us that efficient search is possible if appropriate pruning is employed to discard highly unlikely paths. In fact, all the search techniques use two strategies: sharing and pruning. *Sharing* means that intermediate results can be kept, so that they can be used by other paths without redundant re-computation. *Pruning* means that unpromising paths can be discarded reliably without wasting time in exploring them further.

Search strategies based on dynamic programming or the Viterbi algorithm with the help of clever pruning, have been applied successfully to a wide range of speech recognition tasks [31], ranging from small-vocabulary tasks, like digit recognition, to unconstraint large-vocabulary (more than 60,000 words) speech recognition. All the efficient search algorithms we discuss in this chapter and the next are considered as variants of dynamic programming or the Viterbi search algorithm.

In Section 12.1, cost (distance) is used as the measure of goodness for graph search algorithms. With Bayes' formulation, searching the minimum-cost path (word sequence) is equivalent to finding the path with maximum probability. For the sake of consistency, we use the inverse of Bayes' posterior probability as our objective function. Furthermore, logarithms are used on the inverse posterior probability to avoid multiplications. That is, the following new criterion is used to find the optimal word sequence $\hat{\mathbf{W}}$:

$$C(\mathbf{W}|\mathbf{X}) = \log\left[\frac{1}{P(\mathbf{W})P(\mathbf{X}|\mathbf{W})}\right] = -\log\left[P(\mathbf{W})P(\mathbf{X}|\mathbf{W})\right] \tag{12.6}$$

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} C(\mathbf{W}|\mathbf{X}) \tag{12.7}$$

For simplicity, we also define the following cost measures to mirror the likelihood for acoustic models and language models:

$$C(\mathbf{X}|\mathbf{W}) = -\log\left[P(\mathbf{X}|\mathbf{W})\right] \tag{12.8}$$

$$C(\mathbf{W}) = -\log\left[P(\mathbf{W})\right] \tag{12.9}$$

12.2.2. Combining Acoustic and Language Models

Although Bayes' equation [Eq. (12.5)] suggests that the acoustic model probability (conditional probability) and language model probability (prior probability) can be combined through simple multiplication, in practice some weighting is desirable. For example, when HMMs are used for acoustic models, the acoustic probability is usually underestimated, owing to the fallacy of the Markov and independence assumptions. Combining the language model probability with an underestimated acoustic model probability according to Eq. (12.5) would give the language model too little weight. Moreover, the two quantities have vastly different dynamic ranges particularly when continuous HMMs are used. One way to balance the two probability quantities is to add a *language model weight* LW to raise the language model probability $P(\mathbf{W})$ to that power $P(\mathbf{W})^{LW}$ [4, 25]. The language model weight LW is typically determined empirically to optimize the recognition performance on a development set. Since the acoustic model probabilities are underestimated, the language model weight LW is typically >1.

Language model probability has another function as a penalty for inserting a new word (or existing words). In particular, when a uniform language model (every word has an equal probability for any condition) is used, the language model probability here can be viewed as purely the penalty of inserting a new word. If this penalty is large, the decoder will prefer fewer longer words in general, and if this penalty is small, the decoder will prefer a greater number of shorter words instead. Since varying the language model weight to match the underestimated acoustic model probability will have some side effect of adjusting the penalty of inserting a new word, we sometimes use another independent *insertion penalty* to adjust the issue of longer or short words. Thus the language model contribution becomes:

$$P(\mathbf{W})^{LW} IP^{N(\mathbf{W})} \tag{12.10}$$

where IP is the insertion penalty (generally $0 < IP \leq 1.0$) and $N(\mathbf{W})$ is the number of words in sentence $\mathbf{W}$. According to Eq. (12.10), insertion penalty is generally a constant that is added to the negative-logarithm domain when extending the search to another new word. In Chapter 9, we described how to compute errors in a speech recognition system and introduced three types of error: substitutions, deletions and insertions. Insertion penalty is so named because it usually affects only insertions. Similar to language model weight, the insertion penalty is determined empirically to optimize the recognition performance on a development set.

12.2.3. Isolated Word Recognition

With isolated word recognition, word boundaries are known. If word HMMs are available, the acoustic model probability $P(\mathbf{X}|\mathbf{W})$ can be computed using the forward algorithm introduced in Chapter 8. The search becomes a simple pattern recognition problem, and the word

$\hat{W}$ with highest forward probability is then chosen as the recognized word. When subword models are used, word HMMs can be easily constructed by concatenating corresponding phoneme HMMs or other types of subword HMMs according to the procedure described in Chapter 9.

12.2.4. Continuous Speech Recognition

Search in continuous speech recognition is rather complicated, even for a small vocabulary, since the search algorithm has to consider the possibility of each word starting at any arbitrary time frame. Some of the earliest speech recognition systems took a two-stage approach towards continuous speech recognition, first hypothesizing the possible word boundaries and then using pattern matching techniques for recognizing the segmented patterns. However, due to significant cross-word co-articulation, there is no reliable segmentation algorithm for detecting word boundaries other than doing recognition itself.

Let's illustrate how you can extend the isolated-word search technique to continuous speech recognition by a simple example, as shown in Figure 12.10. This system contains only two words, w_1 and w_2. We assume the language model used here is an uniform unigram ($P(w_1) = P(w_2) = 1/2$).

It is important to represent the language structures in the same HMM framework. In Figure 12.10, we add one starting state S and one collector state C. The starting state has a null transition to the initial state of each word HMM with corresponding language model probability (1/2 in this case). The final state of each word HMM has a null transition to the collector state. The collector state then has a null transition back to the starting state in order to allow recursion. Similar to the case of embedding the phoneme (subword) HMMs into the word HMM for isolated speech recognition, we can embed the word HMMs for w_1 and w_2 into a new HMM corresponding to structure in Figure 12.10. Thus, the continuous speech search problem can be solved by the standard HMM formulations.

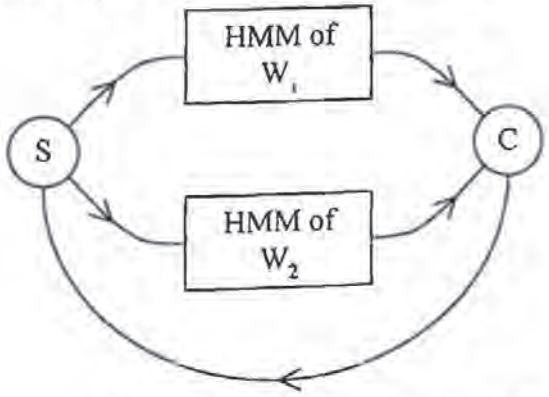


Figure 12.10 A simple example of continuous speech recognition task with two words w_1 and w_2. A uniform unigram language model is assumed for these words. State S is the starting state while state C is a collector state to save fully expanded links between every word pair.

The composite HMMs shown in Figure 12.10 can be viewed as a stochastic finite state network with transition probabilities and output distributions. The search algorithm is essentially producing a match between the acoustic observation $\mathbf{X}$ and a path[4] in the stochastic finite state network. Unlike isolated word recognition, continuous speech recognition needs to find the optimal word sequence $\hat{\mathbf{W}}$. The Viterbi algorithm is clearly a natural choice for this task since the optimal state sequence $\hat{\mathbf{S}}$ corresponds to the optimal word sequence $\hat{\mathbf{W}}$. Figure 12.11 shows the HMM Viterbi trellis computation for the two-word continuous speech recognition example in Figure 12.10. There is a cell for each state in the stochastic finite state network and each time frame t in the trellis. Each cell $C_{s,t}$ in the trellis can be connected to a cell corresponding to time t or $t+1$ and to states in the stochastic finite state network that can be reached from s. To make a word transition, there is a null transition to connect the final state of each word HMM to the initial state of the next word HMM that can be followed. The trellis computation is done *time-synchronously* from left to right, i.e., each cell for time t is completely computed before proceeding to time $t+1$.

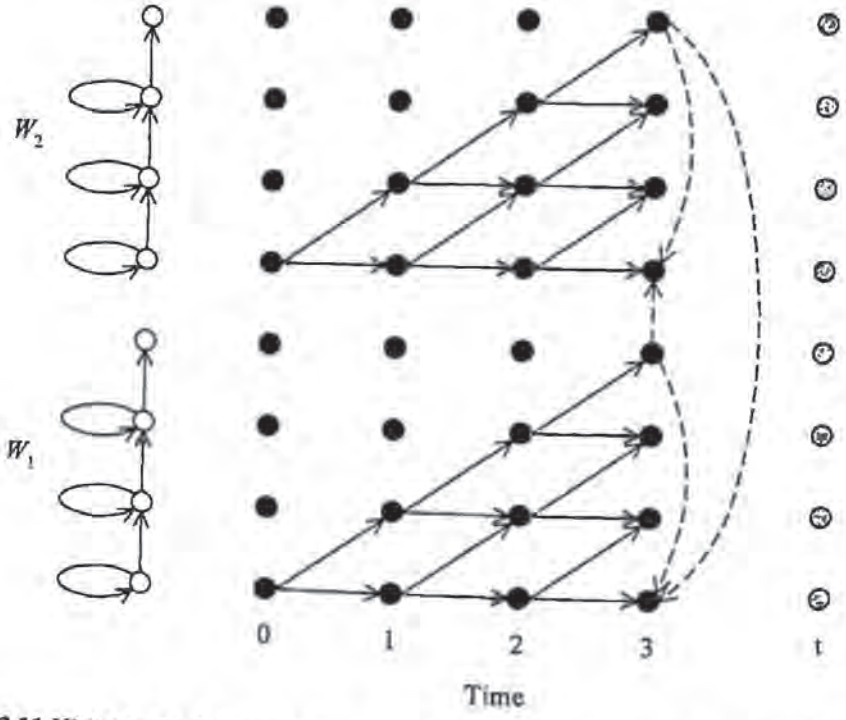


Figure 12.11 HMM trellis for continuous speech recognition example in Figure 12.10. When the final state of the word HMM is reached, a null arc (indicated by a dashed line) is linked from it to the initial state of the following word.

[4] A path here means a sequence of states and transitions.

12.3. LANGUAGE MODEL STATES

The state-space is a good indicator of search complexity. Since the HMM representation for each word in the lexicon is fixed, the state-space is determined by the language models. According to Chapter 11, every language model (grammar) is associated with a state machine (automata). Such a state machine is expanded to form the state-space for the recognizer. The states in such a state machine are referred to as *language models states*. For simplicity, we will use the concepts of state-space and language model states interchangeably. The expansion of language model states to HMM states will be done implicitly. The language model states for isolated word recognition are trivial. They are just the union of the HMM states of each word. In this section we look at the language model states for various grammars for continuous speech recognition.

12.3.1. Search Space with FSM and CFG

As described in Chapter 8, the complexity for the Viterbi algorithm is $O(N^2T)$, where N is the total number of states in the composite HMM and T is the length of input observation. A full time-synchronous Viterbi search is quite efficient for moderate tasks (vocabulary $\leq$ 500). We have already demonstrated in Figure 12.11 how to search for a two-word continuous speech recognition task with a uniform language model. The uniform language model, which allows all words in the vocabulary to follow every word with the same probability, is suitable for connected-digit task. In fact, most small vocabulary tasks in speech recognition applications usually use a finite state grammar (FSG).

Figure 12.12 shows a simple example of an FSM. Similar to the process described in Sections 12.2.3 and 12.2.4, each of the word arcs in an FSG can be expanded as a network of phoneme (subword) HMMs. The word HMMs are connected with null transitions with the grammar state. A large finite state HMM network that encodes all the legal sentences can be constructed based on the expansion procedure. The decoding process is achieved by performing a time-synchronous Viterbi search on this composite finite state HMM.

In practice, FSGs are sufficient for simple tasks. However, when an FSG is made to satisfy the constraints of sharing of different sub-grammars for compactness and support for dynamic modifications, the resulting non-deterministic FSG is very similar to context-free grammar (CFG) in terms of implementation. The CFG grammar consists of a set of productions or rules, which expand nonterminals into a sequence of terminals and nonterminals. Nonterminals in the grammar tend to refer to high-level task-specific concepts such as dates, names, and commands. The terminals are words in the vocabulary. A grammar also has a non-terminal designated as its start state.

Although efficient parsing algorithms, like chart parsing (described in Chapter 11), are available for CFG, they are not suitable for speech recognition, which requires left-to-right processing. A context-free grammar can be formulated with a recursive transition network (RTN). RTNs are more powerful and complicated than the finite state machines described in

Chapter 11 because they allow arc labels to refer to other networks as well as words. We use Figure 12.13 to illustrate how to embed HMMs into a recursive transition network.

Figure 12.13 is an RTN representation of the following CFG:

```
S→ NP VP
NP→ sam | sam davis
VP → VERB tom
VERB → likes | hates
```

There are three types of arcs in an RTN, as shown in Figure 12.13: CAT(x), PUSH (x), and POP(x). The CAT(x) arc indicates that x is a terminal node (which is equivalent to a word arc). Therefore, all the CAT(x) arcs can be expanded by the HMM network for x. The word HMM can again be a composite HMM built from phoneme (or subword) HMMs. Similar to the finite state grammar case in Figure 12.12, each grammar state acts as a state with incoming and outgoing null transitions to connect word HMMs in the CFG.

During decoding, the search pursues several paths through the CFG at the same time. Associated with each of the paths is a grammar state that describes completely how the path can be extended further. When the decoder hypothesizes the end of the current word of a path, it asks the CFG module to extend the path further by one word. There may be several alternative successor words for the given path. The decoder considers all the successor word possibilities. This may cause the path to be extended to generate several more paths to be considered, each with its own grammar state.

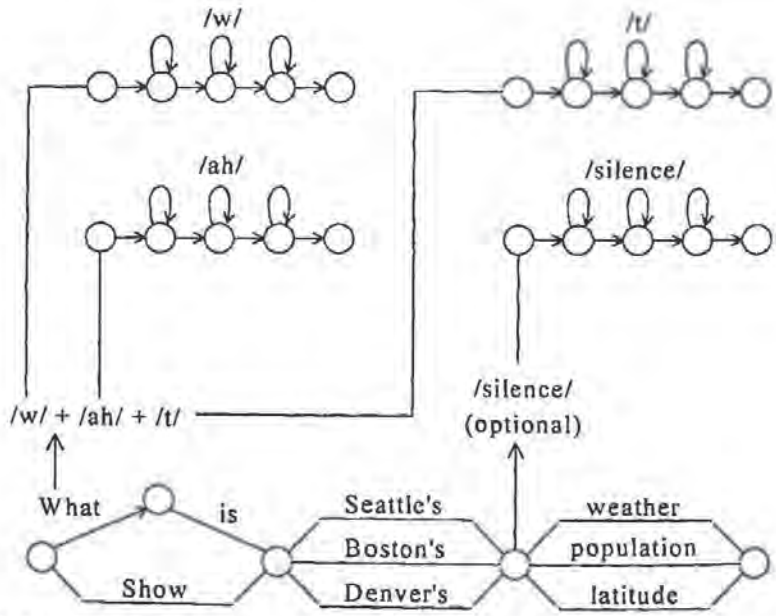


Figure 12.12 An illustration of how to compile a speech recognition task with finite state grammar into a composite HMM.

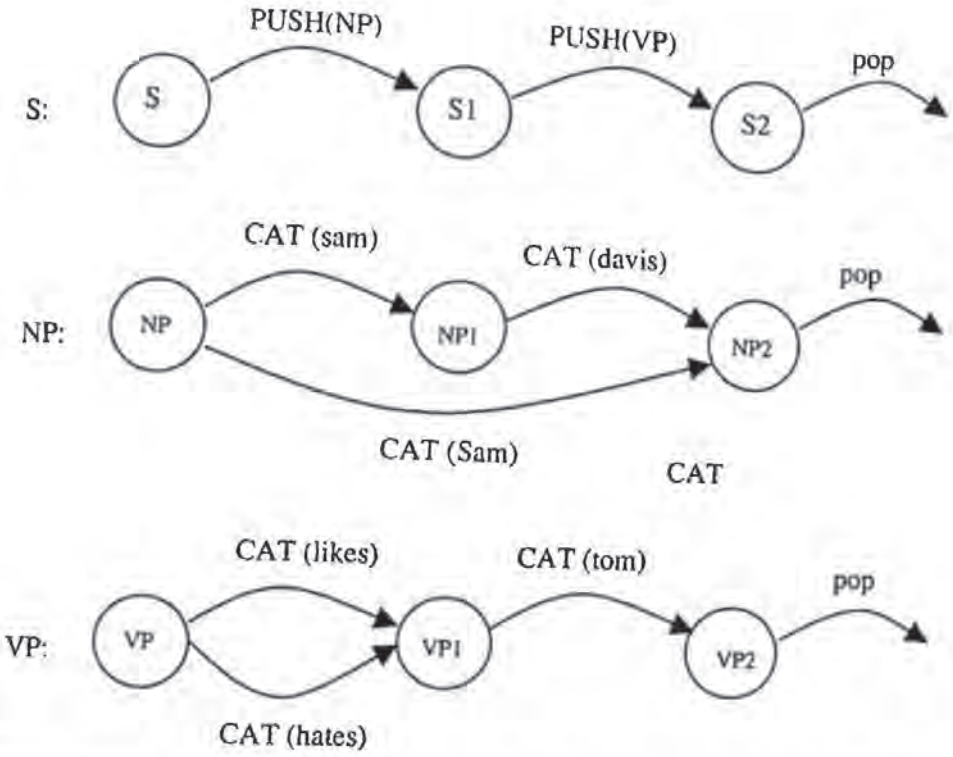


Figure 12.13 A simple RTN example with three types of arcs: CAT(x), PUSH (x), POP.

Readers should note that the same word might be under consideration by the decoder in the context of different paths and grammar states at the same time. For example, there are two word arcs CAT (Sam) in Figure 12.13. Their HMM states should be considered as distinct states in the trellis because they are in completely different grammar states. Two different states in the trellis also means that different paths going into these two states cannot be merged. Since these two partial paths will lead to different successive paths, the search decision needs to be postponed until the end of search. Therefore, when embedding HMMs into word arcs in the grammar network, the HMM state will be assigned a new state identity, although the HMM parameters (transition probabilities and output distributions) can still be shared across different grammar arcs.

Each path consists of a stack of production rules. Each element of the stack also contains the position within the production rule of the symbol that is currently being explored. The search graph (trellis) started from the initial state of CFG (state S). When the path needs to be extended, we look at the next arc (symbol in CFG) in the production. When the search enters a CAT(x) arc (terminal), the path gets extended with the terminal, and the HMM trellis computation is performed on the CAT(x) arc to match the model x against the acoustic data. When the final state of the HMM for x is reached, the search moves on via the null

transition to the destination of the CAT(x) arc. When the search enters a PUSH(x) arc, it indicates a nonterminal symbol x is encountered. In effect, the search is about to enter a subnetwork of x; the destination of the PUSH(x) arc is stored in a last-in first-out (LIFO) stack. When the search reaches a POP arc that signals the end of the current network, the control should jump back to the calling network. In other words, the search returns to the state extracted from the top of the LIFO stack. Finally, when we reach the end of the production rule at the very bottom of the stack, we have reached an accepting state in which we have seen a complete grammatical sentence. For our decoding purpose, that is the state we want to pick as the best score at the end of time frame T to get the search result.

The problem of connected word recognition by finite state or context-free grammars is that the number of states increases enormously when it is applied to more complex grammars. Moreover it remains a challenge to generate such FSGs or CFGs from a large corpus, either manually or automatically. As mentioned in Chapter 11, it is questionable whether FSG or CFG is adequate to describe natural languages or unconstrained spontaneous languages. Instead, n-gram language models are often used for natural languages or unconstrained spontaneous languages. In the next section we investigate how to integrate various n-grams into continuous speech recognition.

12.3.2. Search Space with the Unigram

The simplest n-gram is the unigram that is memory-less and depends only on the current word.

$$P(\mathbf{W}) = \prod_{i=1}^{n} P(w_i) \tag{12.11}$$

Figure 12.14 shows such a unigram grammar network. The final state of each word HMM is connected to the collector state by a null transition, with probability 1.0. The collector state is then connected to the starting state by another null transition, with transition probability equal to 1.0. For word expansion, the starting state is connected to the initial state of each word HMM by a null transition, with transition probability equal to the corresponding unigram probability. Using the collector state and starting state for word expansion allows efficient expansion because it first merges all the word-ending paths[5] (only the best one survives) before expansion. It can cut the total cross-word expansion from N^2 to N.

[5] In graph search, a partial path still under consideration is also referred to as a theory, although we will use paths instead of theories in this book.

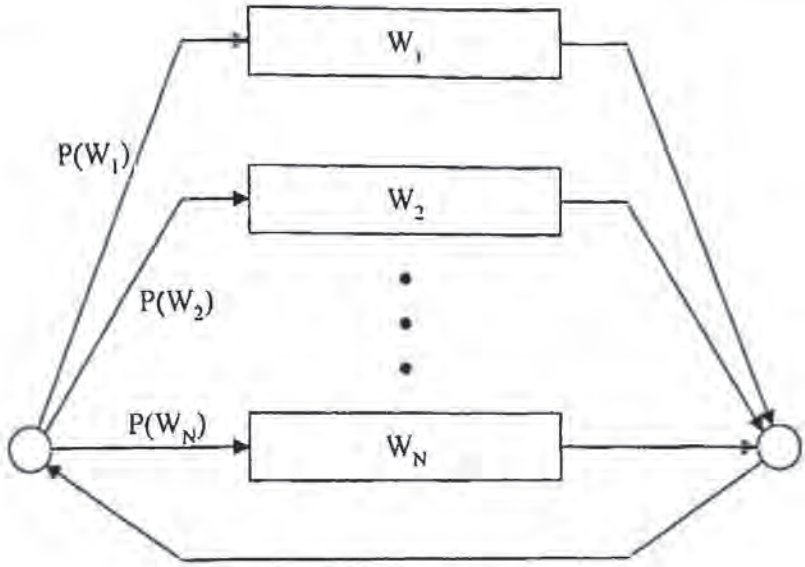


Figure 12.14 A unigram grammar network where the unigram probability is attached as the transition probability from starting state *S* to the first state of each word HMM.

12.3.3. Search Space with Bigrams

When the bigram is used, the probability of a word depends only on the immediately preceding word. Thus, the language model score is:

$$P(\mathbf{W}) = P(w_1 \mid <\text{s}>)\prod_{i=2}^{n} P(w_i \mid w_{i-1}) \tag{12.12}$$

where <s> represents the symbol of starting of a sentence.

Figure 12.15 shows a grammar network using a bigram language model. Because of the bigram constraint, the merge-and-expand framework for unigram search no longer applies here. Instead, the bigram search needs to perform expand-and-merge. Thus, bigram expansion is more expensive than unigram expansion. For a vocabulary size N, the bigram would need N^2 word-to-word transitions in comparison to N for the unigram. Each word transition has a transition probability equal to the corresponding bigram probability. Fortunately, the total number of states for bigram search is still proportional to the vocabulary size N.

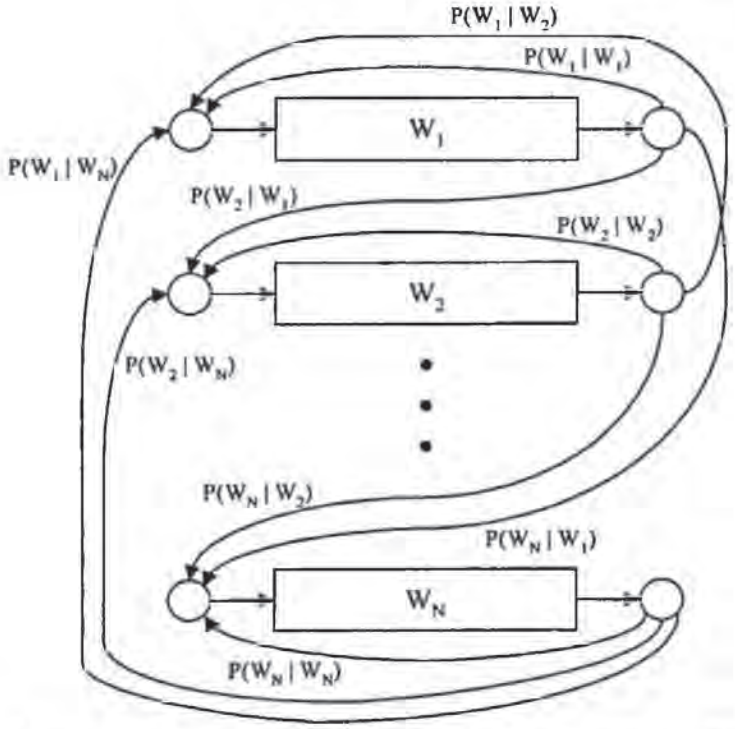


Figure 12.15 A bigram grammar network where the bigram probability $P(w_j \mid w_i)$ is attached as the transition probability from word w_i to w_j [19].

Because the search space for bigram is kept manageable, bigram search can be implemented very efficiently. Bigram search is a good compromise between efficient search and effective language models. Therefore, bigram search is arguably the most widely used search technique for unconstrained large-vocabulary continuous speech recognition. Particularly for the multiple-pass search techniques described in Chapter 13, a bigram search is often used in the first pass search.

12.3.3.1. Backoff Paths

When the vocabulary size N is large, the total bigram expansion N^2 can become computationally prohibitive. As described in Chapter 11, only a limited number of bigrams are observable in any practical corpora for a large vocabulary size. Suppose the probabilities for unseen bigrams are obtained through Katz's backoff mechanism. That is, for unseen bigram $P(w_j \mid w_i)$,

$$P(w_j \mid w_i) = \alpha(w_i)P(w_j) \tag{12.13}$$

where $\alpha(w_i)$ is the backoff weight for word w_i.

Using the backoff mechanism for unseen bigrams, the bigram expansion can be significantly reduced [12]. Figure 12.16 shows the new word expansion scheme. Instead of full bigram expansion, only observed bigrams are connected by direct word transitions with correspondent bigram probabilities. For backoff bigrams, the last state of word w_i is first connected to a central backoff node with transition probability equal to backoff weight $\alpha(w_i)$. The backoff node is then connected to the beginning of each word w_j with transition probability equal to its corresponding unigram probability $P(w_j)$. Readers should note that there are now two paths from w_i to w_j for an observed bigram $P(w_j \mid w_i)$. One is the direct link representing the observable bigram $P(w_j \mid w_i)$, and the other is the two-link backoff path representing $\alpha(w_i)P(w_j)$. For a word pair whose bigram exists, the two-link backoff path is likely to be ignored since the backoff unigram probability is almost always smaller than the observed bigram $P(w_j \mid w_i)$. Suppose there are only N_b different observable bigrams, this scheme requires $N_b + 2N$ instead of N^2 word transitions. Since under normal circumstance $N_b \ll N^2$, this backoff scheme significantly reduces the cost of word expansion.

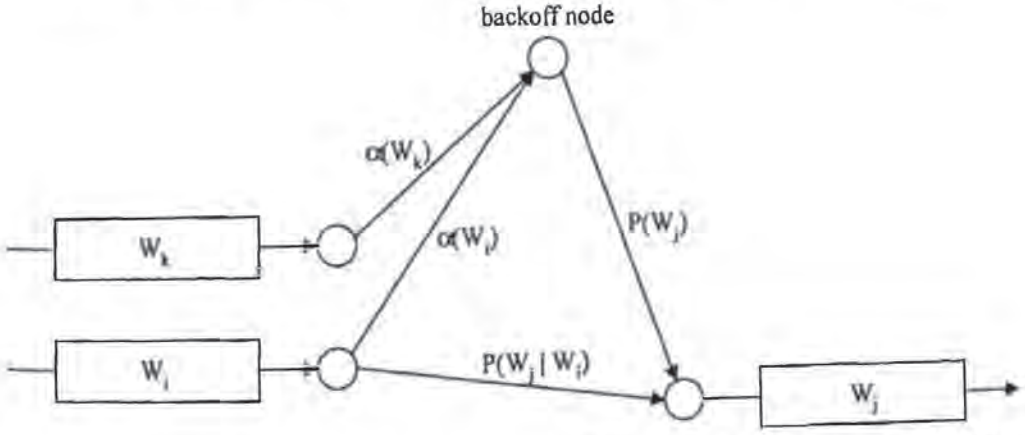


Figure 12.16 Reducing bigram expansion in a search by using the backoff node. In addition to normal bigram expansion arcs for all observed bigrams, the last state of word w_i is first connected to a central backoff node with transition probability equal to backoff weight $\alpha(w_i)$. The backoff node is then connected to the beginning of each word w_j with its corresponding unigram probability $P(w_j)$ [12].

12.3.4. Search Space with Trigrams

For a trigram language model, the language model score is:

$$P(\mathbf{W}) = P(w_1 \mid <s>)P(w_2 \mid <s>, w_1)\prod_{i=3}^{n} P(w_i \mid w_{i-2}, w_{i-1}) \tag{12.14}$$

The search space is considerably more complex, as shown in Figure 12.17. Since the equivalence grammar class is the previous two words w_i and w_j, the total number of grammar states is N^2. From each of these grammar states, there is a transition to the next word [19].

Obviously, it is very expensive to implement large-vocabulary trigram search given the complexity of the search space. It becomes necessary to dynamically generate the trigram search graph (trellis) via a graph search algorithm. The other alternative is to perform a multiple-pass search strategy, in which the first-pass search uses less detailed language models, like bigrams, to generate an n-best list or word lattice, and then a second-pass detailed search can use trigrams on a much smaller search space. Multiple-pass search strategy is discussed in Chapter 13.

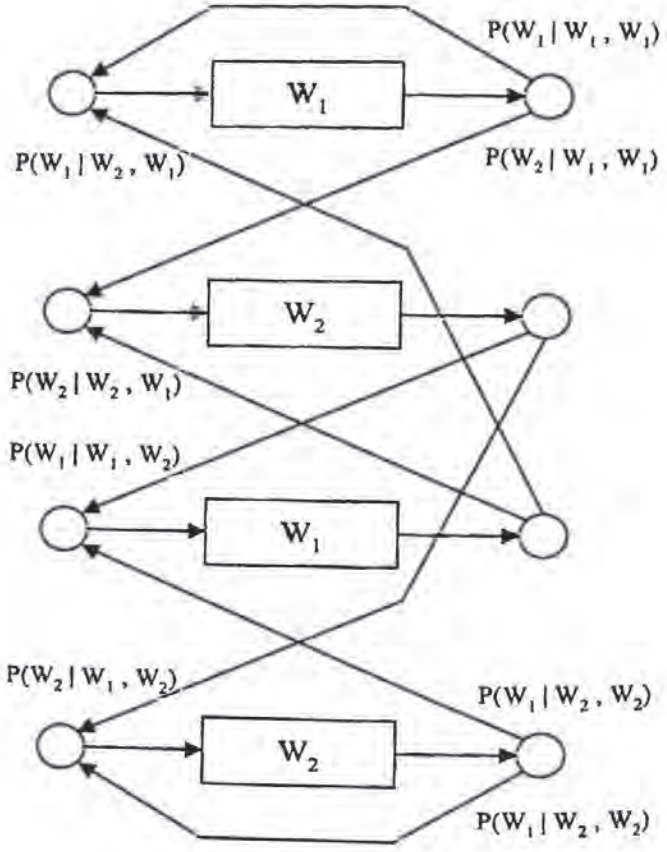


Figure 12.17 A trigram grammar network where the trigram probability $P(w_k \mid w_i, w_j)$ is attached to transition from grammar state word w_i, w_j to the next word w_k. Illustrated here is a two-word vocabulary, so there are four grammar states in the trigram network [19].

12.3.5. How to Handle Silences Between Words

In continuous speech recognition, there are unavoidable pauses (silences) between words or sentences. The pause is often referred to as silence or a non-speech event in continuous speech recognition. Acoustically, the pause is modeled by a silence model[6] that models background acoustic phenomena. The silence model is usually modeled with a topology flexible enough to accommodate a wide range of lengths, since the duration of a pause is arbitrary.

It can be argued that silences are actually linguistically distinguishable events, which contribute to prosodic and meaning representation. For example, people are likely to pause more often in phrasal boundaries. However, these patterns are so far not well understood for unconstrained natural speech (particularly for spontaneous speech). Therefore, the design of almost all automatic speech recognition systems today allows silences occurring just about anywhere between two lexical tokens or between sentences. It is relatively safe to assume that people pause a little bit between sentences to catch breath, so the silences between sentences are assumed mandatory while silences between words are optional. In most systems, silence is often modeled as a special lexicon entry with special language model probability. This special language model probability is also referred to as silence insertion penalty that is set to adjust the likelihood of inserting such an optional silence between words.

It is relatively straightforward to handle the optional silence between words. We need only to replace all the grammar states connecting words with a small network like the one shown in Figure 12.18. This arrangement is similar to that of the optional silence in training continuous speech, described in Chapter 9. The small network contains two parallel paths. One is the original null transition acting as the direct transition from one word to another, while the other path will need to go through a silence model with the silence insertion penalty attached in the transition probability before going to the next word.

One thing to clarify in the implementation of Figure 12.18 is that this silence expansion needs to be done for every grammar state connecting words. In the unigram grammar network of Figure 12.14, since there is only one collector node to connect words, the silence expansion is required only for this collector node. On the other hand, in the bigram grammar network of Figure 12.15, there is a collector node for every word before expanding to the next word. In this case, the silence expansion is required for every collector node. For a vocabulary size $|V|$, this means there are $|V|$ numbers of silence networks in the grammar search network. This requirement lies in the fact that in bigram search we cannot merge paths before expanding into the next word. Optional silence can then be regarded as part of the search effort for the previous word, so the word expansion needs to be done after finishing the optional silence. Therefore, we treat each word as having two possible pronunciations, one with the silence at the end and one without. This viewpoint integrates silence in the word pronunciation network like the example shown in Figure 12.19.

[6] Some researchers extend the context-dependent modeling to silence models. In that case, there are several silence models based on surrounding contexts.

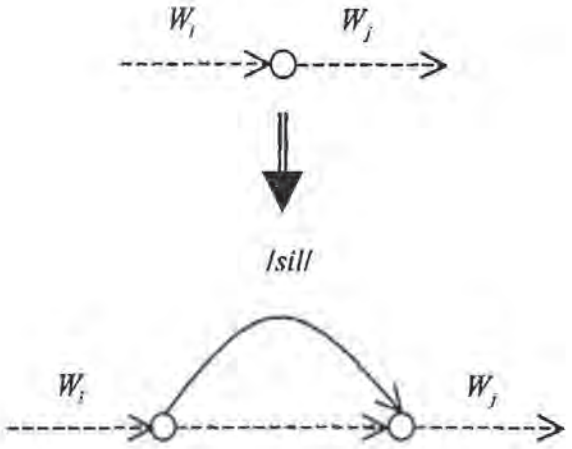


Figure 12.18 Incorporating optional silence (a non-speech event) in the grammar search network where the grammar state connecting different words is laced by two parallel paths. One is the original null transition directly from one word to the other, while the other first goes through the silence word to accommodate the optional silence.

For efficiency reasons, a single silence is sometimes used for large-vocabulary continuous speech recognition using higher order n-gram language model. Theoretically, this could be a source of pruning errors.[7] However, the error could turn out to be so small as to be negligible because there are, in general, very few pauses between word for continuous speech. On the other hand, the overhead of using multiple silences should be very minimal because it is less likely to visit those silence models at the end of words due to pruning.

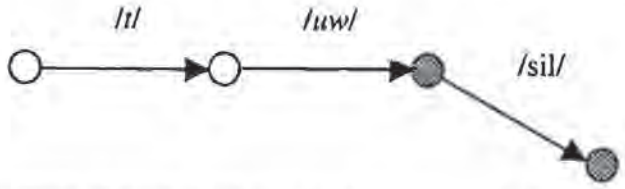


Figure 12.19 An example of treating silence as of the pronunciation network of word TWO. The shaded nodes represent possible word-ending nodes: one without silence and the other one with silence.

12.4. TIME-SYNCHRONOUS VITERBI BEAM SEARCH

When HMMs are used for acoustic models, the acoustic model score (likelihood) used in search is by definition the forward probability. That is, all possible state sequences must be considered. Thus,

[7] Speech recognition errors due to sub-optimal search or heuristic pruning are referred to as *pruning errors*, which will be described in detail in Chapter 13.

$$P(\mathbf{X} \mid \mathbf{W}) = \sum_{\text{all possible } s_0^T} P(\mathbf{X}, s_0^T \mid \mathbf{W}) \tag{12.15}$$

where the summation is to be taken over all possible state sequences S with the word sequence W under consideration. However, under the trellis framework (as in Figure 12.11), more bookkeeping must be performed since we cannot add scores with different word sequence history. Since the goal of decoding is to uncover the best word sequence, we could approximate the summation with the maximum to find the best state sequence instead. The Bayes' decision rule, Eq. (12.5), becomes

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W}) \cong \arg\max_{\mathbf{W}} \left\{ P(\mathbf{W}) \max_{s_0^T} P(\mathbf{X}, s_0^T \mid \mathbf{W}) \right\} \tag{12.16}$$

Equation (12.16) is often referred to as the *Viterbi approximation*. It can be literally translated to "the *most likely word sequence* is approximated by the *most likely state sequence*." Viterbi search is then sub-optimal. Although the search results by using forward probability and Viterbi probability could, in principle, be different, in practice this is rarely the case. We use this approximation for the rest of this chapter.

The Viterbi search has already been discussed as a solution to one of the three fundamental HMM problems in Chapter 8. It can be executed very efficiently via the same trellis framework. To briefly reiterate, the Viterbi search is a time-synchronous search algorithm that completely processes time t before going on to time $t+1$. For time t, each state is updated by the best score (instead of the sum of all incoming paths) from all states in at time $t-1$. This is why it is often called *time-synchronous Viterbi search*. When one update occurs, it also records the backtracking pointer to remember the most probable incoming state. At the end of search, the most probable state sequence can be recovered by tracing back these backtracking pointers. The Viterbi algorithm provides an optimal solution for handling nonlinear time warping between hidden Markov models and acoustic observation, word boundary detection and word identification in continuous speech recognition. This unified Viterbi search algorithm serves as the basis for all search algorithms as described in the rest of the chapter.

It is necessary to clarify the backtracking pointer for time-synchronous Viterbi search for continuous word recognition. We are generally not interested in the optimal state sequence for speech recognition.[8] Instead, we are only interested in the optimal word sequence indicated by Eq. (12.16). Therefore, we use the backtrack pointer just to remember the word history for the current path, so the optimal word sequence can be recovered at the end of search. To be more specific, when we reach the final state of a word, we create a history node containing the word identity and current time index and append this history node to the existing backtrack pointer. This backtrack pointer is then passed onto the successor node if it

[8] While we are not interested in optimal state sequences for ASR, they are very useful in deriving phonetic segmentation, which could provide important information for developing ASR systems.

is the optimal path leading to the successor node for both intra-word and inter-word transition. The side benefit of keeping this backtrack pointer is that we no longer need to keep the entire trellis during the search. Instead, we only need space to keep two successive time slices (columns) in the trellis computation (the previous time slice and the current time slice) because all the backtracking information is now kept in the backtrack pointer. This simplification is a significant benefit in the implementation of a time-synchronous Viterbi search.

Time-synchronous Viterbi search can be considered as a *breadth-first search* with dynamic programming. Instead of performing a tree search algorithm, the dynamic programming principle helps create a search graph where multiple paths leading to the same search state are merged by keeping the best path (with minimum cost). The Viterbi trellis is a representation of the search graph. Therefore, all the efficient techniques for graph search algorithms can be applied to time-synchronous Viterbi search. Although so far we have described the trellis in an explicit fashion—the whole search space needs to be explored before the optimal path can be found—it is not necessary to do so. When the search space contains an enormous number of states, it becomes impractical to pre-compile the composite HMM entirely and store it in the memory. It is preferable to dynamically build and allocate portions of the search space sufficient to search the promising paths. By using the graph search algorithm described in Section 12.1.1, only part of the entire Viterbi trellis is generated explicitly. By constructing the search space dynamically, the computation cost of the search is proportional only to the number of active hypotheses, independent of the overall size of the potential search space. Therefore, dynamically generated trellises are key to heuristic Viterbi search for efficient large-vocabulary continuous speech recognition, as described in Chapter 13.

12.4.1. The Use of Beam

Based on Chapter 8, the search space for Viterbi search is $O(NT)$ and the complexity is $O(N^2T)$, where N is the total number of HMM states and T is the length of the utterance. For large-vocabulary tasks these numbers are astronomically large even with the help of dynamic programming. In order to avoid examining the overwhelming number of possible cells in the HMM trellis, a heuristic search is clearly needed. Different heuristics generate or explore portions of the trellis in different ways.

A simple way to prune the search space for breadth-first search is the beam search described in Section 12.1.3.2. Instead of retaining all candidates (cells) at every time frame, a threshold T is used to keep only a subset of promising candidates. The state at time t with the lowest cost $D_{\min}$ is first identified. Then each state at time t with cost $> D_{\min} + T$ is discarded from further consideration before moving on to the next time frame $t+1$. The use of the beam alleviates the need to process all the cells. In practice, it can lead to substantial savings in computation with little or no loss of accuracy.

Although beam search is a simple idea, the combination of time-synchronous Viterbi and beam search algorithms produces the most powerful search strategy for large-vocabulary speech recognition. Comparing paths with equal length under a time-

synchronous search framework makes beam search possible. That is, for two different word sequences $\mathbf{W}_1$ and $\mathbf{W}_2$, the posterior probabilities $P(\mathbf{W}_1 \mid \mathbf{x}_0^t)$ and $P(\mathbf{W}_2 \mid \mathbf{x}_0^t)$ are always compared based on the same partial acoustic observation $\mathbf{x}_0^t$. This makes the comparison straightforward because the denominator $P(\mathbf{x}_0^t)$ in Eq. (12.5) is the same for both terms and can be ignored. Since the score comparison for each time frame is fair, the only assumption of beam search is that an optimal path should have a good enough partial-path score for each time frame to survive under beam pruning.

The time-synchronous framework is one of the aspects of Viterbi beam search that is critical to its success. Unlike the time-synchronous framework, time-asynchronous search algorithms such as stack decoding require the normalization of likelihood scores over feature streams of different time lengths. This, as we will see in Section 12.5, is the Achilles' heel of that approach.

The straightforward time-synchronous Viterbi beam search is ineffective in dealing with the gigantic search space of high perplexity tasks. However, with a better understanding of the linguistic search space and the advent of techniques for obtaining n-best lists from time-synchronous Viterbi search, described in Chapter 13, time-synchronous Viterbi beam search has turned out to be surprisingly successful in handling tasks of all sizes and all different types of grammars, including FSG, CFG, and n-gram [2, 14, 18, 28, 34, 38, 44]. Therefore, it has become the predominant search strategy for continuous speech recognition.

12.4.2. Viterbi Beam Search

To explain the time-synchronous Viterbi beam search in a formal way [31], we first define some quantities:

$D(t; s_t; w) \equiv$ total cost of the best path up to time t that ends in state s_t of grammar word state w.

$h(t; s_t; w) \equiv$ backtrack pointer for the best path up to time t that ends in state s_t of grammar word state w.

Readers should be aware that w in the two quantities above represents a grammar word state in the search space. It is different from just the word identity since the same word could occur in many different language model states, as in the trigram search space shown in Figure 12.17.

There are two types of dynamic programming (DP) transition rules [30], namely intra-word and inter-word transition. The intra-word transition is just like the Viterbi rule for HMMs and can be expressed as follows:

$$D(t; s_t; w) = \min_{s_{t-1}} \{ d(\mathbf{x}_t, s_t \mid s_{t-1}; w) + D(t-1; s_{t-1}; w) \} \tag{12.17}$$

$$h(t; s_t; w) = h(t-1, b_{\min}(t; s_t; w); w) \tag{12.18}$$

where $d(\mathbf{x}_t, s_t \mid s_{t-1}; w)$ is the cost associated with taking the transition from state s_{t-1} to state s_t while generating output observation $\mathbf{x}_t$, and $b_{\min}(t; s_t; w)$ is the optimal predecessor state of cell $D(t; s_t; w)$. To be specific, they can be expressed as follows:

$$d(\mathbf{x}_t, s_t \mid s_{t-1}; w) = -\log P(s_t \mid s_{t-1}; w) - \log P(\mathbf{x}_t \mid s_t; w) \tag{12.19}$$

$$b_{\min}(t; s_t; w) = \arg\min_{s_{t-1}} \left\{ d(\mathbf{x}_t, s_t \mid s_{t-1}; w) + D(t-1; s_{t-1}; w) \right\} \tag{12.20}$$

The inter-word transition is basically a null transition without consuming any observation. However, it needs to deal with creating a new history node for the backtracking pointer. Let's define $F(w)$ as the final state of word HMM w and $I(w)$ as the initial state of word HMM w. Moreover, state η is denoted as the pseudo initial state. The inter-word transition can then be expressed as follows:

$$D(t; \eta; w) = \min_{v} \left\{ \log P(w \mid v) + D(t; F(v); v) \right\} \tag{12.21}$$

$$h(t; \eta; w) = \left\langle v_{\min}, t \right\rangle :: h(t, F(v_{\min}); v_{\min}) \tag{12.22}$$

where $v_{\min} = \arg\min_{v} \left\{ \log P(w \mid v) + D(t; F(v); v) \right\}$ and :: is a link appending operator.

The time-synchronous Viterbi beam search algorithm assumes that all the intra-word transitions are evaluated before inter-word null transitions take place. The same time index is used intentionally for inter-word transition since the null language model state transition does not consume an observation vector. Since the initial state $I(w)$ for word HMM w could have a self-transition, the cell $D(t; I(w); w)$ might already have an active path. Therefore, we need to perform the following check to advance the inter-word transitions.

$$\begin{aligned} &\text{if } D(t; \eta; w) < D(t; I(w); w) \\ &\quad D(t; I(w); w) = D(t; \eta; w) \text{ and } h(t; I(w); w) = h(t; \eta; w) \end{aligned} \tag{12.23}$$

The time-synchronous Viterbi beam search can be summarized as in Algorithm 12.6. For large-vocabulary speech recognition, the experimental results show that only a small percentage of the entire search space (the beam) needs to be kept for each time interval t without increasing error rates. Empirically, the beam size has typically been found to be between 5% and 10% of the entire search space. In Chapter 13 we describe strategies of using different level of beams for more effectively pruning.

12.5. STACK DECODING (A* SEARCH)

If some reliable heuristics are available to guide the decoding, the search can be done in a depth-first fashion around the best path early on, instead of wasting efforts on unpromising paths via the time-synchronous beam search. Stack decoding represents the best attempt to

ALGORITHM 12.6: TIME-SYNCHRONOUS VITERBI BEAM SEARCH

Step 1: Initialization: For all the grammar word states w which can start a sentence,

$$D(0;I(w);w) = 0$$

$$h(0;I(w);w) = null$$

Step 2: Induction: For time $t = 1$ to T do

For all active states do

Intra-word transitions according to Eq. (12.17) and (12.18)

$$D(t;s_t;w) = \min_{s_{t-1}} \left\{ d(\mathbf{x}_t, s_t \mid s_{t-1};w) + D(t-1;s_{t-1};w) \right\}$$

$$h(t;s_t;w) = h(t-1, b_{min}(t;s_t;w);w)$$

For all active word-final states do

Inter-word transitions according to Eq. (12.21), (12.22) and (12.23)

$$D(t;\eta;w) = \min_{v} \left\{ \log P(w \mid v) + D(t;F(v);v) \right\}$$

$$h(t;\eta;w) = \left\langle v_{min}, t \right\rangle :: h(t, F(v_{min});v_{min})$$

if $D(t;\eta;w) < D(t;I(w);w)$

$$D(t;I(w);w) = D(t;\eta;w) \text{ and } h(t;I(w);w) = h(t;\eta;w)$$

Pruning: Find the cost for the best path and decide the beam threshold

Prune unpromising hypotheses

Step 3: Termination: Pick the best path among all the possible final states of grammar at time T. Obtain the optimal word sequence according to the backtracking pointer $h(t;\eta;w)$

use A* search instead of time-synchronous beam search for continuous speech recognition. Unfortunately, as we will discover in this section, such a heuristic function $h(\bullet)$ (defined in Section 12.1.3) is very difficult to attain in continuous speech recognition, so search algorithms based on A* search are in general less efficient than time-synchronous beam search.

Stack decoding is a variant of the heuristic A* search based on the forward algorithm, where the evaluation function is based on the forward probability. It is a tree search algorithm, which takes a slightly different viewpoint than the time-synchronous Viterbi search. Time-synchronous beam search is basically a breadth-first search, so it is crucial to control the number of all possible language model states as described in Section 12.3. In a typical large-vocabulary Viterbi search with n-gram language models, this number is determined by the equivalent classes of language model histories. On the other hand, stack decoding as a tree search algorithm treats the search as a task for finding a path in a tree whose branches correspond to words in the vocabulary V, non-terminal nodes correspond to incomplete sentences, and terminal nodes correspond to complete sentences. The search tree has a constant branching factor of $|V|$, if we allow every word to be followed by every word. Figure 12.20 illustrates such a search tree for a vocabulary with three words [19].

An important advantage of stack decoding is its consistency with the forward-backward training algorithm. Viterbi search is a graph search, and paths cannot be easily summed because they may have different word histories. In general, the Viterbi search finds the optimal state sequence instead of optimal word sequence. Therefore, Viterbi approximation is necessary to make the Viterbi search feasible, as described in Section 12.4. Stack decoding is a tree search, so each node has a unique history, and the forward algorithm can be used within word model evaluation. Moreover, all possible beginning and ending times (shaded areas in Figure 12.21) are considered [24]. With stack decoding, it is possible to use an objective function that searches for the optimal word string, rather than the optimal state sequence. Furthermore, it is in principle natural for stack decoding to accommodate long-range language models if the heuristics can guide the search to avoid exploring the overwhelmingly large unpromising grammar states.

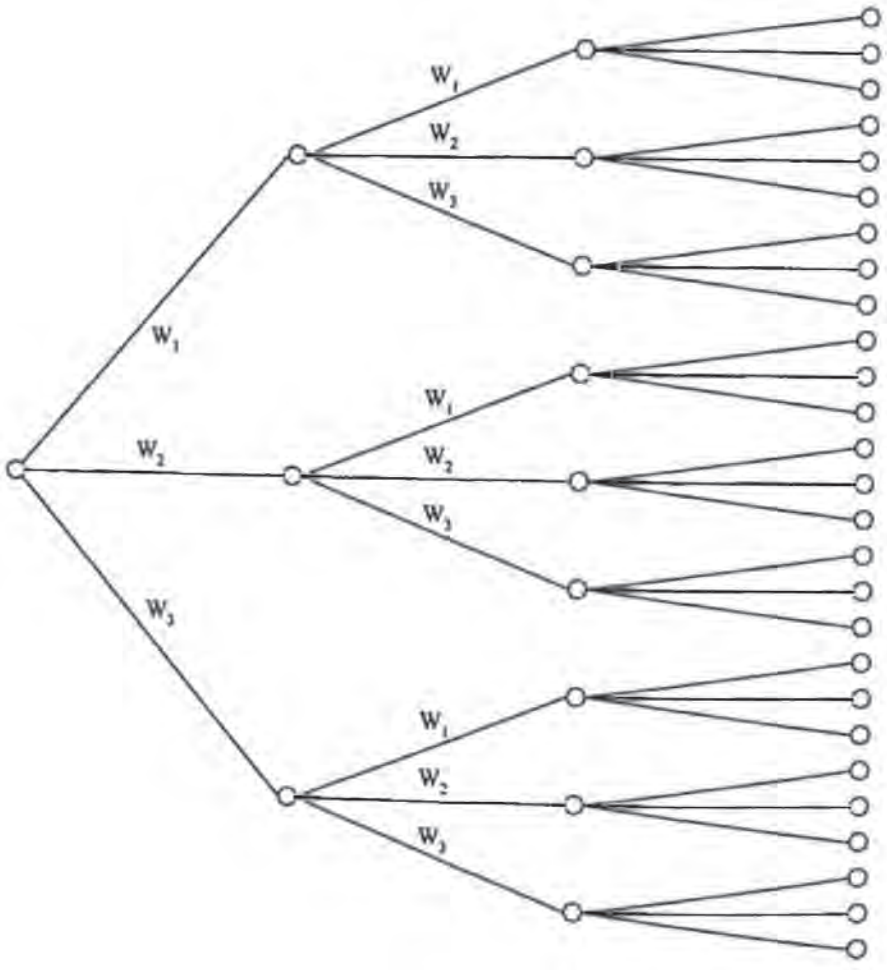


Figure 12.20 A stack decoding search tree for a vocabulary size of three [19].

By formulating stack decoding in a tree search framework, the graph search algorithms described in Section 12.1 can be directly applied to stack decoding. Obviously, blind-search methods, like depth-first and breadth-first search, that do not take advantage of the goodness measurement of how close we are getting to the goal, are usually computationally infeasible in practical speech recognition systems. A* search is clearly attractive for speech recognition, given the hope of a sufficient heuristic function to guide the tree search in a favorable direction without exploring too many unpromising branches and nodes. In contrast to the Viterbi search, it is not time-synchronous and extends paths of different lengths.

The search begins by adding all possible one-word hypotheses to the *OPEN* list. Then the best path is removed from the *OPEN* list, and all paths from it are extended, evaluated, and placed back in the *OPEN* list. This search continues until a complete path that is guaranteed to be better than all paths in the *OPEN* list has been found.

Unlike Viterbi search, where the acoustic probabilities being compared are always based on the same partial input, it is necessary to compare the goodness of partial paths of different lengths to direct the A* tree search. Moreover, since stack decoding is done asynchronously, we need an effective mechanism to determine when to end a phone/word evaluation and move on to the next phone/word. Therefore, the heart and soul of the stack decoding are clearly in

1. Finding an effective and efficient heuristic function for estimating the future remaining input feature stream and
2. Determining when to extend the search to the next word/phone.

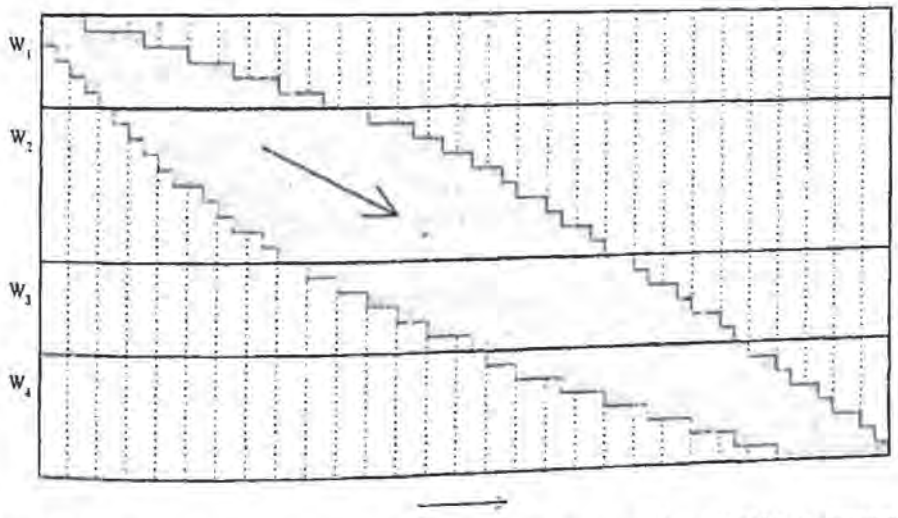


Figure 12.21 The forward trellis space for stack decoding. Each grid point corresponds to a trellis cell in the forward computation. The shaded area represents the values contributing to the computation of the forward score for the optimal word sequence $w_1, w_2, w_3, \ldots$ [24].

In the following section we describe these two critical components. Readers will note that the solutions to these two issues are virtually the same—using a normalization scheme to compare paths of different lengths.

12.5.1. Admissible Heuristics for Remaining Path

The key issue in heuristic search is the selection of an evaluation function. As described in Section 12.1.3, the heuristic function of the path H_N going through node N includes the cost up to the node and the estimate of the cost to the target node from node N. Suppose path H_N is going through node N at time t; then the evaluation for path H_N can be expressed as follows:

$$f(H_N^t) = g(H_N^t) + h(H_N^{t,T}) \tag{12.24}$$

where $g(H_N^t)$ is the evaluation function for the partial path of H_N up to time t, and $h(H_N^{t,T})$ is the heuristic function of the remaining path from $t+1$ to T for path H_N. The challenge for stack decoders is to devise an admissible function for $h(\bullet)$.

According to Section 12.1.3.1, an admissible heuristic function is one that always underestimates the true cost of the remaining path from $t+1$ to T for path H_N. A trivially admissible function is the zero function. In this case, it results in a very large OPEN list. In addition, since the longer paths tend to have higher cost because of the gradually accumulated cost, the search is likely to be conducted in a breadth-first fashion, which functions very much like a plain Viterbi search. The evaluation function $g(\bullet)$ can be obtained easily by using the HMM forward score as the true cost up to current time t. However, how can we find an admissible heuristic function $h(\bullet)$? We present the basic concept here [19, 35].

The goal of $h(\bullet)$ is to find the expected cost for the remaining path. If we can obtain the expected cost per frame ψ for the remaining path, the total expected cost, $(T-t)*\psi$, is simply the product of ψ and the length of the remaining path. One way to find such expected cost per frame is to gather statistics empirically from training data.

1. After the final training iteration, perform Viterbi forced alignment[9] with each training utterance to get an optimal time alignment for each word.
2. Randomly select an interval to cover the number of words ranging from two to ten. Denote this interval as $[i \ldots j]$.
3. Compute the average acoustic cost per frame within this selected interval according to the following formula and save the value in a set Λ:

[9] Viterbi forced alignment means that the Viterbi is performed on the HMM model constructed from the known word transcription. The term "forced" is used because the Viterbi alignment is forced to be performed on the correct model. Viterbi forced alignment is a very useful tool in spoken language processing as it can provide the optimal state-time alignment with the utterances. This detailed alignment can then be used for different purposes, including discriminant training, concatenated speech synthesis, etc.

$$\frac{-1}{j-i}\log P(\mathbf{x}_i^j \mid \mathbf{w}_{i...j}) \tag{12.25}$$

where $\mathbf{w}_{i...j}$ is the word string corresponding to interval $[i...j]$.

4. Repeat Steps 2 and 3 for the entire training set.

5. Define $\psi_{\min}$ and ψ_{avg} as the minimum and average value found in set Λ.

Clearly, $\psi_{\min}$ should be a good under-estimate of the expected cost per frame for the future unknown path. Therefore, the heuristic function $h(H_N^{t,T})$ can be derived as:

$$h(H_N^{t,T}) = (T-t)\psi_{\min} \tag{12.26}$$

Although $\psi_{\min}$ is obtained empirically, stack decoding based on Eq. (12.26) will generally find the optimal solution. However, the search using $\psi_{\min}$ usually runs very slowly, since Eq. (12.26) always under-estimates the true cost for any portion of speech. In practice, a heuristic function like ψ_{avg} that may over-estimate has to be used to prune more hypotheses. This speeds up the search at the expense of possible search errors, because ψ_{avg} should represent the average cost per frame for any portion of speech. In fact, there is an argument that one might be able to use a heuristic function even more than ψ_{avg}. The argument is that ψ_{avg} is derived from the correct path (training data) and the average cost per frame for all paths during search should be more than ψ_{avg} because the paths undoubtedly include correct and incorrect ones.

12.5.2. When to Extend New Words

Since stack decoding is executed asynchronously, it becomes necessary to detect when a phone/word ends, so that the search can extend to the next phone/word. If we have a cost measure that indicates how well an input feature vector of any length matches the evaluated model sequence, this cost measure should drop slowly for the correct phone/word and rise sharply for an incorrect phone/word. In order to do so, it implies we must be able to compare hypotheses of different lengths.

The first thing that comes to mind for this cost measure is simply the forward cost $-\log P(\mathbf{x}_1^t, s_t \mid w_1^k)$, which represents the likelihood of producing acoustic observation $\mathbf{x}_1^t$ based on word sequence w_1^k and ending at state s_t. However, it is definitely not suitable because it is deemed to be smaller for a shorter acoustic input vector. This causes the search to almost always prefer short phones/words, resulting in many insertion errors. Therefore, we must derive some normalized score that satisfies the desired property described above. The normalized cost $\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k)$ can be represented as follows [6, 24]:

$$\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k) = -\log\left[\frac{P(\mathbf{x}_1^t, s_t \mid w_1^k)}{\gamma^t}\right] = -\log\left[P(\mathbf{x}_1^t, s_t \mid w_1^k)\right] + t\log\gamma \tag{12.27}$$

where γ $(0<\gamma<1)$ is a constant normalization factor.

Suppose the search is now evaluating a particular word w_k; we can define $\hat{C}_{\min}(t)$ as the minimum cost for $\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k)$ for all the states of w_k, and $\alpha_{\max}(t)$ as the maximum forward probability for $P(\mathbf{x}_1^t, s_t \mid w_1^k)$ for all the states of w_k. That is,

$$\hat{C}_{\min}(t) = \min_{s_t \in w_k} \left[\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k) \right] \tag{12.28}$$

$$\alpha_{\max}(t) = \max_{s_t \in w_k} \left[P(\mathbf{x}_1^t \mid w_1^k, s_t) \right] \tag{12.29}$$

We want $\hat{C}_{\min}(t)$ to be near 0 just as long as the phone/word we are evaluating is the correct one and we have not gone beyond its end. On the other hand, if the phone/word we are evaluating is the incorrect one or we have already passed its end, we want the $\hat{C}_{\min}(t)$ to be rising sharply. Similar to the procedure of finding the admissible heuristic function, we can set the normalized factor γ empirically during training so that $\hat{C}_{\min}(T) = 0$ when we know the correct word sequence $\mathbf{W}$ that produces acoustic observation sequence $\mathbf{x}_1^T$. Based on Eq. (12.27), γ should be set to:

$$\gamma = \sqrt[T]{\alpha_{\max}(T)} \tag{12.30}$$

Figure 12.22 shows a plot of $\hat{C}_{\min}(t)$ as a function of time for correct match. In addition, the cost for the final state $FS(w_k)$ of word w_k, $\hat{C}(\mathbf{x}_1^t, s_t = FS(w_k) \mid w_1^k)$, which is the score for w_k-ending path, is also plotted. There should be a valley centered around 0 for $\hat{C}(\mathbf{x}_1^t, s_t = FS(w_k) \mid w_1^k)$, which indicates the region of possible ending time for the correct phone/word. Sometimes a stretch of acoustic observations may match better than the average cost, pushing the curve below 0. Similarly, a stretch of acoustic observations may match worse than the average cost, pushing the curve above 0.

There is an interesting connection between the normalized factor γ and the heuristic estimate of the expected cost per frame, ψ, defined in Eq. (12.25). Since the cost is simply the logarithm on the inverse posterior probability, we get the following equation:

$$\psi = \frac{-1}{T} \log P(\mathbf{x}_1^T \mid \hat{\mathbf{W}}) = -\log \left[\alpha_{\max}(T)^{1/T} \right] = -\log \gamma \tag{12.31}$$

Equation (12.31) reveals that these two quantities are basically the same estimate. In fact, if we subtract the heuristic function $f(H_N^t)$ defined in Eq. (12.24) by the constant $\log(\gamma^T)$, we get exactly the same quantity as the one defined in Eq. (12.27). Decisions on which path to extend first based on the heuristic function and when to extend the search to the next word/phone are basically centered on comparing partial theories with different lengths. Therefore, the normalized cost $\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k)$ can be used for both purposes.

Based on the connection we have established, the heuristic function, $f(H_N^t)$, which estimates the goodness of a path is simply replaced by the normalized evaluation function $\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k)$. If we plot the un-normalized cost $C(\mathbf{x}_1^t, s_t \mid w_1^k)$ for the optimal path and other

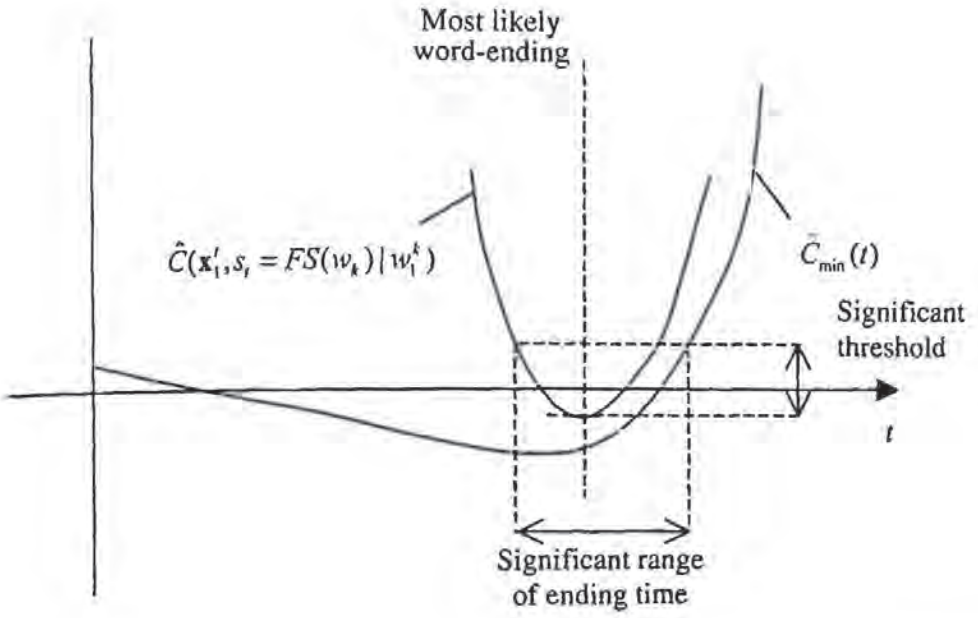


Figure 12.22 $\hat{C}_{\min}(t)$ and $\hat{C}(\mathbf{x}_1^t, s_t = FS(w_k) \mid w_1^k)$ as functions of time t. The valley region represents possible ending times for the correct phone/word.

competing paths as the function time t, the cost values increase as paths get longer (illustrated in Figure 12.23) because every frame adds some non-negative cost to the overall cost. It is clear that using un-normalized cost function $C(\mathbf{x}_1^t, s_t \mid w_1^k)$ generally results in a breadth-first search. What we want is an evaluation that decreases slightly along the optimal path, and hopefully increases along other competing paths. Clearly, the normalized cost function $\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k)$ fulfills this role, as shown in Figure 12.24.

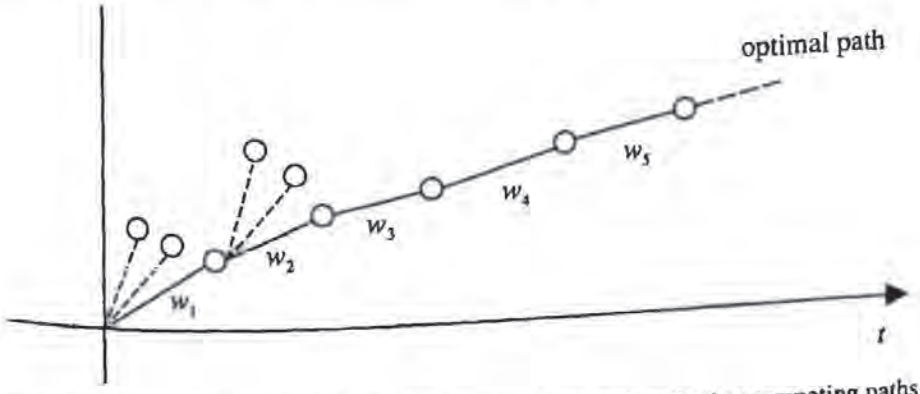


Figure 12.23 Unnormalized cost $C(\mathbf{x}_1^t, s_t \mid w_1^k)$ for optimal path and other competing paths as a function of time.

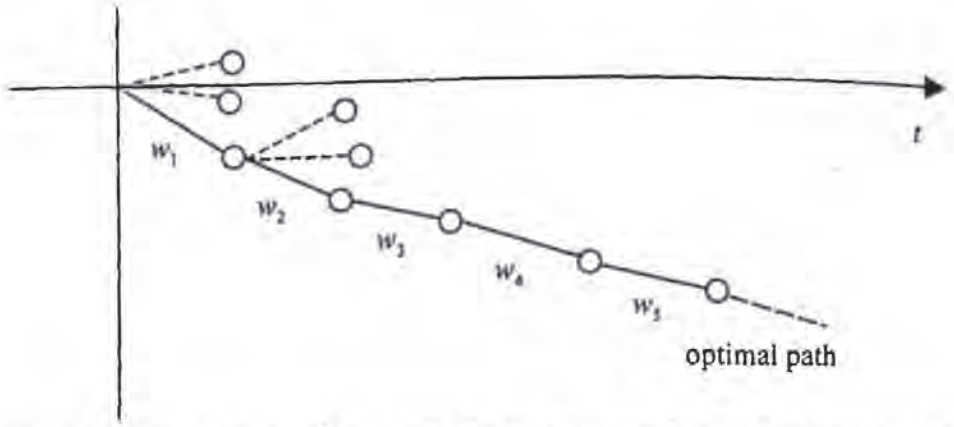


Figure 12.24 Normalized cost $\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^t)$ for the optimal path and other competing paths as a function of time.

Equation (12.30) is a context-less estimation of the normalized factor, which is also referred to as zero-order estimate. To improve the accuracy of the estimate, you can use context-dependent higher-order estimates like [24]:

$$\gamma_t = \gamma(\mathbf{x}_t) \qquad \text{first-order estimate}$$
$$\gamma_t = \gamma(\mathbf{x}_t, \mathbf{x}_{t-1}) \qquad \text{second-order estimate}$$
$$\gamma_t = \gamma(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-N+1}) \qquad n\text{-order estimate}$$

Since the normalized factor γ is estimated from the training data that is also used to train the parameters of the HMMs, the normalized factor γ_t tends to be an overestimate. As a result, $\alpha_{\max}(t)$ might rise slowly for test data even when the correct phone/word model is evaluated. This problem is alleviated by introducing some other scaling factor $\delta < 1$ so that $\alpha_{\max}(t)$ falls slowly for test data for when evaluating the correct phone/word model. The best solution for this problem is to use an independent data set other than the training data to derive the normalized factor γ_t.

12.5.3. Fast Match

Even with an efficient heuristic function and mechanism to determine the ending time for a phone/word, stack decoding could still be too slow for large-vocabulary speech recognition tasks. As described in Section 12.5.1, an effective underestimated heuristic function for the remaining portion of speech is very difficult to derive. On the other hand, a heuristic estimate for the immediate short segment that usually corresponds to the next phone or word may be feasible to attain. In this section, we describe the fast-match mechanism that reduces phone/word candidates for detailed match (expansion).

In asynchronous stack decoding, the most expensive step is to extend the best subpath. For a large-vocabulary search, it implies the calculation of $P(\mathbf{x}_t^{t+k} \mid w)$ over the entire vocabulary size $|V|$. It is desirable to have a fast computation to quickly reduce the possible

words starting at a given time t to reduce the search space. This process is often referred to as *fast match* [15, 35]. In fact, fast match is crucial to stack decoding, of which it becomes an integral part. Fast match is a method for the rapid computation of a list of candidates that constrain successive search phases. The expensive *detailed match* can then be performed after fast match. In this sense, fast match can be regarded as an additional pruning threshold to meet before new word/phone can be started.

Fast match, by definition, needs to use only a small amount of computation. However, it should also be accurate enough not to prune away any word/phone candidates that participate in the best path eventually. Fast match is, in general, characterized by the approximations that are made in the acoustic/language models in order to reduce computation. There is an obvious trade-off between these two objectives. Fortunately, many systems [15] have demonstrated that one needs to sacrifice very little accuracy in order to speed up the computation considerably.

Similar to *admissibility* in A* search, there is also an *admissibility* property in fast match. A fast match method is called admissible if it never prunes away the word/phone candidates that participate in the optimal path. In other words, a fast match is admissible if the recognition errors that appear in a system using the fast match followed by a detailed match are those that would appear if the detailed match were carried out for all words/phones in the vocabulary. Since fast match can be applied to either word or phone level, as we describe in the next section, we explain the admissibility for the case of word-level fast match for simplicity. The same principle can be easily extended to phone-level fast match.

Let V be the vocabulary and $C(\mathbf{X} \mid w)$ be the cost of a detailed match between input $\mathbf{X}$ and word w. Now $F(\mathbf{X} \mid w)$ is an estimator of $C(\mathbf{X} \mid w)$ that is accurate enough and fast to compute. A word list selected by fast match estimator can be attained by first computing $F(\mathbf{X} \mid w)$ for each word w of the vocabulary. Suppose w_b is the word for which the fast match has a minimum cost value:

$$w_b = \arg\min_{w \in V} F(\mathbf{X} \mid w) \tag{12.32}$$

After computing $C(\mathbf{X} \mid w_b)$, the detailed match cost for w_b, we form the fast match word list, Λ, from the word w in the vocabulary such that $F(\mathbf{X} \mid w)$ is no greater than $C(\mathbf{X} \mid w_b)$. In other words,

$$\Lambda = \{w \in V \mid F(\mathbf{X} \mid w) \le C(\mathbf{X} \mid w_b)\} \tag{12.33}$$

Similar to the admissibility condition for A* search [3, 33], the fast match estimator $F(\bullet)$ conducted in the way described above is admissible if and only if $F(\mathbf{X} \mid w)$ is always an under-estimator (lower bound) of detailed match $C(\mathbf{X} \mid w)$. That is,

$$F(\mathbf{X} \mid w) \le C(\mathbf{X} \mid w) \qquad \forall \mathbf{X}, w \tag{12.34}$$

The proof is straightforward. If the word w_c has a lower detailed match cost $C(\mathbf{X} \mid w_c)$, you can prove that it must be included in the fast match list Λ because

$$C(\mathbf{X} \mid w_c) \le C(\mathbf{X} \mid w_b) \text{ and } F(\mathbf{X} \mid w_c) \le C(\mathbf{X} \mid w_c) \Rightarrow F(\mathbf{X} \mid w_c) \le C(\mathbf{X} \mid w_b)$$

Therefore, based on the definition of Λ, $w_c \in \Lambda$.

Now the task is to find an admissible fast match estimator. Bahl et al. [6] proposed one fast match approximation for discrete HMMs. As we will see later, this fast match approximation is indeed equivalent to a simplification of the HMM structure. Given the HMM for word w and an input sequence x_1^T of codebook symbols describing the input signal, the probability that the HMM w produces the VQ sequence x_1^T is given by (according to Chapter 8):

$$P(x_1^T \mid w) = \sum_{s_1, s_2, \ldots s_T} \left[P_w(s_1, s_2, \ldots s_T) \prod_{i=1}^{T} P_w(x_i \mid s_i) \right] \tag{12.35}$$

Since we often use Viterbi approximation instead of the forward probability, the equation above can be approximated by:

$$P(x_1^T \mid w) \cong \max_{s_1, s_2, \ldots s_T} \left[P_w(s_1, s_2, \ldots s_T) \prod_{i=1}^{T} P_w(x_i \mid s_i) \right] \tag{12.36}$$

The detailed match cost $C(\mathbf{X} \mid w)$ can now be represented as:

$$C(\mathbf{X} \mid w) = \min_{s_1, s_2, \ldots s_T} \left\{ -\log \left[P_w(s_1, s_2, \ldots s_T) \prod_{i=1}^{T} P_w(x_i \mid s_i) \right] \right\} \tag{12.37}$$

Since the codebook size is finite, it is possible to compute, for each model w, the highest output probability for every VQ label c among all states s_k in HMM w. Let's define $m_w(c)$ to be the following:

$$m_w(c) = \max_{s_k \in w} P_w(c \mid s_k) = \max_{s_k \in w} b_k(c) \tag{12.38}$$

We can further define the $q_{\max}(w)$ as the maximum state sequence with respect to T, i.e., the maximum probability of any complete path in HMM w.

$$q_{\max}(w) = \max_{T} \left[P_w(s_1, s_2, \ldots s_T) \right] \tag{12.39}$$

Now let's define the fast match estimator $F(\mathbf{A} \mid w)$ as the following:

$$F(\mathbf{X} \mid w) = -\log \left[q_{\max}(w) \prod_{i=1}^{T} m_w(x_i) \right] \tag{12.40}$$

It is easy to show the fast match estimator $F(\mathbf{X} \mid w) \le C(\mathbf{X} \mid w)$ is admissible based on Eq. (12.38) to Eq. (12.40).

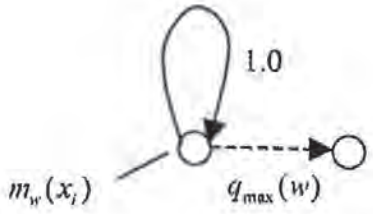


Figure 12.25 The equivalent one-state HMM corresponding to fast match computation defined in Eq. (12.40) [15].

The fast match estimator defined in Eq. (12.40) requires $T+1$ additions for a vector sequence of length T. The operation can be viewed as equivalent to the forward computation with a one-state HMM of the form shown in Figure 12.25. This correspondence can be interpreted as a simplification of the original multiple-state HMM into such a one-state HMM. It thus explains why fast match can be computed much faster than detailed match. Readers should note that this HMM is not actually a true HMM by strict definition, because the output probability distribution $m_w(c)$ and the transition probability distribution do not add up to one.

The fast match computation defined in Eq. (12.40) discards the sequence information with the model unit since the computation is independent of the order of input vectors. Therefore, one needs to decide the acoustic unit for fast match. In general, the longer the unit, the faster the computation is. and, therefore, the smaller the under-estimated cost $F(\mathbf{X}|w)$ is. It thus becomes a trade-off between accuracy and speed.

Now let's analyze the real speedup by using fast match to reduce the vocabulary V to the list Λ, followed by the detailed match. Let $|V|$ and $|\Lambda|$ be the sizes for the vocabulary V and the fast match short list Λ. Suppose t_f and t_d are the times required to compute one fast match score and one detailed match score for one word, respectively. Then, the total time required for the fast match followed by the detailed match is $t_f|V|+t_d|\Lambda|$, whereas the time required in doing the detailed match alone for the entire vocabulary is $t_d|V|$. The speed-up ratio is then given as follows:

$$\frac{1}{\left(\frac{t_f}{t_d}+\frac{|\Lambda|}{|V|}\right)} \tag{12.41}$$

We need t_f to be much smaller than t_d and $|\Lambda|$ to be much smaller than $|V|$ to have a significant speed-up using fast match. Using our admissible fast match estimator in Eq. (12.40), the time complexity of the computation for $F(\mathbf{X}|w)$ is T instead of N^2T for $C(\mathbf{X}|w)$, where N is the number of states in the detailed acoustic model. Therefore, the t_f/t_d saving is about N^2.

In general, in order to make $|\Lambda|$ much smaller than $|V|$, one needs a very accurate fast match estimator that could result in $t_f \approx t_d$. This is why we often relax the constraint of admissibility, although it is a nice principle to adhere to. In practice, most real-time speech recognition systems don't necessarily obey the admissibility principle with the fast match. For example, Bahl et al. [10], Laface et al., [22] and Roe et al., [36] used several techniques

to construct off-line groups of acoustically similar words. Armed with this grouping, they can use an aggressive fast match to select only a very short list of words, and words acoustically similar to the words in this list are added to form the short word list Λ for further detailed match processing. By doing so, they are able to report a very efficient fast match method that misses the correct word only 2% of the time. When non-admissible fast match is used, one needs to minimize the additional search error introduced by fast match empirically.

Bahl et al. [6] use a one-state HMM as their fast match units and a tree-structure lexicon similar to the lexical tree structures introduced in Chapter 13 to construct the short word list Λ for next-word expansion in stack decoding. Since the fast match tree search is also done in an asynchronous way, the ending time of each phone is detected using normalized scores similar to those described in Section 12.5.2. It is based on the same idea that this normalized score rises slowly for the correct phone, while it drops rapidly once the end of phone is encountered (so the model is starting to go toward the incorrect phones). During the asynchronous lexical tree search, the unpromising hypotheses are also pruned away by a pruning threshold that is constantly changing once a complete hypothesis (a leaf node) is obtained. On a 20,000-word dictation task, such a fast match scheme was about 100 times faster than detailed match and achieved real-time performance on a commercial workstation with only 0.34% increase in the word error rate being introduced by the fast match process.

12.5.4. Stack Pruning

Even with efficient heuristic functions, the mechanism to determine the ending time for phone/word, and fast match, stack decoding might still be too slow for large-vocabulary speech recognition tasks. A beam within the stack, which saves only a small number of promising hypotheses in the *OPEN* list, is often used to reduce search effort. This *stack pruning* is very similar to beam search. A predetermined threshold ϵ is used to eliminate hypotheses whose cost value is much worse than the best path so far.

Both fast match and stack pruning could introduce search errors where the eventual optimal path is thrown away prematurely. However, the impact could be reduced to a minimum by empirically adjusting the thresholds in both methods.

The implementation of stack decoding is, in general, more complicated, particularly when some inevitable pruning strategies are incorporated to make the search more efficient. The difficulty of devising both an effectively admissible heuristic function for $h(\bullet)$ and an effective estimation of normalization factors for boundary determination has limited the advantage that stack decoders have over Viterbi decoders. Unlike stack decoding, time-synchronous Viterbi beam search can use an easy comparison of same-length path without heuristic determination of word boundaries. As described in the earlier sections, these simple and unified features of Viterbi beam search allow researchers to incorporate various sound techniques to improve the efficiency of search. Therefore, time-synchronous Viterbi Beam search enjoys a much broader popularity in the speech community. However, the principle of stack decoding is essential particularly for n-best and lattice search. As we describe in Chapter 13, stack decoding plays a very crucial part in multiple-pass search strate-

gies for n-best and lattice search because the early pass is able to establish a near-perfect estimate of the remaining path.

12.5.5. Multistack Search

Even with the help of normalized factor γ or heuristic function $h(\bullet)$, it is still more effective to compare hypotheses of the same length than those of different lengths, because hypotheses with the same length are compared based on the true forward matching score. Inspired by the time-synchronous principle in Viterbi beam search, researchers [8, 35] propose a variant stack decoding based on multiple stacks.

Multistack search is equivalent to a best-first search algorithm running on multiple stacks time-synchronously. Basically, the search maintains a separate stack for each time frame t, so it never needs to compare hypotheses of different lengths. The search runs time-synchronously from left to right just like time-synchronous Viterbi search. For each time frame t, multistack search extracts the best path out of the t-stack, computes one-word extensions, and places all the new paths into the corresponding stacks. When the search finishes, the top path in the last stack is our optimal path. Algorithm 12.7 illustrates the multistack search algorithm.

This time-synchronous multistack search is designed based on the fact that by the time the t^{th} stack is extended, it already contains the best paths that could ever be placed into it. This phenomenon is virtually a variant of the dynamic programming principle introduced in Chapter 8. To make multistack more efficient, some heuristic pruning can be applied to reduce the computation. For example, when the top path of each stack is extended for one more word, we could only consider extensions between minimum and maximum duration. On the other hand, when some heuristic pruning is integrated into the multistack search, one might need to use a small beam in Step 2 of Algorithm 12.7 to extend more than just the best path to guarantee the admissibility.

ALGORITHM 12.7: *MULTISTACK SEARCH*

Step 1: Initialization: for each word v in vocabulary V
  for $t = 1, 2, \ldots, T$
  Compute $C(\mathbf{x}_1^t \mid v)$ and insert it to t^{th} stack

Step 2: Iteration: for $t = 1, 2, \ldots, T-1$
  Sort the t^{th} stack and pop the top path $C(\mathbf{x}_1^t \mid w_1^k)$ out of the stack
  for each word v in vocabulary V
    for $\tau = t+1, t+2, \ldots, T$
    Extend the path $C(\mathbf{x}_1^t \mid w_1^k)$ by word v to get $C(\mathbf{x}_1^\tau \mid w_1^{k+1})$
    where $w_1^{k+1} = w_1^k \,\|\, v$ and $\|$ means string concatenation
    Place $C(\mathbf{x}_1^\tau \mid w_1^{k+1})$ in τ^{th} stack

Step 3: Termination: Sort the T^{th} stack and the top path is the optimal word sequence

12.6. HISTORICAL PERSPECTIVE AND FURTHER READING

Search has been one of the most important topics in artificial intelligence since the origins of the field. It plays the central role in general problem solving [29] and computer games. [43], Nilsson's *Principles of Artificial Intelligence* [32] and Barr and Feigenbaum's *The Handbook of Artificial Intelligence* [11] contain a comprehensive introduction to state-space search algorithms. A* search was first proposed by Hart et al. [17]. A* was thought to be derived from Dijkstra's algorithm [13] and Moore's algorithm [27]. A* search is similar to the *branch-and-bound* algorithm [23, 39], widely used in *operations research*. The proof of admissibility of A* search can be found in [32].

The application of *beam search* in speech recognition was first introduced by the HARPY system [26]. It wasn't widely popular until BBN used it for their BYBLOS system [37]. There are some excellent papers with detailed description of the use of time-synchronous Viterbi beam search for continuous speech recognition [24, 31]. Over the years, many efficient implementations and improvements have been introduced for time-synchronous Viterbi beam search, so real-time large-vocabulary continuous speech recognition can be realized on a general-purpose personal computer.

On the other hand, stack decoding was first developed by IBM [9]. It is successfully used in IBM's large-vocabulary continuous speech recognition systems [3, 16]. Lacking a time-synchronous framework, comparing theories of different lengths and extending theories are more complex as described in this chapter. Because of the complexity of stack decoding, far fewer publications and systems are based on it than on Viterbi beam search [16, 19, 20, 35]. With the introduction of multistack search [8], stack decoding in essence has actually come very close to time-synchronous Viterbi beam search.

Stack decoding is typically integrated with fast match methods to improve its efficiency. Fast match was first implemented for isolated word recognition to obtain a list of potential word candidates [5, 7]. The paper by Gopalakrishnan et al. [15] contains a comprehensive description of fast match techniques to reduce the word expansion for stack decoding. Besides the fast match techniques described in this chapter, there are a number of alternative approaches [5, 21, 41]. Waast's fast match [41], for example, is based on a binary classification tree built automatically from data that comprise both phonetic transcription and acoustic sequence.

REFERENCES

[1] Aho, A., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, 1974, Addison-Wesley Publishing Company.

[2] Alleva, F., X. Huang, and M. Hwang, "An Improved Search Algorithm for Continuous Speech Recognition," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 307-310.

[3] Bahl, L.R. and et. al., "Large Vocabulary Natural Language Continuous Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, Glasgow, Scotland, pp. 465-467.

[4] Bahl, L.R., *et al.*, "Language-Model/Acoustic Channel Balance Mechanism," *IBM Technical Disclosure Bulletin*, 1980, **23**(7B), pp. 3464-3465.

[5] Bahl, L.R., *et al.*, "Obtaining Candidate Words by Polling in a Large Vocabulary Speech Recognition System," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1988, pp. 489-492.

[6] Bahl, L.R., *et al.*, "A Fast Approximate Acoustic Match for Large Vocabulary Speech Recognition," *IEEE Trans. on Speech and Audio Processing*, 1993(1), pp. 59-67.

[7] Bahl, L.R., *et al.*, "Matrix Fast Match: a Fast Method for Identifying a Short List of Candidate Words for Decoding," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, Glasgow, Scotland, pp. 345-347.

[8] Bahl, L.R., P.S. Gopalakrishnan, and R.L. Mercer, "Search Issues in Large Vocabulary Speech Recognition," *Proc. of the 1993 IEEE Workshop on Automatic Speech Recognition*, 1993, Snowbird, UT.

[9] Bahl, L.R., F. Jelinek, and R. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1983(2), pp. 179-190.

[10] Bahl, L.R., *et al.*, "Constructing Candidate Word Lists Using Acoustically Similar Word Groups," *IEEE Trans. on Signal Processing*, 1992(1), pp. 2814-2816.

[11] Barr, A. and E. Feigenbaum, *The Handbook of Artificial Intelligence: Volume I*, 1981, Addison-Wesley.

[12] Cettolo, M., R. Gretter, and R.D. Mori, "Knowledge Integration" in *Spoken Dialogues with Computers*, R.D. Mori, Editor 1998, London, Academic Press, pp. 231-256.

[13] Dijkstra, E.W., "A Note on Two Problems in Connection with Graphs," *Numerische Mathematik*, 1959, **1**, pp. 269-271.

[14] Gauvain, J.L., *et al.*, "The LIMSI Speech Dictation System: Evaluation on the ARPA Wall Street Journal Corpus," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1994, Adelaide, Australia, pp. 129-132.

[15] Gopalakrishnan, P.S. 2and L.R. Bahl, "Fast Match Techniques," in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds., 1996, Norwell, MA, Kluwer Academic Publishers, pp. 413-428.

[16] Gopalakrishnan, P.S., L.R. Bahl, and R.L. Mercer, "A Tree Search Strategy for Large-Vocabulary Continuous Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, MI, pp. 572-575.

[17] Hart, P.E., N.J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. on Systems Science and Cybernetics*, 1968, 4(2), pp. 100-107.

[18] Huang, X., *et al.*, "Microsoft Windows Highly Intelligent Speech Recognizer: Whisper," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, pp. 93-96.

[19] Jelinek, F., *Statistical Methods for Speech Recognition*, 1998, Cambridge, MA, MIT Press.

[20] Kenny, P., *et al.*, "A*—Admissible Heuristics for Rapid Lexical Access," *IEEE Trans. on Speech and Audio Processing*, 1993, **1**, pp. 49-58.

[21] Kenny, P., *et al.*, "A New Fast Match for Very Large Vocabulary Continuous Speech Recognition," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 656-659.

[22] Laface, P., L. Fissore, and F. Ravera, "Automatic Generation of Words toward Flexible Vocabulary Isolated Word Recognition," *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan, pp. 2215-2218.

[23] Lawler, E.W. and D.E. Wood, "Branch-and-Bound Methods: A Survey," *Operations Research*, 1966(14), pp. 699-719.

[24] Lee, K.F. and F.A. Alleva, "Continuous Speech Recognition" in *Recent Progress in Speech Signal Processing*, S. Furui and M. Sondhi, eds., 1990, Marcel Dekker, Inc.

[25] Lee, K.F., H.W. Hon, and R. Reddy, "An Overview of the SPHINX Speech Recognition System," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1990, **38**(1), pp. 35-45.

[26] Lowerre, B.T., *The HARPY Speech Recognition System*, PhD Thesis in Computer Science Department, 1976, Carnegie Mellon University.

[27] Moore, E.F., "The Shortest Path Through a Maze," *Int. Symp. on the Theory of Switching*, 1959, Cambridge, MA, Harvard University Press, pp. 285-292.

[28] Murveit, H., *et al.*, "Large Vocabulary Dictation Using SRI's DECIPHER Speech Recognition System: Progressive Search Techniques," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 319-322.

[29] Newell, A. and H.A. Simon, *Human Problem Solving*, 1972, Englewood Cliffs, NJ, Prentice Hall.

[30] Ney, H. and X. Aubert, "Dynamic Programming Search: From Digit Strings to Large Vocabulary Word Graphs," in *Automatic Speech and Speaker Recognition*, C.H. Lee, F. Soong and K.K. Paliwal, eds., 1996, Boston, Kluwer Academic Publishers, pp. 385-412.

[31] Ney, H. and S. Ortmanns, *Dynamic Programming Search for Continuous Speech Recognition*, in *IEEE Signal Processing Magazine*, 1999, pp. 64-83.

[32] Nilsson, N.J., *Principles of Artificial Intelligence*, 1982, Berlin, Germany, Springer Verlag.

[33] Nilsson, N.J., *Artificial Intelligence: A New Synthesis*, 1998, Academic Press/Morgan Kaufmann.

[34] Normandin, Y., R. Cardin, and R.D. Mori, "High-Performance Connected Digit Recognition Using Maximum Mutual Information Estimation," *IEEE Trans. on Speech and Audio Processing*, 1994, **2**(2), pp. 299-311.

[35] Paul, D.B., "An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Francisco, California, pp. 25-28.

[36] Roe, D.B. and M.D. Riley, "Prediction of Word Confusabilities for Speech Recognition," *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan, pp. 227-230.

[37] Schwartz, R., *et al.*, "Context-Dependent Modeling for Acoustic-Phonetic Recognition of Speech Signals," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1985, Tampa, FLA, pp. 1205-1208.

[38] Steinbiss, V., *et al.*, "The Philips Research System for Large-Vocabulary Continuous-Speech Recognition," *Proc. of the European Conf. on Speech Communication and Technology*, 1993, Berlin, Germany, pp. 2125-2128.

[39] Taha, H.A., *Operations Research: An Introduction*, 6th ed, 1996, Prentice Hall.

[40] Tanimoto, S.L., *The Elements of Artificial Intelligence : An Introduction Using Lisp*, 1987, Computer Science Press, Inc.

[41] Waast, C. and L.R. Bahl, "Fast Match Based on Decision Tree," *Proc. of the European Conf. on Speech Communication and Technology*, 1995, Madrid, Spain, pp. 909-912.

[42] Winston, P.H., *Artificial Intelligence*, 1984, Reading, MA, Addison-Wesley.

[43] Winston, P.H., *Artificial Intelligence*, 3rd ed, 1992, Reading, MA, Addison-Wesley.

[44] Woodland, P.C., *et al.*, "Large Vocabulary Continuous Speech Recognition Using HTK," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1994, Adelaide, Australia, pp. 125-128.

CHAPTER 13

Large-Vocabulary Search Algorithms

Chapter 12 discussed the basic search techniques for speech recognition. However, the search complexity for large-vocabulary speech recognition with high-order language models is still difficult to handle. In this chapter we describe efficient search techniques in the context of time-synchronous Viterbi beam search, which becomes the choice for most speech recognition systems because it is very efficient. We use Microsoft Whisper as our case study to illustrate the effectiveness of various search techniques. Most of the techniques discussed here can also be applied to stack decoding.

With the help of beam search, it is unnecessary to explore the entire search space or the entire trellis. Instead, only the promising search state-space needs to be explored. Please keep in mind the distinction between the implicit search graph specified by the grammar network and the explicit partial search graph that is actually constructed by the Viterbi beam search algorithm.

In this chapter we first introduce the most critical search organization for large-vocabulary speech recognition—tree lexicons. Tree lexicons significantly reduce potential search space, although they introduce many practical problems. In particular, we need to

address problems such as reentrant lexical trees, factored language model probabilities, subtree optimization, and subtree polymorphism.

Various other efficient techniques also are introduced. Most of these techniques aim for clever pruning with the hope of sparing the correct paths. For more effective pruning, different layers of beams are usually used. While fast match techniques described in Chapter 12 are typically required for stack decoding, similar concepts and techniques can be applied to Viterbi beam search. In practice, the look-ahead strategy is equally effective for Viterbi beam search.

Although it is always desirable to use all the knowledge sources (KSs) in the search algorithm, some are difficult to integrate into the left-to-right time-synchronous search framework. One alternative strategy is to first produce an ordered list of sentence hypotheses (a.k.a. *n-best list*), or a lattice of word hypotheses (a.k.a. *word lattice*) using relatively inexpensive KSs. More expensive KSs can be used to rescore the *n*-best list or the word lattice to obtain the refined result. Such a multipass strategy has been explored in many large-vocabulary speech recognition systems. Various algorithms to generate sufficient *n*-best lists or the word lattices are described in the section on multipass search strategies.

Most of the techniques described in this chapter rely on nonadmissible heuristics. Thus, it is critical to derive a framework to evaluate different search strategies and pruning parameters.

13.1. Efficient Manipulation of a Tree Lexicon

The lexicon entry is the most critical component for large-vocabulary speech recognition, since the search space grows linearly along with increased linear vocabulary. Thus an efficient framework for handling large vocabulary undoubtedly becomes the most critical issue for efficient search performance.

13.1.1. Lexical Tree

The search space for *n*-gram discussed in Chapter 12 is organized based on a straightforward linear lexicon, i.e., each word is represented as a linear sequence of phonemes, independent of other words. For example, the phonetic similarity between the words *task* and *tasks* is not leveraged. In a large-vocabulary system, many words may share the same beginning phonemes. A tree structure is a natural representation for a large-vocabulary lexicon, as many phonemes can be shared to eliminate redundant acoustic evaluations. The lexical tree-based search is thus essential for building a real-time[1] large-vocabulary speech recognizer.

[1] The term *real-time* means the decoding process takes no longer than the duration of the speech. Since the decoding process can take place as soon as the speech starts, such a real-time decoder can provide real instantaneous responses after speakers finish talking.

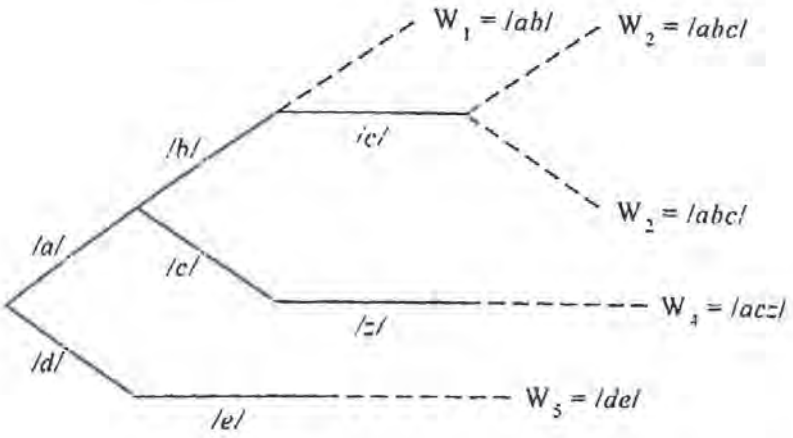


Figure 13.1 An example of a lexical tree, where each branch corresponds to a shared phoneme and the leaf corresponds to a word.

Figure 13.1 shows an example of such a lexical tree, where common beginning phonemes are shared. Each leaf corresponds to a word in the vocabulary. Please note that an extra null arc is used to form the leaf node for each word. This null arc has the following two functions:

1. When the pronunciation transcription of a word is a prefix of other ones, the null arc can function as one branch to end the word.
2. When there are homophones in the lexicon, the null arcs can function as linguistic branches to represent different words such as *two* and *to*.

The advantage of using such a lexical tree representation is obvious: it can effectively reduce the state search space of the trellis. Ney et al. [32] reported that a lexical tree representation of a 12,306-word lexicon with only 43,000 phoneme arcs had a saving of a factor of 2.5 over the linear lexicon with 100,800 phoneme arcs. Lexical trees are also referred to as *prefix trees*, since they are efficient representations of lexicons with sharing among lexical entries that have a common prefix. Table 13.1 shows the distribution of phoneme arcs for this 12,306-word lexical tree. As one can see, even in the fifth level the number of phoneme arcs is only about one-third of the total number of words in the lexicon.

Table 13.1 Distribution of the tree phoneme arcs and active tree phoneme arc for a 12,306-word lexicon using a lexical tree representation [32].

Level	1	2	3	4	5	6	≥7
Phoneme arcs	28	331	1511	3116	4380	4950	29.200
Average active arcs	23	233	485	470	329	178	206

The saving by using a lexical tree is substantial, because it not only results in considerable memory saving for representing state-search space but also saves tremendous time by searching far fewer potential paths. Ney et al. [32] report that a tree organization of the lexicon reduces the total search effort by a factor of 7 over the linear lexicon organization. This is because the lion's share of hypotheses during a typical large-vocabulary search is on the first and second phonemes of a word. Haeb-Umbach et al. [23] report that for a 12,306-word dictation task, 79% and 16% of the state hypotheses are in the first and second phonemes, when analyzing the distribution of the state hypotheses over the state position within a word. Obviously, the effect is caused by the ambiguities at the word boundaries. The lexical tree representation reduces that effort by evaluating common phonetic prefixes only once. Table 13.1 also shows the average number of active phoneme arcs in the layers of the lexical tree [32]. Based on this table, you can expect that the overall search cost is far less than the size of the vocabulary. This is the key reason why lexical tree search is widely used for large-vocabulary continuous speech recognition systems.

The lexical tree search requires a sophisticated implementation because of a fundamental deficiency—*a branch in a lexical tree representation does not correspond to a single word with the exception of branches ending in a leaf.* This deficiency translates to the fact that a unique word identity is not determined until a leaf of the tree is reached. This means that any decision about the word identity needs to be delayed until the leaf node is reached, which results in the following complexities.

- Unlike a linear lexicon, where the language model score can be applied when starting the acoustic search of a new word, the lexical tree representation has to delay the application of the language model probability until the leaf is reached. This may result in an increased search effort, because the pruning needs to be done on a less reliable measure, unless a factored language model is used, as discussed in Section 13.1.3.
- Because of the delay of language model contribution by one word, we need to keep a separate copy of an entire lexical tree for each unique language model history.

13.1.2. Multiple Copies of Pronunciation Trees

A simple lexical tree is sufficient if no language model or a unigram is used. This is because the decision at time t depends on the current word only. However, for higher-order n-gram models, the linguistic state cannot be determined locally. A tree copy is required for each language model state. For bigrams, a tree copy is required for each predecessor word. This may seem to be astonishing, because the potential search space is increased by the vocabulary size. Fortunately, experimental results show only a small number of tree copies are required, because efficient pruning can eliminate most of the unneeded ones. Ney et al. [32] report that the search effort using bigrams is increased by only a factor of 2 over the unigram

case. In general, when more detailed (better) acoustic and/or language models are used, the effect of a potentially increased search space is often compensated by a more focused beam search from the use of more accurate models. In other words, although the static search space might increase significantly by using more accurate models, the dynamic search space can be under control (sometimes even smaller), thanks to improved evaluation functions.

To deal with tree copies [19, 23, 37], you can create redundant subtrees. When copies of lexical trees are used to disambiguate active linguistic contexts, many of the active state hypotheses correspond to the same redundant unigram state, due to the postponed application of language models. To apply the language model sooner, and to eliminate redundant unigram state computations, a successor tree, T_i, can be created for each linguistic context i. T_i encodes the nonzero n-grams of the linguistic context i as an isomorphic subgraph of the unigram tree, T_0. Figure 13.2 shows the organization of such successor trees and unigram tree for bigram search. For each word w a successor tree, T_w is created with the set of successor words that have nonzero bigram probabilities. Suppose u is a successor of w; the bigram probability $P(u \mid w)$ is attached to the transition connecting the leaf corresponding to u in the successor tree T_w, with the root of the successor tree T_u. The unigram tree is a full-size lexical tree and is shared by all words as the back-off lexical tree. Each leaf of the unigram tree corresponds to one of $|V|$ words in the vocabulary and is linked to the root of its bigram successor tree (T_u) by an arc with the corresponding unigram probability $P(u)$. The backoff weight, $\alpha(u)$, of predecessor u is attached to the arc which links the root of successor tree T_u to the root of the unigram tree.

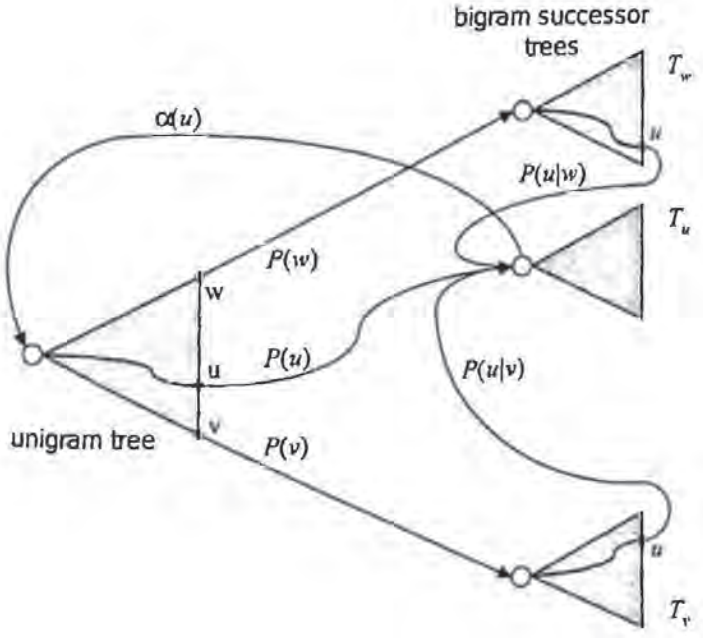


Figure 13.2 Successor trees and unigram trees for bigram search [13].

A careful search organization is required to avoid computational overhead and to guarantee a linear time complexity for exploring state hypotheses. In the following sections we describe techniques to achieve efficient lexical tree recognizers. These techniques include factorization of language model probabilities, tree optimization, and exploiting subtree dominance.

13.1.3. Factored Language Probabilities

As mentioned in Section 13.1.2, search is more efficient if a detailed knowledge source can be applied at an early stage. The idea of *factoring* the language model probabilities across the tree is one such example [4, 19]. When more than one word shares a phoneme arc, the upper bound of their probability can be associated to that arc.[2] The factorization can be applied to both the full lexical tree (unigram) and successor trees (bigram or other higher-order language models).

An unfactored tree only has language model probabilities attached to the leaf nodes, and all the internal nodes have probability 1.0. The procedure for factoring the probabilities across the tree computes the maximum of each node n in the tree according to Eq. (13.1). The tree can then be factored according to Eq. (13.2) so when you traverse the tree you can multiply $F^*(n)$ along the path to get the needed language probability.

$$P^*(n) = \max_{x \in child(n)} P(x) \tag{13.1}$$

$$F^*(n) = \frac{P^*(n)}{P^*(parent(n))} \tag{13.2}$$

An illustration of the factored probabilities is shown in Table 13.2. Using this lexicon, we create the tree depicted in Figure 13.3(a). In this figure the unlabeled internal nodes have a probability of 1.0. We distribute the probabilities according to Eq. (13.1) in Figure 13.3(b), which is factored according to Eq. (13.2), resulting in Figure 13.3(c).

Table 13.2 Sample probabilities $P(w_j)$ and their pseudoword pronunciations [4].

w_j	**Pronunciation**	$P(w_j)$
w_0	/a b c/	0.1
w_1	/a b c/	0.4
w_2	/a c z/	0.3
w_3	/d e/	0.2

[2] The choice of upper bound is because it is an admissible estimate of the path no matter which word will be chosen later.

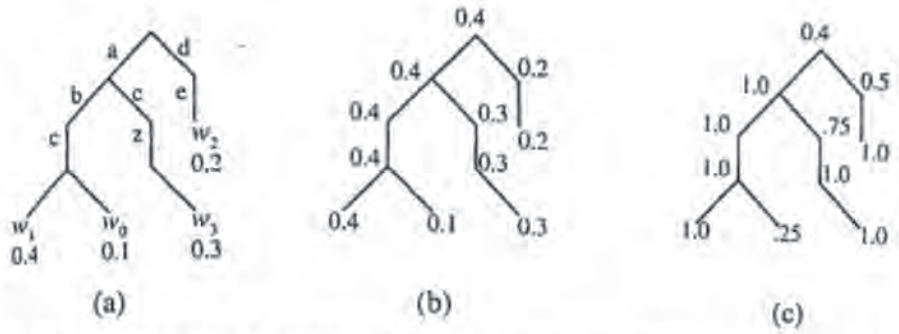


Figure 13.3 (a) Unfactored lexical tree; (b) distributed probabilities with computed $P^*(n)$; (c) factored tree $F^*(n)$ [4].

Using the upper bounds in the factoring algorithm is not an approximation, since the correct language model probabilities are calculated by the product of values traversed along each path from the root to the leaves. However, you should note that the probabilities of all the branches of a node do not sum to one. This can solved by replacing the upper-bound (max) function in Eq. (13.1) with the sum.

$$P^*(n) = \sum_{x \in child(n)} P(x) \tag{13.3}$$

To guarantee that all the branches sum to one, Eq. (13.2) should also be replaced by the following equation:

$$F^*(n) = \frac{P^*(n)}{\sum_{x \in child(parent(n))} P^*(x)} \tag{13.4}$$

A new illustration of the distribution of LM probabilities by using sum instead of upper bound is shown in Figure 13.4. Experimental results have shown that the factoring method with either sum or upper bound has comparable search performance.

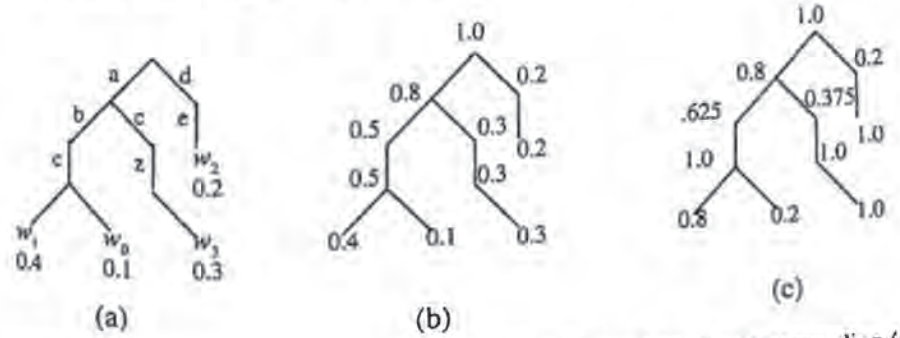


Figure 13.4 Using sum instead of upper bound when factoring tree, the corresponding (a) unfactored lexical tree; (b) distributed probabilities with computed $P^*(n)$; (c) factored tree with computed $F^*(n)$ [4].

One interesting observation is that the language model score can be regarded as a heuristic function to estimate the linguistic expectation of the current word to be searched. In a linear representation of the pronunciation lexicon, application of the linguistic expectation was straightforward, since each state is associated with a unique word. Therefore, given the context defined by the hypothesis under consideration, the expectation for the first phone of word w_i is just $P(w_i \mid w_1^{i-1})$. After the first phone, the expectation for the rest of the phones becomes 1.0, since there is only one possible phone sequence when searching the word w_i. However, for the tree lexicon, it is necessary to compute $E(p_j \mid p_1^{j-1}, w_1^{i-1})$, the expectation of phone p_j given the phonetic prefix p_1^{j-1} and the linguistic context w_1^{i-1}. Let $\phi(j, w_k)$ denote the phonetic prefix of length j for w_k. Based on Eqs. (13.1) and (13.2), we can compute the expectation as:

$$E(p_j \mid p_1^{j-1}, w_1^{i-1}) = \frac{P(w_c \mid w_1^{i-1})}{P(w_p \mid w_1^{i-1})} \tag{13.5}$$

where $c = \arg\max_k (w_k \mid w_1^{i-1}, \phi(j, w_k) = p_1^j)$ and $p = \arg\max_k (w_k \mid w_1^{i-1}, \phi(j-1, w_k) = p_1^{j-1})$. Based on Eq. (13.5), an arbitrary n-gram model or even a stochastic context-free grammar can be factored accordingly.

13.1.3.1. Efficient Memory Organization of Factored Lexical Trees

A major drawback to the use of successor trees is the large memory overhead required to store the additional information that encodes the structure of the tree and the factored linguistic probabilities. For example, the 5.02 million bigrams in the 1994 NABN (North American Business News) model require 18.2 million nodes. Given a compact binary tree representation that uses 4 bytes of memory per node, 72.8 million bytes are required to store the predecessor-dependent lexical trees. Furthermore, this tree representation is not as amenable to data compression techniques as the linear bigram representation.

The factored probability of successor trees can be encoded as efficiently as the n-gram model based on Algorithm 13.1, i.e., one n-gram record results in one constant-sized record. Step 3 is illustrated in Figure 13.5(b), where the heavy line ends at the most recently visited node that is not a direct ancestor. The encoding result is shown in Table 13.3.

ALGORITHM 13.1: *ENCODING THE LEXICAL SUCCESSOR TREES (LST)*

For each linguistic context:
Step 1: Distribute the probabilities according to Eq. (13.1).
Step 2: Factor the probabilities according to Eq. (13.2).
Step 3: Perform a depth-first traversal of the LST and encode each leaf record,
(a) the depth of the most recently visited node that is not a direct ancestor,
(b) the probability of the direct ancestor at the depth in (a),
(c) the word identity.

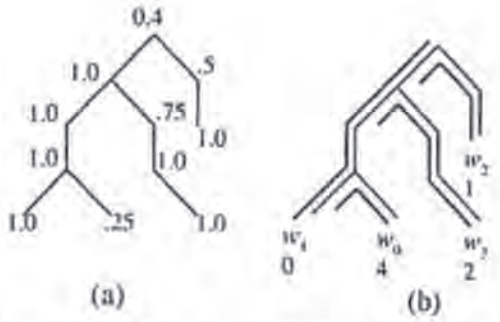


Figure 13.5 (a) Factored tree; (b) tree with common prefix-length annotation.

Clearly the new data structure meets the requirements set forth, and, in fact, it only requires additional $\log(n)$ bits per record (n is the depth of the tree). These bits encode the common prefix length for each word. Naturally this requires some modification to the decoding procedure. In particular, the decoder must scan a portion of the n-gram successor list in order to determine which tree nodes should be activated. Depending on the structure of the tree (which is determined by the acoustic model, the lexicon, and language model), the tree structure can be interpreted at runtime or cached for rapid access if memory is available.

Table 13.3 Encoded successor lexical tree; each record corresponds to one augmented factored n-gram.

w_j	Depth	$F^*(w_j)$
w_1	0	0.4
w_0	4	0.25
w_3	2	0.75
w_2	1	0.5

13.1.4. Optimization of Lexical Trees

We now investigate ways to handle the huge search network formed by the multiple copies of lexical trees in different linguistic contexts. The factorization of lexical trees actually makes it easier to search. First, after the factorization of the language model, the intertree transitions shown in Figure 13.2 no longer have the language model scores attached because they are already applied completely before leaving the leaves. Moreover, as illustrated in Figure 13.3, many transitions toward the end of a single-word path now have an associated transition probability that is equal to 1. This observation implies that there could be many duplicated subtrees in the network. Those duplicated subtrees can then be merged to save both space and computation by eliminating redundant (unnecessary) state evaluation. Unlike pruning, this saving is based on the dynamic programming principle, without introducing any potential error.

13.1.4.1. Optimization of Finite State Network

One way to compress the lexical tree network is to use a similar algorithm for optimizing the number of states in a deterministic finite state automaton. The optimization algorithm is based on the *indistinguishable* property of states in a finite state automaton. Suppose that s_1 and s_2 are the initial states for automata T_1 and T_2, then s_1 and s_2 are said to be *indistinguishable* if the languages accepted by automata T_1 and T_2 are exactly the same. If we consider our lexical tree network as a finite state automaton, the symbol emitted from the transition arc includes not only the phoneme identity, but also the factorized language model probability.

The general set-partitioning algorithm [1] can be used for the reduction of finite state automata. The algorithm starts with an initial partition of the automaton states and iteratively refines the partition so that two states s_1 and s_2 are put in the same block B_i if and only if $f(s_1)$ and $f(s_2)$ are both in the same block B_j. For our purpose, $f(s_1)$ and $f(s_2)$ can be defined as the destination state given a phone symbol (in the factored trees, the pair <*phone, LM-probability*> can be used). Each time a block is partitioned, the smaller subblock is used for further partitioning. The algorithm stops when all the states that transit to some state in a particular block with arcs labeled with the same symbol are in the same block. When the algorithm halts, each block of the resulting partition is composed of *indistinguishable* states, and those states within each block can then be merged. The algorithm is guaranteed to find the automaton with the minimum number of states. The algorithm has a time complexity of $O(MN \log N)$, where M is the maximum number of branching (fan-out) factors in the lexical tree and N is the number of states in the original tree network.

Although the above algorithm can give optimal finite state networks in terms of number of states, such an optimized network may be difficult to maintain, because the original lexical tree structure could be destroyed and it may be troublesome to add any new word into the tree network [1].

13.1.4.2. Subtree Isomorphism

The finite state optimization algorithm described above does not take advantage of the tree structure of the finite state network, though it generates a network with a minimum number of states. Since our finite state network is a network of trees, the indistinguishability property is actually the same as the definition of subtree isomorphism. Two subtrees are said to be *isomorphic* to each other if they can be made equivalent by permuting the successors. It should be straightforward to prove that two states are indistinguishable, if and only if their subtrees are isomorphic.

There are efficient algorithms [1] to detect whether two subtrees are isomorphic. For all possible pairs of states u and v, if the subtrees starting at u and v, $ST(u)$ and $ST(v)$, are isomorphic, v is merged into u and $ST(v)$ can be eliminated. Note that only internal nodes need to be considered for subtree isomorphism check. The time complexity for this algorithm is $O(N^2)$ [1].

13.1.4.3. Sharing Tails

A *linear tail* in a lexical tree is defined as a subpath ending in a leaf and going through states with a unique successor. It is often referred as a *single-word subpath*. It can be proved that such a linear tail has unit probability attached to its arcs according to Eqs. (13.1) and (13.2). This is because LM probability factorization *pushes forward* the LM probability attached to the last arc of the linear tail, leaving arcs with unit probability. Since all the tails corresponding to the same word w in different successor trees are linked to the root of successor tree T_w,[1] the subtree starting from the first state of each linear tail is isomorphic to the subtree starting from one of the states forming the longest linear tail of w. A simple algorithm to take advantage of this share-tail topology can be employed to reduce the lexical tree network.

Figure 13.6 and Figure 13.7 show a lexical tree network before and after shared-tail optimization. For each word, only the longest linear tail is kept. All other tails can be removed by linking them to an appropriate state in the longest tail, as shown in Figure 13.7.

Shared-tail optimization is not global optimization, because it considers only some special topology optimization. However, there are some advantages associated with shared-tail optimization. First, in practice, duplicated linear tails account for most of the redundancy in lexical tree networks [12]. Moreover, shared-tail optimization has a nice property of maintaining the basic lexical tree structure for the optimized tree network.

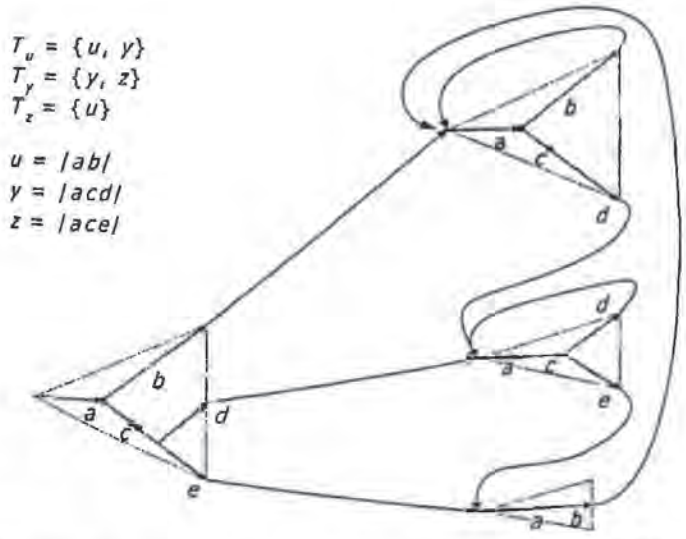


Figure 13.6 An example of a lexical tree network without shared-tail optimization [12]. The vocabulary includes three words, u, y, and z. T_u, T_y, and T_z are the successor trees for u, y, and z respectively [13].

[1] We assume bigram is used in the discussion of "sharing tails."

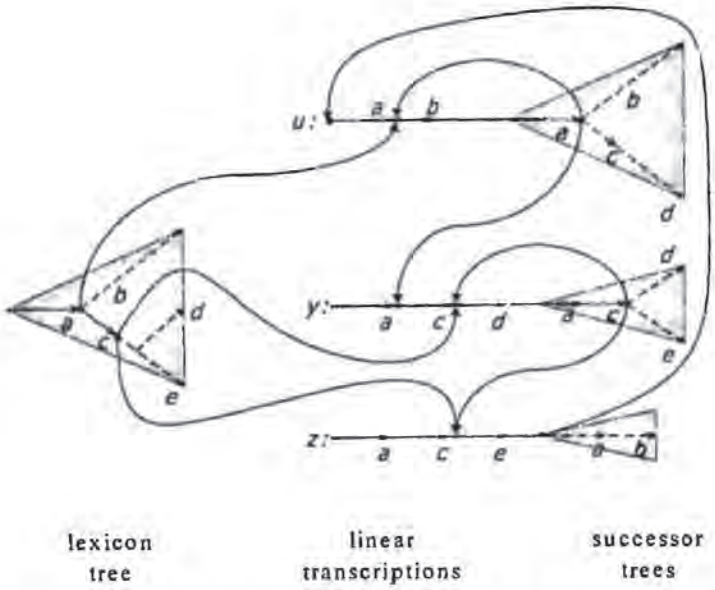


Figure 13.7 The lexical tree network in Figure 13.6 after shared-tail optimization [12].

13.1.5. Exploiting Subtree Polymorphism

The techniques of optimizing the network of successor lexical trees can only eliminate identical subtrees in the network. However, there are still many subtrees that have the same nodes and topology but with different language model scores attached to the arcs. The acoustic evaluation for those subtrees is unnecessarily duplicated. In this section we exploit *subtree dominance* for additional saving.

A subtree instance is *dominated* when the best outcome in that subtree is not better than the worst outcome in another instance of that subtree. The evaluation becomes redundant for the dominated subtree instance. Subtree isomorphism and shared-tail are cases of subtree dominance, but they require prearrangement of the lexical tree network as described in the previous section.

If we need to implement lexical tree search dynamically, the network optimization algorithms are not suitable. Although subtree dominance can be computed using minimax search [35] during runtime, this requires that information regarding subtree isomorphism be available for all corresponding pairs of states for each successor tree T_w. Unfortunately, it is not practical in terms of either computation or space.

In place of computing strict subtree dominance, a *polymorphic* linguistic context assignment to reduce redundancy is employed by estimating subtree dominance based on local information and ignoring the subgraph isomorphism problem. Polymorphic context assignment involves keeping a single copy of the lexical tree and allowing each state to assume the linguistic context of the most promising history. The advantage of this approach is that it employs maximum sharing of data structures and information, so each node in the tree is

evaluated, at most, once. However, the use of local knowledge to determine the dominant context could introduce significant errors because of premature pruning. Whisper [4] reports a 65.7% increase in error rate when only the dominant context is kept, based on local knowledge.

To recover the errors created by using local linguistic information to estimate subtree dominance, you need to delay the decision regarding which linguistic context is most promising. This can be done by keeping a heap of contexts at each node in the tree. The heap maintains all contexts (linguistic paths) whose probabilities are within a constant threshold ε, of that of the best global path. The effect of the ε -heap is that more contexts are retained for high-probability states in the lexical tree. The pseudocode fragment in Algorithm 13.2 [3] illustrates a transition from state s_n in context c to state s_m. The terminology used in Algorithm 13.2 is listed as follows:

- $(-\log P(s_m \mid s_n, c))$ is the cost associated with applying acoustic model matching and language model probability of state s_m transited from s_n in context c.
- $InHeap(s_m, c)$ is true if context c is in the heap corresponding to state s_m.
- $Cost(s_m, c)$ is the cost for context c in state s_m.
- $StateInfo(s_m, c)$ is the auxiliary state information associated with context c in state s_m.
- $Add(s_m, c)$ adds context c to the state s_m heap.
- $Delete(s_m, c)$ deletes context c from state s_m heap.
- $WorstContext(s_m)$ retrieves the worst context from the heap of state s_m.

ALGORITHM 13.2: *HANDLING MULTIPLE LINGUISTIC CONTEXTS IN A LEXICAL TREE*

1. $d = Cost(s_n, c) \div (-\log P(s_m \mid s_n, c))$
2. If $InHeap(s_m, c)$ then
   If $d < Cost(s_m, c)$ then
   $Cost(s_m, c) = d$
   $StateInfo(s_m, c) = StateInfo(s_n, c)$
   else if $d < BestCost(s_m) + \varepsilon$ then
   $Add(s_m, c)$; $StateInfo(s_m, c) = StateInfo(s_n, c)$
   $Cost(s_m, c) = d$
   else
   $w = WorstContext(s_m)$
   if $d < Cost(s_m, w)$ then
   $Delete(s_m, w)$
   $Add(s_m, c)$; $StateInfo(s_m, c) = StateInfo(s_n, c)$
   $Cost(s_m, c) = d$

When higher-order n-gram is used for lexical tree search, the potential heap size for lexical tree nodes (some also refer to *prefix nodes*) could be unmanageable. With decent acoustic models and efficient pruning, as illustrated in Algorithm 13.2, the average heap size for active nodes in the lexical tree is actually very modest. For example, Whisper's average heap size for active nodes in the 20,000-word WSJ lexical tree decoder is only about 1.6 [3].

13.1.6. Context-Dependent Units and Inter-Word Triphones

So far, we have implicitly assumed that context-independent models are used in the lexical tree search. When context-dependent phonetic or subphonetic models, as discussed in Chapter 9, are used for better acoustic models, the construction and use of a lexical tree become more complicated.

Since senones represent both subphonetic and context-dependent acoustic models, this presents additional difficulty for use in lexical trees. Let's assume that a three-state context-dependent HMM is formed from three senones, one for each state. Each senone is context-dependent and can be shared by different allophones. If we use allophones as the units for lexical tree, the sharing may be poor and fan-out unmanageable. Fortunately, each HMM is uniquely identified by the sequence of senones used to form the HMM. In this way, different context-dependent allophones that share the same *senone sequence* can be treated as the same. This is especially important for lexical tree search, since it reduces the order of the fan-out in the tree.

Interword triphones that require significant fan-ins for the first phone of a word and fan-outs for the last phones usually present an implementation challenge for large-vocabulary speech recognition. A common approach is to delay full interword modeling until a subsequent rescoring phase.[4] Given a sufficiently rich lattice or word graph, this is a reasonable approach, because the static state space in the successive search has been reduced significantly. However, as pointed out in Section 13.1.2, the size of the dynamic state space can remain under control when detailed models are used to allow effective pruning. In addition, a multipass search requires an augmented set of acoustic models to effectively model the biphone contexts used at word boundaries for the first pass. Therefore, it might be desirable to use genuine interword acoustic models in the single-pass search.

Instead of expanding all the fan-ins and fan-outs for inter-word context-dependent phone units in the lexical tree, three *metaunits* are created.

1. The first metaunit, which has a known right context corresponding to the second phone in the word, but uses open left context for the first phone of a word (sometimes referred to as the *word-initial unit*). In this way, the fan-in is represented as a subgraph shared by all words with the same initial left-context-dependent phone.

[4] Multipass search strategy is described in Section 13.3.5.

2. Another metaunit, which has a known left context corresponding to the second-to-last phone of the word, but uses open right context for the last phone of a word (sometimes referred to as the *word-final unit*). Again, the fan-out is represented as a subgraph shared by all words with the same final right-context-dependent phone.
3. The third metaunit, which has both open left and right contexts, and is used for single-phone word unit.

By using these metaunits we can keep the states for the lexical trees under control, because the fan-in and fan-out are now represented as a single node.

During recognition, different left or right contexts within the same metaunit are handled using Algorithm 13.2, where the different acoustic contexts are treated similarly as different linguistic contexts. The open left-context metaunit (fan-ins) can be dealt with in a straightforward way using Aglorithm 13.2, because the left context is always known (the last phone of the previous word) when it is initiated. On the other hand, the open right-context metaunit (fan-out) needs to explore all possible right contexts because the next word is not known yet. To reduce unnecessary computation, fast match algorithms (described in Section 13.2.3) can be used to provide both expected acoustic and language scores for different context-dependent units to result in early pruning of unpromising contexts.

13.2. OTHER EFFICIENT SEARCH TECHNIQUES

Tree structured lexicon represents an efficient framework of manipulation of search space. In this section we present some additional implementation techniques, which can be used to further improve the efficiency of search algorithms. Most of these techniques can be applied to both Viterbi beam search and stack decoding. They are essential ingredients for a practical large-vocabulary continuous speech recognizer.

13.2.1. Using Entire HMM as a State in Search

The state in state-search space based on HMM-trellis computation is, by definition, a Markov state. Phonetic HMM models are the basic unit in most speech recognizers. Even though subphonetic HMMs, like senones, might be used for such a system, the search is often based on phonetic HMMs.

Treating the entire phonetic HMM as a state in state-search has many advantages. The first obvious advantage is that the number of states the search program needs to deal with is smaller. Note that using the entire phonetic HMM does not in effect reduce the number of states in the search. The entire search space is unchanged. All the states within a phonetic HMM are now bundled together. This means that all of them are either kept in the beam, if the phonetic HMM is regarded as promising, or all of them are pruned away. For any given time, the minimum cost among all the states within the phonetic HMM is used as the cost for the phonetic HMM. For pruning purposes, this cost is used to determine the promising

degree of this phonetic HMM, i.e., the fate of all the states within this phonetic HMM. Although this does not actually reduce the beam beyond normal pruning, it has the effect of processing fewer candidates in the beam. In programming, this means less checking and bookkeeping, so some computation savings can be expected.

You might wonder if this organization might be ineffective for beam search, since it forces you to keep or prune all the states within a phonetic HMM. In theory, it is possible that only one or two states in the phonetic HMM need to be kept, while other states can be pruned due to high cost score. However, this is, in reality, very rare, since a phone is a small unit and all the states within a phonetic HMM should be relatively promising when the search is near the acoustic region corresponding to the phone.

During the trellis computation, all the phonetic HMM states need to advance one time step when processing one input vector. By performing HMM computation for all states together, the new organization can reduce memory accesses and improve cache locality, since the output and transition probabilities are held in common by all states. Combining this organization strategy with lexical tree search further enhances the efficiency. In lexical tree search, each hypothesis in the beam is associated with a particular node in the lexical tree. These hypotheses are linked together in the heap structure described in Algorithm 13.2 for the purposes of efficient evaluation and heuristic pruning. Since the node corresponds to a phonetic HMM, the HMM evaluation is guaranteed to execute once for each hypothesis sharing this node.

In summary, treating the entire phonetic HMM as a state in state-search space allows you to explore the effective data structure for better sharing and improved memory locality.

13.2.2. Different Layers of Beams

Because of the complexity of search, it often requires pruning of various levels of search to make search feasible. Most systems thus employ different pruning thresholds to control what states participate. The most frequently used thresholds are listed below:

- τ_s controls what states (either phone states or senone states) to retain. This is the most fundamental beam threshold.
- τ_p controls whether the next phone is extended. Although this might not be necessary for both stack decoding and linear Viterbi beam search, it is crucial for lexical tree search, because pruning unpromising phonetic prefixes in the lexical trees could improve search efficiency significantly.
- τ_h controls whether hypotheses are extended for the next word. Since the branching factor for word boundaries is very large, we need this threshold to limit search to only the promising ones.
- τ_c controls where a linguistic context is created in a lexical tree search using higher-order language models. This is also known as ε-heap in Algorithm 13.2.

Pruning can introduce search errors if a state is pruned that would have been on the globally best path. The principle applied here is that the more constraints you have available, the more aggressively you decide whether this path will participate in the globally best path. In this case, at the state level, you have the least constraints. At the phonetic level there are more, and there are the most at the word level. In general, the number of word hypotheses tends to drop significantly at word boundaries. Different thresholds for different levels allow the search designer to fine-tune those thresholds for their tasks to achieve best search performance without significant increase in error rates.

13.2.3. Fast Match

As described in Chapter 12, fast match is a crucial part of stack decoding, which mainly reduces the number of possible word expansions for each path. Similarly, fast match can be applied to the most expensive part—extending the phone HMM fan-outs within or between lexical trees. Fast match is a method for rapidly deriving a list of candidates that constrain successive search phases in which a computationally expensive *detailed match* is performed. In this sense, fast match can be regarded as an additional pruning threshold to meet before a new word/phone can be started.

Fast match is typically characterized by the approximations that are made in the acoustic/language models to reduce computation. The factorization of language model scores among tree branches in lexical trees described in Section 13.1.3 can be viewed as fast match using a language model. The factorized method is also an admissible estimate of the language model scores for the future word. In this section we focus on acoustic model fast match.

13.2.3.1. Look-Ahead Strategy

Fast match, when applied in time-synchronous search, is also called *look-ahead* strategy, since it basically searches ahead of the time-synchronous search by a few frames to determine which words or phones are likely to extend. Typically the look-ahead frames are fixed, and the fast match is also done in time-synchronous fashion with another specialized beam for efficient pruning. You can also use simplified models, like the one-state HMMs or context-independent models [4, 32]. Some systems [21, 22] have tried to simplify the level of details in the input feature vectors by aggregating information from several frames into one. A straightforward way for compressing the feature stream is to skip every other frame of speech for fast match. This allows a longer-range look-ahead, while keeping computation under control. The approach of simplifying the input feature stream instead of simplifying the acoustic models can reuse the fast match results for detailed match.

Whisper [4] uses phoneme look-ahead fast match in lexical tree search, in which pruning is applied based on the estimation of the score of possible phone fan-outs that may follow a given phone. A context-independent phone-net is searched synchronously with the

search process but offset N frames into the future. In practice, significant savings can be obtained in search efforts without increase in error rates.

The performance of word and phoneme look-ahead clearly depends on the length of the look-ahead frames. In general, the larger the look-ahead window, the longer is the computation and the shorter the word/phone Λ list. Empirically, the window is a few tens of milliseconds for phone look-ahead and a few hundreds of milliseconds for word look-ahead.

13.2.3.2. The Rich-Get-Richer Strategy

For systems employing continuous-density HMMs, tens of mixtures of Gaussians are often used for the output probability distribution for each state. The computation of the mixtures is one of the bottlenecks when many context-dependent models are used. For example, Whisper uses about 120,000 Gaussians. In addition to using various beam pruning thresholds in the search, there could be significant savings if we have a strategy to limit the number of Gaussians to be computed.

The *Rich-Get-Richer* (RGR) strategy enables us to focus on most promising paths and treat them with detailed acoustic evaluations and relaxed path-pruning thresholds. On the contrary, the less promising paths are extended with less expensive acoustic evaluations and less forgiving path-pruning thresholds. In this way, locally optimal candidates continue to receive the maximum attention while less optimal candidates are retained but evaluated using less precise (computationally expensive) acoustic and/or linguistic models. The RGR strategy gives us finer control in the creation of new paths that has potential to grow exponentially.

RGR is used to control the level of acoustic details in the search. The goal is to reduce the number of context-dependent senone probability (Gaussian) computations required. The context-dependent senones associated with a phone instance p would be evaluated according to the following condition:

$$\begin{aligned} &Min[ci(p)] * \alpha + LookAhead[ci(p)] < \text{threshold} \\ &\text{where } Min[ci(p)] = \min_{s}\{\cos t(s) \mid s \in ci\_phone(p)\} \\ &\text{and } \quad LookAhead[ci(p)] = \text{look-ahead estimate of } ci(p) \end{aligned} \tag{13.6}$$

These conditions state that the context-dependent senones associated with p should be evaluated if there exists a state s corresponding to p, whose cost in linear combination with a look-ahead cost score corresponding to p falls within a threshold. In the event that p does not fall within the threshold, the senone scores corresponding to p are estimated using the context-independent senones corresponding to p. This means the context-dependent senones are evaluated only if the corresponding context-independent senones and the look-ahead start showing promise. RGR strategy should save significant senone computation for clearly unpromising paths. Whisper [26] reports that 80% of senone computation can be avoided without introducing significant errors for a 20,000-word WSJ dictation task.

13.3. *N*-BEST AND MULTIPASS SEARCH STRATEGIES

Ideally, a search algorithm should consider all possible hypotheses based on a unified probabilistic framework that integrates all *knowledge sources* (KSs).[5] These KSs, such as acoustic models, language models, and lexical pronunciation models, can be integrated in an HMM state search framework. It is desirable to use the most detailed models, such as context-dependent models, interword context-dependent models, and high-order n-grams, in the search as early as possible. When the explored search space becomes unmanageable, due to the increasing size of vocabulary or highly sophisticated KSs, search might be infeasible to implement.

As we develop more powerful techniques, the complexity of models tends to increase dramatically. For example, language understanding models in Chapter 17 require long-distance relationships. In addition, many of these techniques are not operating in a left-to-right manner. A possible alternative is to perform a multipass search and apply several KSs at different stages, in the proper order to constrain the search progressively. In the initial pass, the most discriminant and computationally affordable KSs are used to reduce the number of hypotheses. In subsequent passes, progressively reduced sets of hypotheses are examined, and more powerful and expensive KSs are then used until the optimal solution is found.

The early passes of multipass search can be considered fast match that eliminates those unlikely hypotheses. Multipass search is, in general, not admissible because the optimal word sequence could be wrongly pruned prematurely, due to the fact that not all KSs are used in the earlier passes. However, for complicated tasks, the benefits of computation complexity reduction usually outweigh the nonadmissibility. In practice, multipass search strategy using progressive KSs could generate better results than a search algorithm forced to use less powerful models due to computation and memory constraints.

The most straightforward multipass search strategy is the so-called n-best search paradigm. The idea is to use affordable KSs to first produce a list of n most probable word sequences in a reasonable time. Then these n hypotheses are rescored using more detailed models to obtain the most likely word sequence. The idea of the n-best list can be further extended to create a more compact hypotheses representation—namely word lattice or graph. A word lattice is a more efficient way to represent alternative hypotheses. N-best or lattice search is used for many large-vocabulary continuous speech recognition systems [20, 30, 44].

In this section we describe the representation of the n-best list and word lattice. Several algorithms to generate such an n-best-list or word lattice are discussed.

[5] In the field of artificial intelligence, the process of performing search through an integrated network of various knowledge sources is called *constraint satisfaction*.

13.3.1. *N*-best Lists and Word Lattices

Table 13.4 shows an example n-best (10-best) list generated for a North American Business (NAB) sentence. *N*-best search framework is effective only for n of the order of tens or hundreds. If the short n-best list that is generated by using less optimal models does not include the correct word sequence, the successive rescoring phases have no chance to generate the correct answer. Moreover, in a typical n-best list like the one shown in Table 13.4, many of the different word sequences are just one-word variations of each other. This is not surprising, since similar word sequences should achieve similar scores. In general, the number of n-best hypotheses might grow exponentially with the length of the utterance. Word lattices and word graphs are thus introduced to replace n-best list with a more compact representation of alternative hypotheses.

Word lattices are composed by word hypotheses. Each word hypothesis is associated with a score and an explicit time interval. Figure 13.8 shows an example of a word lattice corresponding to the n-best list example in Table 13.4. It is clear that a word lattice is more efficient representation. For example, suppose the spoken utterance contains 10 words and there are 2 different word hypotheses for each word position. The n-best list would need to have $2^{10} = 1024$ different sentences to include all the possible permutations, whereas the word lattice requires only 20 different word hypotheses.

Word graphs, on the other hand, resemble finite state automata, in which arcs are labeled with words. Temporal constraints between words are implicitly embedded in the topology. Figure 13.9 shows a word graph corresponding to the n-best list example in Table 13.4. Word graphs in general have an explicit specification of word connections that don't allow overlaps or gaps along the time axis. Nonetheless, word lattices and graphs are similar, and we often use these terms interchangeably.[6] Since an n-best list can be treated as a simple word lattice, word lattices are a more general representation of alternative hypotheses. *N*-best lists or word lattices are generally evaluated on the following two parameters:

Table 13.4 An example 10-best list for a North American Business sentence.

1.	I will tell you would I think in my office
2.	I will tell you what I think in my office
3.	I will tell you when I think in my office
4.	I would sell you would I think in my office
5.	I would sell you what I think in my office
6.	I would sell you when I think in my office
7.	I will tell you would I think in my office
8.	I will tell you why I think in my office
9.	I will tell you what I think on my office
10.	I Wilson you I think on my office

[6] We will use the term *word lattice* in the rest of this chapter..

- *Density:* In the n-best case, it is measured by how many alternative word sequences are kept in the n-best list. In the word lattice case, it is measured by the number of word hypotheses or word arcs per uttered word. Obviously, we want the density to be as small as possible for successive rescoring modules, provided the correct word sequence is included in the n-best list or word lattice.
- *The lower bound word error rate:* It is the lowest word error rate for any word sequence in the n-best list or the word lattice.

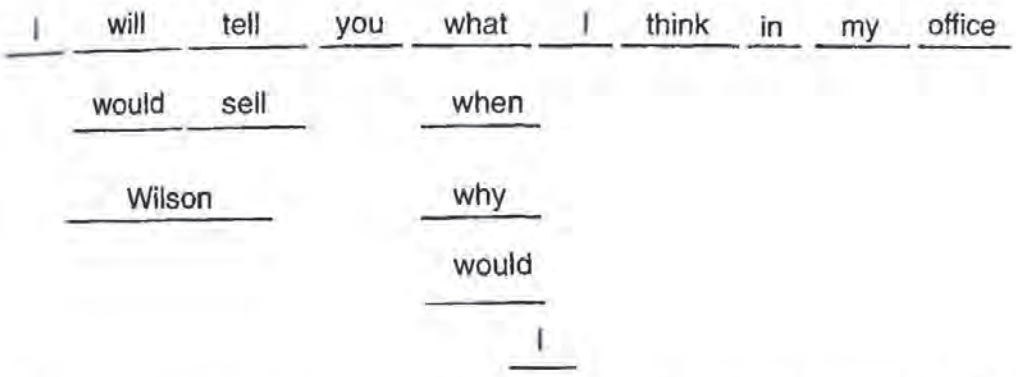


Figure 13.8 A word lattice example. Each word has an explicit time interval associated with it.

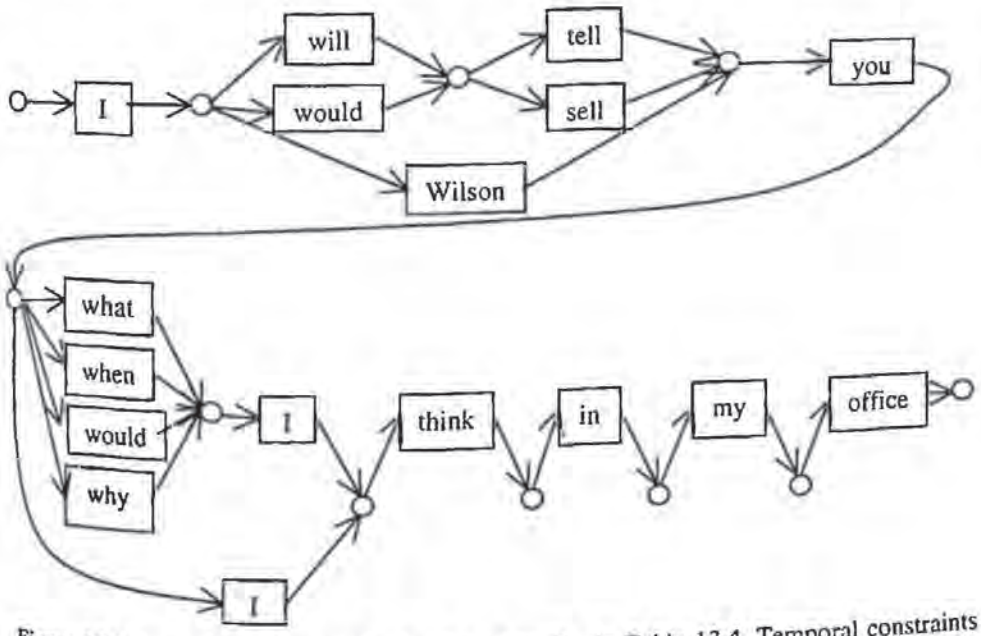


Figure 13.9 A word graph example for the n-best list in Table 13.4. Temporal constraints are implicit in the topology.

Rescoring with highly similar n-best alternatives duplicates computation on common parts. The compact representation of word lattices allows both data structure and computation sharing of the common parts among similar alternative hypotheses, so it is generally computationally less expensive to rescore the word lattice.

Figure 13.10 illustrates the general n-best/lattice search framework. Those KSs providing most constraints, at a lesser cost, are used first to generate the n-best list or word lattice. The n-best list or word lattice is then passed to the rescoring module, which uses the remaining KSs to select the optimal path. You should note that the n-best and word-lattice generators sometimes involve several phases of search mechanisms to generate the n-best list or word lattice. Therefore, the whole search framework in Figure 13.10 could involve several (> 2) phases of search mechanism.

Does the compact n-best or word-lattice representation impose constraints on the complexity of the acoustic and language models applied during successive rescoring modules? The word lattice can be expanded for higher-order language models and detailed context-dependent models, like inter-word triphone models. For example, to use higher-order language models for word lattice entails copying each word in the appropriate context of preceding words (in the trigram case, the two immediately preceding words). To use interword triphone models entails replacing the triphones for the beginning and ending phone of each word with appropriate interword triphones. The expanded lattice can then be used with detailed acoustic and language models. For example, Murveit et al. [30] report this can achieve trigram search without exploring the enormous trigram search space.

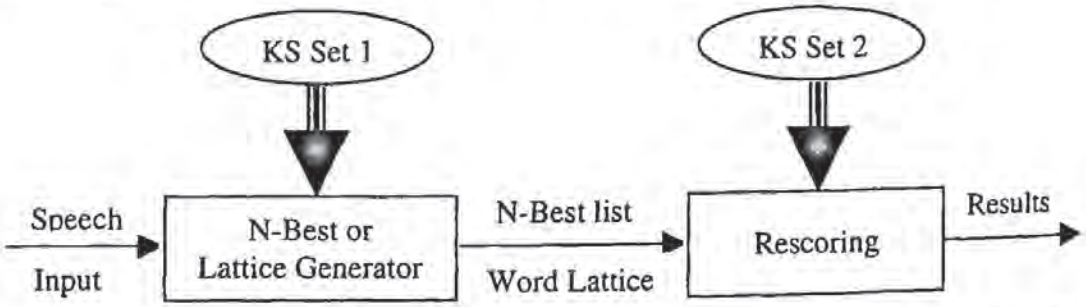


Figure 13.10 N-best/lattice search framework. The most discriminant and inexpensive knowledge sources (KSs 1) are used first to generate the n-best/lattice. The remaining knowledge sources (KSs 2, usually expensive to apply) are used in the rescoring phase to pick up the optimal solution [40].

13.3.2. The Exact *N*-best Algorithm

Stack decoding is the choice of generating n-best candidates because of its best-first principle. We can keep it generating results until it finds n complete paths; these n complete sentences form the n-best list. However, this algorithm usually cannot generate the n best candidates efficiently. The efficient n-best algorithm for time-synchronous Viterbi search was first introduced by Schwartz and Chow [39]. It is a simple extension of time-synchronous Viterbi search. The fundamental idea is to maintain separate records for paths

with distinct histories. The history is defined as the whole word sequence up to the current time t and word w. This exact n-best algorithm is also called *sentence-dependent* n-best algorithm. When two or more paths come to the same state at the same time, paths having the same history are merged and their probabilities are summed together; otherwise, only the n-best paths are retained for each state. As commonly used in speech recognition, a typical HMM state has 2 or 3 predecessor states within the word HMM. Thus, for each time frame and each state, the n-best search algorithm needs to compare and merge 2 or 3 sets of n paths into n new paths. At the end of the search, the n paths in the final state of the trellis are simply re-ordered to obtain the n-best word sequences.

This straightforward n-best algorithm can be proved to be admissible[7] in normal circumstances [40]. The complexity of the algorithm is proportional to $O(n)$, where n is the number of paths kept at each state. This is often too slow for practical systems.

13.3.3. Word-Dependent *N*-best and Word-Lattice Algorithm

Since many of the different entries in the n-best list are just one-word variations of each other, as shown in Table 13.4, one efficient algorithm can be derived from the normal 1-best Viterbi algorithm to generate the n-best hypotheses. The algorithm runs just like the normal time-synchronous Viterbi algorithm for all within-word transitions. However for each time frame t, and each word-ending state, the algorithm stores all the different words that can end at current time t and their corresponding scores in a *traceback* list. At the same time, the score of the best hypothesis at each grammar state is passed forward, as in the normal time-synchronous Viterbi search. This obviously requires almost no extra computation above the normal time-synchronous Viterbi search. At the end of search, you can simply search through the stored traceback list to get all the permutations of word sequences with their corresponding scores. If you use a simple threshold, the traceback can be implemented very efficiently to only uncover the word sequences with accumulated cost scores below the threshold. This algorithm is often referred as *traceback*-based n-best algorithm [29, 42] because of the use of the traceback list in the algorithm.

However, there is a serious problem associated with this algorithm. It could easily miss some low-cost hypotheses. Figure 13.11 illustrates an example in which word w_k can be preceded by two different words w_i and w_j in different time frames. Assuming path w_i-w_k has a lower cost than path w_j-w_k when both paths meet during the trellis search of w_k, the path w_j-w_k will be pruned away. During traceback for finding the n-best word sequences, there is only one best starting time for word w_k, determined by the best boundary between the best preceding word w_i and it. Even though path w_j-w_k might have a very low cost (let's say only marginally higher than that of w_i-w_k), it could be completely overlooked, since the path has a different starting time for word w_k.

[7] Although one can show, in the worst case, when paths with different histories have near identical scores for each state, the search actually needs to keep all paths ($> N$) in order to guarantee absolute admissibility. Under this worst case, the admissible algorithm is clearly exponential in the number of words for the utterance, since all permutations of word sequences for the whole sentence need to be kept.

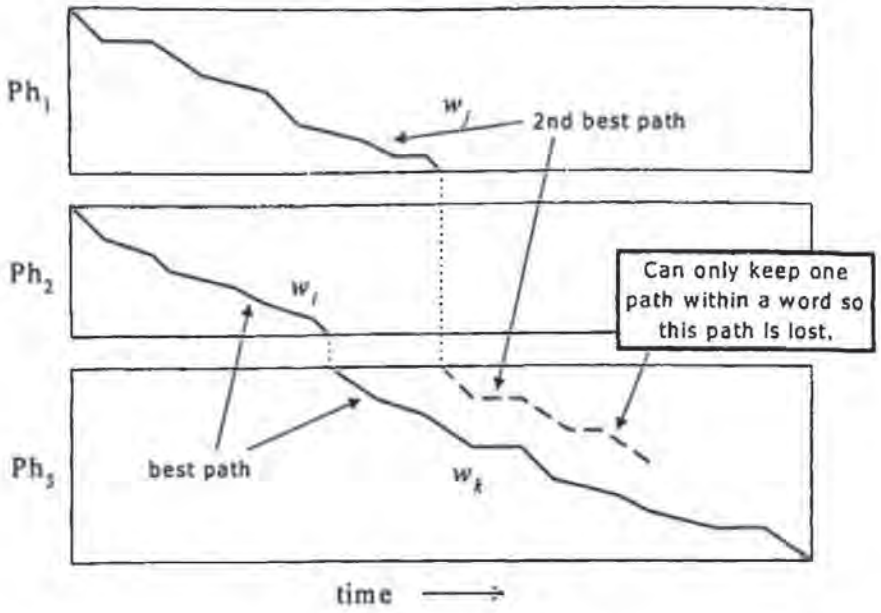


Figure 13.11 Deficiency in traceback-based n-best algorithm. The best subpath, w_i - w_k, will prune away subpath w_j - w_k while searching the word w_k; the second-best subpath cannot be recovered [40].

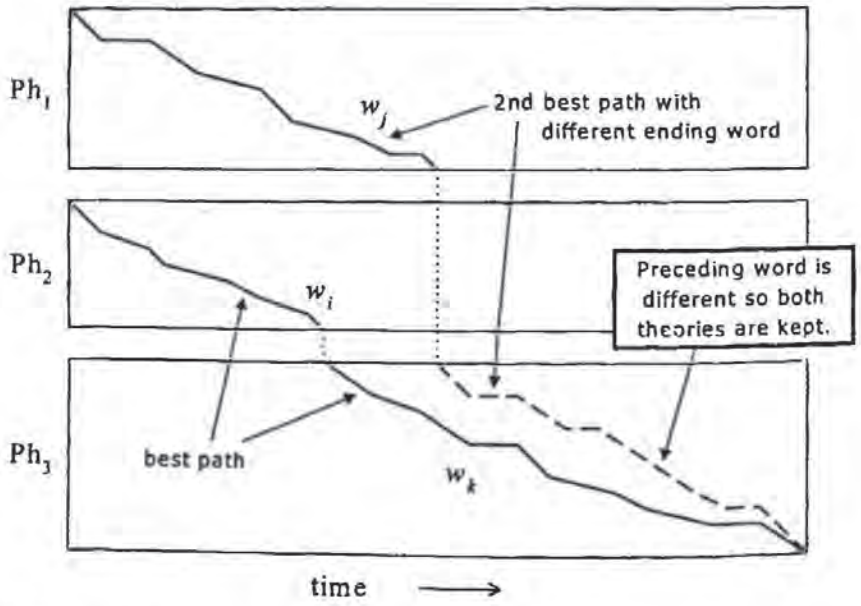


Figure 13.12 Word-dependent n-best algorithm. Both subpaths w_i - w_k and w_j - w_k are kept under the word-dependent assumption [40].

The *word-dependent* n-best algorithm [38] can alleviate the deficiency of the trace-back-based n-best algorithm, in which only one starting time is kept for each word, so the starting time is independent of the preceding words. On the other hand, in the sentence-dependent n-best algorithm, the starting time for a word depends on all the preceding words, since different histories are kept separately. A good compromise is the so-called word-dependent assumption: *The starting time of a word depends only on the immediate preceding word.* That is, given a word pair and its ending time, the boundary between these two words is independent of further predecessor words.

In the word-dependent assumption, the history to be considered for a different path is no longer the entire word sequence; instead, it is only the immediately preceding word. This allows you to keep k ($<< n$) different records for each state and each time frame in Viterbi search. Differing slightly from the exact n-best algorithm, a traceback must be performed to find the n-best list at the end of search. The algorithm is illustrated in Figure 13.12. A word-dependent n-best algorithm has a time complexity proportional to k. However, it is no longer admissible because of the word-dependent approximation. In general, this approximation is quite reasonable if the preceding word is long. The loss it entails is insignificant [6].

13.3.3.1. One-Pass *N*-best and Word-Lattice Algorithm

As presented in Section 13.1, one-pass Viterbi beam search can be implemented very efficiently using a tree lexicon. Section 13.1.2 states that multiple copies of lexical trees are necessary for incorporating language models other than the unigram. When bigram is used in lexical tree search, the successor lexical tree is predecessor-dependent. This predecessor-dependent property immediately translates into the word-dependent property,[8] as defined in Section 13.3.3, because the starting time of a word clearly depends on the immediately preceding word. This means that different word-dependent partial paths are automatically saved under the framework of predecessor-dependent successor trees. Therefore, one-pass predecessor-dependent lexical tree search can be modified slightly to output n-best lists or word graphs.

Ney et al. [31] used a word graph builder with a one-pass predecessor-dependent lexical tree search. The idea is to exploit the word-dependent property inherited from the predecessor-dependent lexical tree search. During predecessor-dependent lexical tree search, two additional quantities are saved whenever a word ending state is processed.

$\tau(t;w_i,w_j)$—Representing the optimal word boundary between word w_i and w_j, given word w_j ending at time t.

$h(w_j;\tau(t;w_i,w_j),t) = -\log P(\mathbf{x}'_\tau \mid w_j)$—Representing the cumulative cost that word w_j produces acoustic vector $\mathbf{x}_\tau, \mathbf{x}_{\tau+1}, \cdots \mathbf{x}_t$.

[8] When higher order n-gram models are used, the boundary dependence will be even more significant. For example, when trigrams are used, the boundary for a word juncture depends on the previous two words. Since we generally want a fast method of generating word lattices/graphs, bigram is often used instead of higher order n-gram to generate word lattices/graphs.

At the end of the utterance, the word lattice or n-best list is constructed by tracing back all the permutations of word pairs recorded during the search. The algorithm is summarized in Algorithm 13.3.

ALGORITHM 13.3: *ONE-PASS PREDECESSOR-DEPENDENT LEXICAL TREE SEARCH FOR N-BEST OR WORD-LATTICE CONSTRUCTION*

Step 1: For $t = 1..T$,
1-best predecessor-dependent lexical tree search;
$\forall(w_i, w_j)$ ending at t
record word-dependent crossing time $\tau(t; w_i, w_j)$;
record cumulative word score $h(w_j; \tau(t; w_i, w_j), t)$;
Step 2: Output 1-best result;
Step 3: Construct n-best or word-lattice by tracing back the word-pair records (τ and h).

13.3.4. The Forward-Backward Search Algorithm

As described Chapter 12, the ability to predict how well the search fares in the future for the remaining portion of the speech helps to reduce the search effort significantly. The one-pass search strategy, in general, has very little chance of predicting the cost for the portion that it has not seen. This difficulty can be alleviated by multipass search strategies. In successive phases the search should be able to provide good estimates for the remaining paths, since the entire utterance has been examined by the earlier passes. In this section we investigate a special type of multipass search strategy—forward-backward search.

The idea is to first perform a forward search, during which partial forward scores α for each state can be stored. Then perform a second pass search backward—that is, the second pass starts by taking the final frame of speech and searches its way back until it reaches the start of the speech. During the backward search, the partial forward scores α can be used as an accurate estimate of the heuristic function or the fast match score for the remaining path. Even though different KSs might be used in forward and backward phases, this estimate is usually close to perfect, so the search effort for the backward phase can be significantly reduced.

The forward search must be very fast and is generally a time-synchronous Viterbi search. As in the multipass search strategy, simplified acoustic and language models are often used in forward search. For backward search, either time-synchronous search or time-asynchronous A* search can be employed to find the n-best word sequences or word lattice.

13.3.4.1. Forward-Backward Search

Stack decoding, as described in Chapter 12, is based on the admissible A* search, so the first complete hypothesis found with a cost below that of all the hypotheses in the stack is guaranteed to be the best word sequence. It is straightforward to extend stack decoding to produce the n-best hypotheses by continuing to extend the partial hypotheses according to the same A* criterion until n different hypotheses are found. These n different hypotheses are destined to be the n-best hypotheses under a proof similar to that presented in Chapter 12. Therefore, stack decoding is a natural choice for producing the n-best hypotheses.

However, as described in Chapter 12, the difficulty of finding a good heuristic function that can accurately under-estimate the remaining path has limited the use of stack decoding. Fortunately, this difficulty can be alleviated by *tree-trellis forward-backward search* algorithms [41]. First, the search performs a time-synchronous forward search. At each time frame t, it records the score of the final state of each word ending. The set of words whose final states are active (surviving in the beam) at time t is denoted as Δ_t. The score of the final state of each word w in Δ_t is denoted as $\alpha_t(w)$, which represents the sum of the cost of matching the utterance up to time t given the most likely word sequence ending with word w and the cost of the language model score for that word sequence. At the end of the forward search, the best cost is obtained and denoted as α^T.

After the forward pass is completed, the second search is run in reverse (backward), i.e., considering the last frame T as the beginning one and the first frame as the final one. Both the acoustic models and language models need to be reversed. The backward search is based on A* search. At each time frame t, the best path is removed from the stack and a list of possible one-word extensions for that path is generated. Suppose this best path at time t is ph_{w_j}, where w_j is the first word of this partial path (the last expanded during backward A* search). The exit score of path ph_{w_j} at time t, which now corresponds to the score of the initial state of the word HMM w_j, is denoted as $\beta_t(ph_{w_j})$.

Let us now assume we are concerned about the one-word extension of word w_i for path ph_{w_j}. Remember that there are two fundamental issues for the implementation of A* search algorithm—(1) finding an effective and efficient heuristic function for estimating the future remaining input feature stream and (2) finding the best crossing time between w_i and w_j.

The stored forward score α can be used for solving both issues effectively and efficiently. For each time t, the sum $\alpha_t(w_i)+\beta_t(ph_{w_j})$ represents the cost score of the best complete path including word w_i and partial path ph_{w_j}. $\alpha_t(w_i)$ clearly represents a very good heuristic estimate of the remaining path from the start of the utterance until the end of the word w_i, because it is indeed the best score computed in the forward path for the same quantity. Moreover, the optimal crossing time t^* between w_i and w_j can be easily computed by the following equation:

$$t^* = \arg\min_t \left[\alpha_t(w_i)+\beta_t(ph_{w_j})\right] \tag{13.7}$$

Finally, the new path ph', including the one-word (w_i) extension, is inserted into the stack, ordered by the cost score $\alpha_t(w_i)+\beta_t(ph_{w_i})$. The heuristic function (forward scores α) allows the backward A* search to concentrate search on extending only a few truly promising paths.

As a matter of fact, if the same acoustic and language models are used in both the forward and backward search, this heuristic estimate (forward scores α) is indeed a perfect estimate of the best score the extended path will achieve. The first complete hypothesis generated by backward A* search coincides with the best one found in the time-synchronous forward search and is truly the best hypothesis. Subsequent complete hypotheses correspond sequentially to the n-best list, as they are generated in increasing order of cost. Under this condition, the size of the stack in the backward A* search need only be N. Since the estimate of future is exact, the $(N+1)^{th}$ path in the stack has no chance to become part of the n-best list. Therefore, the backward search is executed very efficiently to obtain the n-best hypotheses without exploring many unpromising branches. Of course, tree-trellis forward-backward search can also be used like most other multipass search strategies—inexpensive KSs are used in the forward search to get an estimate of α, and more expensive KSs are used in the backward A* search to generate the n-best list.

The same idea of using forward score α can be applied to time-synchronous Viterbi search in the backward search instead of backward A* search [7, 34]. For large-vocabulary tasks, the backward search can run 2 to 3 orders of magnitude faster than a normal Viterbi beam search. To obtain the n-best list from time-synchronous forward-backward search, the backward search can also be implemented in a similar way as a time-synchronous word-dependent n-best search.

13.3.4.2. Word-Lattice Generation

The forward-backward n-best search algorithm can be easily modified to generate word lattices instead of n-best lists. A forward time-synchronous Viterbi search is performed first to compute $\alpha_t(\omega)$, the score of each word ω ending at time t. At the end of the search, this best score α^T is also recorded to establish the global pruning threshold. Then, a backward time-synchronous Viterbi search is performed to compute $\beta_t(\omega)$, the score of each word ω beginning at time t. To decide whether to include word juncture $\omega_i-\omega_j$ in the word lattice/graph at time t, we can check whether the forward-backward score is below a global pruning threshold. Specifically, supposed bigram probability $P(\omega_j \mid \omega_i)$ is used, if

$$\alpha_t(\omega_i)+\beta_t(\omega_j)+\left[-\log P(\omega_j \mid \omega_i)\right] < \alpha^T+\theta \tag{13.8}$$

where θ is the pruning threshold, we will include $\omega_i-\omega_j$ in the word lattice/graph at time t. Once word juncture $\omega_i-\omega_j$ is kept, the search continues looking for the next word-pair, where the first word ω_i will be the second word of the next word-pair.

The above formulation is based on the assumption of using the same acoustic and language models in both forward and backward search. If different KSs are used in forward and backward search, the normalized α and β scores should be used instead.

13.3.5. One-Pass vs. Multipass Search

There are several real-time one-pass search engines [4, 5]. Is it necessary to build a multipass search engine based on n-best or word-lattice rescoring? We address this issue by discussing the disadvantages and advantages of multipass search strategies.

One criticism of multipass search strategies is that they are not suitable for real-time applications. No matter how fast the first pass is, the successive (backward) passes cannot start until users finish speaking. Thus, the search results need to be delayed for at least the time required to execute the successive (backward) passes. This is why the successive passes must be extremely fast in order to shorten the delay. Fortunately, it is possible to keep the delays minimum (under one second) with clever implementation of multipass search algorithms, as demonstrated by Nguyen et al. [18].

Another criticism for multipass search strategies is that each pass has the potential to introduce inadmissible pruning, because decisions made in earlier passes are based on simplified models (KSs). Search is a constraint-satisfaction problem. When a pruning decision in each search pass is made on a subset of constraints (KSs), pruning error is inevitable and is unrecoverable by successive passes. However, inadmissible pruning, like beam pruning and fast match, is often necessary to implement one-pass search in order to cope with the large active search space caused jointly by complex KSs and large-vocabulary tasks. Thus, the problem of inadmissibility is actually shared by both real-time one-pass search and multipass search for different reasons. Fortunately, in both cases, search errors can be reduced to a minimum by clever implementation and by empirically designing all the pruning thresholds carefully, as demonstrated in various one-pass and multipass systems [4, 5, 18].

Despite these concerns regarding multipass search strategies, they remain important components in developing spoken language systems. We list here several important aspects:

1. It might be necessary to use multipass search strategies to incorporate very expensive KSs. Higher-order n-gram, long-distance context-dependent models, and natural language parsing are examples that make the search space unmanageable for one-pass search. Multipass search strategies might be compelling even for some small-vocabulary tasks. For example, there are only a couple of million legal credit card numbers for the authentication task of 16-digit credit card numbers. However, it is very expensive to incorporate all the legal numbers explicitly in the recognition grammar. To first reduce search space down to an n-best list or word lattice/graph might be a desirable approach.
2. Multipass search strategies could be very compelling for spoken language understanding systems. It is problematic to incorporate most natural language

understanding technologies in one-pass search. On the other hand, n-best lists or word lattices provide a trivial interface between speech recognition and natural language understanding modules. Such an interface also provides a convenient mechanism for integrating different KSs in a modular way. This is important because the KSs could come from different modalities (like video or pen) that make one-pass integration almost infeasible. This high degree of modality allows different component subsystems to be optimized and implemented independently.

3. N-best lists or word lattices are very powerful offline tools for developing new algorithms for spoken language systems. It is often a significant task to fully integrate new modeling techniques, such as segment models, into a one-pass search. The complexity could sometimes slow down the progress of the development of such techniques, since recognition experiments are difficult to conduct. Rescoring of n-best list and lattice provides a quick and convenient alternative for running recognition experiments. Moreover, the computation and storage complexity can be kept relatively constant for offline n-best or word lattice/graph search strategies even when experimenting with highly expensive new modeling techniques. New modeling techniques can be experimented with using n-best/word-graph framework first, being integrated into the system only after significant improvement is demonstrated.
4. Besides being an alternative search strategy, n-best generation is also essential for discriminant training. Discriminant training techniques, like MMIE, and MCE described in Chapter 4, often need to compute statistics of all possible rival hypotheses. For isolated word recognition using word models, it is easy to enumerate all the word models as the rival hypotheses. However, for continuous speech recognition, one needs to use an all-phone or all-word model to generate all possible phone sequences or all possible word sequences during training. Obviously, that is too expensive. Instead, one can use n-best search to find all the near-miss sentence hypotheses that we want to discriminate against [15, 36].

13.4. SEARCH-ALGORITHM EVALUATION

Throughout this chapter we are careful in following dynamic programming principles, using admissible criteria as much as possible. However, many heuristics are still unavoidable to implement large-vocabulary continuous speech recognition in practice. Those nonadmissible heuristics include:

- Viterbi score instead of forward score described in Chapter 12.
- Beam pruning or stack pruning described in Section 13.2.2 and Chapter 12.

- Subtree dominance pruning described in Section 13.1.5.
- Fast match pruning described in Section 13.2.3.
- Rich-get-richer pruning described in Section 13.2.3.2.
- Multipass search strategies described in Section 13.3.5.

Nonadmissible heuristics generate suboptimal searches where the found path is not necessarily the path with the minimum cost. The question is, how different is this suboptimal from the true optimal path? Unfortunately, there is no way to know the optimal path unless an exhaustive search is conducted. The practical question is whether the suboptimal search hurts the search result. In a test condition where the true result is specified, you can easily compare the search result with the true result to find whether any error occurs. Errors could be due to inaccurate models (including acoustic and language models), suboptimal search, or end-point detection. The error caused by a suboptimal search algorithm is referred to as *search error* or *pruning error*.

How can we find out whether the search commits a pruning error? One of the procedures most often used is straightforward. Let $\hat{W}$ be the recognized word sequence from the recognizer and $\tilde{W}$ be the true word sequence. We need to compare the cost for these two word sequences:

$$-\log P(\hat{W} \mid X) \propto -\log\left[P(\hat{W})P(X \mid \hat{W})\right] \tag{13.9}$$

$$-\log P(\tilde{W} \mid X) \propto -\log\left[P(\tilde{W})P(X \mid \tilde{W})\right] \tag{13.10}$$

The quantity in Eq. (13.9) is supposed to be minimum among all possible word sequences if the search is admissible. Thus, if the quantity in Eq. (13.10) is greater than that in Eq. (13.9), the error is not attributed to search pruning. On the other hand, if the quantity in Eq. (13.10) is smaller than that in Eq. (13.9), there is a search error. The rationale behind the procedure described here is obvious. In the case of search errors, the suboptimal search (or nonadmissible pruning) has obviously pruned the correct path, because the cost of the correct path is smaller than the one found by the recognizer. Although we can conclude that search errors are found in this case, it does not guarantee that the search result is correct if the search can be made optimal. The reason is simply that there might be one pruned path with an incorrect word sequence and lower cost under the same suboptimal search. Therefore, the search errors represent only the upper bound that one can improve on if an optimal search is carried out. Nonetheless, finding search errors by comparing quantities in Eqs. (13.9) and (13.10) is a good measure in different search algorithms.

During the development of a speech recognizer, it is a good idea to always include the correct path in the search space. By including such a path, and some bookkeeping, one can use the correct path to help in determining all the pruning thresholds. If the correct path is pruned away during search by some threshold, some adjustment can be made to relax such a

threshold to retain the correct path. For example, one can adjust the pruning threshold for fast match if a word in $\tilde{W}$ fails to appear on the list supplied by the fast match.

13.5. CASE STUDY—MICROSOFT WHISPER

We use the decoder of Microsoft's Whisper [26, 27] discussed in Chapter 9 as a case study for reviewing the search techniques we have presented in this chapter. Whisper can handle both context-free grammars for small-vocabulary tasks and n-gram language models for large-vocabulary tasks. We describe these two different cases.

13.5.1. The CFG Search Architecture

Although context-free grammars (CFGs) have the disadvantage of being too restrictive and unforgiving, particularly with novice users, they are still one of the most popular configurations for building limited-domain applications because of the following advantages:

- Compact representation results in a small memory footprint.
- Efficient operation during decoding in terms of both space and time.
- Ease of grammar creation and modification for new tasks.

As mentioned in Chapter 12, the CFG grammar consists of a set of productions or rules that expand nonterminals into a sequence of terminals and nonterminals. Nonterminals in the grammar tend to refer to high-level task-specific concepts such as dates, font names, and commands. The terminals are words in the vocabulary. A grammar also has a nonterminal designated as its start state. Whisper also allows some regular expression operators on the right-hand side of the production for notational convenience. These operators are: or '|'; repeat zero or more times '*'; repeat one or more times '+'; and optional ([ ]). The following is a simple CFG example for *binary number:*

```
%start BINARY_NUMBER
BINARY_NUMBER: (zero | one)*
```

Without losing generality, Whisper disallows the left recursion for ease of implementation [2]. The grammar is compiled into a binary linked list format. The binary format currently has a direct one-to-one correspondence with the text grammar components, but is more compact. The compiled format is used by the search engine during decoding. The binary representation consists of variable-sized nodes linked together. The grammar format achieves sharing of subgrammars through the use of shared nonterminal definition rules.

The CFG search is conducted according to RTN framework (see Chapter 12). During decoding, the search engine pursues several paths through the CFG at the same time. Associated with each of the paths is a grammar state that describes completely how the path can be extended further. When the decoder hypothesizes the end of the current word of a path, it asks the grammar module to extend the path further by one word. There may be several alternative successor words for the given path. The decoder considers all the successor word

possibilities. This may cause the path to be extended to generate several more paths to be considered, each with its own grammar state. Also note that the same word might be under consideration by the decoder in the context of different paths and grammar states at the same time.

The decoder uses beam search to prune unpromising paths with three different beam thresholds. The state pruning threshold τ_s and new phone pruning threshold τ_p work as described in Section 13.2.2. When extending a path, if the score of the extended path does not exceed the threshold τ_h, the path to be extended is put into a pool. At each frame, for each word in the vocabulary, a winning path that extends to that word is picked from the pool, based on the score. All the remaining paths in the pool are pruned. This level of pruning gives us finer control in the creation of new paths that have potential to grow exponentially.

When two paths representing different word sequences thus far reach the end of the current word with the same grammar state at the same time, only the better path of the two is allowed to continue on. This optimization is safe, except that it does not take into account the effect of different interword left acoustic contexts on the scores of the new word that is started.

Besides beam pruning, the RGR strategy, described in Section 13.2.3.2, is used to avoid unnecessary senone computation. The basic idea is to use the linear combination of context-independent senone score and context-independent look-ahead score to determine whether the context-dependent senone evaluation is worthwhile to pursue.

All of these pruning techniques enable Whisper to perform typical 100- to 200-word CFG tasks in real time running on a 486 PC with 2 MB RAM. Readers might think it is not critical to make CFG search efficient on such a low-end platform.[9] However, it is indeed important to keep the CFG engine fast and lean. The speech recognition engine is eventually only part of an integrated application. The application will benefit if the resources (both CPU and memory) used by the speech decoder are kept as small as possible, so there are more resources left for the application module to use. Moreover, in recognition server applications, several channels of speech recognition can be performed on a single server platform if each speech recognition engine takes only a small portion of the total resources.

13.5.2. The *N*-gram Search Architecture

The CFG decoder is best suited for limited domain command and control applications. For dictation or natural conversational systems, a stochastic grammar such as n-grams provides a more natural choice. Using bigrams or trigrams leads to a large number of states to be considered by the search process, requiring an alternative search architecture.

[9] Thanks to the progress predicted by Moore's law, the current mainstream PC configuration is an order of magnitude more powerful than the configuration we list here (486 PC with 2 MB RAM) in both speed and memory.

Whisper's n-gram search architecture is based on lexical tree search as described in Section 13.1. To keep the runtime memory[10] as small as possible, Whisper does not need to allocate the entire lexical tree network statically. Instead, it dynamically builds only the portion that needs to be active. To cope with the problem of delayed application of language model scores, Whisper uses the factorization algorithm described in Section 13.1.3 to distribute the language model probabilities through the tree branches. To reduce the memory overhead of the factored language model probabilities, an efficient data structure is used for representing the lexical tree as described in Section 13.1.3.1. This data structure allows Whisper to encode factored language model probabilities in no more than the space required for the original n-gram probabilities. Thus, there is absolutely no storage overhead for using factored lexical trees.

The basic acoustic subword model in Whisper is a context-dependent senone. It also incorporates inter-word triphone models in the lexical tree search as described in Section 13.1.6. Table 13.5 shows the distribution of phoneme arcs for 20,000-word WSJ lexical tree using senones as acoustic models. Context-dependent units certainly prohibit more prefix sharing when compared with Table 13.1. However, the overall arcs in the lexical tree still represent quite a saving when compared with a linear lexicon with about 140,000 phoneme arcs. Most importantly, similar to the case in Table 13.1, most sharing is realized in the beginning prefixes where most computation resides. Moreover, with the help of context-dependent and interword senone models, the search is able to use more reliable knowledge to perform efficient pruning. Therefore, lexical tree with context-dependent models can still enjoy all the benefits associated with lexical tree search.

The search organization is evaluated on the 1992 development test set for the *Wall Street Journal* corpus with a back-off trigram language model. The trigram language model has on the order of 10^7 linguistic equivalent classes, but the number of classes generated is far fewer due to the constraints provided by the acoustic model. Figure 13.13(a) illustrates that the relative effort devoted to the trigram, bigram, and unigram is constant regardless of total search effort, across a set of test utterances. This is because the ratio of states in the language model is constant. The language model is using $\sim 2 \times 10^6$ trigrams, $\sim 2 \times 10^6$ bigrams, and 6×10^4 unigrams. Figure 13.13(b) illustrates different relative order when word hypotheses are considered. The most common context for word hypotheses is the unigram context, followed by the bigram and trigram contexts. The reason for the reversal from the state-level transitions is the partially overlapping evaluations required by each bigram context. The trigram context is more common than the bigram context for utterances that generate few hypotheses overall. This is likely because the language model models those utterances well.

[10] Here the runtime memory means the virtual memory for the decoder that is the entire image of the decoder.

Table 13.5 Configuration of the first seven levels of the 20,000-word WSJ (*Wall Street Journal*) tree; the large initial fan-out is due to the use of context-dependent acoustic models [4].

Tree Level	Number of Nodes	Fan-Out
1	655	655.0
2	3174	4.85
3	9388	2.96
4	13,703	1.46
5	14,918	1.09
6	13,907	0.93
7	11,389	0.82

To improve efficiency in dealing with tree copies due to the use of higher-order n-gram, one needs to reduce redundant computations in subtrees that are not explicitly part of the given linguistic context. One solution is to use successor trees to include only nonzero successors, as described in Section 13.1.2. Since Whisper builds the search space dynamically, it is not effective for Whisper to use the optimization techniques of the successor-tree network, such as FSN optimization, subtree isomorphism, and sharing tail optimization. Instead, Whisper uses polymorphic linguistic context assignment to reduce redundancy, as described in Section 13.1.5. This involves keeping a single copy of the lexical tree, so that each node in the tree is evaluated at most once. To avoid early inadmissible pruning of different linguistic contexts, an ε-heap of storing paths of different linguistic contexts is created for each node in the tree. The operation of such ε-heaps is in accordance with Algorithm 13.2. The depth of each heap varies dynamically according to a changing threshold that allows more contexts to be retained for promising nodes.

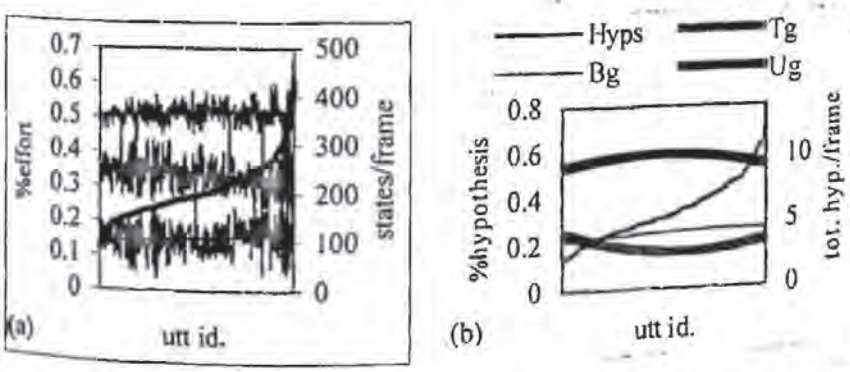


Figure 13.13 (a) Search effort for different linguistic contexts measured by number of active states in each of the three different linguistic contexts. The top series is for the bigram, then the unigram and trigram. The remaining series is the effort per utterance and is plotted on the secondary y-axis. (b) The distribution of word hypotheses with respect to their context. The top line is the unigram context, then the bigram and trigram. The remaining series is the average number of hypotheses per frame for each utterance and is plotted on the secondary y-axis [3].

Table 13.6 illustrates how the depth of the ε-heap, the active states per frame of speech, word error rate, and search time change when the value of threshold ε increases for the 20,000-word WSJ dictation task. As we can see from the table, the average heap size for active nodes is only about 1.6 for the most accurate configuration. Figure 13.14(a) illustrates the distribution of stack depths for a large data set, showing that the stack depth is small even for tree initial nodes. Figure 13.14(b) illustrates the profile of the average stack depth for a sample utterance, showing that the average stack depth remains small across an utterance.

Whisper also employs look-ahead techniques to further reduce the search effort. The acoustic look-ahead technique described in Section 13.2.3.1 attempts to estimate the probability that a phonetic HMM will participate in the final result [3]. Whisper implements acoustic look-ahead by running a CI phone-net synchronously with the search process but offset N frames in the future. One side effect of the acoustic look-ahead is to provide information for the RGR strategy, as described in Section 13.2.3.2, so the search can avoid unnecessary Gaussian computation. Figure 13.15 demonstrates the effectiveness of varying the frame look-ahead from 0 to N frames in terms of states evaluated.

When the look-ahead is increased from 0 to 3 frames, the search effort, in terms of real time, is reduced by ~40% with no loss in accuracy; however, most of that is due to reducing the number of states evaluated per frame. There is no effect on the number of Gaussians evaluated per frame (the system using continuous density) until we begin to negatively impact error rate, indicating that the acoustic space represented by the pruned states is redundant and adequately covered by the retained states prior to the introduction of search errors.

With the techniques discussed here, Whisper is able to achieved real-time performance for the continuous WSJ dictation task (60,000-word) on Pentium-class PCs. The recognition accuracy is identical to that of a standard Viterbi beam decoder with a linear lexicon.

Table 13.6 Effect of heap threshold on contexts/node, states/frame-of-speech (fos), word error rate, and search time [4].

ε	Context / node	states / fos	%error	search time
0	1.000	8805	16.4	1.0x
1.0	1.001	8808	15.5	1.0x
2.0	1.008	8898	14.4	1.0x
3.0	1.018	9252	12.4	1.07x
4.0	1.056	10224	10.5	1.16x
5.0	1.147	11832	10.3	1.36x
6.0	1.315	13749	10.0	1.60x
7.0	1.528	15342	9.9	1.81x
8.0	1.647	15984	9.9	1.86x

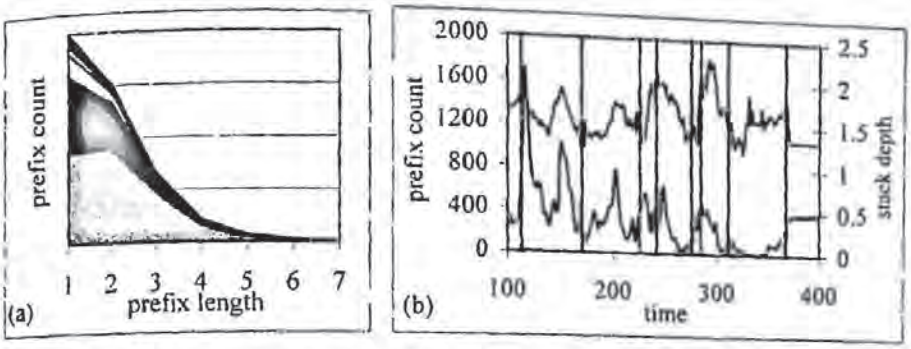


Figure 13.14 (a) A cumulative graph of the prefix count for each stack depth, starting with depth 1, showing the distribution according to prefix length. (b) The prefix count and the average stack depth with respect to one utterance. The vertical bars show the word boundaries [3].

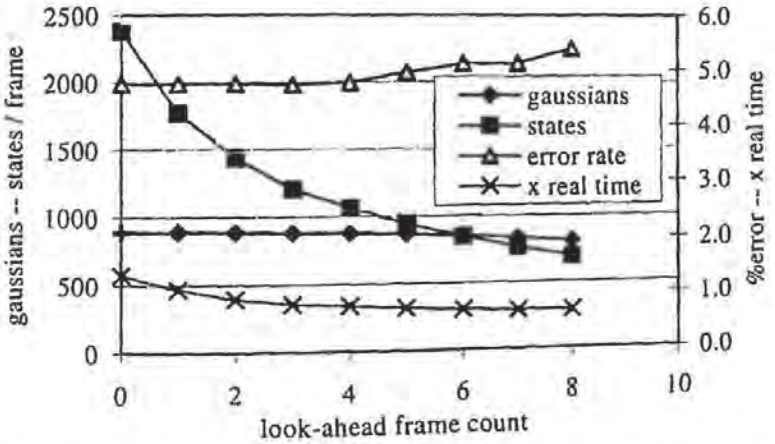


Figure 13.15 Search effort, percent error rate, and real-time factor as a function of the acoustic look-ahead. Note that search effort is the number of Gaussians evaluated per frame and the number of states evaluated per frame [3].

13.6. Historical Perspective and Further Reading

Large-vocabulary continuous speech recognition is a computationally intensive task. Real-time systems started to emerge in the late 1980s. Before that, most systems achieved real-time performance with the help of special hardware [11, 16, 25, 28]. Thanks to Moore's law and various efficient search techniques, real-time systems became a reality on a single-chip general-purpose personal computer in the 1990s [4, 34, 43].

Common wisdom in 1980s saw stack decoding as more efficient for large-vocabulary continuous speech recognition with higher-order n-grams. Time-synchronous Viterbi beam search, as described in Sections 13.1 and 13.2, emerged as the most efficient search frame-

work. It has become the most widely used search technique today. The lexical tree representation was first used by IBM as part of its allophonic fast match system [10]. Ney proposed the use of the lexical tree as the primary representation for the search space [32]. The ideas of language model factoring [4, 19] [5] and subtree polymorphism [4] enabled real-time single-pass search with higher-order language models (bigrams and trigrams). Alleva [3] and Ney [33] are two excellent articles regarding the detailed Viterbi beam search algorithm with lexical tree representation.

As mentioned in Chapter 12, fast match was first invented to speed up stack decoding [8, 9]. Ney and Ortmanns [33] and Alleva [3] extended the fast match idea to phone look-ahead in time-synchronous search by using context-independent model evaluation. In Haeb-Umbach et al. [22], a word look-ahead is implemented for a 12.3k-word speaker-dependent continuous speech recognition task. The look-ahead is performed on a lexical tree, with beam search executed every other frame. The results show a factor of 3–5 times of reduction for search space compared to the standard Viterbi beam search, while only 1–2% extra errors are introduced by word look-ahead.

The idea of multipass search strategy has long existed for knowledge-based speech recognition systems [17], where first a phone recognizer is performed, then a lexicon hypothesizer is used to locate all the possible words to form a word lattice, and finally a language model is used to search for the most possible word sequence. However, HMM's popularity predominantly shifted the focus to the unified search approach to achieve global optimization. Computation concerns led many researchers to revisit the multipass search strategy. The first n-best algorithm, described in Section 13.3.2, was published by researchers at BBN [39]. Since then, n-best and word-lattice based multipass search strategies have become important search frameworks for rapid system deployment, research tools, and spoken language understanding systems. Schwartz et al.'s paper [40] is a good tutorial on the n-best or word-lattice generation algorithms. Most of the n-best search algorithms can be made to generate word lattices/graphs with minor modifications. Other excellent discussions of multipass search can be found in [14, 24, 30].

REFERENCES

[1] Aho, A., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, 1974, Addison-Wesley Publishing Company.

[2] Aho, A.V., R. Sethi, and J.D. Ullman, *Compilers: Principles, Techniques, and Tools*, 1985, Addison-Wesley.

[3] Alleva, F., "Search Organization in the Whisper Continuous Speech Recognition System," *IEEE Workshop on Automatic Speech Recognition*, 1997.

[4] Alleva, F., X. Huang, and M.Y. Hwang, "Improvements on the Pronunciation Prefix Tree Search Organization," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1996, Atlanta, Georgia, pp. 133-136.

[5] Aubert, X., *et al.*, "Large Vocabulary Continuous Speech Recognition of *Wall Street Journal* Corpus," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1994, Adelaide, Australia, pp. 129-132.

[6] Aubert, X. and H. Ney, "Large Vocabulary Continuous Speech Recognition Using Word Graphs," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, MI, pp. 49-52.

[7] Austin, S., R. Schwartz, and P. Placeway, "The Forward-Backward Search Algorithm for Real-Time Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada, pp. 697-700.

[8] Bahl, L.R., *et al.*, "Obtaining Candidate Words by Polling in a Large Vocabulary Speech Recognition System," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1988, pp. 489-492.

[9] Bahl, L.R., *et al.*, "Matrix Fast Match: a Fast Method for Identifying a Short List of Candidate Words for Decoding," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, Glasgow, Scotland, pp. 345-347.

[10] Bahl, L.R., P.S. Gopalakrishnan, and R.L. Mercer, "Search Issues in Large Vocabulary Speech Recognition," *Proc. of the 1993 IEEE Workshop on Automatic Speech Recognition*, 1993, Snowbird, UT.

[11] Bisiani, R., T. Anantharaman, and L. Butcher, "BEAM: An Accelerator for Speech Recognition," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, pp. 782-784.

[12] Brugnara, F. and M. Cettolo, "Improvements in Tree-Based Language Model Representation," *Proc. of the European Conf. on Speech Communication and Technology*, 1995, Madrid, Spain, pp. 1797-1800.

[13] Cettolo, M., R. Gretter, and R.D. Mori, "Knowledge Integration" in *Spoken Dialogues with Computers*, R.D. Mori, ed., Academic Press, 1998, London, pp. 231-256.

[14] Cettolo, M., R. Gretter, and R.D. Mori, "Search and Generation of Word Hypotheses" in *Spoken Dialogues with Computers*, R.D. Mori, ed., 1998, London, Academic Press, pp. 257-310.

[15] Chou, W., C.H. Lee, and B.H. Juang, "Minimum Error Rate Training Based on N-best String Models," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 652-655.

[16] Chow, Y.L., *et al.*, "BYBLOS: The BBN Continuous Speech Recognition System," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1987, pp. 89-92.

[17] Cole, R.A., *et al.*, "Feature-Based Speaker Independent Recognition of English Letters," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1983, pp. 731-734.

[18] Davenport, J.C., R. Schwartz, and L. Nguyen, "Towards A Robust Real-Time Decoder," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1999, Phoenix, Arizona, pp. 645-648.

[19] Federico, M., *et al.*, "Language Modeling for Efficient Beam-Search," *Computer Speech and Language*, 1995, pp. 353-379.

[20] Gauvain, J.L., L. Lamel, and M. Adda-Decker, "Developments in Continuous Speech Dictation using the ARPA WSJ Task," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, MI, pp. 65-68.

[21] Gillick, L.S. and R. Roth, "A Rapid Match Algorithm for Continuous Speech Recognition," *Proc. of the Speech and Natural Language Workshop*, 1990, Hidden Valley, PA, pp. 170-172.

[22] Haeb-Umbach, R. and H. Ney, "A Look-Ahead Search Technique for Large Vocabulary Continuous Speech Recognition," *Proc. of the European Conf. on Speech Communication and Technology*, 1991, Genova, Italy, pp. 495-498.

[23] Haeb-Umbach, R. and H. Ney, "Improvements in Time-Synchronous Beam-Search for 10000-Word Continuous Speech Recognition," *IEEE Trans. on Speech and Audio Processing*, 1994, 2(4), pp. 353-365.

[24] Hetherington, I.L., *et al.*, "A* Word Network Search for Continuous Speech Recognition," *Proc. of the European Conf. on Speech Communication and Technology*, 1993, Berlin, Germany, pp. 1533-1536.

[25] Hon, H.W., *A Survey of Hardware Architectures Designed for Speech Recognition*, 1991, Carnegie Mellon University, Pittsburgh, PA.

[26] Huang, X., *et al.*, "From Sphinx II to Whisper: Making Speech Recognition Usable," in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds. 1996, Norwell, MA, Kluwer Academic Publishers, pp. 481-508.

[27] Huang, X., *et al.*, "Microsoft Windows Highly Intelligent Speech Recognizer: Whisper," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, pp. 93-96.

[28] Jelinek, F., "The Development of an Experimental Discrete Dictation Recognizer," *Proc. of the IEEE*, 1985, 73(1), pp. 1616-1624.

[29] Marino, J. and E. Monte, "Generation of Multiple Hypothesis in Connected Phonetic-Unit Recognition by a Modified One-Stage Dynamic Programming Algorithm," *Proc. of EuroSpeech*, 1989, Paris, pp. 408-411.

[30] Murveit, H., *et al.*, "Large Vocabulary Dictation Using SRI's DECIPHER Speech Recognition System: Progressive Search Techniques," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 319-322.

[31] Ney, H. and X. Aubert, "A Word Graph Algorithm for Large Vocabulary," *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan, pp. 1355-1358.

[32] Ney, H., *et al.*, "Improvements in Beam Search for 10000-Word Continuous Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Francisco, California, pp. 9-12.

[33] Ney, H. and S. Ortmanns, *Dynamic Programming Search for Continuous Speech Recognition*, in *IEEE Signal Processing Magazine*, 1999, pp. 64-83.

[34] Nguyen, L., *et al.*, "Search Algorithms for Software-Only Real-Time Recognition with Very Large Vocabularies," *Proc. of ARPA Human Language Technology Workshop*, 1993, Plainsboro, NJ, pp. 91-95.

[35] Nilsson, N.J., *Problem-Solving Methods in Artificial Intelligence*, 1971, New York, McGraw-Hill.

[36] Normandin, Y., "Maximum Mutual Information Estimation of Hidden Markov Models" in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds. 1996, Norwell, MA, Kluwer Academic Publishers.

[37] Odell, J.J., *et al.*, "A One Pass Decoder Design for Large Vocabulary Recognition," *Proc. of the ARPA Human Language Technology Workshop*, 1994, Plainsboro, NJ, pp. 380-385.

[38] Schwartz, R. and S. Austin, "A Comparison of Several Approximate Algorithms for Finding Multiple (N-BEST) Sentence Hypotheses," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada, pp. 701-704.

[39] Schwartz, R. and Y.L. Chow, "The N-Best Algorithm: an Efficient and Exact Procedure for Finding the N Most Likely Sentence Hypotheses," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1990, Albuquerque, New Mexico, pp. 81-84.

[40] Schwartz, R., L. Nguyen, and J. Makhoul, "Multiple-Pass Search Strategies" in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds., 1996, Norwell, MA, Klewer Academic Publishers, pp. 57-81.

[41] Soong, F.K. and E.F. Huang, "A Tree-Trellis Based Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada, pp. 705-708.

[42] Steinbiss, V., "Sentence-Hypotheses Generation in a Continuous Speech Recognition," *Proc. of EuroSpeech*, 1989, Paris, pp. 51-54.

[43] Steinbiss, V., *et al.*, "The Philips Research System for Large-Vocabulary Continuous-Speech Recognition," *Proc. of the European Conf. on Speech Communication and Technology*, 1993, Berlin, Germany, pp. 2125-2128.

[44] Woodland, P.C., *et al.*, "Large Vocabulary Continuous Speech Recognition Using HTK," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1994, Adelaide, Australia, pp. 125-128.

PART IV

TEXT-TO-SPEECH SYSTEMS

CHAPTER 14

Text and Phonetic Analysis

Text-to-speech can be viewed as a speech coding system that yields an extremely high compression ratio coupled with a high degree of flexibility in choosing style, voice, rate, pitch range, and other playback effects. In this view of TTS, the text file that is input to a speech synthesizer is a form of coded speech. Thus, TTS subsumes coding technologies discussed in Chapter 7 with the following goals:

- *Compression ratios superior to digitized wave files*—Compression yields benefits in many areas, including fast Internet transmission of spoken messages.
- *Flexibility in output characteristics*—Flexibility includes easy change of gender, average pitch, pitch range, etc., enabling application developers to give their systems' spoken output a unique individual personality. Flexibility also implies easy change of message content; it is generally easier to retype text than it is to record and deploy a digitized speech file.
- *Ability for perfect indexing between text and speech forms*—Preservation of the correspondence between textual representation and the speech wave form allows synchronization with other media and output modes, such as word-by-word reverse video highlighting in a literacy tutor reading aloud.

- *Alternative access of text content*—TTS is the most effective alternative access of text for the blind, hands-free/eyes-free and displayless scenarios.

At first sight, the process of converting text into speech looks straightforward. However, when we analyze how complicated speakers read a text aloud, this simplistic view quickly falls apart. First, we need to convert words in written forms into speakable forms. This process is clearly nontrivial. Second, to sound natural, the system needs to convey the intonation of the sentences properly. This second process is clearly an extremely challenging one. One good analogy is to think how difficult it is to drop a foreign accent when speaking a second language—a process still not quite understood by human beings.

The ultimate goal of simulating the speech of an understanding, effective human speaker from plain text is as distant today as the corresponding Holy Grail goals of the fields of speech recognition and machine translation. This is because such humanlike rendition depends on common-sense reasoning about the world and the text's relation to it, deep knowledge of the language itself in all its richness and variability, and even knowledge of the actual or expected audience—its goals, assumptions, presuppositions, and so on. In typical audio books or recordings for the visually challenged today, the human reader has enough familiarity with and understanding of the text to make appropriate choices for rendition of emotion, emphasis, and pacing, as well as handling both dialog and exposition. While computational power is steadily increasing, there remains a substantial knowledge gap that must be closed before fully human-sounding simulated voices and renditions can be created.

While no TTS system to date has approached optimal quality in the Turing test,[1] a large number of experimental and commercial systems have yielded fascinating insights. Even the relatively limited-quality TTS systems of today have found practical applications.

The basic TTS system architecture is illustrated in Chapter 1. In the present chapter we discuss text analysis and phonetic analysis whose objective is to convert words into speakable phonetic representation. The techniques discussed here are relevant to what we discussed for language modeling in Chapter 11 (like text normalization before computing *n*-gram) and for pronunciation modeling in Chapter 9. The next two modules—prosodic analysis and speech synthesis—are treated in the next two chapters.

14.1. MODULES AND DATA FLOW

The text analysis component, guided by presenter controls, is typically responsible for determining document structure, conversion of nonorthographic symbols, and parsing of language structure and meaning. The phonetic analysis component converts orthographic words to phones (unambiguous speech sound symbols). Some TTS systems assume dependency between text analysis, phonetic analysis, prosodic analysis, and speech synthesis, particularly systems based on very large databases containing long stretches of original, unmodified

[1] A test proposed by British mathematician Allan Turing of the ability of a computer to flawlessly imitate human performance on a given speech or language task [29].

digitized speech with their original pitch contours. We discuss our high-level linguistic description of those modules, based on modularity, transparency, and reusability of components, although some aspects of text and phonetic analysis may be unnecessary for some particular systems.

We assume that the entire text (word, sentence, paragraph, document) to be spoken is contained in a single, wholly visible buffer. Some systems may be faced with special requirements for continuous flow-through or visibility of only small (word, phrase, sentence) *chunks* at a time, or extremely complex timing and synchronization requirements. The basic functional processes within the text and phonetic analysis are shown schematically in Figure 14.1.

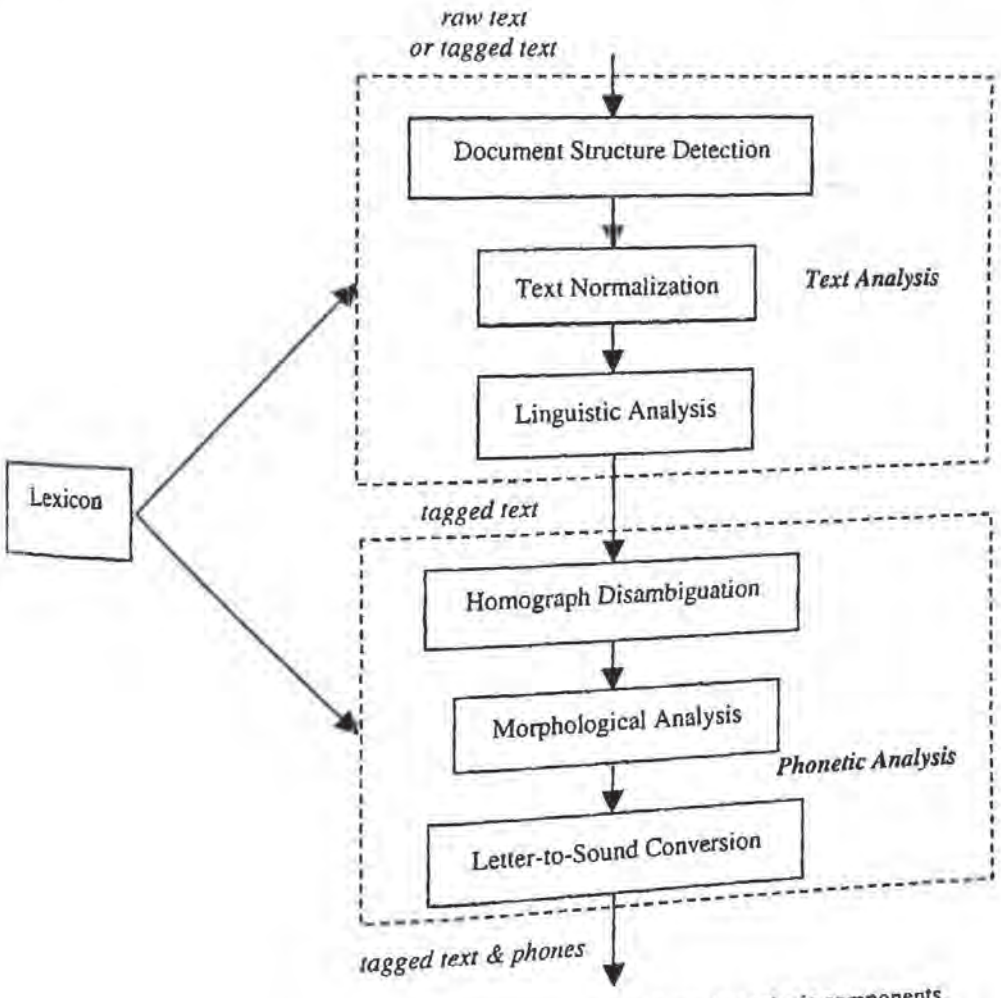


Figure 14.1 Modularized functional blocks for text and phonetic analysis components.

The architecture in Figure 14.1 brings the standard benefits of modularity and transparency. Modularity in this case means that the analysis at each level can be supplied by the most expert knowledge source, or a variety of different sources, as long as the markup conventions for expressing the analysis are uniform. Transparency means that the results of each stage could be reused by other processes for other purposes.

14.1.1. Modules

The *text analysis module* (TAM) is responsible for indicating all knowledge about the text or message that is not specifically phonetic or prosodic in nature. Very simple systems do little more than convert nonorthographic items, such as numbers, into words. More ambitious systems attempt to analyze whitespaces and punctuations to determine document structure, and perform sophisticated syntax and semantic analysis on sentences to determine attributes that help the phonetic analysis to generate correct phonetic representation and prosodic generation to construct superior pitch contours. As shown in Figure 14.1, text analysis for TTS involves three related processes:

- *Document structure detection*—Document structure is important to provide a context for all later processes. In addition, some elements of document structure, such as sentence breaking and paragraph segmentation, may have direct implications for prosody.
- *Text normalization*—Text normalization is the conversion from the variety of symbols, numbers, and other nonorthographic entities of text into a common orthographic transcription suitable for subsequent phonetic conversion.
- *Linguistic analysis*—Linguistic analysis recovers the syntactic constituency and semantic features of words, phrases, clauses, and sentences, which is important for both pronunciation and prosodic choices in the successive processes.

The task of the phonetic analysis is to convert lexical orthographic symbols to phonemic representation along with possible diacritic information, such as stress placement. Phonetic analysis is thus often referred to as grapheme-to-phoneme conversion. The purpose is obvious, since phonemes are the basic units of sound, as described in Chapter 2. Even though future TTS systems might be based on word sounding units with increasing storage technologies, homograph disambiguation and phonetic analysis for new words (either true new words being invented over time or morphologically transformed words) are still necessary for systems to correctly utter every word.

Grapheme-to-phoneme conversion is trivial for languages where there is a simple relationship between orthography and phonology. Such a simple relationship can be well captured by a handful of rules. Languages such as Spanish and Finnish belong to this category and are referred to as *phonetic languages*. English, on the other hand, is remote from pho-

netic language because English words often have many distinct origins. It is generally believed that the following three services are necessary to produce accurate pronunciations.

- *Homograph disambiguation*—It is important to disambiguate words with different senses to determine proper phonetic pronunciations, such as object (*/ah b jh eh k t/*) as a verb or as a noun (*/aa b jh eh k t/*).
- *Morphological analysis*—Analyzing the component morphemes provides important cues to attain the pronunciations for inflectional and derivational words.
- *Letter-to-sound conversion*—The last stage of the phonetic analysis generally includes general letter-to-sound rules (or modules) and a dictionary lookup to produce accurate pronunciations for any arbitrary word.

All the processes in text and phonetic analysis phases above need not to be deterministic, although most TTS systems today have deterministic processes. What we mean by *not deterministic* is that each of the above processes can generate multiple hypotheses with the hope that the later process can disambiguate those hypotheses by using more knowledge. For example, often it might not be trivial to decide whether the punctuation "." is a sentence ending mark or abbreviation mark during document structure detection. The document structure detection process can pass both hypotheses to the later processes, and the decision can then be delayed until there is enough information to make an informed decision in later modules, such as the text normalization or linguistic analysis phases. When generating multiple hypotheses, the process can also assign probabilistic information if it comprehends the underlying probabilistic structure. This flexible pipeline architecture avoids the mistakes made by early processes based on insufficient knowledge.

Much of the work done by the text/phonetic analysis phase of a TTS system mirrors the processing attempted by *natural language process* (NLP) systems for other purposes, such as automatic proofreading, machine translation, database document indexing, and so on. Increasingly sophisticated NL analysis is needed to make certain TTS processing decisions in the examples illustrated in Table 14.1. Ultimately all decisions are context driven and probabilistic in nature, since, for example, dogs might be cooked and eaten in some cultures.

Table 14.1 Examples of several ambiguous text normalization cases.

Examples	Alternatives	Techniques
Dr. Smith	*doctor* or *drive*?	abbreviation analysis, case analysis
Will you go?	yes-no or wh-question?	syntactic analysis
I ate a hot dog.	accent on *dog*?	semantic, verb/direct object likelihood
I saw a hot dog.	accent on *dog*?	discourse, pragmatic analysis

Most TTS systems today employ specialized natural language processing modules for front-end analysis. In the future, it is likely that less emphasis will be placed on construction of TTS-specific text/phonetic analysis components such as those described in [27], while more resources will likely go to general-purpose NLP systems with cross-functional potential [23]. In other words, all the modules above only perform simple processing and pass all possible hypotheses to the later modules. At the end of the text/phonetic phase, a unified NLP module then performs extensive syntactic/semantic analysis for the best decisions. The necessity for such an architectural approach is already visible in markets where language issues have forced early attention to common lexical and tokenization resources, such as Japan. Japanese system services and applications can usually expect to rely on common cross-functional linguistic resources, and many benefits are reaped, including elimination of bulk, reduction of redundancy and development time, and enforcement of systemwide consistent behavior. For example, under Japanese architectures, TTS, recognition, sorting, word processing, database, and other systems are expected to share a common language and dictionary service.

14.1.2. Data Flows

It is arguable that text input alone does not give the system enough information to express and render the intention of the text producer. Thus, more and more TTS systems focus on providing an infrastructure of standard set of markups (tags), so that the text producer can better express their semantic intention with these markups in addition to plain text. These kinds of markups have different levels of granularity, ranging from simple speed settings specified in *words per minute* up to elaborate schemes for semantic representation of concepts that may bypass the ordinary text analysis module altogether.[2] The markup can be done by internal proprietary conventions or by some standard markup, such as XML (Extensible Markup Language [35]). Some of these markup capabilities will be discussed in Sections 14.3 and 14.4.

For example, an application may know a lot about the structure and content of the text to be spoken, and it can apply this knowledge to the text, using common markup conventions, to greatly improve spoken output quality. On the other hand, some applications may have certain broad requirements such as rate, pitch, callback types, etc. For engines providing such supports, the text and/or phonetic analysis phase can be skipped, in whole or in part. Whether the application or the system has provided the text analysis markup, the structural conventions should be identical and must be sufficient to guide the phonetic analysis. The phonetic analysis module should be presented only with markup tags indicating structure or functions of textual chunks, and words in standard orthography. The similar phonetic markups could also be presented to the phonetic analysis module, the module could be skipped.

[2] This latter type of system is sometimes called *concept-to-speech* or *message-to-speech*, which is described in Chapter 17. It generally generates better speech rendering when domain-specific knowledge is provided to the system.

Internal architectures, data structures, and interfaces may vary widely from system to system. However, most modern TTS systems initially construct a simple description of an utterance or paragraph based on observable attributes, typically text words and punctuation, perhaps augmented by control annotations. This minimal initial *skeleton* is then augmented with many layers of structure hypothesized by the TTS system's internal analysis modules. Beginning with a surface stream of words, punctuation, and other symbols, typical layers of detected structure that may be added include:

- Phonemes
- Syllables
- Morphemes
- Words derived from nonwords (such as dates like "9/10/99")
- Syntactic constituents
- Relative importance of words and phrases
- Prosodic phrasing
- Accentuation
- Duration controls
- Pitch controls

We can now consider how the information needed to support synthesis of a sentence is developed in processing an example sentence such as: "A skilled electrician reported."

In Figure 14.2, the information that must be inferred from text is diagrammed. The flow proceeds as follows:

- **W(ords) → Σ, C(ontrols)**: the syllabic structure (Σ) and the basic phonemic form of a word are derived from lexical lookup and/or the application of rules. The Σ tier shows the syllable divisions (written in text form for convenience). The C tier, at this stage, shows the basic phonemic symbols for each word's syllables.
- **W(ords) → S(yntax/semantics)**: The word stream from text is used to infer a syntactic and possibly semantic structure (S tier) for an input sentence. Syntactic and semantic structure above the word would include syntactic constituents such as Noun Phrase (NP), Verb Phrase (VP), etc. and any semantic features that can be recovered from the current sentence or analysis of other contexts that may be available (such as an entire paragraph or document). The lower-level phrases such as NP and VP may be grouped into broader constituents such as Sentence (S), depending on the parsing architecture.
- **S(yntax/semantics) → P(rosody)**: The P(rosodic) tier is also called the symbolic prosodic module. If a word is semantically *important* in a sentence, that importance can be reflected in speech with a little extra phonetic prominence, called an accent. Some synthesizers begin building a prosodic structure by

placing metrical foot boundaries to the left of every accented syllable. The resulting metrical foot structure is shown as F1, F2, etc. in Figure 14.2 (some *feet* lack an accented head and are 'degenerate'). Over the metrical foot structure, higher-order prosodic constituents, with their own characteristic relative pitch ranges, boundary pitch movements, etc. can be constructed, shown in the figure as intonational phrases IP1, IP2. The details of prosodic analysis, including the meaning of those symbols, are described in Chapter 15.

The final phonetic form of the words to be spoken will reflect not only the original phonetics, but decisions made in the S and P tiers as well. For example, the P(rosody) tier adds detailed pitch and duration controls to the C(ontrol) specification that is passed to the voice synthesis component. Obviously, there can be a huge variety of particular architectures and components involved in the conversion process. Most systems, however, have some analog to each of the components presented above.

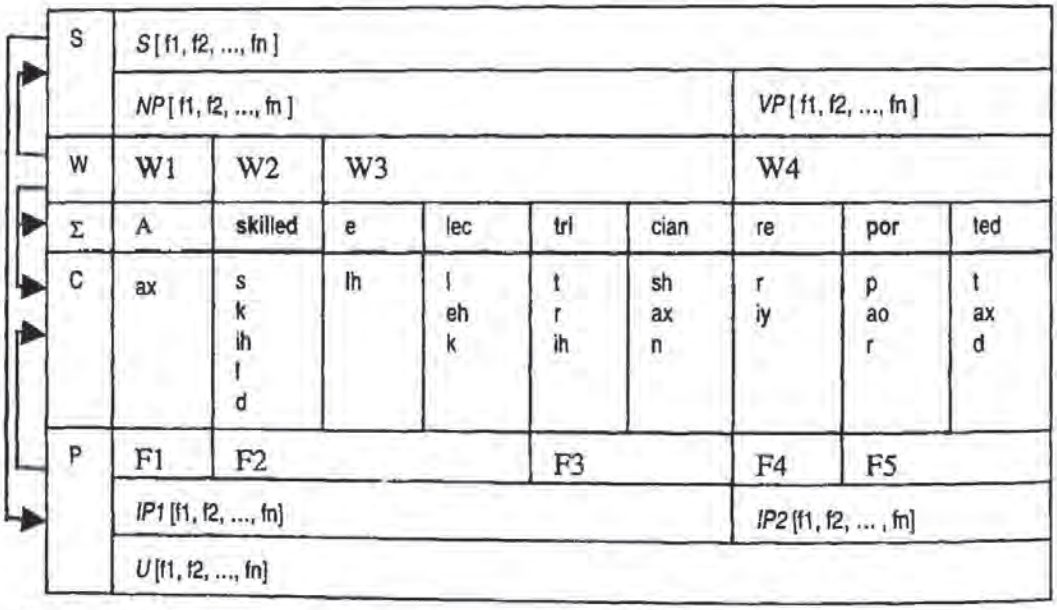


Figure 14.2 Annotation tiers indicating incremental analysis based on an input (text) sentence "A skilled electrician reported." Flow of incremental annotation is indicated by arrows on the left side.

14.1.3. Localization Issues

A major issue in the text and phonetic analysis components of a TTS system is internationalization and localization. While most of the language processing technologies in this book are exemplified by English case studies, an internationalized TTS architecture enabling minimal expense in localization is highly desirable. From a technological point of view, the text conventions and writing systems of language communities may differ substantially in arbitrary ways, necessitating serious effort in both specifying an internationalized architec-

ture for text and phonetic analysis, and localizing that architecture for any particular language.

For example, in Japanese and Chinese, the unit of *word* is not clearly identified by spaces in text. In French, interword dependencies in pronunciation realization exist (liaison). Conventions for writing numerical forms of dates, times, money, etc. may differ across languages. In French, number groups separated by spaces may need to be integrated as single amounts, which rarely occurs in English. Some of these issues may be more serious for certain types of TTS architectures than others. In general, it is best to specify a rule architecture for text processing and phonetic analysis based on some fundamental formalism that allows for language-particular data tables, and which is powerful enough to handle a wide range of relations and alternatives.

14.2. LEXICON

The most important resource for text and phonetic analysis is the TTS system lexicon (also referred to as a dictionary). As illustrated in Figure 14.1, the TTS system lexicon is shared with almost all components. The lexical service should provide the following kinds of content in order to support a TTS system:

- Inflected forms of lexicon entries
- Phonetic pronunciations (support multiple pronunciations), stress and syllabic structure features for each lexicon entry
- Morphological analysis capability
- Abbreviation and acronym expansion and pronunciation
- Attributes indicating word status, including proper-name tagging, and other special properties
- List of speakable names of all common single characters. Under modern operating systems, the characters should include all Unicode characters.
- Word part-of-speech (POS) and other syntactic/semantic attributes
- Other special features, e.g., how likely a word is to be accented, etc.

It should be clear that the requirements for a TTS system lexical service overlap heavily with those for more general-purpose NLP.

Traditionally, TTS systems have been rule oriented, in particular for grapheme-to-phoneme conversion. Often, tens of so called *letter-to-sound* (LTS) rules (described in detail in Section 14.8) are used first for grapheme-to-phoneme conversion, and the role of the lexicon has been minimized as an *exception list*, whose pronunciations cannot be predicted on the basis of such LTS rules. However, this view of the lexicon's role has increasingly been adjusted as the requirement of a sophisticated NLP analysis for high-quality TTS systems has become apparent. There are a number of ways to optimize a dictionary system. For a good overview of lexical organization issues, please see [4].

To expose different contents about a lexicon entry listed above for different TTS module, it calls for a consistent mechanism. It can be done either through a database query or a function call in which the caller sends a key (usually the orthographic representation of a word) and the desired attribute. For example, a TTS module can use the following function call to look up a particular attribute (like phonetic pronunciations or POS) by passing the attribute *att* and the result will be stored in the pointer *val* upon successful lookup. Moreover, when the lookup is successful (the word is found in the dictionary) the function returns true, otherwise it will return false instead.

```
BOOLEAN DictLookup (string word, ATTTYPE att, (VOID *) val)
```

We should also point out that this functional view of dictionary could further expand the physical dictionary as a service. The morphological analysis and letter-to-sound modules (described in Sections 14.7 and 14.8) can all be incorporated into the same lexical service. That is, underneath dictionary lookup, operation and analysis is encapsulated from users to form a uniform service.

Another consideration in the system's runtime dictionary is compression. While many standard compression algorithms exist, and should be judiciously applied, the organization and extent of the vocabulary itself can also be optimized for small space and quick search. The kinds of American English vocabulary relevant to a TTS system include:

- Grammatical function words (closed class)—about several hundred
- Very common vocabulary—about 5,000 or more
- College-level core vocabulary base forms—about 60,000 or more
- College-level core vocabulary inflected form—about 120,000 or more
- Scientific and technical vocabulary, by field—e.g., legal, medical, engineering, etc.
- Personal names—e.g., family, given, male, female, national origin, etc.
- Place names—e.g., countries, cities, rivers, mountains, planets, stars, etc.
- Slang
- Archaisms

The typical sizes of reasonably complete lists of the above types of vocabulary run from a few hundred function or closed-class words (such as prepositions and pronouns) to 120,000 or so inflected forms of college-level vocabulary items, up to several million surnames and place names. Careful analysis of the likely needs of typical target applications can potentially reduce the size of the runtime dictionary. In general, most TTS systems maintain a system dictionary with a size between 5000 and 200,000 entries. With advanced technologies in database and hashing, search is typically a nonissue for dictionary lookup. In addition, since new forms are constantly produced by various creative processes, such as acronyms, borrowing, slang acceptance, compounding, and morphological manipulation, some means of analyzing words that have not been stored must be provided. This is the topic of Sections 14.7 and 14.8.

14.3. DOCUMENT STRUCTURE DETECTION

For the purpose of discussion, we assume that all input to the TAM is an XML document, though perhaps largely unmarked, and the output is also a (more extensively marked) XML document. That is to say, all the knowledge recovered during the TAM phase is to be expressed as XML markup. This confirms the independence of the TAM from phonetic and prosodic considerations, allowing a variety of resources, some perhaps not crafted with TTS in mind, to be brought to bear by the TAM on the text. It also implies that that output of the TAM is potentially usable by other, non-TTS processes, such as normalization of language-model training data for building statistical language models (see Chapter 11). This fully modular and transparent view of TTS allows the greatest flexibility in document analysis, provides for direct *authoring* of structure and other customization, while allowing a split between expensive, multipurpose natural language analysis and the core TTS functionality. Although other text format or markup language, such as Adobe Acrobat or Microsoft Word, can be used for the same purpose, the choice of XML is obvious because it is the widely open standard, particularly for the Internet.

XML is a set of conventions for indicating the semantics and scope of various entities that combine to constitute a document. It is conceptually somewhat similar to Hypertext Markup Language (HTML), which is the exchange code for the World Wide Web. In these markup systems, properties are identified by tags with explicit scope, such as `"<b>make this phrase bold</b>"` to indicate a heavy, dark print display. XML in particular attempts to enforce a principled separation between document structure and content, on one hand, and the detailed formatting or presentation requirements of various *uses* of documents, on the other. Since we cannot provide a tutorial on XML here, we freely introduce example tags that indicate document and linguistic structure. The interpretations of these are intuitive to most readers, though, of course, the analytic knowledge underlying decisions to insert tags may be very sophisticated. It will be some time before commercial TTS engines come to a common understanding on the wide variety of text attributes that should be marked, and accept a common set of conventions. Nevertheless, it is reasonable to adopt the idea that TAM should be independent and reusable, thus allowing XML documents (which are expected to proliferate) to function for speech just as for other modalities, as indicated schematically in Figure 14.3.

TTS is regarded in Figure 14.3 as a factored process, with the text analysis perhaps carried out by human editors or by natural language analysis systems. The role of the TTS engine per se may eventually be reduced to the interpretation of structural tags and provision of phonetic information. While commercial engines of the present day are not structured with these assumptions in mind, modularity and transparency are likely to become increasingly important. The increasing acceptance of the basic ideas underlying an XML documentcentric approach to text and phonetic analysis for TTS can be seen in the recent proliferation of XML-like speech markup proposals [24, 33]. While not presenting any of these in detail, in the discussion below we adopt informal conventions that reflect and extend their basic assumptions. The structural markup exploited by the TTS systems of the

future may be imposed by XML authoring systems at document creation time, or may be inserted by independent analytical procedures. In any case the distinction between purely automatic structure creation/detection and human annotation and authoring will increasingly blur—just as in natural language translation and information retrieval domains, the distinction between machine-produced results and human-produced results has begun to blur.

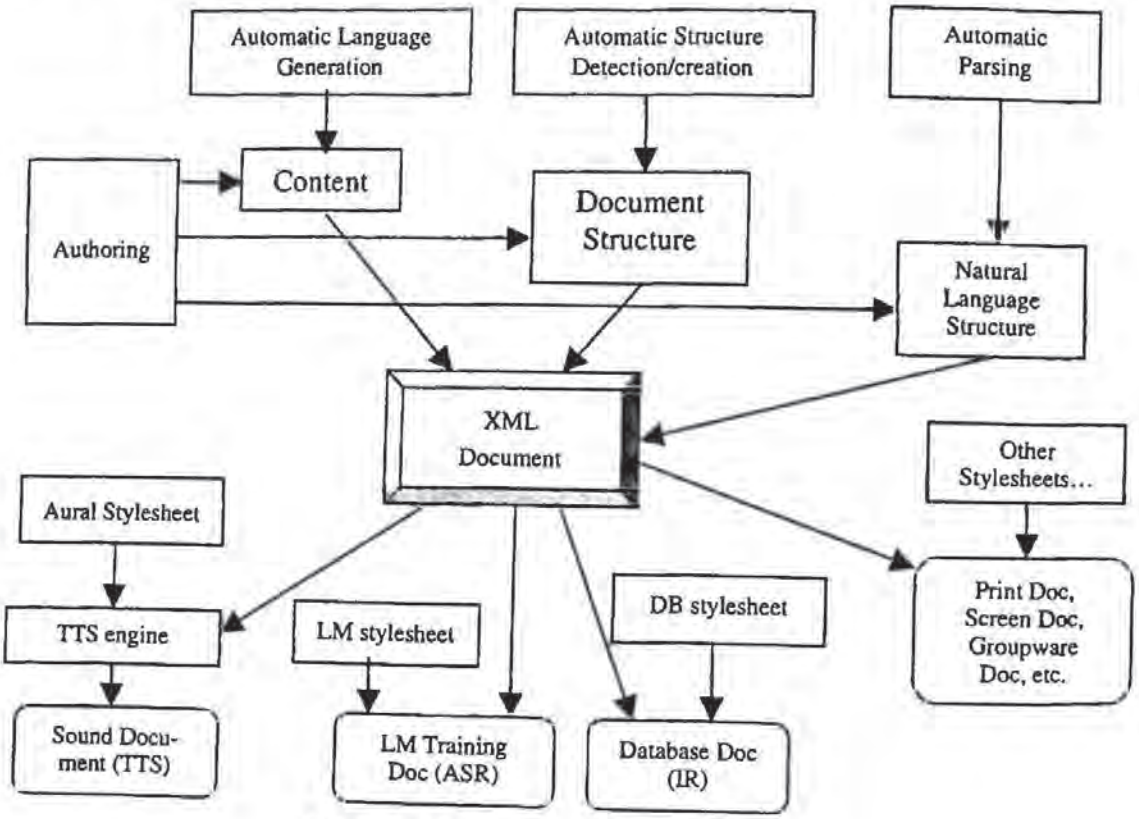


Figure 14.3 A documentcentric view of TTS.

14.3.1. Chapter and Section Headers

Section headers are a standard convention in XML document markup, and TTS systems can use the structural indications to control prosody and to regulate prosodic style, just as a professional reader might treat chapter headings differently. Increasingly, a document created on computer or intended for any kind of electronic circulation incorporates structural markup, and the TTS and audio human-computer-interface systems of the future learn to exploit this (in longer documents, the document structure markup assists in audio navigation, speedup, and skipping). For example, the XML annotation of a book at a high level might follow conventions as shown in Figure 14.4. Viewing a document in this way might lead a TTS system to insert pauses and emphasis correctly, in accordance with the structure

marked. Furthermore, an audio interface system would work jointly with a TTS system to allow easy navigation and orientation within such a structure. If future documents are marked up in this fashion, the concept of audio books, for example, would change to rely less on unstructured prerecorded speech and more on smart, XML-aware, high-quality audio navigation and TTS systems, with the output customization flexibility they provide.

For documents without explicit markup information for section and chapter headers, it is in general a nontrivial task to detect them automatically. Therefore, most TTS systems today do not make such an attempt.

```
<Book>
   <Title>The Pity of War</Title>
    <Subtitle>Explaining World War I</Subtitle>
   <Author>Niall Ferguson</Author>
   <TableOfContents>...</TableOfContents>
   <Introduction>
         <Para>...</Para>
         ...
   </Introduction>
   <Chapter>
   <ChapterTitle>The Myths of Militarism</ChapterTitle>
         <Section>
               <SectionTitle>Prophets</SectionTitle>
               <Para> ... </Para>
               ...
         </Section>
    </Chapter>
    ...
</Book>
```

Figure 14.4 An example of the XML annotation of a book.

14.3.2. Lists

Lists or bulleted items may be rendered with distinct intonational contours to indicate aurally their special status. This kind of structure might be indicated in XML as shown in Figure 14.5. Again, TTS engine designers need to get used to the idea of accepting such markup for interpretation, or incorporating technologies that can detect and insert such markup as needed by the downstream phonetic processing modules. Similar to chapter and section headers, most TTS systems today do not make an attempt to detect list structures automatically.

```
<UL>
<LI>compression</LI>
<LI>flexibility</LI>
<LI>text-waveform correspondence</LI>
</UL>
<Caption>The advantages of TTS</Caption>
```

Figure 14.5 An example of a list marked by XML.

14.3.3. Paragraphs

The paragraph has been shown to have direct and distinctive implications for pitch assignment in TTS [26]. The pitch range of good readers or speakers in the first few clauses at the start of a new paragraph is typically substantially higher than that for mid-paragraph sentences, and it narrows further in the final few clauses, before resetting for the next paragraph. Thus, to mimic a high-quality reading style in future TTS systems, the paragraph structure has to be detected from XML tagging or inferred from inspection of raw formatting. Obviously, relying on independently motivated XML tagging is, as always, the superior option, especially since this is a very common structural annotation in XML documents.

In contrast to other document structure information, paragraphs are probably among the easiest to detect automatically. The character <CR> (carriage return) or <NL> (new line) is usually a reliable clue for paragraphs.

14.3.4. Sentences

While sentence breaks are not normally indicated in XML markup today, there is no reason to exclude them, and knowledge of the sentence unit can be crucial for high-quality TTS. In fact, some XML-like conventions for text markup of documents to be rendered by synthesizers (e.g., SABLE) provide for a DIV (division) tag that could take paragraph, sentence, clause, etc. as attribute [24]. If we define sentence broadly as a primal linguistic unit that makes up paragraphs, attributes could be added to a `Sent` tag to express whatever linguistic knowledge exists about the type of the sentence as a whole:

```
<Sent type="yes-no question">
Is life so dear, or peace so sweet, as to be purchased at the price of chains and slavery?
</Sent>
```

Again, as emphasized throughout this section, such annotation could be either applied during creation of the XML documents (of the future) or inserted by independent processes. Such structure-detection processes may be motivated by a variety of needs and may exist outside the TTS system per se.

If no independent markup of sentence structure is available from an external, independently motivated document analysis or natural language system, a TTS system typically relies on simple internal heuristics to guess at sentence divisions. In email and other relatively informal written communications, sentence boundaries may be very hard to detect. In contrast to English, sentence breaking could be trivial for some other written languages. In Chinese, there is a designated symbol (a small circle ◦) for marking the end of a sentence, so the sentence breaking could be done in a totally straightforward way. However, for most Asian languages, such as Chinese, Japanese, and Thai, there is in general no space within a sentence. Thus, tokenization is an important issue for Asian languages.

In more formal English writing, sentence boundaries are often signaled by terminal punctuation from the set: { . ! ? } followed by whitespaces and an upper-case initial word. Sometimes additional punctuation may trail the '?' and '!' characters, such as close quotation marks and/or close parenthesis. The character '.' is particularly troubling, because it is, in programming terms, heavily *overloaded*. Apart from its uses in numerical expressions and Internet addresses, its other main use is as a marker of abbreviation, itself a difficult problem for text normalization (see Section 14.4). Consider this pathological jumble of potentially ambiguous cases:

> Mr. Smith came by. He knows that it costs $1.99, but I don't know when he'll be back (he didn't ask, "when should I return?")... His Web site is www.mrsmithhhhhh.com. The car is 72.5 in. long (we don't know which parking space he'll put his car in.) but he said "...and the truth shall set you free," an interesting quote.

Some of these can be resolved in the linguistic analysis module. However for some cases, only probabilistic guesses can be made, and even a human reader may have difficulty. The ambiguous sentence breaking can also be resolved in an abbreviation-processing module (described in Section 14.4.1). Any period punctuation that is not taken to signal an abbreviation and is not part of a number can be taken as end-of-sentence. Of course, as we have seen above, abbreviations are also confusable with words that can naturally end sentences, e.g., "*in*." For the measure abbreviations, an examination of the left context (checking for numeric) may be sufficient. In any case, the complexity of sentence breaking illustrates the value of passing multiple hypotheses and letting later, more knowledgeable modules (such as an abbreviation or linguistic analysis module) make decisions. Algorithm 14.1 shows a simple sentence-breaking algorithm that should be able to handle most cases.

For advanced sentence breakers, a weighted combination of the following kinds of considerations may be used in constructing algorithms for determining sentence boundaries (ordered from easiest/most common to most sophisticated):

- Abbreviation processing—Abbreviation processing is one of the most important tasks in text normalization and will be described in detail in Section 14.4.
- Rules or CART built (Chapter 4) upon features based on: document structure, whitespace, case conventions, etc.
- Statistical frequencies on sentence-initial word likelihood

- Statistical frequencies of typical lengths of sentences for various genres
- Streaming syntactic/semantic (linguistic) analysis—Syntactic/semantic analysis is also essential for providing critical information for phonetic and prosodic analysis. Linguistic analysis will be described in Section 14.5.

As you can see, a deliberate sentence breaking requires a fair amount of linguistic processing, like abbreviation processing and syntactic/semantic analysis. Since this type of analysis is typically included in the later modules (text normalization or linguistic analysis), it might be a sensible decision to delay the decision for sentence breaking until later modules, either text normalization or linguistic analysis. In effect, this arrangement can be treated as the document structure module passing along multiple hypotheses of sentence boundaries, and it allows later modules with deeper linguistic knowledge (text normalization or linguistic analysis) to make more intelligent decisions.

Finally, if a long buffer of unpunctuated words is presented, TTS systems may impose arbitrary limits on the length of a sentence for later processing. For example, the writings of the French author Marcel Proust contain some sentences that are several hundred words long (average sentence length for ordinary prose is about 15 to 25 words).

ALGORITHM 14.1: *A SIMPLE SENTENCE-BREAKING ALGORITHM*

1. **If** found punctuation ./!/? advance one character and **goto 2.**
   else advance one character and **goto 1.**
2. **If** not found whitespace advance one character and **goto 1.**
3. **If** the character is period (.) **goto 4.**
   else goto 5.
4. Perform abbreviation analysis.
   If not an abbreviation **goto 5.**
   else advance one character and **goto 1.**
5. Declare a sentence boundary and sentence type ./!/?
   Advance one character and **goto 1.**

14.3.5. Email

TTS could be ideal for reading email over the phone or in an eyes-busy situation such as when driving a motor vehicle. Here again we can speculate that XML-tagged email structure, minimally something like the example in Figure 14.6, will be essential for high-quality prosody, and for controlling the audio interface, allowing skips and speedups of areas the user has defined as less critical, and allowing the system to announce the function of each block. For example, the `sig` (signature) portion of email certainly has a different semantic function than the main message text and should be clearly identified as such, or skipped, at the listener's discretion. Modern email systems are providing increasingly sophisticated

support for structure annotation such as that exemplified in Figure 14.6. Obviously, the email document structure can be detected only with appropriate tags (like XML). It is very difficult for a TTS system to detect it automatically.

```
<message>
  <header>
      <date>11 June 1998</date>
      <from>Leslie</from>
      <to>Jo</to>
      <subject>Surf's Up!</subject>
  </header>
  <body> ... </body>
  <sig>Freedom's just another word for nothing left to lose</sig>
</message>
```

Figure 14.6 An example of email marked by XML.

14.3.6. Web Pages

All the comments about TTS reliance on XML markup of document structure can be applied to the case of HTML-marked Web page content as well. In addition to sections, headers, lists, paragraphs, etc., the TTS systems should be aware of XML/HTML conventions such as links (link name) and perhaps apply some distinctive voice quality or prosodic pitch contour to highlight these. The size and color of the section of text also provides useful hints for emphasis. Moreover, the TTS system should also integrate the rendering of audio and video contents on the Web page to create a genuine multimedia experience for the users. More could be said about the rendition of Web content, whether from underlying XML documents or HTML-marked documents prepared specifically for Web presentation. In addition, the World Wide Web Consortium has begun work on standards for aural stylesheets that can work in conjunction with standard HTML to provide special direction in aural rendition [33].

14.3.7. Dialog Turns and Speech Acts

Not all text to be rendered by a TTS system is standard written prose. The more expressive TTS systems could be tasked with rendering natural conversation and dialog in a spontaneous style. As with written documents, the TTS system has to be guided by XML markup of its input. Various systems for marking *dialog turns* (change of speaker) and *speech acts* (the mood and functional intent of an utterance)[3] are used for this purpose, and these annotations will trigger particular phonetic and prosodic rules in TTS systems. The speech act coding

[3] Dialog modeling and the concepts of *dialog turns* and *speech acts* are described in detail in Chapter 17.

schemes can help, for example, in identifying the speaker's intent with respect to an utterance, as opposed to the utterance's structural attributes. The prosodic contour and voice quality selected by the TTS system might be highly dependent on this functional knowledge.

For example, a syntactically well-formed question might be used as information solicitation, with the typical utterance-final pitch upturn as shown in the following:

<REQUEST_INFO>Can you hand me the wrench?</REQUEST_INFO>

But if the same utterance is used as a command, the prosody may change drastically.

<DIRECTIVE>Can you hand me the wrench.</DIRECTIVE>

Research on speech act markup-tag inventories (see Chapter 17) and automatic methods for speech act annotation of dialog is ongoing, and this research has the property considered desirable here, in that it is independently motivated (useful for enhancing speech recognition and language understanding systems). Thus, an advanced TTS system should be expected to exploit dialog and speech act markups extensively.

14.4. TEXT NORMALIZATION

Text often include abbreviations (e.g., FDA for *Food and Drug Administration*) and acronyms (SWAT for *Special Weapons And Tactics*). Novels and short stories may include *spoken* dialog interspersed with exposition; technical manuals may include mathematical formulae, graphs, figures, charts and tables, with associated captions and numbers; email may require interpretation of special conventional symbols such as *emoticons* [e.g., :-) means smileys], as well as Web and Internet address formats, and special abbreviations (e.g., IMHO means *in my humble opinion*). Again, any text source may include part numbers, stock quotes, dates, times, money and currency, and mathematical expressions, as well as standard ordinal and cardinal formats. Without context analysis or prior knowledge, even a human reader would sometimes be hard pressed to give a perfect rendition of every sequence of nonalphabetic characters or of every abbreviation. *Text normalization* (TN) is the process of generating normalized orthography (or, for some systems, direct generation of phones) from text containing words, numbers, punctuation, and other symbols. For example, a simple example is given as follows:

The 7% Solution → THE SEVEN PER CENT SOLUTION

Text normalization is an essential requirement not only for TTS, but also for the preparation of training text corpora for acoustic-model and language-model construction.[4] In addition, speech dictation systems face an analogous problem of *inverse* text normalization for document creation from recognized words, and such systems may depend on knowledge sources similar to those described in this section. The example of an inverse text normalization for the example above is given as follows:

[4] For details of acoustic and language modeling, please refer to Chapters 9 and 11.

THE SEVEN PER CENT SOLUTION → The 7% Solution

Modular text normalization components, which may produce output for multiple downstream consumers, mark up the exemplary text along the following lines:

The <tn snor="SEVEN PER CENT">7%</tn> Solution

The `snor` tag stands for *Standard Normalized Orthographic Representation.*[5] For TTS, input text may include multisentence paragraphs, numbers, dates, times, punctuation, symbols of all kinds, as well as interpretive annotations in a TTS markup language, such as tags for word emphasis or pitch range. Text analysis for TTS is the work of converting such text into a stream of normalized orthography, with all relevant input tagging preserved and new markup added to guide the subsequent modules. Such interpretive annotations added by text analysis are critical for phonetic and prosodic generation phases to produce desired output. The output of the text normalizer may be deterministic, or may preserve a full set of interpretations and processing history with or without probabilistic information to be passed along to later stages. We once again assume that XML markup is an appropriate format for expressing knowledge that can be created by a variety of external processes and exploited by a number of technologies in addition to TTS.

Since today's TTS systems typically cannot expect that their input be independently marked up for text normalization, they incorporate internal technology to perform this function. Future systems may piggyback on full natural language processing solutions developed for independent purposes. Presently, many incorporate minimal, TTS-specific hand-written rules [1], while others are loose agglomerations of modular, task-specific statistical evaluators [3].

For some purposes, an architecture that allows for a set or lattice of possible alternative expansions may be preferable to deterministic text normalization, like the *n-best* lists or *word graph* offered by the speech recognizers described in Chapter 13. Alternatives known to the system can be listed and ranked by probabilities that may be learnable from data. Later stages of processing (linguistic analysis or speech synthesis) can either add knowledge to the lattice structure or recover the best alternative, if needed. Consider the fragment "*at 8 am I . . .* " in some informal writing such as email. Given the flexibility of writing conventions for pronunciation, *am* could be realized as either *A. M.* (the numeric context seems to cue at times) or the auxiliary verb *am*. Both alternatives could be noted in a descriptive lattice of covering interpretations, with confidence measures if known (Table 14.2).

Table 14.2 Two alternative interpretations for sentence fragment "*At 8 am I ...*".

At 8 am I ...	At <time> eight am </time> I ...
At 8 am I ...	At <number> eight </number> am I ...

[5] SNOR, or Standard Normalized Orthographic Representation, is a uniform way of writing words and sentences that corresponds to spoken rendition. SNOR-format sentence texts are required as reference material for many Defense Advanced Research Project Agency and National Institutes of Standards and Technology-sponsored standard speech technology evaluation procedures.

If the potential ambiguity in the interpretation of *am* in the above pair of examples is simply noted, and the alternatives retained rather than suppressed, the choice can be made by a later stage of syntactic/semantic processing. Note another feature of this example—the rough irregular abbreviation form for antemeridian, which by prescriptive convention hopes that high-quality TTS processing can rely entirely on *standard* stylistic conventions. That observation also applies to the *obligatory* use of "?" for all questions.

Specific architectures for the text normalization component of TTS may be highly variable, depending on the system architect's answers to the following questions:

- Are cross-functional language processing resources mandated, or available?
- If so, are phonetic forms, with stress or accent, and normalized orthography, available?
- Is a full syntactic and semantic analysis of input text mandated, or available?
- Can the presenting application add interpretive knowledge to structure the input (text)?
- Are there interface or pipelining requirements that preclude lattice alternatives at every stage?

Because of this variability in requirements and resources, we do not attempt to formally specify a single, all-purpose architectural solution here. Rather, we concentrate on describing the text normalization challenges any system has to face. We note where solutions to these challenges are more readily realized under particular architectural assumptions.

All text normalization consists of two phases: identification of type, and expansion to SNOR or other unambiguous representation. Much of the identification phase, dealing with phenomena of sentence boundary determination, abbreviation expansion, number spell-out, etc., can be modeled as regular expression (see Chapter 11). This raises an interesting architectural issue. Imagine a system based entirely on regular *finite state transducers* (FST, see Chapter 11), as in [27], which enforces an appealing uniformity of processing mechanism and internal structure description. The FST permits a lattice-style representation that does not require premature resolution of any structural choice. An entire text analysis system can be based on such a representation. However, as long as a system confines its attention to issues that commonly come under the heading of text normalization, such as number formats, abbreviations, and sentence breaking, a simpler regular-expression-based uniform mechanism for rule specification and structure representation may be adequate.

Alternatively, TTS systems could make use of advanced tools such as, for example, the lex and yacc tools [17], which provide frameworks for writing customized lexical analyzers and context-free grammar parsers, respectively. In the discussion of typical text normalization requirements below, examples will be provided and then a fragment of Perl pattern-matching code will be shown that allows matching of the examples given. Perl notation [36] is used as a convenient short-hand representing any equivalent regular expressionparsing system and can be regarded as a subset of the functionality provided by any regular expression, FST, or context-free grammar tool set that a TTS software architect may choose to employ. Only a small subset of the simple, fairly standard Perl conventions

to employ. Only a small subset of the simple, fairly standard Perl conventions for regular expression matching are used, and comments are provided in our discussion of text normalization.

A text normalization system typically adds identification information to assist subsequent stages in their tasks. For example, if the TN subsystem has determined with some confidence that a given digit string is a phone number, it can associate XML-like tags with its output, identifying the corresponding normalized orthographic chunk as a candidate for special phone-number intonation. In addition, the identification tags can guide the lexical disambiguation of terms for other processes, like phonetic analysis in TTS systems and training data preparation for speech recognition.

Table 14.3 shows some examples of input fragments with a relaxed form of output normalized orthography. It illustrates a possible ambiguity in TN output. In the (contrived) example, the ambiguity is between a place name and a hypothetical individual named perhaps *Steve* or *Samuel* Asia. Two questions arise in such cases. The first is format of specification. The data between submodules in a TTS system can be passed (or be placed in a centrally viewable *blackboard* location) as tagged text or in a binary format. This is an implementation detail. Most important is that all possibilities known to the TN system be specified in the output, and that confidence measures from the TN, if any, be represented. For example, in many contexts, *South Asia* is the more likely spell-out of *S. Asia*, and this should be indicated implicitly by ordering output strings, or explicitly with probability numbers. The decision could then be delayed until one has enough information in the later module (like linguistic analysis) to make the decision in an informed manner.

Table 14.3 Examples of the normalized output using XML-like tags for text normalization.

Dr. King	<title> DOCTOR </title> KING
7%	<number>SEVEN<ratio>PERCENT</ratio> </number>
S. Asia	<toponym> SOUTH ASIA </toponym> OR <psn_name><initial>S</initial>ASIA</psn_name>

14.4.1. Abbreviations and Acronyms

As noted above, a period is an important but not completely reliable clue to the presence of an abbreviation. Periods may be omitted or misplaced in text for a variety of reasons. For similar reasons of stylistic variability and a writer's (lack of) care and skill, capitalization, another potentially important clue, can be variable as well. For example, all the representations of the abbreviation for *post script* listed below have been observed in actual mail and email. A system must therefore combine knowledge from a variety of contextual sources, such as document structure and origin, when resolving abbreviations:

PS. Don't forget your hat.
Ps. Don't forget your hat.
P.S. Don't forget your hat.
P.s. Don't forget your hat.

And *P.S.*, when examined out of context, could be personal name initials as well. Of course, a given TTS system's user may be satisfied with the simple spoken output */p iy ae s/* in cases such as the above, obviating the need for full interpretation. But at a minimum, when *fallback to letter pronunciation* is chosen, the TTS system must attempt to ensure that some obvious spell-out is not being overlooked. For example, a system should not render the title in *Dr. Jones* as letter names */d iy aa r/*.

Actually, any abbreviation is potentially ambiguous, and there are several distinct types of ambiguity. For example, there are abbreviations, typically quantity and measure terms, which can be realized in English as either plural or singular depending on their numeric coefficient, such as *mm* for *millimeter(s)*. This type of ambiguity can get especially tricky in the context of conventionally frozen items. For example, *9mm ammunition* is typically spoken as *nine millimeter ammunition* rather than *nine millimeters ammunition*.

Next, there are forms that can, with appropriate syntactic context, be interpreted either as abbreviations or as simple English words, such as in (inches), particularly at the end of sentences.

Finally, many, perhaps most, abbreviations have entirely different abbreviation spell-outs depending on semantic context, such as *DC* for *direct current* or *District of Columbia*. This variability makes it unlikely that any system ever performs perfectly. However, with sufficient training data, some statistical guidelines for interpretation of common abbreviations in context can be derived. Table 14.4 shows a few more examples of this most difficult type of ambiguity.

An advanced TTS system should attempt to convert reliably at least the following abbreviations:

- Title—Dr., MD, Mr., Mrs., Ms., St. (Saint), ... etc.
- Measure—ft., in., mm, cm (centimeter), kg (kilogram), ... etc.
- Place names—CO, LA, CA, DC, USA, St. (street), Dr. (drive), ... etc.

Table 14.4 Some ambiguous abbreviations.

CO	Colorado	commanding officer
	conscientious objector	carbon monoxide
IRA	Individual Retirement Account	Irish Republican Army
MD	Maryland	doctor of medicine
	muscular dystrophy	

Abbreviation disambiguation usually can be resolved by POS (part-of-speech) analysis. For example, whether *Dr.* is *Doctor* or *Drive* can be resolved by examining the POS features of the previous and following words. If the abbreviation is followed by a capitalized personal name, it can be expanded as *Doctor*, whereas if the abbreviation is preceded by a capitalized place name, a number, or an alphanumeric (like 120th), it will be expanded as *Drive*. Although the example above is resolved via a series of heuristic rules, the disambiguation (POS analysis) can also be done by a statistical approach. In [6], the POS tags are determined based on the most likely POS sequence using POS trigram and lexical-POS unigram. Since an abbreviation can often be distinguished by its POS feature, the most likely POS sequence of the sentence discovered by the trigram search then provides the best guess of the POS (thus the usage) for abbreviations. We describe POS tagging in more detail in Section 14.5.

Other than POS information, the lexical entries for abbreviations should include all features and alternatives necessary to generate a lattice of possible analyses. For example, a typical abbreviation's entry might include information as to whether it could be a word (like *in*), whether period(s) are optional or required, whether plural variants must be generated and if so under what circumstances, whether numerical specification is expected or required, etc.

Acronyms are words created from the first letters or parts of other words. For example, SCUBA is an acronym for *self-contained underwater breathing apparatus*. Generally, to qualify as a true acronym, a letter sequence should reflect normal language phonotactics, such as a reasonable alternation of consonants and vowels. From a TTS system's point of view, the distinctions between acronyms, abbreviations, and plain new or unknown words can be unclear. Many acronyms can be entered into the TTS system lexicon just as ordinary words would be. However, unknown acronyms (not listed in the lexicon) may occasionally be encountered. Although an acronym's case property can be a significant clue to identification, it is often unclear how to speak a given sequence of upper-case letters. Most TTS systems, failing to locate the sequence in the acronym dictionary, spell it out letter-by-letter. Other systems attempt to determine whether the sequence is inherently *speakable*. For example, DEC might be inherently speakable, while FCC is not formed according to normal word phonotactics. When something speakable is found, it is processed via the normal letter-to-sound rules, while something *unspeakable* would be spelled out letter-by-letter. Yet other systems might simply feed the sequence directly to the letter-to-sound rules (see Section 14.8), just as they would any other unknown word. As with all such problems, a larger lexicon usually provides superior results.

The general algorithm for abbreviations and acronyms expansion in text normalization is summarized in Algorithm 14.2. The algorithm assumes that tokenization and POS tagging have been done for the whole sentence. Abbreviation expansion is determined by the POS tags of the potential abbreviation candidates. Acronym expansion is done exclusively by table lookup, and letter-by-letter spell-out is used when acronyms cannot be found in the acronym table.

ALGORITHM 14.2: *ABBREVIATIONS AND ACRONYMS EXPANSION*

1. If word token *w* is not in abbreviation table and *w* contains only capital letters **goto 3.**
2. **Abbreviation Expansion**
   If the POS tag of *w* and the correspondent abbreviation match
   Abbreviation expansion by inserting SNOR and interpretive annotation tags
   Advance one word and **goto 1.**
3. **Acronym Expansion**
   If *w* is in the predefined acronym table
   Acronym expansion by inserting SNOR and interpretive annotation tags
   according to acronym expansion table
   else spell out *w* letter-by-letter
4. Advance one word and **goto 1.**

14.4.2. Number Formats

Numbers occur in a wide variety of formats and have a wide variety of contextually dependent reading styles. For example, the digits 370 in the context of the product name *IBM 370 mainframe computer* typically are read as *three seventy*, while in other contexts 370 would be read as *three hundred seventy* or *three hundred and seventy*. In a phone number, such as 370-1111, the string would normally be read as *three seven oh*, while in still other contexts it might be rendered as *three seven zero*. A text analysis system can incorporate rules, perhaps augmented by probabilities, for these situations, but might never achieve perfection in all cases. Phone numbers are a practical place to start, and their treatment illustrates some of the general issues relevant to the other number formats which are covered below.

14.4.2.1. Phone Numbers

Phone numbers may include prefixes and area codes and may have dashes and parentheses as separators. Examples are shown in Table 14.5.

The first two examples have *prefix* codes, while the next four have area codes with minor formatting differences. The final two examples are possible international-format phone numbers. A basic Perl regular expression pattern to subsume the commonality in all the local domestic numbers can be defined as follows:

```
$us_basic = '([0-9]{3}\-[0-9]{4})';
```

This defines a pattern subpart to match 3 digits, followed by a separator dash, followed by another 4 digits. Then the pattern to match the prefix type would be:

```
/([0-9]{1})[\/ -]($us_basic)/
```

Table 14.5 Some different written representations of phone numbers.

9-999-4118
9 345-5555
(617) 932-9209
(617) 932-9209
716-123-4568
409/845-2274
+49 (228) 550-381
+49-228-550-381

In the first example above, this leaves the system pattern variable $1 (corresponding to the first set of capture parentheses in the pattern) set to 9, and $2 (the second set of capture parentheses) set to 999-4118. Then a separate set of tables, indexed by the rule name and the pattern variable contents, could provide orthographic spell-outs for the digits. Clearly a balance has to be struck between the number of pattern variables provided in the expression and the overall complexity of the expression, vis-à-vis the complexity and sophistication of the indexing scheme of the spell-out tables. For example, the $us_basic could be defined to incorporate parentheses capture on the first three digits and the remaining four separately, which might lead to a simpler spell-out table in some cases.

The pattern to match the area code types could be:

```
/(\([0-9]{3}\))[\/ -]($us_basic)/
```

These patterns could be endlessly refined, expanded, and layered to match strings of almost arbitrary complexity. A balance has to be struck between number and complexity of distinct patterns. In any case, no matter how sophisticated the matching mechanism, arbitrary or at best probabilistic decisions have to be made in constructing a TTS system. For example, in matching an area code type, the rule architect must decide how much and what kind of whitespace separation the matching system tolerates between the area code and the rest of the number before a phone-number match is considered unlikely. Or, as another example, does the rule architect allow new lines or other formatting characters to appear between the area code and the basic phone number? These kinds of decisions must be explicitly considered, or made by default, and should be specified to a reasonable degree in user documentation. There are a great many other phone number formats and issues that are beyond the scope of this treatment.

Once a certain type of pattern requires a conversion to normalized orthography, the question of how to perform the conversion arises. The conversion characters can be aligned with the identification, so that conversion occurs implicitly during the pattern matching process. Another way is to separate the conversion from the identification phase. This may or may not lead to gains in efficiency and elimination of redundancy, depending on the

overall architecture of the system and whether and how components are expected to be reused. A version of this second approach is sketched here.

Suppose that the pattern match variable $1 has been set to 617 by one of the identification-phase pattern matches described above. Another list can provide pointers to conversion tables, indexed by the rule name or number and the variable name. So for the rule that can match area codes, the relevant entry would be:

```
Identification rule     Variable     Spellout table
Area-Phone              $1           LITERAL_DIGIT
```

The LITERAL_DIGIT spell-out rule set, when presented with the 617 character sequence (the value of $1), simply generates the normalized orthography *six one seven*, by table lookup. In this simple and straightforward approach, spell-out tables such as LITERAL_DIGIT can be reused for portions of a wide variety of identification rules. Other simple numeric spell-out tables would cover different styles of numeric reading, such as *pairwise* style (e.g., *six seventeen*), full decimal with tens, hundreds, thousands units (*six hundred seventeen*), and so on. Some spellout tables may require processing code to supplement the basic table lookup. Additional examples of spell-out tables are not provided for the various other types of text normalization entities exemplified below, but would function similarly.

14.4.2.2. Dates

Dates may be specified in a wide variety of formats, sometimes with a mixture of orthographic and numeric forms. Note that dates in TTS suffer from a mild form of the century-date-change uncertainty (the infamous Y2K bug), so a form such as 5/7/37 may in the future be ambiguous, in its full form, between 1937 and 2037. The safest course is to say as little as possible, i.e., "*five seven thirty seven*", or even "*May seventh, thirty seven*", rather than attempt "*May seventh, nineteen thirty seven*". Table 14.6 shows a variety of date formats and associated normalized orthography.

Table 14.6 Various date formats.

12/19/94 (US)	December nineteenth ninety four
19/12/94 (European)	December nineteenth ninety four
04/27/1992	April twenty seventh nineteen ninety two
May 27, 1995	May twenty seventh nineteen ninety five
July 4, 94	July fourth ninety four
1,994	one thousand nine hundred and ninety four
1994	nineteen ninety four

One issue that comes up with certain number formats, including dates, is range checking. A form like 13/19/94 is basically uninterpretable as a date. This kind of checking, if included in the initial pattern matching, may be slow and may increase formal requirements for power of the pattern matching system. Therefore, range checking can be done at spell-out time (see below) during normalized orthography generation, as long as a backtracking or redo option is present. If range checking is desired as part of the basic identification phase of text normalization, some regular expression matching systems allow for extensions. For example, the following pattern variable matches only numbers less than or equal to 12, the valid month specifications. It can be included as part of a larger, more complex date matching *pattern:*

```
$month = '/(0[123456789]/1[012]/'
```

14.4.2.3. Times

Times may include hours, minute, seconds, and duration specifications as shown in Table 14.7. Time formats exemplify yet another area where linguistic concerns have to intersect with architecture. If simple, flat normalized orthography is generated during a text normalization phase, a later stage may still find a form like *am* ambiguous in pronunciation. If a lattice of alternative interpretations is provided, it should be supplemented with interpretive information on the linguistic status of the alternative text analyses. Alternatively, a single best guess can be made, but even in this case, some kind of interpretive information indicating the status of the choice as, e.g., a time expression, should be provided for later stages of syntactic, semantic, and prosodic interpretation. This reiterates the importance of TTS text analysis systems to generate interpretive annotations tags for subsequent modules' use whenever possible, as discussed in Section 14.4. In some cases, unique text formatting of the choice, corresponding to the system's lexical contents, may be sufficient. That is, in some systems, generation of *A.M.*, for example, may uniquely correspond to the lexicon's entry for that portion of a time expression, which specifies the desired pronunciation and grammatical treatment.

Table 14.7 Several examples for time expressions.

11:15	eleven fifteen
8:30 pm	eight thirty pm
5:20 am	five twenty am
12:15:20	twelve hours fifteen minutes and twenty seconds
07:55:46	seven hours fifty-five minutes and forty-six seconds

14.4.2.4. Money and Currency

As illustrated in Table 14.8, money and currency processing should correctly handle at least the currency indications $, £, DM, ¥, and €, standing for dollars, British pounds, deutsche marks, Japanese yen, and euros, respectively. In general, $ and £ have to precede the numeral; DM, ¥, and € have to follow the numeral. Other currencies are often written in full words and have to follow the numeral, though abbreviations for these are sometimes found, such as *100 francs* and *20 lira*.

Table 14.8 Several money and currency expressions.

$40	forty dollars
£200	two hundred pounds
5 ¥	five yen
25 DM	twenty five deutsche marks
300 €	three hundred euros

14.4.2.5. Account Numbers

Account numbers may refer to bank accounts or social security numbers. Commercial product part numbers often have these kinds of formats as well. In some cases these cannot be readily distinguished from mathematical expressions or even phone numbers. Some examples are shown below:

```
123456-987-125456
000-1254887-87
049-85-5489
```

The other popular number format is that of credit card number, such as

```
4446-2289-2465-7065
3745-122267-22465
```

To process formats like these, it may eventually be desirable for TTS systems to provide customization capabilities analogous to the pronunciation customization features for words found in current TTS systems. Regular expression formalisms of the type exemplified above for phone number, would, if exposed to applications and developers through suitable editors, be adequate for most such needs.

14.4.2.6. Ordinal Numbers

Ordinal numbers are those referring to rank or placement in a series. Examples include:

1st, 2nd, 3rd, 4th, 10th, 11th, 12th, 20th, 100th, 1000th, etc.
1st, 2nd, 3rd, 4th, 10th, 11th, 12th, 20th, 21st, 32nd, 100th, 1000th, etc.

The system's ordinal processing may also be used to generate the denominators of fractions, except for halves, as shown in Table 14.9. Notice that the ordinal must be plural for numerators other than 1.

Table 14.9 Some examples of fractions.

1/2	one half
1/3	one third
1/4	one quarter or one fourth
1/10	one tenth
3/10	three tenths

14.4.2.7. Cardinal Numbers

Cardinal numbers are, loosely speaking, those forms used in simple counting or the statement of amounts. If a given sequence of digits fails to fit any of the more complex formats above, it may be a simple cardinal number. These may be explicitly negative or positive or assumed positive. They may include decimal or fractional specifications. They may be read in several different styles, depending on context and/or aesthetic preferences. Table 14.10 gives some examples of cardinal numbers and alternatives for normalized orthography.

The number-expansion algorithm is summarized in Algorithm 14.3. In this algorithm the text normalization module maintains an extensive pattern table. Each pattern in the table contains its associated pattern in regular expression or Perl format along with a pointer to a rule in the conversion table, which guides the expansion process.

Table 14.10 Some cardinal number types.

123	one two three	one hundred (and) twenty three
1,230	one thousand two hundred (and) thirty	
2426	two four two six	twenty four twenty six
	two thousand four hundred (and) twenty six	

A regular expression to match well-formed cardinals with commas grouping chunks of three digits of the type from 1,000,000 to 999,999,999 might appear as:

```
if ($item =~ /^([0-9]{1,3}),([0-9]{3}),([0-9]{3})/
 {      $NewFrame->{"millions"} = $1;
  $NewFrame->{"thousands"} = $2;
  $NewFrame->{"hundreds"} = $3;
   print "Grouped cardinal found: $item\n";
  return $NewFrame;   }
```

ALGORITHM 14.3: *NUMBER EXPANSION*

1. **Pattern Matching**
   If a match is found **goto 2.**
   else goto 3.
2. **Number Expansion**
   Insert SNOR and interpretive annotation tags according to the associated rule
   Advance the pointer to the right of the match pattern and **goto 1.**
3. **Finish**

14.4.3. Domain-Specific Tags

In keeping with the theme of this section—that is, the increasing importance of independently generated precise markup of text entities—we present a little-used but interesting example.

14.4.3.1. Mathematical Expressions

Mathematical expressions are regarded by some systems as the domain of special-purpose processors. It is a serious question how far to go in mathematical expression parsing, since providing some capability in this area may raise users' expectations to an unrealistic level. The World Wide Web Consortium has developed MathML (mathematical markup language) [34], which provides a standard way of describing math expressions. MathML is an XML extension for describing mathematical expression structure and content to enable mathematics to be served, received, and processed on the Web, similar to the function HTML has performed for text. As XML becomes increasingly pervasive, MathML could possibly be used to guide interpretation of mathematical expressions. For the notation $(x + 2)^2$ a possible MathML representation such as that below might serve as an initial guide for a spoken rendition.

```
<EXPR>
 <EXPR>
 x
 <PLUS/>
 2
 </EXPR>
 <POWER/>
 2
</EXPR>
```

This might be generated by an application or by a specialized preprocessor within the TTS system itself. Prosodic rules or data tables appropriate for math expressions could then be triggered.

14.4.3.2. Chemical Formulae

As XML becomes increasingly common and exploitable by TTS text normalization, other areas follow. For example, Chemical Markup Language (CML [22]) now provides a standard way to describe molecular structure or chemical formulae. CML is an example of how standard conventions for text markup are expected increasingly to replace ad hoc, TTS-internal heuristics.

In CML, the chemical formula C_2OCOH_4 would appear as:

```
<FORMULA>
  <XVAR BUILTIN="STOICH">
  C C O C O H H H H
  </XVAR>
</FORMULA>
```

It seems reasonable to expect that TTS engines of the future will be increasingly devoted to interpreting such precise conventions in high-quality speech renditions rather than endlessly replicating NL heuristics that fail as often as they succeed in guessing the identity of raw text strings.

14.4.4. Miscellaneous Formats

A random list illustrating the range of other types of phenomena for which an English-oriented TTS text analysis module must generate normalized orthography might include:

- Approximately/tilde: The symbol ~ is spoken as *approximately* before (Arabic) numeral or currency amount, otherwise it is the character named *tilde*.

- Folding of accented Roman characters to *nearest* plain version: If the TTS system has no knowledge of dealing with foreign languages, like French or German, a table of folding characters can be provided so that for a term such as *Über-mensch*, rather than spell out the word *Über*, or ignore it, the system can convert it to its *nearest* English-orthography equivalent: *Uber*. The ultimate way to process such foreign words should integrate a language identification module with a multi-lingual TTS system, so that language-specific knowledge can be utilized to produce appropriate text normalization of all text.
- Rather than simply ignore high ASCII characters in English (characters from 128 to 255), the text analysis lexicon can incorporate a table that gives *character names* to all the printable high ASCII characters. These names are either the full Unicode character names, or an abbreviated form of the Unicode names. This would allow speaking the names of characters like © (*copyright sign*), ™ (*trademark*), @ (*at*), ® (*registered mark*), and so on.
- Asterisk: in email, the symbol '*' may be used for emphasis and for setting off an item for special attention. The text analysis module can introduce a little pause to indicate possible emphasis when this situation is detected. For the example of "*Larry has *never* been here*," this may be suppressed for asterisks spanning two or more words. In some texts, a word or phrase appearing completely in UPPER CASE may also be a signal for special emphasis.
- Emoticons: There are several possible emoticons (emotion icons).

  1. :-) or :) SMILEY FACE (humor, laughter, friendliness, sarcasm)
  2. :-(or :(FROWNING FACE (sadness, anger, or disapproval)
  3. ;-) or ;) WINKING SMILEY FACE (naughty)
  4. :-D OPEN-MOUTHED SMILEY FACE (laughing out loud)

  Smileys, of which there are dozens of types, may be tacked onto word start or word end or even occur interword without spaces, as in the following examples.

  :)hi!
  Hi:)
  Hi:)Hi!

14.5. LINGUISTIC ANALYSIS

Linguistic analysis (sometimes also referred to as syntactic and semantic parsing) of natural language (NL) constitutes a major independent research field. Often commercial TTS systems incorporate some minimal parsing heuristics developed strictly for TTS. Alternatively, the TTS systems can also take advantage of independently motivated natural language proc-

essing (NLP) systems, which can produce structural and semantic information about sentences. Such linguistically analyzed documents can be used for many purposes other than TTS—information retrieval, machine translation system training, etc.

Provision of some parsing capability is useful to TTS systems in several areas. Parsers may be used in disambiguating the text normalization alternatives described above. Additionally, syntactic/semantic analysis can help to resolve grammatical features of individual words that may vary in pronunciation according to sense or abstract inflection, such as *read*. Finally, parsing can lay a foundation for derivation of a prosodic structure useful in determining segmental duration and pitch contour.

The fundamental types of information desired for TTS from a parsing analysis are summarized below:

- Word part of speech (POS) or word type, e.g., proper name or verb.
- Word sense, e.g., river *bank* vs. money *bank*.
- Phrasal cohesion of words, such as idioms, syntactic phrases, clauses, sentences.
- Modification relations among words.
- Anaphora (co-reference) and synonymy among words and phrases.
- Syntactic type identification, such as questions, quotes, commands, etc.
- Semantic focus identification (emphasis).
- Semantic type and speech act identification, such as requesting, informing, narrating, etc.
- Genre and style analysis.

Here we confine ourselves to discussion of the kind of information that a good parser could, in principle, provide to enable the TTS-specific functionality.

Linguistic analysis supports the phonetic analysis and prosodic generation phases. The modules of phonetic analysis are covered in Sections 14.6, 14.7, and 14.8. A linguistic parser can contribute in several ways to the process of generating (symbolic) phonetic forms from orthographic words found in text. One function of a parser is to provide accurate part-of-speech (POS) labels. This can aid in resolving the pronunciation of several hundred American English homographs, such as *object* and *absent*. Homographs are discussed in greater detail in Section 14.6. Parsers can also aid in identifying names and other special classes of vocabulary for which specialized pronunciation rule sets may exist [32].

Prosody generation deals mainly with assignment of segmental duration and pitch contour that have close relationship with prosodic phrasing (pause placement) and accentuation. Parsing can contribute useful information, such as the syntactic type of an utterance. (e.g., yes/no question contours typically differ from *wh*-question contours, though both are marked simply by '?' in text), as well as semantic relations of synonymy, anaphora, and focus that may affect accentuation and prosodic phrasing. Information from discourse analysis and text genre characterization may affect pitch range and voice quality settings. Further

examination of the contribution of parsing specifically to prosodic phrasing, accentuation, and other prosodic interpretation is provided in Chapter 15.

As mentioned earlier, TTS can employ either a general-purpose NL analysis engine or a pipeline of a number of very narrowly targeted, special-purpose NL modules together for the requirement of TTS linguistic analysis. Although we focus on linguistic information for supporting phonetic analysis and prosody generation here, a lot of the information and services are beneficial to document structure detection and text normalization described in previous sections.

The minimum requirement for such a linguistic analysis module is to include a lexicon of the *closed-class* function words, of which only several hundred exist in English (at most), and perhaps homographs. In addition, a minimal set of modular functions or services would include:

- *Sentence breaking*—Sentence breaking has been discussed in Section 14.3.4 above.
- *POS tagging*—POS tagging can be regarded as a two-stage process. The first is POS guessing, which is the process of determining, through a combination of a (possibly small) dictionary and some morphological heuristics or a specialized morphological parser, the POS categories that might be appropriate for a given input term in isolation. The second is POS choosing—that is, the resolution of the POS in context, via local short-window syntactic rules, perhaps combined with probabilistic distribution for the POS guesses of a given word. Sometimes the guessing and choosing functions are combined in a single statistical framework. In [6], lexical probabilities are unigram frequencies of assignments of categories to words estimated from corpora. In the original formulation of the model, the lexical probabilities [$P(c_i \mid w_i)$, where c_i is the hypothesized POS for word w_i], were estimated from the hand-tagged Brown corpus [8]. For Example, the word *see* appeared 771 times as a verb and once as an interjection. Thus the probability that *see* is a verb is estimated to be 771/772 or 0.99. Trigrams are used for contextual probability [$P(c_i \mid c_{i-1}c_{i-2}\cdots c_1) = P(c_i \mid c_{i-1}c_{i-2})$]. Lexical probabilities and trigrams over category sequences are used to score all possible assignments of categories to words for a given input word sequence. The entire set of possible assignments of categories to words in sequence is calculated, and the best-scoring sequence is used. Likewise, simple methods have been used to detect noun phrases (NPs), which can be useful in assigning pronunciation, stress, and prosody. The method described in [6] relies on a table of probabilities for inserting an NP begin bracket '[' between any two POS categories, and similarly for an NP end bracket ']'. This was also trained on the POS-labeled Brown corpus, with further augmentation for the NP labels. For example, the probability of inserting an NP begin bracket after an article was found to be

much lower than that of begin-bracket insertion between a verb and a noun, thus automatically replicating human intuition.

- *Homograph disambiguation*—Homograph disambiguation in general refers to the case of words with the same orthographic representation (written form) but having different semantic meanings and sometimes even different pronunciations. Sometimes it is also referred as sense disambiguation. Examples include "The boy used the *bat* to hit a home run" vs. "We saw a large *bat* in the zoo" (the pronunciation is the same for two *bat*) and "You *record* your voice" vs. "I'd like to buy that *record*" (the pronunciations are different for the two *record*). The linguistic analysis module should at least try to resolve the ambiguity for the case of different pronunciations because it is absolutely required for correct phonetic rendering. Typically, the ambiguity can be resolved based on POS and lexical features. Homograph disambiguation is described in detail in Section 14.6.
- *Noun phrase (NP) and clause detection*—Basic NP and clause information could be critical for a prosodic generation module to generate segmental durations. It also provides useful cues to introduce necessary pauses for intelligibility and naturalness. Phrase and clause structure are well covered in any parsing techniques.
- *Sentence type identification*—Sentence types (declarative, yes-no question, etc.) are critical for macro-level prosody for the sentence. Typical techniques for identifying sentence types have been covered in Section 14.3.4.

If a more sophisticated parser is available, a richer analysis can be derived. A so-called *shallow parse* is one that shows syntactic bracketing and phrase type, based on the POS of words contained in the phrases. A training corpus of shallow-parsed sentences has been created for the Linguistic Data Consortium [16]. The following example illustrates a shallow parse for sentence : "For six years, Marshall Hahn Jr. has made corporate acquisitions in the George Bush mode: kind and gentle."

```
For/IN[six/CD years/NNS],/,[T./NNP Marshall/NNP
Hahn/NNP Jr./NNP]has/VBZ made/VBN[corporate/JJ acquisi-
tions/NNS]in/IN[the/DT George/NNP Bush/NNP mode/NN]
:/:[kind/JJ]and/CC[gentle/JJ]./.
```

The POS labels used in this example are described in Chapter 2 (Table 2.14). A TTS system uses the POS labels in the parse to decide alternative pronunciations and to assign differing degrees of prosodic prominence. Additionally, the bracketing might assist in deciding where to place pauses for great intelligibility. A fuller parse would incorporate more higher-order structure, including sentence type identification, and more semantic analysis, including co-reference.

14.6. HOMOGRAPH DISAMBIGUATION

For written languages, *sense* ambiguities occur when words have different syntactic/semantic meanings. Those words with different senses are called *polysemous* words. For example, *bat* could mean either a kind of animal or the equipment to hit a baseball. Since the pronunciations for the two different senses of *bat* are identical, we are in general only concerned[6] about the other type of polysemous words that are homographs (spelled alike but vary in pronunciation), such as *bass* for a kind of fish (*/b ae s/*) or an instrument (*/b ey s/*).

Homograph variation can often be resolved on POS (grammatical) category. Examples include *object, minute, bow, bass, absent*, etc. Unfortunately, correct determination of POS (whether by a parsing system or statistical methods) is not always sufficient to resolve pronunciation alternatives. For example, simply knowing that the form *bow* is a noun does not allow us to distinguish the pronunciation appropriate for the instrument of archery from that for the front part of a boat. Even more subtle is the pronunciation of *read* in "If you *read* the book, he'll be angry." Without contextual clues, even human readers cannot resolve the pronunciation of *read* from the given sentence alone. Even though the past tense is more likely in some sense, deep semantic and/or discourse analysis would be required to resolve the tense ambiguity.

Several hundred English homographs extracted from the 1974 *Oxford Advanced Learners Dictionary* are listed in [10]. Here are some examples:

- Stress homographs: noun with front-stress vowel, verb with end-stress vowel
  "an *absent* boy" vs. "Do you choose to *absent* yourself?"
- Voicing: noun/verb or adjective/verb distinction made by voice final consonant
  "They will *abuse* him." vs. "They won't take *abuse*."
- –ate words: noun/adjective sense uses schwa, verb sense uses a full vowel
  "He will *graduate*." vs. "He is a *graduate*."
- Double stress: front-stressed before noun, end-stressed when final in phrase
  "an *overnight* bag" vs. "Are you staying *overnight*?"
- -ed adjectives with matching verb past tenses
  "He is a *learned* man." vs. "He *learned* to play piano."
- Ambiguous abbreviations: already described in Section 14.4.1
  in, am, SAT (Saturday vs. Standard Aptitude Test)
- Borrowed words from other languages—They could sometimes be distinguishable based on capitalization.
  "El Camino *Real* road in California" vs. "*real* world"
  "*polish* shoes" vs. "*Polish* accent"

[6] Sometimes, a polysemous word with the same pronunciation could have impact for prosodic generation because different semantic properties could have different accentuation effects. Therefore, a high-quality TTS system can definitely be benefited from word-sense disambiguation beyond homograph disambiguation.

- Miscellaneous
  "The *sewer* overflowed." vs. "a *sewer* is not a tailor."
  "He *moped* since his parents refused to buy a *moped*."
  "*Agape* is a Greek word." vs. "His mouth was *agape*."

As discussed earlier, abbreviation/acronym expansion and linguistic analysis described in Sections 14.4.1 and 14.5 are two main sources of information for TTS systems to resolve homograph ambiguities.

We close this section by introducing two special sources of pronunciation ambiguity that are not fully addressed by current TTS systems. The first one is a variation of dialects (or even personal dialect—*idiolect*). For example, some might say *tom[ey]to*, while some others might say *tom[aa]to*. Another example is that Boston natives tend to reduce the */r/* sound in sentences like "Park your car in Harvard yard." Similarly, some people use the spelling pronunciation *in-ter-es-ting* as opposed to *intristing*. Finally, speech rate and formality level can influence pronunciation. For example, the */g/* sound in *recognize* may be omitted in faster speech. It might be a sensible decision to output all possible pronunciations as a multiple pronunciation list and hope the synthesis back end picks the one with better acoustic/prosodic voice rendition. While true homographs may be resolved by linguistic and discourse analysis, achieving a consistent presentation of dialectal and stylistic variation is an even more difficult research challenge.

The other special source of ambiguity in TTS is somewhat different from what we have considered so far, but may be a concern in some markets. Most borrowed or foreign single words and place names are realized naturally with pronunciation normalized to the main presentation language. Going beyond that, language detection refers to the ability of a TTS system to recognize the intended language of a multiword stretch of text. For example, consider the fragment "Well, as for the next department head, that is simply *une chose entendue*." The French phrase "*une chose entendue*" (something clearly understood) might be realized in a proper French accent and phone pronunciation by a bilingual English/French reader. For a TTS system to mimic the best performance, the system must have:

- language identification capability
- dictionaries and rules for both languages
- voice rendition capability for both languages

14.7. MORPHOLOGICAL ANALYSIS

General issues in morphology are covered in Chapter 2. Here, we consider issues of relating a surface orthographic form to its pronunciation by analyzing its component morphemes, which are minimal, meaningful elements of words, such as prefixes, suffixes, and stem words themselves. This decomposition process is referred as *morphological analysis* [28]. When a dictionary does not list a given orthographic form explicitly, it is sometimes possible to analyze the new word in terms of shorter forms already present. These shorter forms

EX-1018
U.S. Patent 10,755,699

may combine as prefixes, one or more stems or roots, and suffixes to generate new forms. If a word can be so analyzed, the listed pronunciations of the pieces can be combined, perhaps with some adjustment (phonological rules), to yield a phonetic form for the word as a whole.

The prefixes and suffixes are generally considered bound, in the sense that they cannot stand alone but must combine with a stem. A stem, however, can stand alone. A word such as *establishment* may be decomposed into a "stem" *establish* and a suffix *-ment*. In practice, it is not always clear where this kind of analysis should stop. That is, should a system attempt to further decompose the stem *establish* into *establ* and *-ish*? These kinds of questions ultimately belong to etymology, the study of word origins, and there is no final answer. However, for practical purposes, having three classes of entries corresponding to prefixes, stems, and suffixes, where the uses of the affixes are intuitively obvious to educated native speakers, is usually sufficient. In practical language engineering, a difference that makes no difference is no difference, and unless there is a substantial gain in compression or analytical power, it is best to be conservative and list only obvious and highly productive affixes.

The English language presents numerous genuine puzzles in morphological analysis. For example, there is the issue of abstraction: is the word *geese* one morpheme, or two (base *goose* + abstract pluralizing morpheme)? For practical TTS systems, relying on large dictionaries, it is generally best to deal with concrete, observable forms where possible. In such a lexically oriented system, the word geese probably should appear in the lexicon as such, with attached grammatical features including plurality. Likewise, it is simpler to include *children* in the lexical listing rather than create a special pluralizing suffix *-ren* whose use is restricted to the single base *child*.

The morphological analyzer must attempt to cover an input word in terms of the affixes and stems listed in the morphological lexicon. The covering(s) proposed must be legal sequences of forms, so that often a word grammar is supplied to express the allowable patterns of combinations. A word grammar might, for example, restrict suffixation to the final or rightmost stem of a compound, thus allowing plurality on the final element of *businessmen* but not in the initial stem (*businessesman*). In support of the word grammar, all stems and affixes in the lexicon would be listed with morphological combinatory class specifications, usually subtyped in accordance with the base POS categories of the lexicon entries. That is, verbs would typically accept a different set of affixes than nouns or adjectives. In addition, spelling changes that sometimes accompany affixation must be recognized and undone during analysis. For example, the word *stopping* has undergone final consonant doubling as part of accommodating the suffix *ing*.

A morphological analysis system might be as simple as a set of *suffix-stripping* rules for English. If a word cannot be found in the lexicon, a suffix-stripping rule can be applied to first strip out the possible suffix, including *–s, -'s, -ing, -ed, -est, -ment*, etc. If the stripped form can be found in the lexicon, a morphological decomposition is attained. Similarly, *prefix-stripping* rules can be applied to find prefix-stem decomposition for prefixes like *in-, un-, non-, pre-, sub-, etc.*, although in general prefix stripping is less reliable.

Suffix and prefix stripping gives an analysis for many common inflected and some derived words such as *helped, cats, establishment, unsafe, predetermine, subword*, etc. It helps in saving system storage. However, it does not account for compounding, issues of legality

of sequence (word grammar), or spelling changes. It can also make mistakes (from a synchronic point of view: *basement* is not *base* + *-ment*), some of which will have consequences in TTS rendition. A more sophisticated version could be constructed by adding elements such as POS type on each suffix/prefix for a rudimentary legality check on combinations. However, a truly robust morphological capability would require more powerful formal machinery and a more thorough analysis. Therefore, adding irregular morphological formation into a system dictionary is always a desirable solution.

Finally, sometimes in commercial product names the compounding structure is signaled by word-medial case differences, e.g., AltaVista™, which can aid phonetic conversion algorithms. These can be treated as two separate words and will often sound more natural if rendered with two separate main stresses. This type of decomposition can be expanded to find compound words that are formed by two separate nouns. Standard morphological analysis algorithms employing suffix/prefix stripping and compound word decomposition are summarized in Algorithm 14.4. Note that the algorithm can be easily modified to handle words constructed by a combination of prefix, suffix, and compound.

ALGORITHM 14.4: *MORPHOLOGICAL ANALYSIS*

1. **Dictionary Lookup**
   Look up word w in lexicon
   If found
      Output attributes of the found lexical entry and exit
2. **Suffix Stripping**
   If word ends in *-s, -'s, -ing, -ed, -est, -ment,* etc.
      Strip the suffix from word w to form u
   If stripped form u found in lexicon
      Output attributes of the stem and suffix and exit
3. **Prefix Stripping**
   If word begins with *in-, un-, non-, pre-, sub-,* etc.
      Strip the prefix from word w to form u
   If stripped form u found in lexicon
      Output attributes of the prefix and stem and exit
4. **Compound word decomposition**
   If detect word-medial case differences within word w
      Break word w into a multiple words u_1, u_2, u_3, ...according to case changes
      For words u_1, u_2, u_3, **goto 1.**
   Else if word w can be decomposed into two nouns u_1, u_2 in lexicon
      Output attributes of the u_1, u_2 and exit
5. Pass word w to letter-to-sound module

14.8. LETTER-TO-SOUND CONVERSION

The best resource for generating (symbolic) phonetic forms from words is an extensive word list. The accuracy and efficiency of such a solution is limited only by the time, effort, and knowledge brought to bear on the dictionary construction process. As described in Section 14.2, a general lexicon service is a critical resource for the TTS system. Thus, the first and the most reliable way for grapheme-to-phoneme conversion is via dictionary lookup.

Where direct dictionary lookup fails, rules may be used to generate phonetic forms. Under earlier naïve assumptions about the regularity and coverage of simple descriptions of English orthography, rules have traditionally been viewed as the primary source of phonetic conversion knowledge, since no dictionary covers every input form and the TTS system must always be able to speak any word. A general *letter-to-sound* (LTS) conversion is thus required to provide phonetic pronunciation for any sequence of letters.

Inspired by the phonetic languages, letter-to-sound conversion is usually carried out by a set of rules. These rules can be thought of as dictionaries of fragments with some special conventions about lookup and context. Typically, rules for phonetic conversion have mimicked phonological rewriting in phonological theory [5], including conventions of ordering, such as *most specific first*. In phonological rules, a target is given and the rewrite is indicated, with context following. For example, a set of rules that changes orthographic *k* to a velar plosive */k/* except when the *k* is word-initial ('[') followed by *n* might appear as:

```
k -> /sil/ % [ _ n
k -> /k/
```

The rule above reads that *k* is rewritten as (phonetic) silence when in word initial position and followed by *n*, otherwise *k* is rewritten as (phonetic) */k/*. The underscore in the first line is a placeholder for the *k* itself in specifying the context. This little set properly treats *k* in *knight*, *darkness*, and *kitten*. These are formally powerful, context-sensitive rules. Generally a TTS system require hundreds or even thousands of such rules to cover words not appearing in the system dictionary or *exception list*. Typically rules are specified in terms of single-letter targets, such as the example for *k* above. However, some systems may have rules for longer fragments, such as the special vowel and consonant combinations in words like *neighbor* and *weigh*. In practice, a binary format for compression, a corresponding fragment matching capability, and a rule index must be defined for efficient system deployment.

Rules of this type are tedious to develop manually. As with any *expert system*, it is difficult to anticipate all possible relevant cases and sometimes hard to check for rule interference and redundancy. In any case, the rules must be verified over a test list of words with known transcriptions. Generally, if prediction of main stress location is not attempted, such rules might account for up to 70% of the words in a test corpus of general English. If prediction of main stress is attempted, the percentage of correct phonetic pronunciations is much lower, perhaps below 50%. The correct prediction of stress depends in part on morphology, which is not typically explicitly attempted in this type of simple rule system (though fragments corresponding to affixes are frequently used, such as *tion -> /ah ax n/*). Certainly, such rules can be made to approach dictionary accuracy, as longer and more explicit mor-

phological fragments are included. One extreme case is to create one specific rule (containing exact contexts for the whole word) for each word in the dictionary. Obviously this is not desirable, since it is equivalent to putting the word along with its phonetic pronunciation in the dictionary.

In view of how costly it is to develop LTS rules, particularly for a new language, attempts have been made recently to automate the acquisition of LTS conversion rules. These self-organizing methods believe that, given a set of words with correct phonetic transcriptions (the offline dictionary), an automated learning system could capture significant generalizations. Among them, classification and regression trees (CART) have been demonstrated to give satisfactory performances for letter-to-sound conversion. For basic and theoretic description of CART, please refer to Chapter 4.

In the system described in [14], CART methods and phoneme trigrams were used to construct an accurate conversion procedure. All of the experiments were carried on two databases. The first is the NETALK [25], which has hand-labeled alignment between letter and phoneme transcriptions. The second is the CMU dictionary, which does not have any alignment information. The NETALK database consists of 19,940 entries, of which 14,955 were randomly selected as a training set and the remaining 4951 were reserved for testing. Those 4951 words correspond to 4985 entries in the database because of multiple pronunciations. The hand-labeled alignments were used directly to train the CART for LTS conversion. The CMU dictionary has more than 100,000 words, of which the top 60,000 words were selected based on unigram frequencies trained from North American Business News. Among them, 52,415 were used for training and 9719 reserved for testing. Due to multiple pronunciations, those 9719 words have 10,520 entries in the dictionary. Due to lack of alignment information, dynamic programming was used to align each letter to the corresponding phoneme before training the LTS CART.

The basic CART component includes a set of *yes-no* questions and a procedure to select the best question at each node to grow the tree from the root. The basic *yes-no* question for LTS conversion looks like "*Is the second right letter 'p'?*" or "*Is the first left output phoneme /ay/?*" The questions for letters could be on either the left or the right side. For phones, only questions on the left side were used, for simplicity. The range of question positions must be long enough to cover the long-distance phonological variations. It was found that the 11-letter window (5 for left letter context and 5 for right letter context) and 3-phoneme window for left phoneme context are generally sufficient. A primitive set of questions would be the set of all the singleton questions about each letter or phoneme identity. When growing the tree, the question that had the best entropy reduction was chosen at each node. We observed that if we allow the node to have a complex question that is a combination of primitive questions, the depth of the tree will be greatly reduced and the performance improved. For example, the complex question "*Is the second left letter 't' and the first left letter 'i' and the first right letter 'n'?*" can capture '*o*' in common suffix "*tion*" and convert it to the right phoneme. Complex questions can also alleviate the data fragment problem caused by greedy nature of the CART algorithm. This way of finding such complex questions is similar to those used in Chapter 4. The baseline system built using the above techniques has error rates as listed in Table 14.11.

Table 14.11 LTS baseline results using CART [13].

Database	Phoneme	Word
CMU Lexicon	9.7%	35.0%
NETTALK	9.5%	42.3%

The CART LTS system [14] further improved the accuracy of the system via the following extensions and refinements:

- *Phoneme trigram rescoring*: A statistical model of phoneme co-occurrence, or phonotactics, was constructed over the training set. A phonemic trigram was generated from the training samples with back-off smoothing, and this was used to rescore the n-best list generated by LTS.
- *Multiple tree combination:* The training data was partitioned into two parts and two trees were trained. When the performance of these two trees was tested, it was found that they had a great overlap but also behaved differently, as each had a different focus region. Combining them together greatly improved the coverage. To get a better overall accuracy, the tree trained by all the samples was used together with two other trees, each trained by half of the samples. The leaf distributions of three trees were interpolated together with equal weights and then phonemic trigram was used to rescore the n-best output lists.

By incrementally experimenting with addition of these extensions and refinements, the results improved, as shown in Table 14.12.

These experiments did not include prediction of stress location. Stress prediction is a difficult problem, as we pointed out earlier. It requires information beyond the letter string. In principle, one can incorporate more lexical information, including POS and morphologic information, into the CART LTS framework, so it can be more powerful to learn the phonetic correspondence between the letter string and lexical properties.

Table 14.12 LTS using multiple trees and phonemic trigram rescoring [13].

Database	Phoneme	Word
CMU Lexicon	8.2%	26.9%
NETTALK	8.1%	34.2%

14.9. EVALUATION

Ever since the early days of TTS research [21, 31], evaluation has been considered an integral part of the development of TTS systems. End users and application developers are

mostly interested in the end-to-end evaluation of TTS systems. This *monolithic* type of whole-system evaluation is often referred to as *black-box* evaluation. On the other hand, *modular* (component) testing is more appropriate for TTS researchers when working with isolated components of the TTS system, for diagnosis or regression testing. We often refer to this type of evaluation as *glass-box* evaluation. We discuss the modular evaluations in each modular TTS chapter, while leaving the evaluation of the whole system to Chapter 16.

For text and phonetic analysis, automated, analytic, and objective evaluation is usually feasible, because the input and output of such module is relatively well defined. The evaluation focuses mainly on symbolic and linguistic level in contrast to the acoustic level, with which prosodic generation and speech synthesis modules need to deal. Such tests usually involve establishing a test corpus of correctly tagged examples of the tested materials, which can be automatically checked against the output of a text analysis module. It is not particularly productive to discuss such testing in the abstract, since the test features must closely track each system's design and implementation. Nevertheless, a few typical areas for testing can be noted. In general, tests are simultaneously testing the linguistic model and content as well as the software implementation of a system, so whenever a discrepancy arises, both possible sources of error must be considered.

For automatic detection of document structures, the evaluation typically focuses on sentence breaking and sentence type detection. Since the definitions of these two types of document structures are straightforward, a standard evaluation database can be easily established.

In the basic level, the evaluation for the text normalization component should include large regression test databases of text micro-entities: addresses, Internet and email entities, numbers in many formats (ordinal, cardinal, mathematical, phone, currency, etc.), titles, and abbreviations in a variety of contexts. These would be paired with the correct reference forms in something like the SNOR used in ASR output evaluation. In its simplest form, this would consist of a database of automatically checkable paired entries like 7% vs. *seven percent*, and $1.20 vs. *one dollar and twenty cents*. If you want to evaluate the semantic capability of text normalization, the regression database might include markups for semantic tags, so that we have 7% vs. "<number>SEVEN<ratio>PERCENT</ratio></number>", and $1.20 vs. "<money>ONE DOLLAR AND TWENTY CENTS</money>". The regression database could include domain-specific entries. This implies some dependence on the system's API—its markup capabilities or tag set. In the examples given in Table 14.13, the first one is a desirable output for domain-independent input, while the second one is suitable for normalization of the same expression in mathematical formula domain.

Table 14.13 Two examples to test domain independent/dependent text normalization.

3-4	*three to four*
	three four
<math_exp> 3-4 </math_exp>	*three minus four*

Some systems may not have a discrete level of orthographic or SNOR representation that easily lends itself to the type of evaluation described in this section. Such systems may have to evaluate their text normalization component in terms of LTS conversion.

An automated test framework for the LTS conversion analysis minimally includes a set of test words and their phonetic transcriptions for automated lookup and comparison tests. The problem is the infinite nature of language: there are always *new words* that the system does not convert correctly, and many of these will initially lack a transcription of record even to allow systematic checking. Therefore, a comprehensive test program for test of phonetic conversion accuracy needs to be paired with a data development effort. The data effort has two goals: to secure a continuous source of potential new words, such as a 24-hour newswire feed, and to maintain and construct an offline test dictionary, where reference transcriptions for new words are constantly created and maintained by human experts. This requirement illustrates the codependence of automated and manual aspects of evaluation. Different types and sources of vocabulary need to be considered separately, and they may have differing testing requirements, depending, again, on the nature of the particular system to be evaluated. For example, some systems have elaborate subsystems targeted specifically for name conversion. Such systems may depend on other kinds of preprocessing technologies, such as name identification modules, that might be tested independently.

The correct phonetic representation of a word usually depends on its sentence and even discourse contexts, as described in Section 14.6. Therefore, the adequacy of LTS conversion should not, in principle, be evaluated on the basis of isolated word pronunciations. However, a list of isolated word pronunciations is often used in LTS conversion because of its simplicity. Discourse contexts are, in general, difficult to represent unless specific applications and markup tags are available to the evaluation database. A reasonable compromise is to use a list of independent sentences with their corresponding phonetic representation for the evaluation of grapheme-to-phoneme conversion.

Error analysis should be treated as equally important as the evaluation itself. For example, if a confusability matrix shows that a given system frequently confuses central and schwa-like unstressed vowels, this may be viewed as less serious than other kinds of errors. Other subareas of LTS conversion that could be singled out for special diagnosis and testing include morphological analysis and stress placement. Of course, testing with phonemic transcriptions is the ultimate *unit test* in the sense that it contains nothing to insure that the correctly transcribed words, when spoken by the system's artificial voice and prosody, are intelligible or pleasant to hear. Phone transcription accuracy is, thus, a necessary but not a sufficient condition of quality.

14.10. Case Study: Festival

The University of Edinburgh's *Festival* [3] has been designed to take advantage of modular subcomponents for various standard functions. Festival provides a complete text and phonetic analysis with modules organized in sequence roughly equivalent to Figure 14.1. Festival outputs speech of quality comparable to many commercial synthesizers. While default

routines are provided for each stage of processing, the system is architecturally designed to accept alternative routines in modular fashion, as long as the data transfer protocols are followed. This variant of the traditional TTS architecture is particularly attractive for commercial purposes (development, maintenance, testing, scalability) as well as research. Festival can be called in various ways with a variety of switches and filters, set from a variety of sanctioned programming and scripting languages. These control options are beyond the scope of this overview.

14.10.1. Lexicon

Festival employs phonemes as the basic sounding units, which are used not only as the atoms of word transcriptions in the lexicons, but also as the organizing principle for unit selection (see Chapter 16) in the synthesizer itself. Festival can support a number of distinct phone sets and it supports mapping from one to another. A phone defined in a set can have various associated phonological features, such as vowel, high, low, etc.

The Festival lexicon, which may contain several components, provides pronunciations for words. The *addenda* is an optional list of words that are unique to a particular user, document, or application. The addenda is searched linearly. The main system lexicon is expected to be large enough to require compression and is assumed to reside on a disk or other external storage. It is accessed via binary search. The lexical entry also contains POS information, which can be modified according to the preference of the system configurer. A typical lexical entry consists of the word key, a POS tag, and phonetic pronunciation (with stress and possible syllabification indicated in parentheses):

("walkers" N (((w ao) 1) ((k er z) 0)))

If the syllables structure is not shown with parentheses, a syllabification rule component can be invoked. Separate entry lines are used for words with multiple pronunciations and/or POS, which can be resolved by later processing.

14.10.2. Text Analysis

Festival has been partially integrated with research on the use of automatic identification of document and discourse structures. The discourse tagging is done by a separate component, called SOLE [11]. The tags produced by SOLE indicate features that may have relevance for pitch contour and phrasing in later stages of synthesis (see Chapter 15). These must be recognized and partially interpreted at the text analysis phrase. The SOLE tags tell Festival when the text is comparing or contrasting two objects, when it's referring to old or new information, when it's using a parenthetical or starting a new paragraph, etc., and Festival will decide, based on this information, that it needs to pause, to emphasize or deemphasize, to modify its pitch range, etc.

Additionally, as discussed in Section 14.3, when document creators have knowledge about the structure or content of documents, they can express the knowledge through an XML-based synthesis markup language. A document to be spoken is first analyzed for all such tags, which can indicate alternative pronunciations, semantic or quasi-semantic attributes (different uses of numbers by context for example), as well as document structures, such as explicit sentence or paragraph divisions. The kinds of information potentially supplied by the SABLE tags[7] are exemplified in Figure 14.7.

```
<SABLE>
    <SPEAKER NAME="male1">
    The boy saw the girl in the park <BREAK/> with the telescope.
    The boy saw the girl <BREAK/> in the park with the telescope.

    Good morning <BREAK /> My name is Stuart, which is spelled
    <RATE SPEED="-40%">
    <SAYAS MODE="literal">stuart</SAYAS> </RATE>
    though some people pronounce it
    <PRON SUB="stoo art">stuart</PRON>. My telephone number
    is <SAYAS MODE="literal">2787</SAYAS>.

    I used to work in <PRON SUB="Buckloo">Buccleuch</PRON> Place,
    but no one can pronounce that.
    </SPEAKER>
</SABLE>
```

Figure 14.7 A document fragment augmented with SABLE tags can be processed by the Festival system [3].

For untagged input, or for input inadequately tagged for text division (<BREAK/>), sentence breaking is performed by heuristics, similar to Algorithm 14.1, which observe whitespace, punctuation, and capitalization. A linguistic unit roughly equivalent to a sentence is created by the system for the subsequent stages of processing.

Tokenization is performed by system or user-supplied routines. The basic function is to recognize potentially speakable items and to strip irrelevant whitespace or other non-speakable text features. Note that some punctuation is retained as a feature on its nearest word.

Text normalization is implemented by token-to-word rules, which return a standard orthographic form that can, in turn, be input to the phonetic analysis module. The token-to-word rules have to deal with text normalization issues similar to those presented in Section 14.4. As part of this process, token-type-specific rule sets may be applied to disambiguate tokens whose pronunciations are highly context dependent. For example, a disambiguation routine may be required to examine context for deciding whether *St.* should be realized as *Saint* or *street*. For general English-language phenomena, such as numbers and various

[7] SABLE and other TTS markup systems are discussed further in Chapter 15.

symbols, a standard token-to-word routine is provided. One interesting feature of the Festival system is a utility for helping to automatically construct decision trees to serve text normalization rules, when system integrators can gather some labeled training data.

The linguistic analysis module for the Festival system is mainly a POS analyzer. An *n*-gram based trainable POS tagger is used to predict the likelihoods of POS tags from a limited set given an input sentence. The system uses both a priori probabilities of tags given a word and *n*-grams for sequences of tags. The basic underlying technology is similar to the work in [6] and is described in Section 14.5. When lexical lookup occurs, the predicted most likely POS tag for a given word is input with the word orthography, as a compound lookup key. Thus, the POS tag acts as a secondary selection mechanism for the several hundred words whose pronunciation may differ by POS categories.

14.10.3. Phonetic Analysis

The homograph disambiguation is mainly resolved by POS tags. When lexical lookup occurs, the predicted most likely POS tag for a given word is input with the word orthography as a compound lookup key. Thus, the POS tag acts as a secondary selection mechanism for the several hundred words whose pronunciation may differ by POS categories.

If a word fails lexical lookup, LTS rules may be invoked. These rules may be created by hand, formatted as shown below:

```
( # [ c h ] C = /k/ )      // ch at word start, followed by a consonant, is /k/, e.g.,
Chris
```

Alternatively, LTS rules may be constructed by automatic statistical methods, much as described in Section 14.8 above, where CART LTS systems were introduced. Utility routines are provided to assist in using a system lexicon as a training database for CART rule construction.

In addition, Festival system employs *post-lexical rules* to handle *context coarticulation*. Context coarticulation occurs when surrounding words and sounds, as well as speech style, affect the final form of pronunciation of a particular phoneme. Examples include reduction of consonants and vowels, phrase final devoicing, and *r*-insertion. Some coarticulation rules are provided for these processes, and users may also write additional rules.

14.11. Historical Perspective and Further Reading

Text-to-speech has a long and rich history. You can hear samples and review almost a century's worth of work at the Smithsonian's Speech Synthesis History Project [19]. A good source for multilingual samples of various TTS engines is [20].

The most influential single published work on TTS has been *From Text to Speech: The MITalk System* [1]. This book describes the MITalk system, from which a large number

of research and commercial systems were derived during the 1980s, including the widely used DECTalk system [9]. The best compact overall historical survey is Klatt's *Review of Text-to-Speech Conversion for English* [15]. For deeper coverage of more recent architectures, refer to [7]. For an overview of some of the most promising current approaches and pressing issues in all areas of TTS and synthesis, see [30]. One of the biggest upcoming issues in TTS text processing is the architectural relation of specialized TTS text processing as opposed to general-purpose natural language or document structure analysis. One of the most elaborate and interesting TTS-specific architectures is the multilingual text processing engine described in [27]. This represents a commitment to providing exactly the necessary and sufficient processing that speech synthesis requires, when a general-purpose language processor is unavailable.

However, it is expected that natural language and document analysis technology will become more widespread and important for a variety of other applications. To get an idea of what capabilities the natural language analysis engines of the future may incorporate, refer to [12] or [2]. Such generalized engines would serve a variety of clients, including TTS, speech recognition, information retrieval, machine translation, and other services which may seem exotic and isolated now but will increasingly share core functionality. This convergence of NL services can be seen in a primitive form today in Japanese *input method editors* (IME), which offload many NL analysis tasks from individual applications, such as word processors and spreadsheets, and unify these functions in a single common processor [18].

For letter-to-sound rules, NETalk [25], which describes automatic learning of LTS processes via neural network, was highly influential. Now, however, most systems have converged on decision-tree systems similar to those described in [14].

REFERENCES

[1] Allen, J., M.S. Hunnicutt, and D.H. Klatt, *From Text to Speech: the MITalk System*, 1987, Cambridge, UK, University Press.

[2] Alshawi, H., *The Core Language Engine*, 1992, Cambridge, US, MIT Press.

[3] Black, A.W., P. Taylor, and R. Caley, "The Architecture of the Festival Speech Synthesis System," *3rd ESCA Workshop on Speech Synthesis*, 1998, Jenolan Caves, Australia, University of Edinburgh, pp. 147-151.

[4] Boguraev, B. and E.J. Briscoe, *Computational Lexicography for Natural Language Processing*, 1989, London, Longmans.

[5] Chomsky, N. and M. Halle, *The Sound Patterns of English*, 1968, Cambridge, MIT Press.

[6] Church, K., "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text," *Proc. of the Second Conf. on Applied Natural Language Processing*, 1988, Austin, Texas, pp. 136-143.

[7] Dutoit, T., *An Introduction to Text-to-Speech Synthesis*, 1997, Kluwer Academic Publishers.

[8] Francis, W. and H. Kucera, *Frequency Analysis of English Usage*, 1982, New York, N.Y., Houghton Mifflin.

[9] Hallahan, W.I., "DECtalk Software: Text-to-Speech Technology and Implementation," *Digit Technical Journal*, 1995, 7(4), pp. 5-19.

[10] Higgins, J., *Homographs*, 2000, http://www.stir.ac.uk/celt/staff/higdox/wordlist/homogrph.htm.

[11] Hitzeman, J., *et al.*, "On the Use of Automatically Generated Discourse-Level Information in a Concept-to-Speech Synthesis System," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 2763-2766.

[12] Jensen, K., G. Heidorn, and S. Richardson, *Natural Language Processing: the PLNLP Approach*, 1993, Boston, MASS, Kluwer Academic Publishers.

[13] Jiang, L., H.W. Hon, and X. Huang, "Improvements on a Trainable Letter-to-Sound Converter," *Proc. of Eurospeech*, 1997, Rhodes, Greece, pp. 605-608.

[14] Jiang, L., H.W. Hon, and X.D. Huang, "Improvements on a Trainable Letter-to-Sound Converter," *Eurospeech'97*, 1997, Rhodes, Greece.

[15] Klatt, D., "Review of Text-to-Speech Conversion for English," *Journal of Acoustical Society of America*, 1987, **82**, pp. 737-793.

[16] LDC, *Linguistic Data Consortium*, 2000, http://www.ldc.upenn.edu/ldc/noframe.html.

[17] Levine, J., Mason, T., Brown, D., *Lex and Yacc*, 1992, Sebastopol, CA, O'Rielly & Associates.

[18] Lunde, K., *CJKV Information Processing Chinese, Japanese, Korean & Vietnamese Computing*, 1998, O'Reilly.

[19] Maxey, H., *Smithsonian Speech Synthesis History Project*, 2000, http://www.mindspring.com/~dmaxey/ssshp/.

[20] Möhler, G., *Examples of Synthesized Speech*, 1999, http://www.ims.uni-stuttgart.de/phonetik/gregor/synthspeech/.

[21] Nye, P.W., *et al.*, "A Plan for the Field Evaluation of an Automated Reading System for the Blind," *IEEE Trans. on Audio and Electroacoustics*, 1973, **21**, pp. 265-268.

[22] OMF, *CML - Chemical Markup Language*, 1999, http://www.xml-cml.org/.

[23] Richardson, S.D., W.B. Dolan, and L. Vanderwende, "MindNet: Acquiring and Structuring Semantic Information from Text," *ACL'98: 36th Annual Meeting of the Assoc. for Computational Linguistics and 17th Int. Conf. on Computational Linguistics*, 1998, pp. 1098-1102.

[24] Sable, *The Draft Specification for Sable version 0.2*, 1998, http://www.cstr.ed.ac.uk/projects/sable_spec2.html.

[25] Sejnowski, T.J. and C.R. Rosenberg, *NETtalk: A Parallel Network that Learns to Read Aloud*, 1986, Johns Hopkins University.

[26] Sluijter, A.M.C. and J.M.B. Terken, "Beyond Sentence Prosody: Paragraph Intonation in Dutch," *Phonetica*, 1993, **50**, pp. 180-188.

[27] Sproat, R., *Multilingual Text-To-Speech Synthesis: The Bell Labs Approach*, 1998, Dordrecht, Kluwer Academic Publishers.

[28] Sproat, R. and J. Olive, "An Approach to Text-to-Speech Synthesis," in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds. 1995, Amsterdam, pp. 611-634, Elsevier Science.

[29] Turing, A.M., "Computing Machinery and Intelligence," *Mind*, 1950, **LIX**(236), pp. 433-460.

[30] van Santen, J., *et al.*, *Progress in Speech Synthesis*, 1997, New York, Springer-Verlag.

[31] van Santen, J., *et al.*, "Report on the Third ESCA TTS Workshop Evaluation Procedure," *Third ESCA Workshop on Speech Synthesis*, 1998, Sydney, Australia.

[32] Vitale, T., "An Algorithm for High Accuracy Name Pronunciation by Parametric Speech Synthesizer," *Computational Linguistics*, 1991, **17**(3), pp. 257-276.

[33] W3C, *Aural Cascading Style Sheets (ACSS)*, 1997, http://www.w3.org/TR/WD-acss-970328.

[34] W3C, *W3C's Math Home Page*, 1998, http://www.w3.org/Math/.

[35] W3C, *Extensible Markup Language (XML)*, 1999, http://www.w3.org/XML/.

[36] Wall, L., Christiansen, T., Schwartz, R., *Programming Perl*, 1996, Sebastopol, CA, O'Rielly & Associates.

CHAPTER 15

Prosody

It isn't **what** you said; it's **how** you said it! Sheridan pointed out the importance of prosody more than 200 years ago [53]:

> *Children are taught to read sentences, which they do not understand; and as it is impossible to lay the emphasis right, without perfectly comprehending the meaning of what one reads, they get a habit either of reading in a monotone, or if they attempt to distinguish one word from the rest, as the emphasis falls at random, the sense is usually perverted, or changed into nonsense.*

Prosody is a complex weave of physical, phonetic effects that is being employed to express attitude, assumptions, and attention as a parallel channel in our daily speech communication. The semantic content of a spoken or written message is referred to as its *denotation*, while the emotional and attentional effects intended by the speaker or inferred by a listener are part of the message's *connotation*. Prosody has an important supporting role in guiding a listener's recovery of the basic messages (denotation) and a starring role in signaling connotation, or the speaker's attitude toward the message, toward the listener(s),

ing connotation, or the speaker's attitude toward the message, toward the listener(s), and toward the whole communication event.

From the listener's point of view, prosody consists of systematic perception and recovery of a speaker's intentions based on:

- **Pauses**: to indicate phrases and to avoid running out of air.
- **Pitch**: rate of vocal-fold cycling (fundamental frequency or F0) as a function of time.
- **Rate/relative duration**: phoneme durations, timing, and rhythm.
- **Loudness**: relative amplitude/volume.

Pitch is the most expressive of the prosodic phenomena. As we speak, we systematically vary our fundamental frequency to express our feelings about what we are saying, or to direct the listener's attention to especially important aspects of our spoken message. If a paragraph is spoken on a constant, uniform pitch with no pauses, or with uniform pauses between words, it sounds highly unnatural.

In some languages, the pitch variation is partly constrained by lexical and syntactic conventions. For example, Japanese words usually exhibit a sharp pitch fall at a certain vowel on a consistent, word-specific basis. In Mandarin Chinese [52], word meaning depends crucially on shape and register distinctions among four highly stylized syllable pitch contour types. This is a grammatical and lexical use of pitch. However, every language, and especially English, allows some range of pitch variation that can be exploited for emotive and attentional purposes. While this chapter concentrates primarily on American English, the use of some prosodic effects to indicate emotion, mood, and attention is probably universal, even in languages that also make use of pitch for signaling word identity, such as Chinese. It is tempting to speculate that speakers of some languages use expressive and affective lexical particles and interjections to express some of the same emotive effects for which American English speakers typically rely on prosody.

We discuss pausing, pitch generation, and duration separately, because it is convenient to separate them when building systems. Bear in mind, however, that all the prosodic qualities are highly correlated in human speech production. The effect of loudness is not nearly as important in synthesizing speech as the effect of the other two factors and thus is not discussed here. In addition, for many concatenative systems this is generally embedded in the speech segment.

15.1. THE ROLE OF UNDERSTANDING

To date, most work on prosody for TTS has focused exclusively on the utterance, which is the literal content of the message. That is, a TTS system learns whatever it can from the isolated, textual representation of a single sentence or phrase to aid in prosodic generation. Typically a TTS system may rely on word identity, word part-of-speech, punctuation, length of a sentence or phrase, and other superficial characteristics. As more sophisticated NLP

capabilities are deployed for use by TTS systems, deeper properties of an utterance, including its document or discourse context, can be taken into account.

Good prosody depends on a speaker or reader's understanding of the text's or message's meaning. As noted in [64], the *golden rule* of the Roman orator Quintilian (c. A.D. 90) states [32] "*That to make a man speak well, and pronounce with a right emphasis, he ought thoroughly to understand all that he says, be fully persuaded of it, and bring himself to have, those affections which he desires to infuse in others.*" This is clearly a tall order for today's computers! How important is understanding of the text's meaning, in generation of appropriately engaging prosody? Consider a stanza from Lewis Carroll's nonsense poem "Jabberwocky" [10]:

Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.

Here, a full interpretation is not possible, owing primarily to lexical uncertainty (our ignorance of the meaning of words like *brillig*). However, you can recover a great deal of information from this passage that aids prosodic rendition. Foremost is probably the metrical structure of the poetic meter. This imposes a rhythmic constraint on prosodic phrasing (cadence, timing, and pause placement). Second, the function words such as *and*, *the*, *in*, etc. are interpretable and give rich clues about the general type and direction of action being specified. They also give us contextual hints about the part-of-speech of the neighboring nonsense words, which is a first crude step in interpreting those words' meaning. Third, punctuation is also important in this case. Using these three properties, with some analogical guesses about LTS conversions and stress locations in the nonsense words, would allow most speakers of English to render fairly pleasant and appropriate prosody for this poem.

Can a computer do the same? Where will a computer fall short of a human's performance on this task, and why? First, the carrier voice quality of a human reader is generally superior to synthesized voices. The natural human voice is more pleasant to a listener, all else being equal. As for the prosody per se, most TTS systems today use a fairly simple method to derive prosody, based on a distinction between closed-class function words, such as determiners and prepositions, which are thought to receive lesser emphasis, and open-ended sets of content words such as nouns like *wabe*, which are more likely to be accented. For this nonsense poem, that is essentially what most human readers do. Thus, if accurate LTS conversions are supplied, including main stress locations, a TTS system with a good synthetic voice and a reasonable default pitch algorithm of this type could probably render this stanza fairly well. Again, though the computer does not recognize it explicitly, the constrained rhythmic structure of the poem may be assisting.

But listeners to nonsense poems are generally not fully participating in the unconscious interpretive dialog, the attempt on the part of the listener to actively construct useful meaning from prosodic and message-content cues supplied in good faith by the speaker. Therefore, judgments of the prosodic quality of uninterpretable nonsense materials must always be suspect. In ordinary prose, the definition and recovery of *meaning* remains a slip-

pery question. Consider the passage below [56], which is not metrically structured, has few or no true nonsense words, and, yet, was deliberately constructed to be essentially meaningless.

> *In mathematical terms, Derrida's observation relates to the invariance of the Einstein field equation under nonlinear space-time diffeomorphisms (self-mappings of the space-time manifold which are infinitely differentiable but not necessarily analytic). The key point is that this invariance group 'acts transitively': this means that any space-time point, if it exists at all, can be transformed into any other. In this way the infinite-dimensional invariance group erodes the distinction between observer and observed; the pi of Euclid and the G of Newton, formerly thought to be constant and universal, are now perceived in their ineluctable historicity; and the putative observer becomes fatally decentered, disconnected from any epistemic link to a space-time point that can no longer be defined by geometry alone.*

Should the fact that, say, a professional news broadcaster with no prior knowledge of the author's intent could render this supposedly meaningless passage rather well, make us suspicious of any claims regarding the necessity of deep semantic analysis for high-quality prosody? Though perhaps meaningless when taken as a whole, once again, the educated human reader can certainly recover fragments of meaning from this text sufficient to support reasonable prosody. The morphology and syntax is all standard English, which takes us a long way. The quality of the announcer's rendition degrades somewhat under the condition the computer truly faces, which can be simulated by replacing the *content words* of a sentence above with content words randomly chosen from "Jabberwocky":

> *In brillig toves, Derrida's wabe gimbles to the bandersnatch of the Tumtum whiffling raths under frumious slithy diffeomorphisms (borogoves of the mimsy mome which are beamishly vorpal but not frabjously uffish).*

It is likely the human reader can still outperform the computer by reliance on morphological and syntactic cues, such as the parallelism determining the accent placements in the contrastive structure "*...which ARE...but NOT ...*" Nevertheless, the degree of *understanding* of a message's content that is required for convincing prosodic rendition remains a subtle question. Clearly, the more the machine or human reader knows, the better the prosodic rendition, but some of the most important knowledge is surprisingly shallow and accessible.

There is no rigorous specification or definition of meaning. The meaning of the rendition event itself is more significant than the inherent meaning of the text, if any. The meaning of the rendition event is determined primarily by the goals of the speaker and listener(s). While textual attributes such as metrical conventions, syntax, morphology, lexical semantics, topic, etc., contribute to the construction of both kinds of meaning, the meaning of the rendition event incorporates more important pragmatic and contextual elements, such as goals of the communication event, and speaker identity and attitude projection. Thus the concept-to-speech discussed in Chapter 17 has a much better chance of generating good prosody, since the content of the sentence is known by the SLU system.

15.2. PROSODY GENERATION SCHEMATIC

Figure 15.1 shows schematically the elements of prosodic generation in TTS, from pragmatic abstraction to phonetic realization. The input of the prosody module in Figure 15.1 is parsed text with a phoneme string, and the output specifies the duration of each phoneme and the pitch contour. One possible output representation of that output prosody is shown in Figure 15.2 for the sentence *The cat sat*. Up to four points per phoneme were included in this example. Often one point per phoneme is more than sufficient, except for words like *john*, where two points are needed for the phoneme */aol* to achieve a natural prosody.

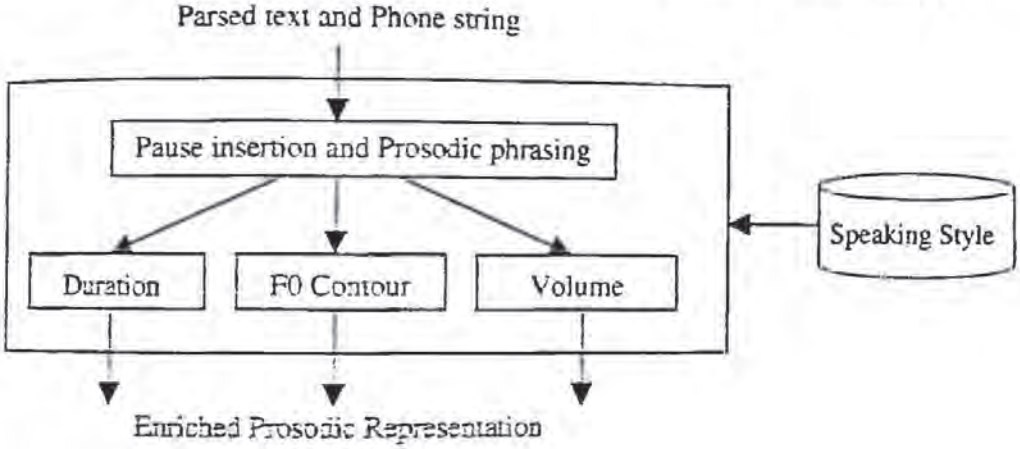


Figure 15.1 Block diagram of a prosody generation system: different prosodic representations are obtained depending on the speaking style we use.

```
DE,  24                 0,278 ;
AE0, 206
#;
I.   81                 25,278)     50,256)     (75,202);
AE.  152                1,226       25,223      (50,206)    (75,193);
T.   61                 1,178       25,178      (50,176)    (75,172);
#;
S.   206                1,278       25,272      (50,280)    (75,289);
AE.  206                1,288       25,288      (50,288)    (75,287);
T.   ...                1,220       200,220);
#;
```

Figure 15.2 Enriched prosody representation, where each line specifies the phoneme duration using the phoneme identity, the phoneme duration in milliseconds, and a number of pitch points specifying pitch and possibly volume. Each pitch point is represented by a time point, expressed as a percentage of the phoneme's duration, and the corresponding pitch value in Hz. The symbol # is a word delimiter. For example, the fourth line specifies a phoneme /t/ which lasts 85 ms and has three pitch points: the first is located at 25% of the phoneme duration, at 278 Hz, the second at 50% of the phoneme duration, at 256 Hz, and the third at the end of the phoneme, at 202 Hz. The last line specifies a silence of 200 ms, during which the pitch remains constant.

In the next sections we describe the modules of Figure 15.1: the speaking style, symbolic prosody (including pause insertion), duration assignment, and pitch generation in that order, as usually followed by most TTS systems.

15.3. SPEAKING STYLE

Prosody depends not only on the linguistic content of a sentence. Different people generate different prosody for the same sentence. Even the same person generates a different prosody depending on his or her mood. The *speaking style* of the voice in Figure 15.1 can impart an overall tone to a communication. Examples of such global settings include a low register, voice quality (falsetto, creaky, breathy, etc.), narrowed pitch range indicating boredom, depression, or controlled anger, as well as more local effects, such as notable excursion of pitch, higher or lower than surrounding syllables, for a syllable in a word chosen for special emphasis. Another example of a global effect is a very fast speaking rate that might signal excitement, while an example of a local effect would be the typical short, extreme rise in pitch on the last syllable of a *yes-no* question in American English.

15.3.1. Character

Character, as a determining element in prosody, refers primarily to long-term, stable, extralinguistic properties of a speaker, such as membership in a group and individual personality. It also includes sociosyncratic features such as a speaker's region and economic status, to the degree that these influence characteristic speech patterns. In addition, idiosyncratic features such as gender, age, speech defects, etc., affect speech, and physical status may also be a background determiner of prosodic character. Finally, character may sometimes include temporary conditions such as fatigue, inebriation, talking with mouth full, etc. Since many of these elements have implications for both the prosodic and voice quality of speech output, they can be very challenging to model jointly in a TTS system. The current state of the art is insufficient to convincingly render most combinations of the character features listed above.

15.3.2. Emotion

Temporary emotional conditions such as amusement, anger, contempt, grief, sympathy, suspicion, etc. have an effect on prosody. Just as a film director explains the emotional context of a scene to her actors to motivate their most convincing performance, so TTS systems need to provide information on the simulated speaker's state of mind. These are relatively unstable properties, somewhat independent of character as defined above. That is, one could imagine a speaker with any combination of social/dialect/gender/age characteristics being in any of a number of emotional states that have been found to have prosodic correlates, such as anger, grief, happiness, etc. Emotion in speech is actually an important area for future research. A large number of high-level factors go into determining emotional effects in speech. Among these are point of view (can the listener interpret what the speaker is really feeling or expressing?); spontaneous vs. symbolic (e.g., acted emotion vs. real feeling); cul-

ture-specific vs. universal; basic emotions and compositional emotions that combine basic feelings and effects; and strength or intensity of emotion. We can draw a few preliminary conclusions from existing research on emotion in speech [34]:

- Speakers vary in their ability to express emotive meaning vocally in controlled situations.
- Listeners vary in their ability to recognize and interpret emotions from recorded speech.
- Some emotions are more readily expressed and identified than others.
- Similar intensity of two emotions can lead to confusing one with the other.

An additional complication in expressing emotion is that the phonetic correlates appear not to be limited to the major prosodic variables (F0, duration, energy) alone. Besides these, phonetic effects in the voice such as jitter (inter-pitch-period microvariation), or the mode of excitation may be important [24]. In a formant synthesizer supported by extremely sophisticated controls [59], and with sufficient data for automatic learning, such voice effects might be simulated. In a typical time-domain synthesizer (see Chapter 16), the lower-level phonetic details are not directly accessible, and only F0, duration, and energy are available.

Some basic emotions that have been studied in speech include:

- **Anger**, though well studied in the literature, may be too broad a category for coherent analysis. One could imagine a threatening kind of anger with a tightly controlled F0, low in the range and near monotone; while a more overtly expressive type of tantrum could be correlated with a wide, raised pitch range.
- **Joy** is generally correlated with increase in pitch and pitch range, with increase in speech rate. Smiling generally raises F0 and formant frequencies and can be well identified by untrained listeners.
- **Sadness** generally has normal or lower than normal pitch realized in a narrow range, with a slow rate and tempo. It may also be characterized by slurred pronunciation and irregular rhythm.
- **Fear** is characterized by high pitch in a wide range, variable rate, precise pronunciation, and irregular voicing (perhaps due to disturbed respiratory pattern).

15.4. SYMBOLIC PROSODY

Abstract or *symbolic prosodic* structure is the link between the infinite multiplicity of pragmatic, semantic, and syntactic features of an utterance and the relatively limited F0, phone durations, energy, and voice quality. The output of the prosody module of Figure 15.2 is a

set of real values of F0 over time and real values for phoneme durations. Symbolic prosody deals with:

- Breaking the sentence into prosodic phrases, possibly separated by pauses, and
- Assigning labels, such as emphasis, to different syllables or words within each prosodic phrase.

Words are normally spoken continuously, unless there are specific linguistic reasons to signal a discontinuity. The term *juncture* refers to prosodic phrasing—that is, where do words cohere, and where do prosodic breaks (pauses and/or special pitch movements) occur. Juncture effects, expressing the degree of cohesion or discontinuity between adjacent words, are determined by physiology (running out of breath), phonetics, syntax, semantics, and pragmatics. The primary phonetic means of signaling juncture are:

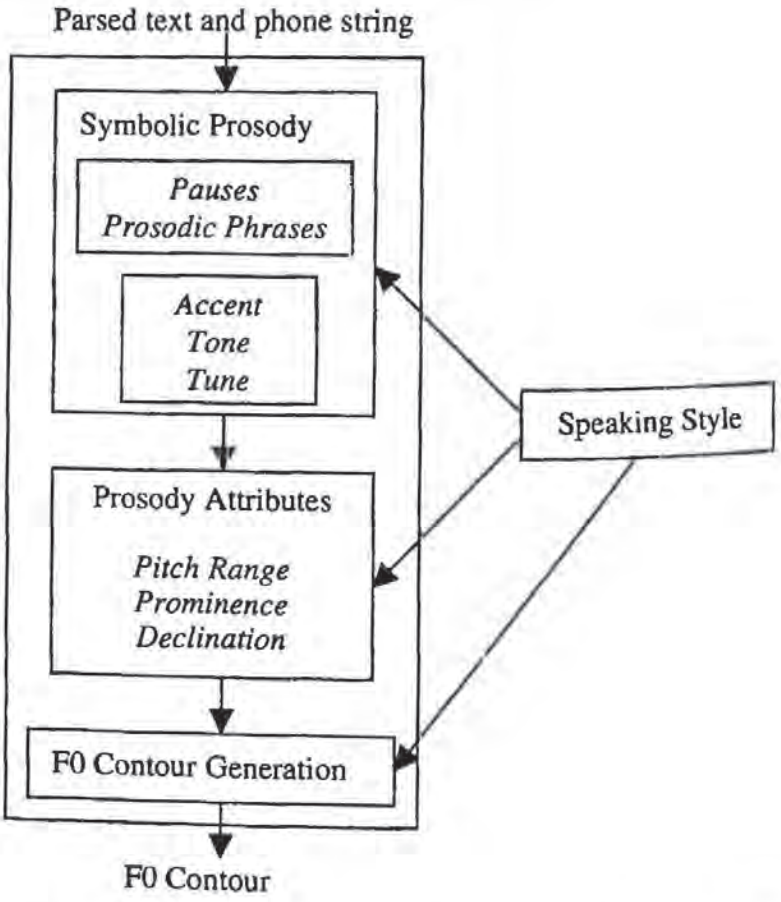


Figure 15.3 Pitch generation decomposed in symbolic and phonetic prosody.

- Silence insertion. This is discussed in Section 15.4.1.
- Characteristic pitch movements in the phrase-final syllable. This is discussed in Section 15.4.4.
- Lengthening of a few phones in the phrase-final syllable. This is discussed in Section 15.5.
- Irregular voice quality such as vocal fry. This is discussed in Chapter 16.

Abstract prosodic structure or annotation typically specifies all the elements in the top block of the pitch-generation schematic in Figure 15.3, including *accents* (corresponding conceptually to *heads* in standard syntactic structure). The accent types are selected from a small inventory of *tones* for American English (e.g., high, low, rising, late-rising, scooped). The sequence of accent and juncture tones in a given prosodic structure may cohere to yield *tune*-like effects that have some holistic semantic interpretation. While we center our description in an abstract representation called ToBI, we also describe other alternate representations in Section 15.4.6. Finally, though in principle the prosody attributes module applies to all prosody variables, it is mostly used for F0 generation in practice, and as such is discussed in Section 15.6.1.

15.4.1. Pauses

In a long sentence, speakers normally and naturally pause a number of times. These pauses have traditionally been thought to correlate with syntactic structure but might more properly be thought of as markers of information structure [58]. They may also be motivated by poorly understood stylistic idiosyncrasies of the speaker, or physical constraints. In spontaneous speech, there is also the possibility that some pauses serve no linguistic function but are merely artifacts of hesitation.

In a typical system, the most reliable indicator of pause location is punctuation. After resolution of abbreviations and special symbols relevant to text normalization (Chapter 14), the remaining punctuation can be reclassified as essentially prosodic in nature. This includes periods, commas, exclamation points, parentheses, ellipsis points, colons, dashes, etc. Each of these can be taken to correspond to a prosodic phrase boundary and can be given a special pitch movement at its end-point.

In predicting pauses, although you have to consider both their occurrence and their duration, the simple presence or absence of a silence (of greater than 30 ms) is the most significant decision, and its exact duration is secondary, based partially on the current rate setting and other extraneous factors.

There are many reasonable places to pause in a long sentence, but a few where it is critical *not* to pause. The goal of a TTS system should be to avoid placing pauses anywhere that might lead to ambiguity, misinterpretation, or complete breakdown of understanding. Fortunately, most decent writing (apart from email) incorporates punctuation according to exactly this metric: no need to punctuate after every word, just where it aids interpretation.

Therefore, by simply following punctuation in many writing styles, the TTS system will not go far wrong.

Consider the opening passage from Edgar Allan Poe's classic story *The Cask of Amontillado* (1846) arranged sentence-by-sentence:

1. *The thousand injuries of Fortunato I had borne as I best could, but when he ventured upon insult, I vowed revenge.*
2. *You, who so well know the nature of my soul, will not suppose, however, that I gave utterance to a threat.*
3. *At length I would be avenged; this was a point definitively settled—but the very definitiveness with which it was resolved precluded the idea of risk.*

If we place prosodic pauses at all and only the punctuation sites, the result is acceptable to most listeners, and no definite mistakes occur. Some stretches seem a bit too long, however. Perhaps the second part of sentence 3 could be broken up as follows:

but the very definitiveness with which it was resolved PAUSE precluded the idea of risk.

While commas are particularly useful in signaling pause breaks, as seen above, pauses may be optional following comma-delimited listed words (*berries, melons, and cheese*), though the special small pitch rise typical of a minor (nonpause) break is often present.

Cases where placing a boundary in certain locations critically affects interpretation include tag questions and verb particle constructions (where the verb must not be separated from its particle), such as:

Why did you hit Joe?
Why did you hit PAUSE Joe?

He distractedly threw out the trash.
(NOT ... threw PAUSE out ...)

He distractedly gazed PAUSE out the window.
(NOT ... out PAUSE the ...)

Supplying junctures at the optimal points sometimes requires deep semantic analysis provided by the module described in Chapter 14. The need for independent methods for pause insertion has motivated some researchers to assume that no independent source of natural language analysis is available. The CART discussed in Chapter 4 can be used for pause assignment [36]. You can use POS categories of words, punctuation, and a few structural measures, such as overall length of a phrase, and length relative to neighboring phrases to construct the classification tree. The decision-tree-based system can have correct prediction of 81% for pauses over test sentences with only 4% false prediction rates. As the algo-

rithm proceeds successively left to right through each pair of words, the following questions can be used:

- *Is this a sentence boundary (marked by punctuation)?*
- *Is the left word a content word and the right word a function word?*
- *What is the function word type of word to the right? (Certain function words are more likely to signal a break)*
- *Is either adjacent word a proper name (capitalized)?*
- *How many content words have occurred since the previous function word (If > 4 or 5 words, a break is more likely)*
- *Is there a comma at this location?*
- *What is the current location in the sentence?*
- *What is the length of current proposed major phrase?*

These questions summarize the relevant knowledge, which could be formulated in expert-system rules, and augmented by high-quality syntactic knowledge if available, or trained statistically from tagged corpora.

15.4.2. Prosodic Phrases

An end-of-sentence period may trigger an extreme lowering of pitch, a comma-terminated prosodic phrase may exhibit a small *continuation rise* at its end, signaling more to come, etc. Rules based on these kinds of simple observations are typically found in commercial TTS systems. Certain pitch-range effects over the entire clause or utterance can also be based on punctuation—for example, the range in a parenthetical restrictive clause is typically narrower than that of surrounding material, while exclamations may have a heightened range, or at least higher accent targets throughout.

Prosodic junctures that are clearly signaled by silence (and usually by characteristic pitch movement as well), also called *intonational phrases*, are required between utterances and usually at punctuation boundaries. Prosodic junctures that are not signaled by silence but rather by characteristic pitch movement only, also called *phonological phrases*, may be harder to place with certainty and to evaluate. In fast speech, the silence demarcating fruits in the sentence 'We have blueberries, raspberries, gooseberries, and blackberries.' may disappear, yet a trace of the continuation rise on each 'berries' typically remains. These locations would then still qualify as minor intonation phrases, or phonological phrases.

In analyzing spontaneous speech, the nature and extent of the signaling pitch movement may vary from speaker to speaker. A further consideration for practical TTS systems is a user's preferred rate setting: blind people who depend on TTS to access information in a computer usually prefer a fast rate, at which most sentence-internal pauses should disappear.

To discuss linguistically significant juncture types and pitch movement, it helps to have a simple standard vocabulary. ToBI (for *Tones and Break Indices*) [4, 55] is a proposed

standard for transcribing symbolic intonation of American English utterances, though it can be adapted to other languages as well. The *Tones* part of ToBI is considered in greater detail in Section 15.4.4.

The *Break Indices* part of ToBI specifies an inventory of numbers expressing the strength of a prosodic juncture. The Break Indices are marked for any utterance on their own discrete *break index tier* (or layer of information), with the BI notations aligned in time with a representation of the speech phonetics and pitch track. On the break index tier, the prosodic association of words in an utterance is shown by labeling the end of each word for the subjective strength of its association with the next word, on a scale from 0 (strongest perceived conjoining) to 4 (most disjoint), defined as follows:

- **0** for cases of clear phonetic marks of clitic[1] groups (phrases with appended reduced function words), e.g., the medial affricate in contractions of *did you* or a flap as in *got it*.
- **1** most phrase-medial word boundaries.
- **2** a strong disjuncture marked by a pause or virtual pause, but with no tonal marks, i.e., a well-formed tune continues across the juncture. OR, a disjuncture that is weaker than expected at what is tonally a clear intermediate or full intonation phrase boundary.
- **3** intermediate intonation phrase boundary, i.e., marked by a single phrase tone affecting the region from the last pitch accent to the boundary.
- **4** full intonation phrase boundary, i.e., marked by a final boundary tone after the last phrase tone.

For example, a typical fluent utterance of the following sentence: *Did you want an example?* might have a 0 between *Did* and *you*, indicating palatalization of the */d j/* sequence across the boundary between these words. Similarly, the break index value between *want* and *an* might again be 0, indicating deletion of */t/* and subsequent flapping of */n/*. The remaining break index values would probably be 1 between *you* and *want* and between *an* and *example*, indicating the presence of a mere word boundary, and 4 at the end of the utterance, indicating the end of a well-formed intonation phrase. The annotation is thus:

Did-0 you-1 want-0 an-1 example-4?

Without reference to any other knowledge, therefore, a system would place a 1 after every word, except where utterance-final punctuation motivates placement of 4. Perhaps comma boundaries would be marked by 4. Need any more be done? A BI of 0 correlates with any special phone substitution or modification rules for reduction in clitic groups that a TTS system may attempt. By marking the location of clitic (close association) phonetic reduction, such a BI can serve as a trigger for special duration rules that shorten the segments of the cliticized word. Whatever syntactic/semantic processing was done to propose the cliticization can serve to trigger assignment of 1. The 2 mark is generally more useful for

[1] Pronounced as part of another word, as in *ve* in *I've*.

analysis than for speech generation. You may observe that in the literature on intonation, a 3 break is sometimes referred to as an *intermediate phrase* break, or a *minor phrase* break, while a 4 break is sometimes called an *intonational phrase* break or a *major phrase* break.

15.4.3. Accent

We should briefly clarify use of terms such as stress and accent. *Stress* generally refers to an idealized location in an English word that is a potential site for phonetic prominence effects, such as extruded pitch and/or lengthened duration. This information comes from our standard lexicon. Thus, the second syllable of the word *em-ploy-er* is said to have the abstract property of lexical stress. In an actual utterance, if the word as a whole is sufficiently important, phonetic highlighting effects are likely to fall on the lexically *stressed* syllable:

*Acme **Industries** is the **biggest** **employer** in the **area**.*

Accent is the signaling of semantic salience by phonetic means. In American English, accent is typically realized via extruded pitch (higher or lower than the general trend) and possibly extended phone duration. Although lexical stress as noted in dictionaries is strictly an abstract property, these accent-signaling phonetic effects are usually strongest on the lexically stressed syllable of the word that is singled out for accentuation (e.g., *employer*).

In the sentence above, the word *employer* is not specially focused or contrasted, but it is an important word in the utterance, so its lexically stressed syllable typically receives a prosodic accent (via pitch/duration phonetic mechanisms), along with the other syllables in boldface. In cases of special emphasis or contrast, the lexically specified preferred location of stress in a word may be overridden in utterance accent placement:

*I didn't say employer, I said **employee**.*

It is also possible to override the primary stress of a word with the secondary stress where a neighboring word is accented. While normally we would say *Massachusetts*, we might say *Massachusetts legislature* [51].

Let's consider what might make a word accentable in context. A basic rule based on the use of POS category is to decide accentuation by *accenting all and only the content words*. Such rule is used in the baseline F0 generation system of Section 15.6.2. The content words are major open-class categories such as noun, verb, adjective, adverb, and certain strong closed-class words such as negatives and some quantifiers. Thus, the *function words*, made up of closed-class categories such as prepositions, conjunctions, etc., end up on a kind of *stop list* for accentuation, analogous to the stop lists used traditionally in simple document indexing schemes for information retrieval. This works adequately for many short, isolated sentences, such as "*The **cat** **sat** on the **mat**.*", where the words selected for accentuation appear in boldface. For more complex sentences, appearing in document or dialog context, such an algorithm will sometimes fail.

How often does the POS class-based stop-list approach fail? Let's consider a slightly more elaborate variant on the theme. A model was created using the Lancaster/IBM Spoken

English Corpus (SEC) [3]. This includes a variety of text types, including news, academic lectures, commentary, and magazine articles. Each word in the corpus has a POS tag automatically assigned by an independent process. The model predicts the probability of a word of POS having accent status. The probability is computed based on POS class of a sequence of words in the history in the similar way as n-gram models discussed in Chapter 11. This simple model performed at or above 90% correct predictions for all text types. As for stress predictions that were *incorrect*, we should note that in many cases accents are optional—it is more a game of avoiding plain wrong predictions than it is of finding optimal ones. Clearly, however, there are situations that call for greater power than a simple POS-based model can provide. Even different readings of the exact same text can result in different accents [46].

Consider a simple case where a word or its base form is repeated within a short paragraph. Such words may have the necessary POS to trigger accentuation, but, since they have already been mentioned (perhaps with varying morphological inflection), it can sound strange to highlight them again with accentuation. They are *given* or *old* information the second time around and may be deaccented. For example, the second occurrence of the noun 'switch' below is best not accented:

> *At the **corner** of the keyboard are **two switches**.*
> *The **top** switch is **user-defined**.*

To achieve this, the TTS system can keep a queue of most recently used words or normalized base forms (if morphological capability is present), and block accentuation when the next word has been used recently. The queue should be reset periodically, perhaps at paragraph boundaries [54].

Of course, the surface form of words, even if reduced to a base form or lemma by morphology, won't always capture the deeper semantic relations that govern accentuation. Consider the following fragment extracted from Roger Rosenblatt's essay:

> ***Kids** today are being neglected by the **older** generation. Adults spend **hours** every day on the **StairMaster**, trying to **become** the youth they should be attending to.*

A simple content-word-based accentuation algorithm accents the word *youth*, because it is a noun. In context, however, it is not optimal to accent *youth*, because it is co-referent with the subject of the fragment, which is *kids today*. Thus it is, by some metrics, old or *given* information, and it had better remain unaccented. The surrounding verbs *become*, *should*, and *attending* may get extra prominence. The degree to which coreference relations, from surface identity to deep anaphora, can be exploited depends on the power of the NL analysis supporting the TTS function.

Other confusions can arise in word accentuation due to English complex nominals, where lack of, or location of, an accent may be a lexical (static) rather than a syntactic or dynamic property. Consider:

> *I **invited** her to my **birthday** party, but she **said** she **can't attend** any parties until her **grades** improve.*

One possible accent structure is indicated in boldface. Here *birthday party* functions as a complex nominal, with lexical stress on *birthday*. The word *party* should not receive stress at all, nor should its later stand-alone form *parties*. Accentuation of *improve* is optional: it is a full content word, yet somehow it also feels predictable from the context, so deaccentuation is possible. Some of the complex nominals, like *birthday party*, are fully fixed and can be entered into the lexicon as such. Others form small families of binary or *n*-ary phrases, which may be detected by local syntactic and lexical analysis. Ambiguous cases such as *moving van* or *hot dog*, which could be either nominals or adjective-noun phrases, may have to be resolved by user markup or text understanding processes.

Dwight Bolinger opined that *Accent is predictable—if you're a mind reader* [6], asserting that accentuation algorithms will never achieve perfect performance, because a writer's exact intentions cannot be inferred from text alone, and understanding is needed. However, work in [20] (similar to [3] but incorporating more sophisticated mechanics for name identification and memory of recent accented items), showed that reasonably straightforward procedures, if applied separately and combined intelligently, can yield adequate results on the accentuation task. This research has also determined that improvement occurs when the system learns that not all 'closed-class' categories are equally likely to be deaccented. For example, *closed accented* items include the negative article, negative modals, negative *do*, most nominal pronouns, most nominative and all reflexive pronouns, pre- and postqualifiers (e.g., quite), prequantifiers (e.g., *all*), postdeterminers (e.g., *next*), nominal adverbials (e.g., *here*), interjections, particles, most wh-words, plus some prepositions (e.g., *despite, unlike*).

One area of current and future development is the introduction of discourse analysis to synthesis of dialog. Discourse analysis algorithms attempt to delimit the time within which a given word/concept can be considered newly introduced, given, old, or reintroduced, and combined with analysis of segments within discourse and their boundary cues (turn-taking, digressions, interruptions, summarization, etc.) can supplement algorithms for accent assignment. This kind of work improves the naturalness of computer responses in human-computer dialog, as well as the accentuation in TTS renditions of pure text, when dialog must be performed (e.g., in reading a novel out loud) [44].

As noted above, user- or application-supplied annotations, based on intimate knowledge of the purpose and content of the speech event, can greatly enhance the quality by offloading the task of automatic accent prediction. The /EMPHASIS/ tag described in Section 15.7, with several levels of strength including *reduced accent* and *no accent*, is ideally suited for this purpose.

15.4.4. Tone

Tones can be understood as labels for perceptually salient levels or movements of F0 on syllables. Pitch levels and movements on accented and phrase-boundary syllables can exhibit a bewildering diversity, based on the speaker's characteristics, the nature of the speech event, and the utterance itself, as discussed above. For modeling purposes, it is useful to

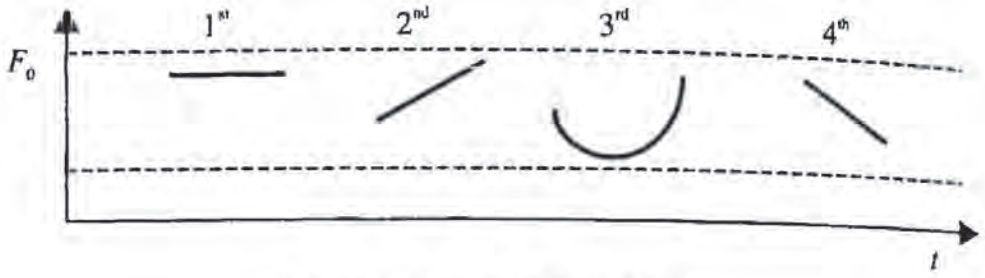


Figure 15.4 The four Chinese tones.

have an inventory of basic, abstract pitch types that could in principle serve as the base inventory for expression of linguistically significant contrasts. Chinese, a lexical tone language, is said to have an inventory of 4 lexical tones (5 if neutral tone is included), as shown in Figure 15.4. Different speakers can realize these tones differently according to their physiology, mood, utterance content, and the speech occasion. But the variance in the tones' shapes, and contrasts with one another, remain fairly predictable, within broad limits.

By analogy, linguists have proposed a relatively small set of tonal primitives for English, which can be used, in isolation or in combination, to specify the gross phonological typology of linguistically relevant contrasts found in theories of English intonational meaning [17, 28, 39]. A basic set of tonal contrasts has been codified for American English as part of the Tones and Break Indices (ToBI) system [4, 55]. These categories can be used for annotation of prosodic training data for machine learning, and also for internal modular control of F0 generation in a TTS system. The set specifies 2 abstract levels, H(igh) and L(ow), indicating a relatively higher or lower point in a speaker's range. The H/L primitive distinctions form the foundations for 2 types of entities: pitch accents, which signal prominence or culmination; and boundary tones, which signal unit completion, or delimitation. The boundary tones are further divided into phrase types and full boundary types, which would mark the ends of intonational phrases or whole utterances.

While useful as a link to syntax/semantics, the term *accent* as defined in Section 15.4.3 is a bit too abstract, even for symbolic prosody. What is required is a way of labeling linguistically significant types of pitch contrast on accented syllables. Such a system could serve as the basis for a theory of intonational meaning. The ToBI standard specifies six types of pitch accents (see Table 15.1) in American English, where the * indicates direct alignment with an accented syllable, two intermediate phrasal tones (see Table 15.2), and five boundary tones [4] (see Table 15.3).

In American English one sometimes hears a string of strictly descending pitch accent levels across a short phrase. When judiciously applied, this *downstep* effect can be pleasantly natural, as in the following sentence:

*"I saw a big-H**

*fat-!H**

pig-!H (L-L%)"*

A basic rule used in the baseline F0 generation system of Section 15.6.2 consists in having all the pitch accents realized as H*, associated with the lexically stressed syllable of accented words. In general, ToBI representations of intonation should be sparse, specifying only what is linguistically significant. So, words lacking accent should not receive ToBI pitch accent annotations, and their pitch must be derived via interpolation over neighbors, or by some other default means. Low excursions can be linguistically significant also, in the crude sense that if you dip very low in your range on a given word, it may be perceived as prominent by listeners. L*+H and L+H* are both F0 rises on the accented syllable, but in the case of L*+H, the association of the starred tone (L*) with the accented syllable may push the realization of H off to the following syllable. !H* can be used for successively lowered high accents, such as might be found on *big red car*, or *tall, dark, and handsome*. A ToBI labeled utterance is shown in Figure 15.5.

A typical boundary tone is the *final lowering*, the marked tendency for the final syllable in all kinds of noninterrogative utterances to be realized on a pitch level close to the absolute bottom of a speaker's range. The final low (L-L%) may 'pull down' the height of some few accents to its left as well [41].

Table 15.1 ToBI pitch accent tones.

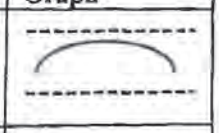
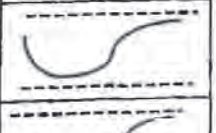
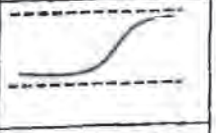
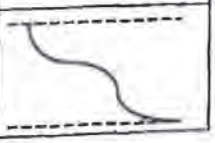
ToBI tone	Description	Graph
H*	*peak accent*—a tone target on an accented syllable which is in the upper part of the speaker's pitch range.	
L*	*low accent*—a tone target on an accented syllable which is in the lowest part of the speaker's pitch range	
L*+H	*scooped accent*—a low tone target on an accented syllable which is immediately followed by a relatively sharp rise to a peak in the upper part of the speaker's pitch range.	
L*+!H	*scooped downstep accent*—a low tone target on an accented syllable which is immediately followed by a relatively flat rise to a downstep peak.	
L+H*	*rising peak accent*—a high peak target on an accented syllable which is immediately preceded by a relatively sharp rise from a valley in the lowest part of the speaker's pitch range.	
!H*	*downstep high tone*—a clear step down onto an accented syllable from a high pitch which itself cannot be accounted for by an H phrasal tone ending the preceding phrase or by a preceding H pitch accent in the same phrase.	

Table 15.2 ToBI intermediate phrasal tones.

ToBI tone	Description
L-	Phrase accent, which occurs at an intermediate phrase boundary (level 3 and above).
H-	Phrase accent, which occurs at an intermediate phrase boundary (level 3 and above).

Ultimately, abstract linguistic categories should correlate with, or provide labels for expressing, contrasts in meaning. While the ToBI pitch accent inventory is useful for generating a variety of English-like F0 effects, the distinction between perceptual contrast, functional contrast, and semantic contrast is particularly unclear in the case of prosody [41]. For example, whether or not the L*, an alternative method of signaling accentual prominence, functions in full linguistic contrast to H* is unclear.

In addition, we have mentioned that junctures are typically marked with perceptible pitch movements that are independent of accent. The ToBI specification also allows for combinations of the H and L primitives that signal phrase, clause, and utterance boundaries. These are called phrasal tones. The ToBI specification further points out that since intonation phrases are composed of one or more intermediate phrases plus a boundary tone, full intonation phrase boundaries have two final tones.

The symbolic ToBI transcription alone is not sufficient to generate a full F0 contour. The remaining components are discussed in Section 15.6.

Table 15.3 ToBI boundary tones.

ToBI tone	Description
L-L%	For a full intonation phrase with an L phrase accent ending its final intermediate phrase and an L% boundary tone falling to a point low in the speaker's range, as in the standard 'declarative' contour of American English.
L-H%	For a full intonation phrase with an L phrase accent closing the last intermediate phrase, followed by an H boundary tone, as in 'continuation rise.'
H-H%	For an intonation phrase with a final intermediate phrase ending in an H phrase accent and a subsequent H boundary tone. as in the canonical 'yes-no question' contour. Note that the H- phrase accent causes 'upstep' on the following boundary tone, so that the H% after H- rises to a very high value.
H- L%	For an intonation phrase in which the H phrase accent of the final intermediate phrase upsteps the L% to a value in the middle of the speaker's range, producing a final level plateau.
%H	High initial boundary tones; marks a phrase that begins relatively high in the speaker's pitch range when not explained by an initial H* or preceding H%.

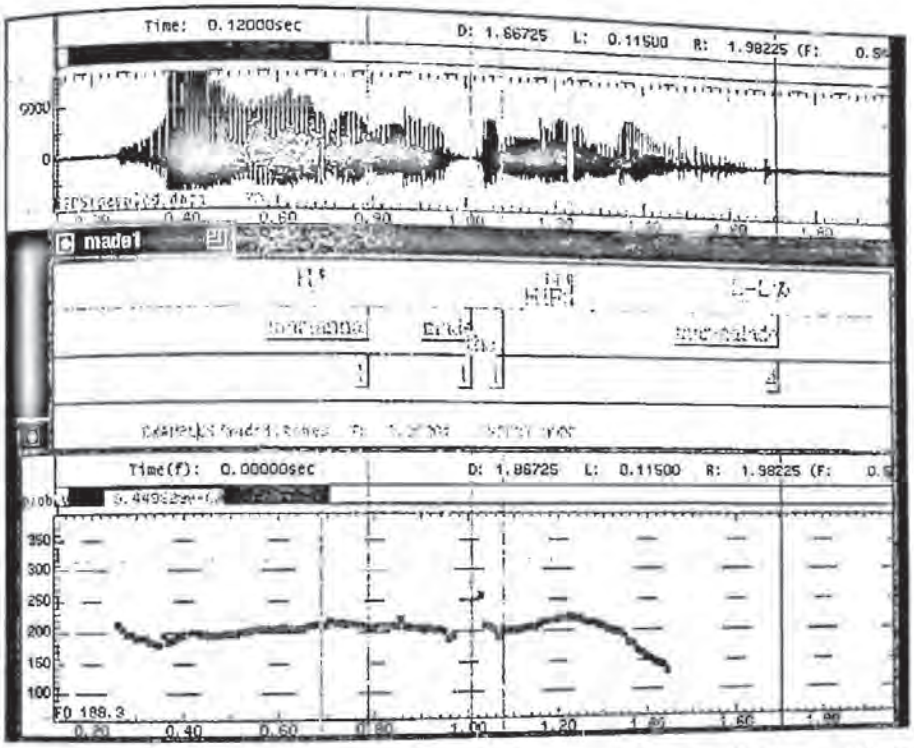


Figure 15.5 "*Marianna made the marmalade*", with an H* accent on *Marianna* and *marmalade*, and a final L-L% marking the characteristic sentence-final pitch drop. Note the use of 1 for the weak inter-word breaks, and 4 for the sentence-final break (after Beckman [4]).

15.4.5. Tune

Nyaah nuh nyaah you get
nyaah nyaah, can't me!
—Children's chant

Some pitch contours appear to be immediately recognizable and emotionally interpretable, independent of lexical content, such as the English *children's chant* above [40]. Can this idea of stylized *tunes*, perhaps decomposable into the *tones* we examined above, be applied to the intonation of ordinary speech? In fact, the ideal use of the ToBI pitch accent labels above would be as primitive elements in holistic prosodic contour descriptions, analogous to the role of phonemes in words. Ultimately, a dictionary of meaningful contours, described abstractly by ToBI tone symbols to allow for variable phonetic realization, would constitute a theory of intonational meaning for American English. Ideally, the meanings of such contours could perhaps be derived compositionally from the meanings of their constituent pitch accent and boundary tones, thus allowing us to dispense with the dictionary altogether. Contour stylization approaches describe contours holistically and index them for

application on the basis of utterance type, usually based on a naïve syntactic typology, e.g., question, declarative, etc.

The holistic representation of contours can perhaps be defended, but the categorizing of types via syntactic description (usually triggered by punctuation) is questionable. Typically, use of punctuation as a rule trigger for pitch effects is making certain hidden assumptions about the relation between punctuation and syntax, and in turn between syntax and prosody. An obvious example is question intonation. If you find a question mark at the end of a sentence, are you justified in applying a final high rise (which might be denoted as H-H% in ToBI)? First, the intonation of yes-no questions in general differs from that of *wh*-questions. *Wh*-questions usually lack the extreme final upturn of F0 heard in some yes-no questions:

i. *Are you going?*
ii. *Where are you going?*

However, there are cases where an extreme final upturn is acceptable on ii. As [8] puts it, "It has been emphasized repeatedly ... that no intonation is an infallible clue to any sentence type: any intonation that can occur with a statement, a command, or an exclamation can also occur with a question."

Admittedly, there is a rough correspondence between syntactic types and speech acts,[2] as shown in Table 15.4. Nevertheless, the correspondence between syntactic types and acts is not deterministic, and prosody in spontaneous speech is definitely mediated via speech acts (the pragmatic context and use of an utterance) rather than syntactic types. Thus, it is difficult to obtain high-quality simulation of spontaneous speech based on linguistic descriptions that do not include speech acts and pragmatics. Likewise, even simulation of prose reading without due consideration of pragmatics and speech acts, and based solely on syntactic types, is difficult because prose that is read may:

- Include acted dialog
- Have limited occurrence of most types other than declarative, lessening variety in practice
- Include long or complex sentences, blunting 'stereotypical' effects based on utterance type
- Lack text cues as to syntactic type, or analysis grammar may be incomplete

Table 15.4 Relationship between syntactic types and speech acts.

Type	Speech Act	Example
interrogative	Questioning	Is it good?
declarative	Stating	It's good.
imperative	Commanding	Be good!
exclamatory	Exclaiming	How good it is!

[2] For a more in-depth coverage of speech acts, consult Chapter 17.

Thus, description of an entire speech event, rather than inferences about text content, is again the ultimate guarantor of quality. This is why the future of automatic prosody lies with *concept-to-speech* systems (see Chapter 17) incorporating explicit pragmatic and semantic context specification to guide message rendition.

For commercial TTS systems that must infer structure from raw text, there are a few characteristic fragmentary pitch patterns that can be taken as tunes and applied to special segments of utterances. These include:

- Phone numbers—downstepping with pauses
- List intonation—downstepping with pauses (melons, pears, and eggplants)
- Tag and quotative tag intonation—low rise on tag (*Never!* he blurted. Come here, Jonathan.)

15.4.6. Prosodic Transcription Systems

ToBI, introduced above, can be used as a notation for transcription of prosodic training data and as a high-level specification for the symbolic phase of prosodic generation. Alternatives to ToBI also exist for these purposes, and some of them are amenable to automated prosody annotation of corpora. Some examples of this type of system are discussed in this section.

PROSPA was developed specially to meet the needs of discourse and conversation analysis, and it has also influenced the Prosody Group in the European ESPRIT 2589 SAM (Multilingual Speech Input/Output Assessment, Methodology and Standardization) project [50]. The system has annotations for general or global trends over long spans shown in Table 15.5, short, accent-lending pitch movements on particular vowels are transcribed in Table 15.6, and the pitch shape after the last accent in a () sequence, or *tail*, is indicated in Table 15.7.

Table 15.5 Annotations for general or global trends over long spans.

( )	extent of a sequence of cohesive accents
F	globally falling intonation
R	globally rising intonation
H	level intonation on high tone level
M	level intonation on middle tone level
L	level intonation on low tone level
H/F	falling intonation on a globally high tone level
...	sequence of weakly accented or unaccented syllables

Table 15.6 Annotations for accent-lending pitch movements on particular vowels.

+	Upward pitch movement
-	Downward pitch movement
=	level pitch accent

Table 15.7 Annotations for pitch shape after the last accent in a () sequence, or *tail*.

	falling tails
/	rising tails
-	level tails
^	combinations of tails (rising-falling here)

INTSINT is a coding system of intonation described in [22]. It provides a formal encoding of the symbolic or phonologically significant events on a pitch curve. Each such target point of the stylized curve is coded by a symbol, either as an absolute tone, scaled globally with respect to the speakers pitch range, or as a relative tone, defined locally in conjunction with the neighboring target points. Absolute tones in INTSINT are defined according to the speaker's pitch range as shown in Table 15.8. Relative tones are notated in INTSINT with respect to the height of the preceding and following target points, as shown in Table 15.9.

Table 15.8 The definition of absolute tones in INTSINT.

T	top of the speaker's pitch range
M	initial, mid value
B	bottom of the speaker's pitch range

Table 15.9 The definition of relative tones in INTSINT.

H	target higher than both immediate neighbours
L	target lower than both immediate neighbours
S	target not different from preceding target
U	target in a rising sequence
D	target in a falling sequence

In a transcription, numerical values can be retained for all F0 target points. TILT [60] is one of the most interesting models of prosodic annotation. It can represent a curve in both its qualitative (ToBI-like) and quantitative (parametrized) aspects. Generally any 'interesting' movement (potential pitch accent or boundary tone) in a syllable can be described in terms of TILT events, and this allows annotation to be done quickly by humans or machines without specific attention to linguistic/functional considerations, which are paramount for ToBI labeling. The linguistic/functional correlations of TILT events can be linked by subsequent analysis of the pragmatic, semantic, and syntactic properties of utterances.

The automatic parametrization of a pitch event on a syllable is in terms of:

- starting F0 value (Hz)
- duration
- amplitude of rise (A_{rise}, in Hz)
- amplitude of fall (A_{fall}, in Hz)
- starting point, time aligned with the signal and with the vowel onset

The tone shape, mathematically represented by its *tilt*, is a value computed directly from the F0 curve by the following formula:

$$tilt = \frac{|A_{rise}| - |A_{fall}|}{|A_{rise}| + |A_{fall}|} \tag{15.1}$$

A likely syllable for tilt analysis in the contour can be automatically detected based on high energy and relatively extreme F0 values or movements. Human annotators can select syllables for attention and label their qualities according to Table 15.10.

Table 15.10 Label scheme for syllables.

sil	Silence
c	Connection
a	Major pitch accent
fb	Falling boundary
rb	Rising boundary
afb	Accent+falling boundary
arb	Accent+rising boundary
m	Minor accent
mfb	Minor accent+falling boundary
mrb	Minor accent+rising boundary
l	Level accent
lrb	Level accent+rising boundary
lfb	Level accent+falling boundary

15.5. DURATION ASSIGNMENT

Pitch and duration are not entirely independent, and many of the higher-order semantic factors that determine pitch contours may also influence durational effects. The relation between duration and pitch events is a complex and subtle area, in which only initial

exploration has been done [63]. Nonetheless, most systems often treat duration and pitch independently because of practical considerations [61].

Numerous factors, including semantics and pragmatic conditions, might ultimately influence phoneme durations. Some factors that are typically neglected include:

- The issue of speech rate relative to speaker intent, mood, and emotion.
- The use of duration and rhythm to possibly signal document structure above the level of phrase or sentence (e.g., paragraph).
- The lack of a consistent and coherent practical definition of the phone such that boundaries can be clearly located for measurement.

15.5.1. Rule-Based Methods

Klatt [1] identified a number of first-order perceptually significant effects that have largely been verified by subsequent research. These effects are summarized in Table 15.11.

Table 15.11. Perceptually significant effects for duration. After Klatt [1].

Lengthening of the final vowel and following consonants in prepausal syllables.
Shortening of all syllabic segments[3] in nonprepausal position.
Shortening of syllabic segments if not in a word final syllable.
Consonants in non-word-initial position are shortened.
Unstressed and secondary stressed phones are shortened.
Emphasized vowels are lengthened.
Vowels may be shortened or lengthened according to phonetic features of their context.
Consonants may be shortened in clusters.

The rule-based duration-modeling mechanism involves table lookup of minimum and inherent durations for every phone type. The minimum duration is rate dependent, so all phones could be globally scaled in their minimum durations for faster or slower rates. The inherent duration is the raw material for the rules above: it may be stretched or contracted by a prespecified percentage attached to each rule type above applied in sequence, then it is finally added back onto the minimum duration to yield a millisecond time for a given phone. The duration of a phone is expressed as

$$d = d_{min} + r(\bar{d} - d_{min}) \tag{15.2}$$

[3] Syllabic segments include vowels and syllabic consonants.

where d_{min} is the minimum duration of the phoneme, $\bar{d}$ is the average duration of the phoneme, and the correction r is given by

$$r = \prod_{i=1}^{N} r_i \tag{15.3}$$

for the case of N rules being applied where each rule has a correction r_i. At the very end, a rule may apply that lengthens vowels when they are preceded by voiceless plosives (/p/, /t/, /k/). This is also the basis for the additive-multiplicative duration model [49] that has been widely used in the field.

15.5.2. CART-Based Durations

A number of generic machine-learning methods have been applied to the duration assignment problem, including CART and linear regression [43, 62]. The voice datasets generally rely on less than the full set of possible joint duration predictors implied in the rule list of Table 15.11. It has been shown that a model restricted to the following features and contexts can compare favorably, in listeners' perceptions, with durations from natural speech [43]:

- Phone identity
- Primary lexical stress (binary feature)
- Left phone context (1 phone)
- Right phone context (1 phone)

In addition, a single rule of vowel and post-vocalic consonant lengthening (rule 1 in Table 15.11) is applied in prepausal syllables. The restriction of phone context to immediate left and right neighbors results in a triphone duration model, congruent with the voice triphone model underlying the basic synthesis in the system [23]. In perceptual testing this simple triphone duration model yielded judgments nearly identical to those elicited by utterances with phone durations from natural speech [43]. From this result, you may conjecture that even the simplified list of *first-order* factors above may be excessive, and that only the handful of factors implicit in the triphones themselves, supplemented by a single-phrase final-syllable coda lengthening rule, is required. This would simplify data collection and analysis for system construction.

15.6. PITCH GENERATION

We now describe the issues involved in generating synthetic pitch contours. Pitch, or F0, is probably the most characteristic of all the prosody dimensions. As discussed in Section 15.8, the quality of a prosody module is dominated by the quality of its pitch-generation component.

Since generating pitch contours is an incredibly complicated problem, pitch generation is often divided into two levels, with the first level computing the so-called symbolic prosody described in Section 15.4 and the second level generating pitch contours from this symbolic prosody. This division is somewhat arbitrary since, as we shall see below, a number of important prosodic phenomena do not fall cleanly on one side or the other but seem to involve aspects of both. Often it is useful to add several other attributes of the pitch contour prior to its generation, which are discussed in Section 15.6.1.

15.6.1. Attributes of Pitch Contours

A pitch contour is characterized not only by its symbolic prosody but also by several other attributes such as pitch range, gradient prominence, declination, and microprosody. Some of these attributes often cross into the realm of symbolic prosody. These attributes are also known in the field as phonetic prosody (termed as an analogy to phonology and phonemics).

15.6.1.1. Pitch Range

Pitch range refers to the high and low limits within which all the accent and boundary tones must be realized: a floor and ceiling, so to speak, which are typically specified in Hz. This may be considered in terms of stable, speaker-specific limits as well as in terms of an utterance or passage. For a TTS system, each voice typically has a characteristic pitch range representing some average of the pitch extremes over test utterances. This speaker-specific range can be set as an initial default for the voice or character. These limits may be changed by an application.

Another sense of pitch range is the actual exploitation of zones within the *hard* limits at any point in time for linguistic purposes, having to do with expression of the content or feeling of the message. Pitch-range variation that is correlated with emotion or other aspects of the speech event is sometimes called *paralinguistic*. This linguistic and paralinguistic use of pitch range includes aspects of both symbolic and phonetic prosody. Since it is quantitative, it certainly is a phonetic property of an utterance's F0 contour. Furthermore, it seems that most linguistic contrasts involving pitch accents, boundary tones, etc. can be realized in any pitch range. These settings can be estimated from natural speech (for research purposes) by calculating F0 mean and variance over an utterance or set of utterances, or by simply adopting the minimum and maximum measurements (perhaps the 5th and 95th percentile to minimize the effect of pitch tracker errors).

But, although pitch range is a phonetic property, it can be systematically manipulated to express states of mind and feeling in ways that other strictly phonetic properties, such as characteristic formant values, rarely are. Pitch range interacts with all the prosodic attributes you have examined above, and certain pitch-range settings may be characteristic of particular styles or utterance events. For example, it is noted [8] that: "*we cannot speak of an into-*

nation of exclamation ... Exclamation draws impartially upon the full repertory of up-down patterns. What characterizes the class is not shape but range: exclamations reach for the extreme—usually higher but sometimes lower." In this sense, then, pitch range cannot be considered an arbitrary or physiological attribute—it is directly manipulated for communicative purposes.

In prosodic research, distinguishing emotive and iconic use of pitch (analogous to gesture) from strictly linguistic (logical, syntactic, and semantic expression, with arbitrary relation between signifier and signified) prosodic phenomena has been difficult. Pitch-range variation seems to straddle emotional, linguistic, and phonetic expression.

A linguistic pitch range may be narrowed or widened, and the zone of current pitch variation may be placed anywhere within a speaker's wider, physically determined range. So, for example, a male speaker might adopt a falsetto speaking style for some purpose, with his pitch range actually narrowed, but with all pitch variation realized in a high portion of his overall range, close to his physical limits.

Pitch range is a gradient property, without categorical bounds. It seems to trade off with other model components: accent, relative prominence, downstep, and declination. For example, if our model of prosody incorporates, say, an accent-strength component, but if we also recognize that pitch range can be manipulated for linguistic purposes, we may have difficulty determining, in analysis, whether a given accent is at partial strength in a wide range or at full strength in a reset, narrower range. This analytic uncertainty may be reflected in the quality of models based on the analysis.

A practical TTS system has to stay within, and make some attempt to maximize the exploitation of, the current system default or user-specified range. Thus, for general TTS purposes, the simplest approach is to use about 90% of the user-set or system default range for general prose reading, most of the time, and use the reserved 10% in special situations, such as the paragraph initial *resets*, exclamations, and emphasized words and phrases.

15.6.1.2. Gradient Prominence

Gradient prominence refers to the relative strength of a given accent position with respect to its neighbors and the current pitch-range setting. The simplest approach, where every accented syllable is realized as a H(igh) tone, at uniform strength, within an invariant range, can sound unnatural. At first glance, the prominence property of accents might appear to be a phonetic detail, in that it is quantitative, and certainly any single symbolic tonal transcription can be realized in a wide variety of relative per-accent prominence settings. However, the relative height of accents can fundamentally alter the information content of a spoken message by determining focus, contrast, and emphasis. You would hope that such linguistic content would be determined by the presence and absence, or perhaps the types (H, L, etc.), of the symbolic accents themselves. But an accented syllable at a low prominence might be perceived as unaccented in some contexts, and there is no guaranteed minimum degree of prominence for accent perception. Furthermore, as noted above, the realization of prominence of an accent is context-sensitive, depending on the current pitch-range setting.

The key knowledge deficit here is a theory of the interpretation of prominence that would allow designers to make sensible decisions. It appears that relative prominence is related to the information status of accent-bearing words and is in that sense linguistic, yet there is no theory of prominence categories that would license any abstraction. For the present, many commercial TTS systems adopt a pseudorandom pattern of alternating stronger/weaker prominence, simply to avoid monotony. If a word is tagged for emphasis, or if its information status can otherwise be inferred, its prominence can be heightened within the local range.

In the absence of information on the relative semantic salience of accented words in the utterance, successive prominence levels are varied in some simple alternating pattern, to avoid monotony. Rather than limiting the system to a single peak F0 value per accented syllable, several points could be specified, which, when connected by interpolation and smoothing, could give varied effects within the syllable, such as rising, falling, and scooped accents.

15.6.1.3. Declination

Related to both pitch range and gradient prominence is the long-term downward trend of accent heights across a typical reading-style, semantically neutral, declarative sentence. This is called *declination*. Although this tendency, if overdone, can simply give the effect of a bored or uncomprehending reader, it is a favorite prosodic effect for TTS systems, because it is simple to implement and licenses some pitch change across a single sentence. If a system uses a 'top line' as a reference for calculating the height of every accent, the slope of that top line can simply be declined across the utterance. Otherwise, each accent's prominence can be realized as a certain percentage of the height of the preceding one. Declination can be reset at utterance boundaries, or within an utterance at the boundaries of certain linguistic structures, such as the beginning of quoted speech. Intrasentence phrase and clause types that typically narrow the pitch range, such as parentheticals and certain relative clauses, can be modeled by suspending the declination, or adopting a new declination line for the temporary narrowed range, then resuming the suspended longer-term trend as the utterance progresses. Needless to say, declination is not a prominent feature of spontaneous speech and in any case should not be overdone.

The minor effect of declination should not be confused with the tendency in all kinds of nonquestioning utterances to end with a very low pitch, close to the bottom of the speaker's range. In prosodic research this is called *final lowering* and is well-attested as a phenomenon that is independent of declination [29]. The ToBI notation used to specify final lowering is the complex boundary tone L-L%. In Figure 15.6 we show the declination line together with the other two *downers of intonation*: downstep and final lowering described in Section 15.4.4.

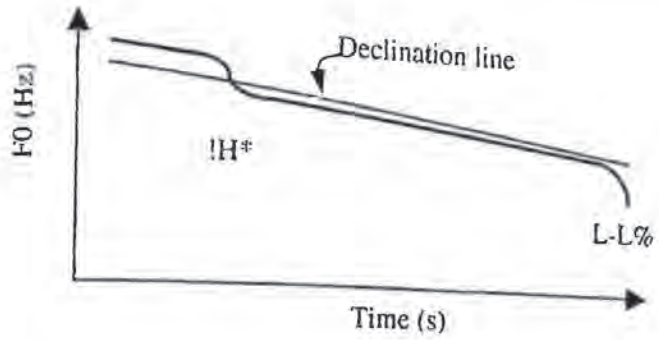


Figure 15.6 The three downers of intonation: the declination line, a downstep (!H*), and the final lowering (L-L%).

15.6.1.4. Phonetic F0—Microprosody

Microprosody refers to those aspects of the pitch contour that are unambiguously phonetic and that often involve some interaction with the speech carrier phones. These may be regarded as *second-order* effects, in the sense that rendering them well cannot compensate for incorrect accentuation or other mistakes at the symbolic level. Conversely, making no attempt to model these but putting a great deal of care into the semantic basis for determining accentuation, contrast, focus, emphasis, phrasing, etc. can result in a system of reasonable quality. Nevertheless, all else being equal, it is advisable to make some attempt to capture the local phonetic properties of natural pitch contours.

If the strength of accents is controlled semantically, by having equal degrees of focus on words of differing phonetic makeup, it has been observed that high vowels described in Chapter 2 carrying H* accents are uniformly higher in the phonetic pitch range than low vowels with the same kinds of accent. The distinction between high and low vowels correlates with the position of the tongue in articulation (high or low in the mouth). The highest English vowels by this metric are */iy/* (as in *bee*) and */uw/* (as in *too*), while the lowest vowel is */aa/* as in *father*. The predictability of F0 under these conditions may relate to the degree of tension placed on the laryngeal mechanisms by the raised tongue position in the high vowels as opposed to the low. In any case, this effect, while probably perceptually important in natural speech, is challenging for a synthesizer. The reason relates again to the issue of gradient prominence, discussed above. Apart from experimental prompts in the lab, there is currently no principled way to assign prominence for accent height realization based on utterance content in general TTS. It may therefore be difficult for a listener to correctly factor pitch accent height that is due to correctly (or incorrectly) assigned gradient prominence from height variation related to the lower-level phonetic effects of vowel height.

Another phonetic effect is the level F0 in the early portion of a vowel that follows a voiced obstruent such as /b/, contrasted with the typical fall in F0 following a voiceless obstruent such as /p/. This phonetic conditioning effect, of both preceding and following consonants, can be observed most clearly when identical underlying accent types are assigned to the carrier vowel, and may persist as long as 50 ms or more into the vowel. The exact contribution of the pre-vocalic consonant, the post-vocalic consonant, and the underlying accent type are difficult to untangle, though [54] is a good survey of all research in this area and adds new experimental results. For commercial synthesizers, this is definitely a second-order effect and is probably more important for rule-based formant synthesizers (see Chapter 16), which need to use every possible cue to enforce distinctions among consonants in phoneme perception, than for strictly intonational synthesis. However, in order to achieve completely natural prosody in the future, this area will have to be addressed.

Last, and perhaps least, *jitter* is a variation of individual cycle lengths in pitch-period measurement, and *shimmer* is variation in energy values of the individual cycles. These are distinct concepts, though somewhat correlated. Obviously, this is an influence of glottal pulse shape and strength on the quality of vowels. Speech with jitter and shimmer over 15% sounds pathological, but complete regularity in the glottal pulse may sound unnatural. For a deeper understanding of how these could be controlled, see Chapter 16.

15.6.2. Baseline F0 Contour Generation

We now examine a simple system that generates F0 contours. Although each stage of an F0 contour algorithm ideally requires a complete natural language and semantic analysis system, in practice a number of rules are often used. The system described here illustrates most of the important features common to the pitch-generation systems of commercial synthesizers.

First, let's consider a natural speech sample and describe what initial information is needed to characterize it, and how an artificial pitch contour can be synthesized based on the input analysis. The chosen sample is the utterance *"Don't hit it to Joey!"*, an exclamation, from the ToBI Labeling Guidelines sample utterance set [4]. The natural waveform, aligned pitch contour, and abstract ToBI labels are shown in Figure 15.7. This utterance is about 1.63 seconds and it has three major ToBI pitch events:

H*	high pitch accent on *Don't*
L*+!H	low pitch accent with following downstepped high on *Joey*
L-L%	low utterance-final boundary tone at the very end of utterance

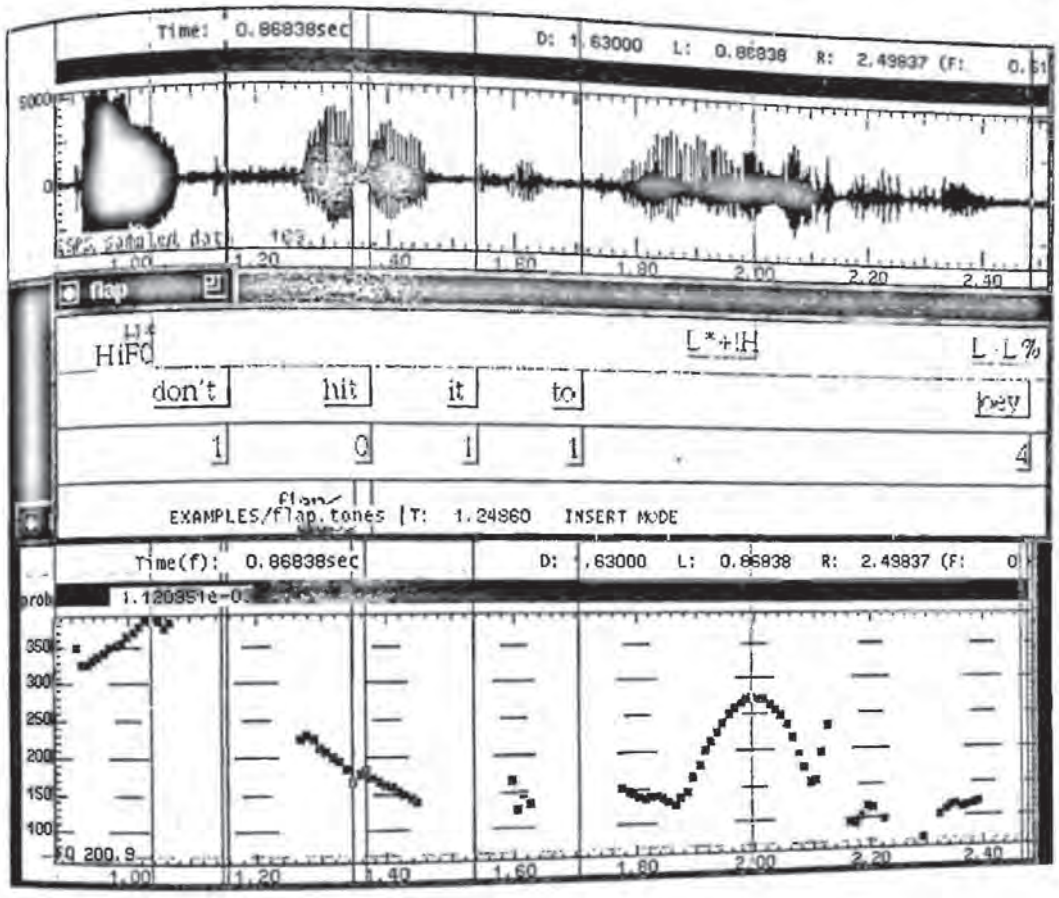


Figure 15.7 Time waveform, segmentation, ToBI marks, and pitch contour for the utterance *"Don't hit it to Joey!"* spoken by a female speaker (after Beckman [4]).

The input to the F0 contour generator includes:

- Word segmentation.
- Phone labels within words.
- Durations for phones, in milliseconds.
- Utterance type and/or punctuation information.
- Relative salience of words as determined by grammatical/semantic analysis.
- Current pitch-range settings for voice.

15.6.2.1. Accent Determination

Although accent determination ideally requires a complete natural language and semantic analysis system (see Section 15.4.3), in practice a number of rules are often used. The first rule is: *Content word categories of noun, verb, adjective, and adverb are to be accented, while the function word categories (everything else, such as pronoun, preposition, conjunction, etc.) are to be left unaccented.* Rules can be used to tune this by specifying which POS is accented or not and in which context.

If we apply that simple metric to the natural sample of Figure 15.7, we see that it does not account for the accentuation of 'hit', which, as a verb, should have been accented. In a real system perhaps we would have accented it, and this might have resulted in the typical overaccented quality of synthetic prosody. For this sample discussion, let's adopt a simplified version of a rule found in some commercial synthesizers: *Monosyllabic common verbs are left unaccented.*

What about *"Don't"*? A simplistic view would state that the POS-based policy has done the right thing, after all *"Don't"* can be regarded as a verbal form. However, usually *do* is considered an auxiliary verb and is not accented. For now we adopt another rule that says: *In a negative imperative exclamation, determined by presence of a second-person negative auxiliary form and a terminal exclamation point, the negative term gets accented.* The adoption of these corollaries to the simple POS-based accentuation rule accounts for our accent placement in the present example, but of course it sometimes fails, as does any rigid policy. So our utterance would now appear (with words selected for accent in upper case) as *"DON'T hit it to JOEY!"*

15.6.2.2. Tone Determination

In the limit, tone determination (see Section 15.4.4) also requires a complete natural language and semantic analysis system, but in practice a number of rules are often used. Generally, in working systems, H* is used for all pitch accent tones, and this is actually very realistic, as H* is the most frequent tone in natural speech.

Sometimes complex tones of the type L*+!H are thrown in for a kind of pseudovariety in TTS. In our sample natural utterance this is the tone that is used, so here we assume that this is the accent type assigned.

We also need to mark punctuation-adjacent and utterance-final phonemes as rise, continue, or fall boundaries. In this case we mark it as L-L%.

15.6.2.3. Pitch Range

To determine the pitch range, we are going to make use of three lines as a frame within which all pitches are calculated. The top line and bottom line would presumably be derived from the current or default pitch-range settings as controlled by an application or user. Here

we set them in accord with the limits of our natural sample. Note that while, for this example, the pitch contour is generated within an actual pitch range, it could also be done within an abstract range of, say, 1–100, which the voice-generation module could map to the current actual setting. So we set the top line at $T = 375$ Hz and the base line at $B = 100$ Hz.

It is more advantageous to work in a logarithmic scale, because it is more easily ported from males to females, and because this better represents human prosody. There are 24 semitones in an octave; thus a semitone corresponds to a ratio of $a = 2^{1/24}$. The pitch range can be expressed in semitones as

$$n = 24 \log_2 (T/B) = 80 \log_{10} (T/B) \tag{15.4}$$

so that we can express frequencies in semitones as

$$f_0 = 80 \log_{10} F_0 \tag{15.5}$$

and its inverse

$$F_0 = 10^{f_0/80} \tag{15.6}$$

Using Eq. (15.5), the top line is $t = 206$ and the base line is $b = 160$. The reference line is a kind of midline for the range, used in the accent scaling calculations, and is set halfway between the bottom and top lines, i.e., $r = 183$, and using Eq. (15.6), $R = 194$ Hz.

15.6.2.4. Prominence Determination

The relative prominence of the words (see Section 15.6.1.2) allows the pitch module to scale the pitch within any given pitch range. Here we assume (arbitrarily) that $N = 5$ degrees of abstract relative prominence are sufficient. This means that, e.g., an H* pitch accent with prominence 5 will be at or near the very top of the current pitch-range setting, while an L* with the same prominence will be at or near the very bottom of the range. Smaller prominence numbers indicate less salience, placing their pitch events closer to the middle of the range.

Converting the abstract tone types plus prominence into pitch numbers is more art than science (but see Section 15.6.4 for a discussion of data-based methods for this process). Here we assume a simple linear relationship between the tone's type and relative prominence:

$$f_0[i] = r + (t - r) * p[i]/N \tag{15.7}$$

In the limit, prominence determination also requires a complete natural language and semantic analysis system, but in practice a number of heuristics are often used. One such heuristic is: *In a negative imperative exclamation, the negative term gets the most emphasis,* leading to a relative prominence assignment of 5 on 'don't.' Using Eqs. (15.7) and (15.6), the anchor equals the top range of 375 Hz.

Then, since the L*+!H involves a *downstepped* term, it must by definition be lower than the preceding H* accent, so we arbitrarily assign it a relative prominence of '2'. The L*+!H is more complex, requiring calculation and placement of two separate anchor points. For simplicity we are using a single prominence value for complex tones like L*+!H, but we could also use a value-per-tone approach, at the cost of greater analytical complexity. Using Eq. (15.7), it corresponds to 192 semitones, and with Eq. (15.6), the value of !H is 251 Hz. For L*, we use a prominence of –2 (we use negative values for L tones), which, using Eq. (15.7), results in an anchor of 174, or alternatively 149 Hz.

The L-L% tone is a boundary tone, so it always goes on the final frame of a syllable-final (in this case, utterance-final) phone. The L-L% in most ToBI systems is not treated as a two-part complex tone but rather as a *super low* L% boundary, falling at the very bottom of the speaker's pitch range, i.e., prominence of 5, for a few frames. Thus the F0 value of these anchor point is 100 Hz.

We also need to set anchors for the initial point. The initial anchor is usually set at some arbitrary but high place within the speaker's range (perhaps a rule looking at u tterance type can be used). A prominence of 4 can be used, yielding a value of 329 Hz.

Finally we need to determine where to place the anchors within the accented syllable. Often they are placed in the middle of the vowel. All the anchor points are shown in Figure 15.8.

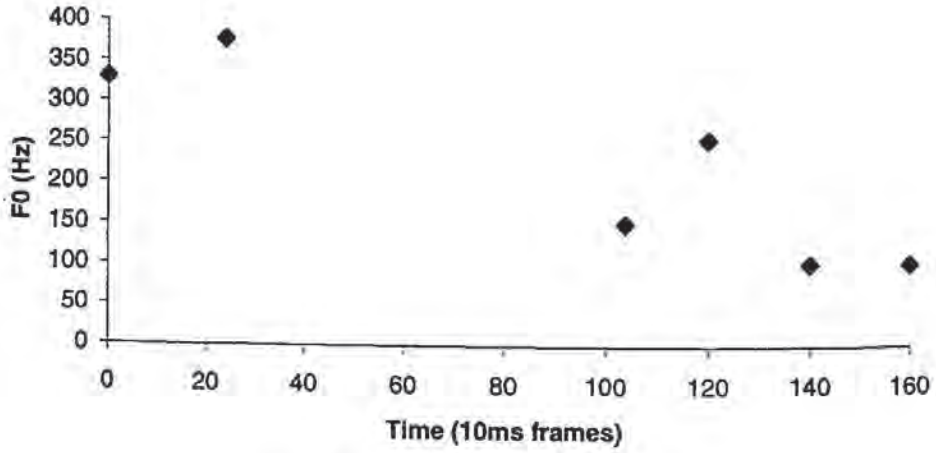


Figure 15.8 Anchor points of the F0 contour.

15.6.2.5. F0 Contour Interpolation

To obtain the full F0 contour we need some kind of interpolation. One way is to interpolate linearly and follow with a multipoint moving-average window over the resulting (angular) contour to smooth it out. Another possibility is a higher-order interpolation polynomial. In this case a cubic interpolation routine is called, which has the advantage of retaining the exact anchor points in a smoothed final contour (as opposed to moving average, which smears the anchor points). In general the choice of interpolation algorithm makes little per-

ceptual difference, as long as no sharp 'corners' remain in the contour. In Figure 15.9 the contour was interpolated fully, without regard to voicing properties of underlying phones. In the graph, the sections corresponding to unvoiced phones have been replaced with zero, for ease of comparison to the sample in Figure 15.7. The interpolation can be done in the linear frequency, as in Figure 15.9, or in the log-frequency.

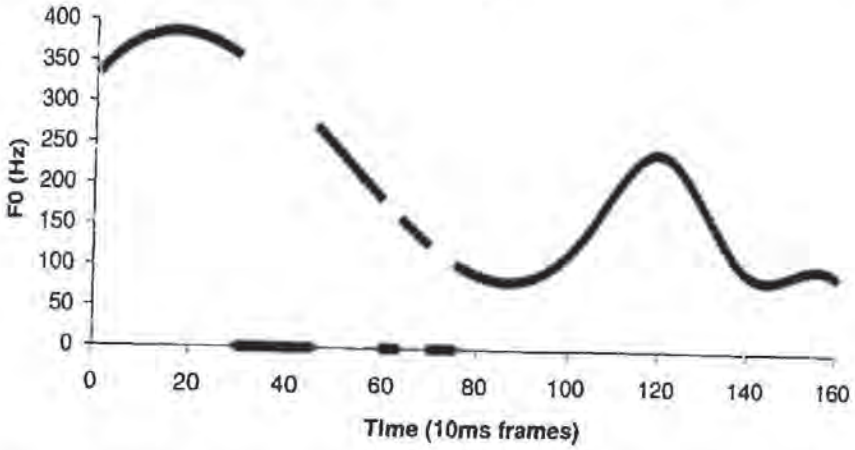


Figure 15.9 F0 contour of Figure 15.8 after cubic interpolation. Sections corresponding to unvoiced phones have been replaced with zero.

In order for the interpolation algorithm to operate properly we need to have phone durations so that the anchor points are appropriately spaced apart. In this baseline algorithm, we followed the algorithm described in Section 15.5.2.

15.6.2.6. Interface to Synthesis Module

Finally, most synthesizers cannot accept an arbitrary number of pitch controls on a given phoneme, nor it this necessary. We can downsample the pitch buffer to allocate a few characteristic points per phoneme record, and, if the synthesizer can interpolate pitch, it may be desirable to skip pitch controls for unvoiced phones altogether. The F0 targets can be placed at default locations (such as the left edge and middle of each phone), or the placements can be indicated by percent values on each target, depending on what the synthesizer supports. This has to be in agreement with the specific interface between the prosody module and the synthesis module as described in Section 15.2.

15.6.2.7. Evaluation and Possible Improvements

In comparing the output contour of Figure 15.9 to the natural one of Figure 15.7, how well have we done? As a first-order approximation, from visual inspection, it is somewhat similar

to the original. Of course, we have used hand-coded information for the accent property, accent type, and prominence! However, these choices were reasonable, and could apply them as defaults to many other utterances. At a minimum, almost exactly the code given above would apply without change and give a decent contour for a whole 'family' of similar utterances, such as *"Don't hit the ball to Joey!"* or *"Never give the baseball to Henry!"* A higher-order discourse module would need to determine that *ball* and *baseball* are not accented, however, in order to use the given contour with the same rhetorical effect (presumably *ball* and *baseball* in these cases could be given/understood information).

Something very much like the system described here has been used in most commercially marketed synthesizers throughout the 1990s. This model seems overly simple, even crude, and presumably it could be substantially augmented, or completely replaced by something more sophisticated.

However, many weaknesses are apparent also. For one thing, the contour appears very smooth. The slight *jitter* of real contours can be easily simulated at a final stage of pitch buffer processing by modifying +/- 3 or 4 Hz to the final value of each frame. The degree to which such niceties actually affect listener perceptions depend entirely on the quality of the synthetic speech and the quality of the pitch-modification algorithms in the synthesizer.

The details of peak placement obviously differ between the natural and synthetic contours. This is partly due to the crude uniform durations used, but in practice synthesizers may incorporate large batteries of rules to decide exactly (for example) which frame of a phone the H* definition point should appear in—early, middle, late? Sometimes this decision is based on surrounding phonetic structure, word and syllable structure, and prosodic context. The degree to which this matters in perception depends partly on synthetic speech quality overall.

15.6.3. Parametric F0 Generation

To realize all the prosodic effects discussed above, some systems make almost direct use of a real speaker's measured data, via table lookup methods. Other systems use data indirectly, via parametrized algorithms with generic structure. The simplest systems use an invariant algorithm that has no particular connection to any single speaker's data, such as the algorithm described in the baseline F0 generation system of Section 15.6.2. Each of these approaches has advantages and disadvantages, and none of them has resulted in a system that fully mimics human prosodic performance to the satisfaction of all listeners. As in other areas of TTS, researchers have not converged on any single standard family of approaches. Once we venture beyond the simplest approaches, we find an interesting variety of systems, based on different assumptions, with differing characteristics. We now discuss a few of the more representative approaches.

Even models that make little or no attempt to analyze the internal components of an F0 contour must be indexed somehow. System designers should choose indexing or predictive factors that are derivable from text analysis, are general enough to cover most prosodic situations, and are powerful enough to specify high-quality prosody. In practice, most mod-

els' predictive factors have a rough correspondence to, or are an elaboration of, the elements of the baseline algorithm of Section 15.6.2. A typical list might include the following:

- Word structure (stress, phones, syllabification)
- Word class and/or POS
- Punctuation and prosodic phrasing
- Local syntactic structure
- Clause and sentence *type* (declarative, question, exclamation, quote, etc.)
- Externally specified focus and emphasis
- Externally specified speech style, pragmatic style, emotional tone, and speech act goals

These factors jointly determine an output contour's characteristics, as listed below. Ideally, any or all of these may be externally and directly specified, or they may be inferred or implied within the F0 generation model itself:

- Pitch-range setting
- Gradient, relative prominence on each syllable
- Global declination trend, if any
- Local shape of F0 movement
- Timing of F0 events relative to phone (carrier) structure

The combinatorial complexity of these predictive factors, and the size of the resulting models, can be serious issues for practical systems that strive for high coverage of prosodic variability and high-quality output. The possibility of using *externally specified* symbolic markups gives the whole system a degree of modularity, in that prosodic annotation can be specified directly by an authoritative outside source or can be derived automatically by the symbolic prosody prediction process that precedes F0 contour generation.

Parametric models propose an underlying architecture of prosodic production or perception that constrains the set of possible outputs to conform to universal constants of the human speech mechanism. Naturally, these models need settings to distinguish different speakers, different styles, and the specifics of utterances. We describe superposition models and ToBI Realization models.

15.6.3.1. Superposition Models

An influential class of parametric models was initiated by the work [35] for Swedish, which proposed additive superposition of component contours to synthesize a complex final F0 track. In the version refined and elaborated in [14], the component contours, which may all have different strengths and decay characteristics, may correspond to longer-term trends, such as phrase or utterance declination, as well as shorter-time events, such as pitch accents

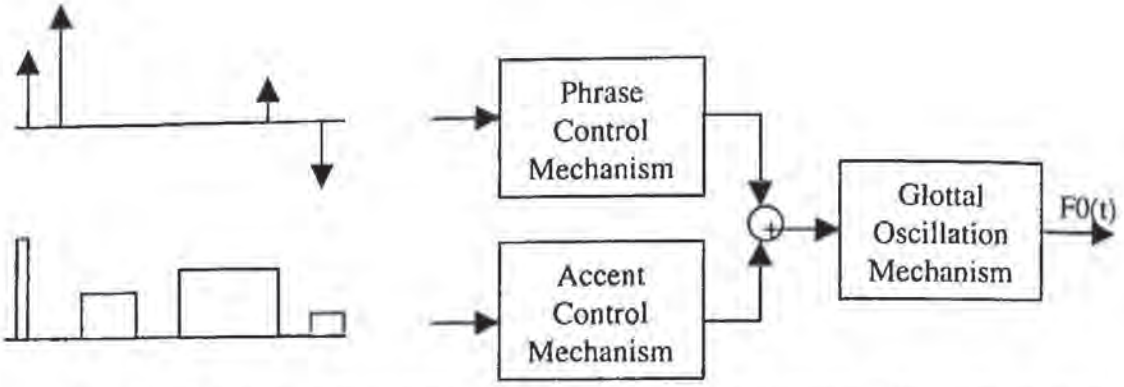


Figure 15.10 Fujisaki pitch model [15]. F0 is a superposition of phrase effects with accent effects. The phrase mechanism controls things like the declination of a declarative sentence or a question, whereas the accent mechanism accounts for accents in individual syllables.

on words. The component contours are modeled as the critically damped responses of second-order systems to impulse functions for the longer-term, slowly decaying phrasal trends, and step or rectangular functions of shorter-term accent events. The components so generated are added and ride a baseline that is speaker specific. The basic ingredients of the system, known as Fujisaki's model [15, 19], are shown in Figure 15.10. The resulting contour is shown in Figure 15.11. Obviously, similar effects can be generated with linear accent shapes as described in the simpler model above, with smoothing. However, there are some plausible claims for the articulatory correlates of the constraints imposed in the second-order damping and superposition effects of this model [33].

Superposition models of this type can, if supplied with accurate parameters in the form of time alignments and strengths of the impulses and steps, generate contours closely mimicking natural examples. In this respect, the remaining quality gap for general application is in the parametric knowledge driving the model, not in the model structure per se. These kinds of models have been particularly successful in replicating the relatively constrained Japanese reading-style. Whether these models can account straightforwardly for the immense variety of a large range of English speakers and text genre, or whether, on the contrary, the parameters proliferate and the settings become increasingly arbitrary, remains to be seen.



Figure 15.11 Composite contour obtained by low-pass filtering the impulses and boxes in the Fujisaki model of Figure 15.10.

15.6.3.2. ToBI Realization Models

One simple parametric model, which in its inherent structure makes only modest claims for principled correspondence to perceptual or articulatory reality, is designed to support prosodic symbols such as the *Tones and Break Indices* (ToBI) system. This model, variants of which are developed in [2, 54], posits two or three control lines, by reference to which ToBI-like prosody symbols can be scaled. This provides for some independence between symbolic and phonetic prosodic subsystems. In the model shown in Figure 15.12, the top line is an upper limit of the pitch range. It can be slanted down to simulate declination. The bottom line represents the bottom of the speaker's range. Pitch accents and boundary tones (as in ToBI) are scaled from a reference line, which is often midway in the range in a logarithmic scale of the pitch range, as described in the baseline algorithm of 15.6.2. You can think of this scaling as operating within a percentage of the current range, rather than absolute values, so a generic method can be applied to any arbitrary pitch-range setting. The quantitative instantiation of accent height is done at the final stage. The accents and boundary tones consist of one or more points, which can be aligned with the carrier phones; then interpolation is applied between points, and smoothing is performed over the resulting contour.

In Figure 15.12, t, r, and b are the top, reference, and baseline pitch values, respectively. They are set from the defaults of the voice character and by user choice. The base b is considered a physiological constraint of voice. P is the prominence of the accent and N is the number of prominence steps. Declination can be modeled by slanting the top and/or reference lines down. The lowered position of the reference in Figure 15.12 reflects the observation that the realization of H(igh) and L(ow) ToBI abstract tones in a given pitch range is asymmetric, with a greater portion available for H, while L saturates more quickly. This is why placing the reference line midway between the top and base lines in a log-frequency scale automatically takes care of this phenomenon. After target points are located and scaled according to their gradient prominence specifications, the (hopefully sparse) targets can be interpolated and the resulting contour smoothed. If the contour is calculated in, say, 10-ms frames, two pitch targets sampled from the contour vector per phone usually suffice to reproduce the intended prosodic effects faithfully.

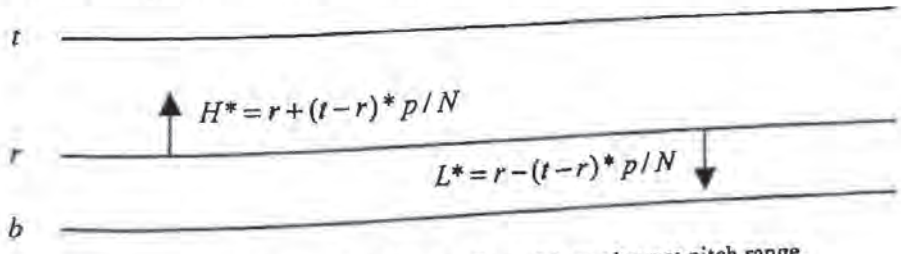


Figure 15.12 A typical model of tone scaling with an abstract pitch range.

If a database of recorded utterances with phone labeling and F0 measurements has been reliably labeled with ToBI pitch annotations, it may be possible to automate the implementation of the ToBI-style parametrized model. This was attempted with some success in [5], where linear regression was used to predict syllable initial, vowel medial, and syllable final F0 based on simple, accurately measurable factors such as:

- ToBI accent type of target and neighbor syllables
- ToBI boundary pitch type of target and neighbor syllables
- Break index on target and neighbor syllables
- Lexical stress of target and neighbor syllables
- Number of syllables in phrase
- Target syllable position phrase
- Number and location of stressed syllable(s)
- Number and location of accented syllable(s)

Models of this sort do not incorporate an explicit mechanism (like the scaling direction from r in Figure 15.12) to distinguish H(igh) from L(ow) tone space, beyond what the data and its annotations imply.

The work in [47] consists of a ToBI realization model in which the 'smoothing' mechanism is built-in as a dynamical system whose parameters are also learnt from data. This work could be viewed as a stochastic realization of Fujisaki's superposition model without the phrase controls and where the accents are given by ToBI labels.

Both the ToBI realization models and the superposition models could, if supplied with sufficiently accurate measurements of an example contour, reproduce it fairly accurately. Both models require much detailed knowledge (global and local pitch range; location, type, and relative strength of accents and boundary tones; degree of declination; etc.) to function at human-level quality for a given utterance. If a system designer is in possession of a completely annotated, high-quality database of fully representative prosodic forms for his/her needs, the question of deployment of the database in a model can be made based on performance tradeoffs, maintenance issues, and other engineering considerations. If, on the other hand, no such database is available for the given application purpose, extremely high prosodic quality, including lively yet principled variation, should not be expected to result simply from choosing the 'mathematically correct' model type.

15.6.4. Corpus-Based F0 Generation

It is possible to have F0 parameters trained from a corpus of natural recordings. The simplest models are the direct models, where an exact match is required. Models that offer more generalization have a library of F0 contours that are indexed either from features from the parse tree or from ToBI labels. Finally, there are F0 generation models from a statistical

network such as a neural network or an HMM. In all cases, once the model is set, the parameters are learned automatically from data.

15.6.4.1. Transplanted Prosody

The most direct approach of all is to store a single contour from a real speaker's utterance corresponding to every possible input utterance that one's TTS system will ever face. This seems to limit the ability to freely synthesize any input text. However, this approach can be viable under certain special conditions and limitations. These controls are so detailed that they are tedious to write manually. Fortunately, they can be generated automatically by speech recognition algorithms.

When these controls (*transplanted prosody*), taken from an authentic digitized utterance, are applied to synthetic voice units, the results can be very convincing, sometimes nearly as good as the original digitized samples [43]. A system with this capability can mix predefined utterances having natural-quality prosody, such as greetings, with flexible synthesis capabilities for system response, using a consistent synthetic voice. The transplanted prosody for the *frozen* phrases can be derived either from the original voice data donor used to create the synthetic voice model, or any other speaker, with global adjustment for pitch-range differences. Another use of the transplanted prosody capability is to compress a spoken message into ASCII (phone labels plus the prosodic controls) for playback, preserving much of the quality, if not the full individuality, of the original speaker's recording.

15.6.4.2. F0 Contours Indexed by Parsed Text

In a more generalized variant of the direct approach, once could imagine collecting and indexing a gigantic database of clauses, phrases, words, or syllables, and then annotating all units with their salient prosodic features. If the terms of annotation (word structure, POS, syntactic context, etc.) can be applied to new utterances at runtime, a prosodic description for the closest matching database unit can be recovered and applied to the input utterance [23]. The advantages here are that prosodic quality can be made arbitrarily high, by collecting enough exemplars to cover arbitrarily large quantities of input text, and that detailed analysis of the deeper properties of the prosodic phenomena can be sidestepped. The potential disadvantages are:

- Data-collection time is long (which affects the capability to create new voices).
- A large amount of runtime storage is required (presumably less important as technology progresses).
- Database annotation may have to be manual, or if automated, may be of poor quality.
- The model cannot be easily modified/extended, owing to lack of fundamental understanding.

- Coverage can never be complete, therefore rulelike generalization, fuzzy match capability, or back-off, is needed.
- Consistency control for the prosodic attributes (to prevent unit boundary mismatches) can be difficult.

The first two disadvantages are self-explanatory. The difficulty of annotating the database, to form the basis of the indexing and retrieval scheme, depends on the type and depth of the indexing parameters chosen. Any such scheme requires annotations to identify the component phones of each unit (syllable, word, or phrase) and their durations. This can usually be obtained from speech recognition tools [23], which may be independently required to create a synthetic voice (see Chapter 16). Lexical or word stress attributes can be extracted from an online dictionary or NLP system, though, as we have seen above, lexical stress is neither a necessary nor a sufficient condition for predicting pitch accent placement.

If only a very high level of description is sought, based primarily on the pragmatics of utterance use and some syntactic typology, it may not be necessary to recover a detailed symbolic pitch analysis. An input text can be described in high-level pragmatic/semantic terms, and pitch from the nearest matching word or phrase from the database can be applied with the expectation that its contour is likely correct. For example, such a system might have multiple prosodic versions of a word that can be used in different pragmatic senses, such as *ok*, which could be a question, a statement, an exclamation, a signal of hesitation or uncertainty, etc. The correct version must be selected based on the runtime requirements of the application.

Direct prosody schemes of this type often preserve the original phone carrier material of each instance in order to assure optimal match between prosody and spectrum. However, with DSP techniques enabling arbitrary modifications of waveforms (see Chapter 16), this is not strictly necessary; the prosodic annotations could stand alone, with phone label annotation only. If more detailed prosodic control is required, such as being aware of the type of accent, its pitch range, prominence, and other features, the annotation task is much more difficult.

A straightforward and elegant formulation of the lookup-table direct model approach can be found in [30]. This system, created for Spanish but generally adaptable, is based on a large single-speaker recorded database of a variety of sentence types. The sentences are linguistically analyzed, and prosodic structure is hypothesized based on syllables, accent groups (groups of syllables with one lexical stress), breath groups (groups of accent-groups between pauses regardless of the duration of the pause), and sentences. Note that these structures are hypothesized based on the textual material alone, and the speaker will not always perform accordingly. Pitch (initial, mid, and final F0) and duration data for each spoken syllable is automatically measured and stored. At runtime, the input sentence is analyzed using the same set of structural attributes inferred from the text, and a vector of candidate syllables from the database is constructed with identical, or similar, structural context and attributes for each successive input syllable position.

The best path through the set of such candidates is selected by minimizing the F0 distance and disjuncture across the utterance. This is a clean and simple approach to jointly

utilizing both shallow linguistic features and genuine phonetic data (duration and F0), with dynamic programming to smooth and integrate the output F0 contour. However, as with any direct modeling approach, it lacks significant generalization capabilities outside the textual material and speaking style specified during the data collection phase, so a number of separate models may have to be constructed.

The CHATR system of ATR (Japan) [9] takes a similar approach, in that optimal prosody is selected from prerecorded units, rather than synthesized from a more general model. The CHATR system consists of a large database of digitized speech, indexed by the speaker identity, the phoneme sequences of the words, and some pragmatic and semantic attributes. Selection of phonemes proceeds by choosing the minimal-cost path from among the similarly indexed database candidate units available for each phoneme or longer segment of speech to be synthesized. This system achieves high quality by allowing the carrier phones to bear only their original prosody—pitch modification of the contour is minimized or eliminated. Of course, the restriction of DSP modification implies a limitation of the generalizability of the database. This type of approach obtains the prosody implicitly from the database [31], and as such combines both the prosody and speech synthesis modules. This type of minimal-cost search is described in more detail in Chapter 16.

15.6.4.3. F0 Contours Indexed by ToBI

The architecture for a simple and straightforward direct model indexed by ToBI is diagrammed in Figure 15.13. This model combines the two often-conflicting goals: it is empirically (corpus) based, but it permits specification in terms of principled abstract prosodic categories. In this model, an utterance to be synthesized is annotated in terms of its linguistic features—perhaps POS, syntactic structure, word emphasis (based on information structure), etc. The utterance so characterized is matched against a corpus of actual utterances that are annotated with linguistic features and ToBI symbols, Corpus (a). A *fuzzy* matching capability based on edit distance or dynamic programming can be incorporated. If Corpus (a) is sufficiently large and varied, a number of possible ToBI renderings of either the entire utterance or selected parts of it may be recovered. At this level of abstraction, the ToBI labels would not encode relative prominence specifications (strength of pitch extrusions) or pitch range. The set of such abstractly described contours can then be fuzzy matched into Corpus (b), a set of ToBI annotated actual contours, and the best set of matches recovered.

Note that while it is possible that Corpus (a) is the exact same base material as Corpus (b), the model does not enforce an identity, and there may be reasons to desire such flexibility and modularity, depending on the degree and quality of data and annotation at each level. Once a number of likely actual contours have been identified, they can be passed to a voice-unit selection module. The module can select the combination of segmental strings (sometimes called 'long units,' since they may combine more than one phoneme) from the voice database whose original prosody is closest to one of the candidate contours, using root-mean-square-error, correlation, or other statistical tests. Those units are then concatenated (with their prosody unmodified) and sent to the application or played out.

A model of this type has some of the disadvantages of direct models as listed above. It also assumes availability of large and varied databases of both prosodic contours and segmental (phone) long units for concatenation (see Chapter 16). It further requires that these databases be annotated, either by human labelers or automated systems. However, it has certain advantages as well:

- It allows for symbolic, phonological coding of prosody.
- It has high-quality natural contours.
- It has high-quality phonetic units, with unmodified pitch.
- Its modular architecture can work with user-supplied prosodic symbols.

It also allows the immediate, temporary use of data that is collected for deeper analysis, in the hope of eventual construction of smaller, parametrized models. The model of Figure 15.13 is a generalization of the prosody system described in [23].

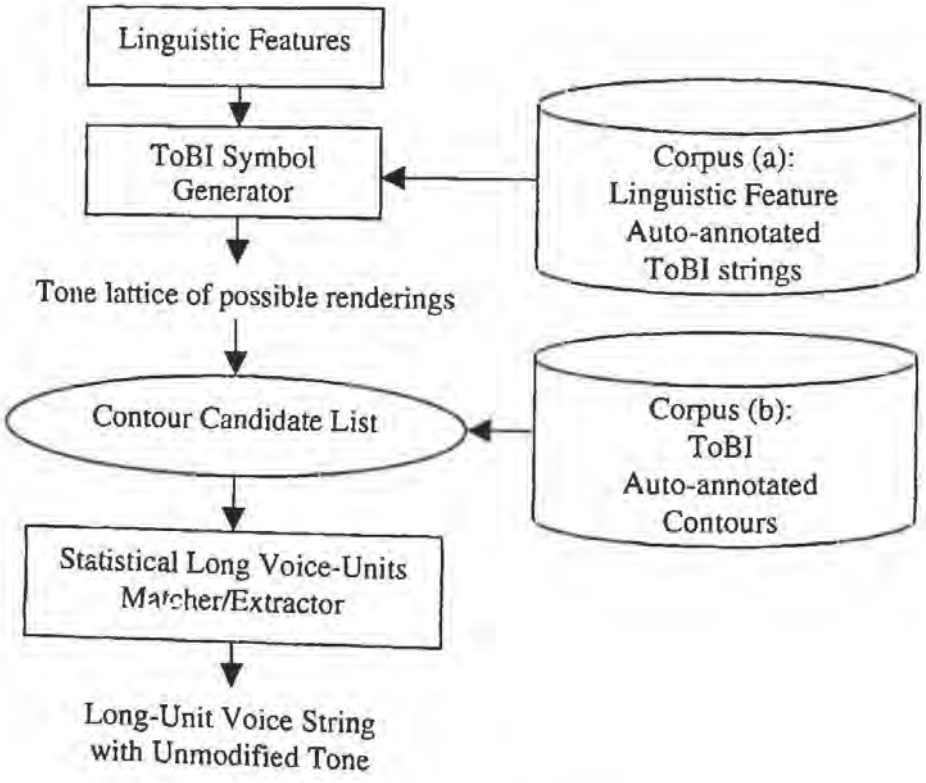


Figure 15.13 A corpus-based prosodic generation model.

15.7. PROSODY MARKUP LANGUAGES

Chapter 14 discussed generalized document markup schemes for text analysis. Most TTS engines provide simple text tags and application programming interface controls that allow at least rudimentary hints to be passed along from an application to a TTS engine. We expect to see more sophisticated speech-specific annotation systems, which eventually incorporate current research on the use of semantically structured inputs to synthesizers, sometimes called concept-to-speech systems. A standard set of prosodic annotation tags would likely include tags for insertion of silence pause, emotion, pitch baseline and range, speed in words-per-minute, and volume. This would be in addition to general tags for specifying the language of origin if not predictable, character of the voice, and text normalization context such as address, date, email, etc.

For prosodic processing, text may be marked with tags that have scope, in the general fashion of XML. Some examples of the form and function of a few common TTS tags for prosodic processing, based loosely on the proposals of [65], are introduced below. Other tags can be added by intermediate subcomponents to indicate variables such as accents and tones. This extension allows for even finer research and prosody models.

- **Pause** or **Break** commands might accept either an absolute duration of silence in milliseconds, or, as in the W3C proposal, a mnemonic describing the relative salience of the pause (Large, Medium, Small, None), or a *prosodic punctuation* symbol from the set ',', '.', '?', '!', '...', etc., which not only indicates a pause insertion but also influences the typical pitch contour of the phone segments entering and leaving the pause area. For example, specifying ',' as the argument of a Pause command might determine the use of a continuation rise on the phones immediately preceding the pause, indicating incompletion or listing intonation.
- **Rate** controls the speed of output. The usual measurement is *words per minute*, which can be a bit vague, since words are of very different durations. However, this metric is familiar to many TTS users and works reasonably well in practice. For non-IndoEuropean languages, different metrics must be contemplated. Some power listeners who use a TTS system routinely can tolerate (in fact, demand) rates of over 300 words per minute, while 150 or fewer might be all that a novice listener could expect to reliably comprehend.
- **Baseline Pitch** specifies the desired average pitch: a level around which, or up from which, pitch is to fluctuate.
- **Pitch Range** specifies within what bounds around the baseline pitch level line the pitch is to fluctuate.
- **Pitch** commands can override the system's default prosody, giving an application or document author greater control. Generally, TTS engines require some freedom to express their typical pitch patterns within the broad limits specified by a Pitch markup.

- **Emphasis** emphasizes or deemphasizes one or more words, signaling their relative importance in an utterance. Its scope could be indicated by XML style. Control over emphasis brings up a number of interesting considerations. For one thing, it may be desirable to have degrees of emphasis. The notion of gradient prominence—the apparent fact that there are no categorical constraints on levels of relative emphasis or accentuation—has been a perpetual thorn in the side for prosodic researchers. This means that in principle any positive real number could be used as an argument to this tag. In practice, most TTS engines would artificially constrain the range of emphasis to a smaller set of integers, or perhaps use semantic labels, such as *strong*, *moderate*, *weak*, *none* for degree of emphasis. Emphasis may be realized with multiple phonetic cues. Thus, if the user or application has, for example, set the pitch range very narrowly, the emphasis effect may be achieved by manipulation of segmental duration or even relative amplitude. The implementation of emphasis by a TTS engine for a given word may involve manipulation (e.g., de-accentuation) of surrounding words as much as it involves heightening the pitch or volume, or stretching the phone durations, of the target word itself. In most cases the main phonetic and perceptual effect of emphasis or accentuation is heard on the lexically main stressed syllable of the word, but this can be violated under special conditions of semantic focus, e.g., "*I didn't say employer, I said employee.*" This would require a more powerful emphasis specification than is currently provided in most TTS systems, but alternatively it could be specified using phone input commands such as "*The <emp>truth</emp>, the <emp>whole truth</emp>, and nothing <emp> but</emp> the truth.*" For more control, future TTS systems may support degree emphasis: "*... nothing <emp level="strong">but</emp> the truth*" or even deemphasis: "*... nothing <emp level= "reduced">but</emp> the truth*". Emphasis is related to prominence, discussed in Section 15.6.1.2.

15.8. PROSODY EVALUATION

Evaluation of a complete TTS system is discussed in Chapter 16. We limit ourselves here to evaluating the prosody component. We assume that the text analysis module has done a perfect job, and that the synthesis module does a perfect job, which cannot be done in general, so that approximations need to be made.

Evaluation can be done automatically or by using listening tests with human subjects. In both cases it's useful to start with some natural recordings with their associated text. We start by replacing the natural prosody with the system's synthetic prosody. In the case of automatic evaluation, we can compare the enriched prosodic representations described in Section 15.2 for both the natural recording and the synthetic prosody. The reference en-

riched prosodic representation can be obtained either manually or by using a pitch tracker and a speech recognizer.

Automated testing of prosody involves the following:

- **Duration.** It can be performed by measuring the average squared difference between each phone's actual duration in a real utterance and the duration predicted by the system.
- **Pitch** contours. It can be performed by using standard statistical measures over a system contour and a natural one. When this is done, duration and phoneme identity should be completely controlled. Measures such as root-mean-square error indicate the characteristic divergence between two contours, while correlation indicates the similarity in shape across difference pitch ranges. In general, RMSE scores of 15 Hz or less for male speech over a long sentence, with correlation of .8 or above, indicate quality that may be close to perceptually identical to the natural reference utterance. In general, such exactness of match is useful only during model training and testing and cannot be expected during training on entirely new utterances from random text.

Listening tests can be performed to evaluate a prosody module. This involves subjects listening to the natural recording and the synthetic speech, or to synthetic speech generated with two different prosody modules. This can lead to a more precise evaluation, as humans are the final consumer of this technology. However, such tests are more expensive to carry out. Furthermore, this method results in testing both the prosody module and the synthesis components together. To avoid this, the original waveform can be modified to have the synthetic prosody using the signal processing techniques described in Chapter 16. Since such techniques introduce some distortions, this measuring method is still somewhat biased. In practice, it has been shown that its effect is much smaller than that of the synthetic prosody [43].

It is shown that synthesizing pitch is more difficult than duration [43]. Subjects scored significantly higher utterances that had natural pitch and synthetic duration than utterances with synthetic pitch and natural duration. In fact, using only synthetic duration had a score quite close to that of the original recording. While duration modeling is not a solved problem, this indicates that generation of pitch contours is more difficult.

15.9. Historical Perspective and Further Reading

Prosodic methods have been incorporated within the traditional fields of rhetoric and elocution for centuries. In ancient Greece, at the time of Plato, written documentation in support of claims in legal disputes was rare. To help litigants plead their cases persuasively, systematic programs of rhetorical instruction were established, which included both content and form of verbal argument. This 'prescriptive' tradition of systematic instruction in verbal

style uncovered issues that remain central to the descriptively oriented prosodic research of today. A masterful and entertaining discussion of this tradition and its possible relevance to the task of teaching computers to *plead a case* can be found in [64]. The Greeks were particularly concerned about an issue that, as usual, is still important for us today: the separation of rhetorical effectiveness from considerations of truth. If you are interested in this, you cannot do better than to begin with Plato's dialog *Phaedrus* [42].

Modern linguists have also considered a related, but more narrowly formulated question: Should prosody be treated as a logical, categorical analog to phonological and syntactic processes? The best discussion of these issues from a prosodic (as opposed to strictly neurological) point of view is found in [7, 8]. If you are interested in the neurological side, you can begin with [13]. For emotional modeling, before slogging through the scattered and somewhat disjointed papers on emotion in speech that have appeared sporadically for years, the reader would be well advised to get a basic grounding in some of the issues related to emotion in computation, as treated in [38].

Going in the other direction, there are many subtle interactions in the phonetics of prosody: the various muscles, their joint possibilities of operation in phonation and articulation, as well as the acoustics properties of the vocal chambers. For an excellent introduction to the whole field, start with [27].

The most complete and accessible overview of modern prosodic analysis as embedded in mainstream linguistic theory is Ladd's *Intonational Phonology* [26], which covers the precursors, current standard practice, and remaining unsolved issues of the highly influential auto segmental theory of intonational phonology, from which ToBI has arisen. ToBI was devised by speech scientists who wanted a prosodic transcription standard to enable sharing of databases [4]. For most practical purposes, the ToBI definitions are sufficient as a starting point for both research and applications, but for those who prefer alternative annotation systems aligned with the British tradition, conversion guidelines have been attempted [45]. Another major phonological approach to English intonation has been the British school described in [11]. Bridging the two is IViE, a labeling system that is philosophically aligned with ToBI but may be more appropriate for non-U.S. dialects of English [18].

The first prosodic synthesis by rule was developed by Ignatius Mattingly in 1968 in Haskins Laboratories. In 1971, Fujisaki [15] developed his superposition model that has been used for many years. The development of the ToBI in 1992 [55] marked a milestone in automatic prosody generation. The application of statistical techniques, such as CART, for phoneme durations during the 1990s constituted a significant step beyond the rule-based methods. Finally, the development of the CHATR system in the mid-1990s ignited interest in the indexing of massive databases. It is possible to attempt smoothing over both the index space and the resulting prosodic data tracks by means of generalized learning methods, such as neural nets or HMMs. These models have built-in generalization over unseen inputs, and built-in smoothing over the concatenated outputs of unit selection. The network described in [57] codes every syllable in a training database in terms of perceived prominence (human judged), a number from 1 to 31, as well as the syllable's phonemes, rising/falling boundary type for phrase-edge syllables, and distance from preceding and following phrase bounda-

ries, for all syllables. When tested with reasonably simple text material of similar type, these networks yielded high-quality simulations.

A potential research area for future generalizations of this system is to increase the degree and accuracy of automation in labeling the training features of the recordings, such as perceived prominence. Another area is to either expand the inventory of model types, or to determine adequate generalization mechanisms. By training HMMs on accented syllables of differing phonetic structure, some of this fine alignment information can be automatically captured [16]. Another approach consists in generating pitch contours directly from a hidden Markov model, which is run in generation mode [66].

Recently, just as in speech synthesis for voice, there has been a realization that the *direct* and *parametric* prosodic models have a great deal in common. Direct models require huge databases of indexed exemplars for unmediated concatenation and playback of contours, in addition to generalized back-off methods, while parametric models are generalized for any input, but also require phonetic databases of sufficient variety to support statistical learning of parameter settings for high quality. We can, therefore, expect to see increasing numbers of hybrid systems. One such system is described in [47], which could be viewed as a stochastic realization of Fujisaki's superposition model without the phrase controls, where the accents are given by ToBI labels and the smoothing is done by means of a dynamical system.

While this chapter has focused on U.S. English, many similar issues arise in prosodic modeling of other languages. An excellent survey of the prosodic systems of every major European language, as well as Arabic and several major East Asian languages, can be found in [21].

Though not explicitly covered in this chapter, analysis of prosody for speech recognition is a small but growing area of study. Anyone who has digested this chapter should be prepared to approach the more specialized work of [25, 37] and the speech recognition prosody studies collected in [48]. Those with a psycholinguistic bent can begin with [12].

REFERENCES

[1] Allen, J., M.S. Hunnicutt, and D.H. Klatt, *From Text to Speech: the MITalk System*, 1987, Cambridge, UK, University Press.

[2] Anderson, M.D., J.B. Pierrehumbert, and M.Y. Liberman, "Synthesis by Rule of English Intonation Patterns," *Proc. of Int. Conf. on Acoustics, Speech and Signal Processing*, 1984, pp. 2.8.1-2.8.4.

[3] Arnfield, S., "Word Class Driven Synthesis of Prosodic Annotations," *Proc. of the Int. Conf. on Spoken Language Processing*, 1996, Philadelphia, PA, pp. 1978-1981.

[4] Beckman, M.E. and G.M. Ayers, *Guidelines for ToBI Labelling*, 1994, http://www.ling.ohio-state.edu/phonetics/ToBI/main.html.

[5] Black, A. and A. Hunt, "Generating F0 Contours from ToBI labels using Linear Regression," *Proc. of the Int. Conf. on Spoken Language Processing*, 1996, pp. 1385-1388.

[6] Bolinger, D., "Accent is predictable (if you're a mind-reader)," *Language*, 1972, **48**, pp. 633-44.

[7] Bolinger, D., *Intonation and its parts*, 1986, Stanford, Stanford University Press.

[8] Bolinger, D., *Intonation and its uses*, 1989, Stanford, Stanford University Press.

[9] Campbell, N., "CHATR: A High-Definition Speech Re-sequencing System," *ASA/ASJ Joint Meeting*, 1996, Honolulu, Hawaii, pp. 1223-1228.

[10] Carroll, L., *Alice in Wonderland, Unabridged ed.*, 1997, Penguin USA.

[11] Crystal, D., *Prosodic Systems and Intonation in English*, 1969, Cambridge University Press.

[12] Crystal, D., "Prosody and Parsing," P. Warren, Editor, 1996, Lawrence Erlbaum Associates.

[13] Emmorey, K., "The Neurological Substrates for Prosodic Aspects of Speech," *Brain and Language*, 1987, **30**, pp. 305-320.

[14] Fujisaki, H., "Prosody, Models, and Spontaneous Speech" in *Computing Prosody*, Y. Sagisaka, N. Campbell, N. Higuchi, Editors, 1997, New York, Springer.

[15] Fujisaki, H. and H. Sudo, "A Generative Model of the Prosody of Connected Speech in Japanese," *Annual Report of Eng. Research Institute*, 1971, **30**, pp. 75-80.

[16] Fukada, T., *et al.*, "A Study on Pitch Pattern Generation Using HMM-based Statistical Information," *Proc. Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan, pp. 723-726.

[17] Goldsmith, J., "English as a Tone Language" in *Phonology in the 1980's*, D. Goyvaerts, Editor 1980, Ghent, Story-Scientia.

[18] Grabe, E., F. Nolan, and K. Farrar, "IViE - a Comparative Transcription System for Intonational Variation in English," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[19] Hirose, H. and H. Fujisaki, "Analysis and Synthesis of Voice Fundamental Frequency Contours of Spoken Sentences," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1982, pp. 950-953.

[20] Hirschberg, J., "Pitch Accent in Context: Predicting Intonational Prominence from Text," *Artificial Intelligence*, 1993, **63**, pp. 305-340.

[21] Hirst, D., A.D. Cristo, and A. Cruttenden, *Intonation Systems: A Survey of Twenty Languages*, 1998, Cambridge, U.K., Cambridge University Press.

[22] Hirst, D.J., "The Symbolic Coding of Fundamental Frequency Curves: from Acoustics to Phonology," *Proc. of Int. Symposium on Prosody*, 1994, Yokohama, Japan.

[23] Huang, X., *et al.*, "Whistler: A Trainable Text-to-Speech System," *Int. Conf. on Spoken Language Processing*, 1996, Philadephia, PA, pp. 2387-2390.

[24] Klasmeyer, G. and W.F. Sendlmeier, "The Classification of Different Phonation Types in Emotional and Neutral Speech," *Forensic Linguistics*, 1997, **4**(1), pp. 104-125.

[25] Kompe, R., *Prosody in Speech Understanding Systems*, 1997, Berlin, Springer.

[26] Ladd, R.D., *Intonational Phonology*, Cambridge Studies in Linguistics, 1996, Cambridge, Cambridge University Press.

[27] Ladefoged, P., *A Course in Phonetics*, 1993, Harcourt Brace Jovanovich.

[28] Liberman, M., *The Intonation System of English*, PhD Thesis in Linguistics and Philosophy, 1975, MIT, Cambridge.

[29] Liberman, M. and J. Pierrehumbert, "Intonational Invariance under Changes in Pitch Range and Length" in *Language and Sound Structure*, M. Aronoff, Oerhle, R., ed., 1984, Cambridge, MA, MIT Press, pp. 157-233.

[30] Lopez-Gonzalo, E., and J.M. Rodriguez-Garcia, "Statistical Methods in Data-Driven Modeling of Spanish Prosody for Text to Speech," *in Proc. ICSLP 1996*, 1996, pp. 1373-1376.

[31] Malfrere, F., T. Dutoit, and P. Mertens, "Automatic Prosody Generation Using Supra-Segmental Unit Selection," *Third ESCA/COCOSCA Workshop on Speech Synthesis*, 1998, Jenolan Caves, Australia, pp. 323-328.

[32] Mason, J., *An Essay on Elocution*, 1st ed, 1748, London.

[33] Möbius, B., "Analysis and Synthesis of German F0 Contours by Means of Fujisaki's Model," *Speech Communication*, 1993, **13**(53-61).

[34] Murray, I. and J. Arnott, "Toward the Simulation of Emotion in Synthetic Speech: A Review of the Literature on Human Vocal Emotion," *Journal Acoustical Society of America*, 1993, **93**(2), pp. 1097-1108.

[35] Öhman, S., *Word and Sentence Intonation: A Quantitative Model*, 1967, KTH, pp. 20-54.

[36] Ostendorf, M., and N. Veilleux, "A Hierarchical Stochastic Model for Automatic Prediction of Prosodic Boundary Location," *Computational Linguistics*, 1994, **20**(1), pp. 27-54.

[37] Ostendorf, M., "Linking Speech Recognition and Language Processing Through Prosody," *Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology*, 1998, **15**(3), pp. 279-303.

[38] Picard, R.W., *Affective Computing*, 1997, MIT Press.

[39] Pierrehumbert, J., *The Phonology and Phonetics of English Intonation*, PhD Thesis in *Linguistics and Philosophy* 1980, MIT, Cambridge, MA.

[40] Pierrehumbert, J., and M. Beckman, *Japanese Tone Structure*, 1988, Cambridge, MA, MIT Press.

[41] Pierrehumbert, J. and J. Hirschberg, "The Meaning of Intonational Contours in the Interpretation of Discourse" in *Intentions in Communication*, P.R. Cohen, J. Morgan, and M. E. Pollack, ed., 1990, Cambridge, MA, MIT Press.

[42] Plato, *The Symposium and The Phaedrus: Plato's Erotic Dialogues*, 1994, State University of New York Press.

[43] Plumpe, M. and S. Meredith, "Which is More Important in a Concatenative Text-to-Speech System: Pitch, Duration, or Spectral Discontinuity," *Third ESCA/COCOSDA Int. Workshop on Speech Synthesis*, 1998, Jenolan Caves, Australia, pp. 231-235.

[44] Prevost, S. and M. Steedman, "Specifying Intonation from Context for Speech Synthesis," *Speech Communication*, 1994, **15**, pp. 139-153.

[45] Roach, P., "Conversion between Prosodic Transcription Systems: 'Standard British' and ToBI," *Speech Communication*, 1994, **15**, pp. 91-97.

[46] Ross, K. and M. Ostendorf, "Prediction of Abstract Prosodic Labels for Speech Synthesis," *Computer, Speech and Language*, 1996, **10**, pp. 155-185.

[47] Ross, K. and M. Ostendorf, "A Dynamical System Model for Generating Fundamental Frequency for Speech Synthesis," *IEEE Trans. on Speech and Audio Processing*, 1999, 7(3), pp. 295-309.

[48] Sagisaka, Y., W.N. Campbell, and N. Higuchi, *Computing Prosody*, 1997, Springer-Verlag.

[49] Santen, J.V., "Contextual Effects on Vowel Duration," *Speech Communication*, 1992, **11**(6), pp. 513-546.

[50] Selting, M., *Prosodie im Gespräch*, 1995, Max Niemeyer Verlag.

[51] Shattuck-Hufnagel, S. and M. Ostendorf, "Stress Shift and Early Pitch Accent Placement in Lexical Items in American English," *Journal of Phonetics*, 1994, **22**, pp. 357-388.

[52] Shen, X.-n.S., *The Prosody of Mandarin Chinese*, 1990, Berkeley, University of California Press.

[53] Sheridan, T., *Lectures on the Art of Reading*, 3rd ed, 1787, London, Dodsley.

[54] Silverman, K., *The Structure and Processing of Fundamental Frequency Contours*, Ph.D. Thesis, 1987, University of Cambridge, Cambridge, UK.

[55] Silverman, K., "ToBI: A Standard for Labeling English Prosody," *Int. Conf. on Spoken Language Processing*, 1992, Banff, Canada, pp. 867-870.

[56] Sokal, A.D., "Transgressing the Boundaries: Towards a Transformative Hermeneutics of Quantum Gravity," *Social Text*, 1996, **46/47**, pp. 217-252.

[57] Sonntag, G., T. Portele, and B. Heuft, "Prosody Generation with a Neural Network: Weighing the Importance of Input Parameters," *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, 1997, pp. 930-934.

[58] Steedman, M., "Information Structure and the Syntax-Phonology Interface," *Linguistic Inquiry*, 2000.

[59] Stevens, K.N., "Control Parameters for Synthesis by Rule," *Proc. of the ESCA Tutorial Day on Speech Synthesis*, 1990, pp. 27-37.

[60] Taylor, P.A., "The Tilt Intonation Model," *Proc. Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[61] van Santen, J., "Assignment of Segmental Duration in Text-to-Speech Synthesis," *Computer Speech and Language*, 1994, **8**, pp. 95-128.

[62] van Santen, J., "Segmental Duration and Speech Timing" in *Computing Prosody*, Y. Sagisaka, N. Campbell, and N. Higuchi, eds., 1997, New York, Springer, pp. 225-250.

[63] van Santen, J. and J. Hirschberg, "Segmental Effects of Timing and Height of Pitch Contours," *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, pp. 719-722.

[64] Vanderslice, R.L., *Synthetic Elocution: Considerations in Automatic Orthographic-to-Phonetic Conversion of English with Special Reference to Prosody*, PhD Thesis, 1968, UCLA, Los Angeles.

[65] W3C, *Speech Synthesis Markup Requirements for Voice Markup Languages*, 2000, http://www.w3.org/TR/voice-tts-reqs/.

[66] Yoshimura, T., *et al.*, "Simultaneous Modeling of Spectrum, Pitch and Duration in HMM-Based Speech Synthesis," *EuroSpeech*, 1999, Budapest, Hungary, pp. 2347-2350.

CHAPTER 16

Speech Synthesis

The speech synthesis module of a TTS system is the component that generates the waveform. The input of traditional speech synthesis components is a phonetic transcription with its associated prosody. The input can also include the original text with tags, as this may help in producing higher-quality speech.

Speech synthesis can be classified into three types according to the model used in the speech generation. *Articulatory synthesis*, described in Section 16.2.4, uses a physical model of speech production that includes all the articulators described in Chapter 2. *Formant synthesis* uses a source-filter model, where the filter is characterized by slowly varying formant frequencies; it is the subject of Section 16.2. *Concatenative synthesis* generates speech by concatenating speech segments and is described in Section 16.3. To allow more flexibility in concatenative synthesis, a number of prosody modification techniques are described in Sections 16.4 and 16.5. Finally, a guide to evaluating speech synthesis systems is included in Section 16.6.

Speech synthesis can also be classified according to the degree of manual intervention in the system design into *synthesis by rule* and *data-driven synthesis*. In the former, a set of manually derived rules is used to drive a synthesizer, and in the latter the synthesizer's parameters are obtained automatically from real speech data. Concatenative systems are, thus, data driven. Formant synthesizers have traditionally used synthesis by rule, since the evolution of formants in a formant synthesizer has been done with hand-derived rules. Nonetheless, formant transitions can also be trained from data, as we show in Section 16.2.3.

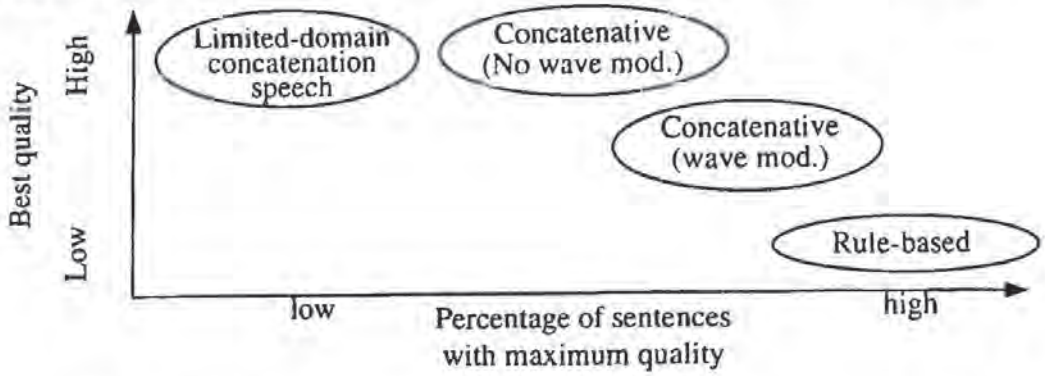


Figure 16.1 Quality and task-independence in speech synthesis approaches.

16.1. ATTRIBUTES OF SPEECH SYNTHESIS

The most important attribute of a speech synthesis system is the quality of its output speech. It is often the case that a single system can sound beautiful on one sentence and terrible on the next. For that reason we need to consider the quality of the best sentences and the percentage of sentences for which such quality is achieved. This tradeoff is illustrated in Figure 16.1, where we compare four different families of speech generation approaches:

- *Limited-domain waveform concatenation.* For a given limited domain, this approach can generate very high quality speech with only a small number of recorded segments. Such an approach, used in most interactive voice response systems, cannot synthesize arbitrary text. Many concept-to-speech systems, described in Chapter 17, use this approach.
- *Concatenative synthesis with no waveform modification.* Unlike the previous approach, these systems can synthesize speech from arbitrary text. They can achieve good quality on a large set of sentences, but the quality can be mediocre for many other sentences where poor concatenations take place.

- *Concatenative systems with waveform modification.* These systems have more flexibility in selecting the speech segments to concatenate because the waveforms can be modified to allow for a better prosody match. This means that the number of sentences with mediocre quality is lower than in the case where no prosody modification is allowed. On the other hand, replacing natural with synthetic prosody can hurt the overall quality. In addition, the prosody modification process also degrades the overall quality.
- *Rule-based systems.* Such systems tend to sound uniformly across different sentences, albeit with quality lower than the best quality obtained in the systems above.

Best-quality and quality variability are possibly two of the most important attributes of a speech synthesis system, but not the only ones. Measuring quality, difficult to do in an objective way, is the main subject of Section 16.6. Other attributes of a speech synthesis system include:

- *Delay.* The time it takes for the synthesizer to start speaking is important for interactive applications and should be less than 200 ms. This delay is composed of the algorithmic delays of the front end and of the speech synthesis module, as well as the computation involved.
- *Memory resources.* Rule-based synthesizers require, on the average, less than 200 KB, so they are a widely used option whenever memory is at a premium. However, required RAM can be an issue for concatenative systems, some of which may require over 100 MB of storage.
- *CPU resources.* With current CPUs, processing time is typically not an issue, unless many channels need to run in the same CPU. Nonetheless, some concatenative synthesizers may require a large amount of computation when searching for the optimal sequence.
- *Variable speed.* Some applications may require the speech synthesis module to generate variable speed, particularly fast speech. This is widely used by blind people who need TTS systems to obtain their information and can accept fast speech because of the increased throughput. Fast speech is also useful when skimming material. Concatenative systems that do not modify the waveform cannot achieve variable speed control, unless a large number of segments are recorded at different speeds.
- *Pitch control.* Some spoken language systems require the output speech to have a specific pitch. This is the case if you want to generate voice for a song. Again, concatenative systems that do not modify the waveform cannot do this, unless a large number of speech segments are recorded at different pitch.

- *Voice characteristics.* Other spoken language systems require specific voices, such as that of a robot, that cannot be recorded naturally, or some, such as monotones, that are tedious to record. Since rule-based systems are so flexible, they are able to do many such modifications.

The approaches described in this chapter assume as input a phonetic string, durations, a pitch contour, and possibly volume. Pauses are signaled by the default phoneme SIL with its corresponding duration. If the parsed text is available, it is possible to do even better in a concatenative system by conducting a matching with all the available information.

16.2. FORMANT SPEECH SYNTHESIS

As discussed in Chapter 6, we can synthesize a stationary vowel by passing a glottal periodic waveform through a filter with the formant frequencies of the vocal tract. For the case of unvoiced speech we can use white random noise as the source instead. In practice, speech signals are not stationary, and we thus need to change the pitch of the glottal source and the formant frequencies over time. The so-called *synthesis-by-rule* refers to a set of rules on how to modify the pitch, formant frequencies, and other parameters from one sound to another while maintaining the continuity present in physical systems like the human production system. Such a system is described in the block diagram of Figure 16.2.

In Section 16.2.1 we describe the second block of Figure 16.2, the formant synthesizer that generates a waveform from a set of parameters. In Section 16.2.2 we describe the first block of Figure 16.2, the set of rules that can generate such parameters. This approach was the one followed by Dennis Klatt and his colleagues [4, 30]. A data-driven approach to this first block is studied in Section 16.2.3. Finally, articulatory synthesis is the topic of Section 16.2.4.

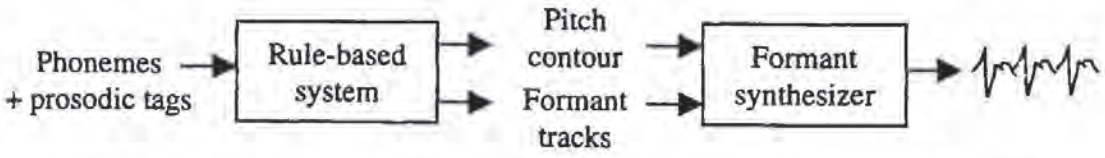


Figure 16.2 Block diagram of a synthesis-by-rule system. Pitch and formants are listed as the only parameters of the synthesizer for convenience. In practice, such system has about 40 parameters.

16.2.1. Waveform Generation from Formant Values

To be able to synthesize speech by rule, a simple model for the filter is needed. Most rule-based synthesizers use the so-called *formant synthesis*, which is derived from models of speech production. The model explicitly represents a number of formant resonances (from 2 to 6). A formant resonance can be implemented (see Chapter 6) with a second-order IIR filter

$$H_i(z) = \frac{1}{1 - 2e^{-\pi b_i} \cos(2\pi f_i) z^{-1} + e^{-2\pi b_i} z^{-2}} \tag{16.1}$$

with $f_i = F_i / F_s$ and $b_i = B_i / F_s$, where F_i, B_i, and F_s are the formant's center frequency, formant's bandwidth, and sampling frequency, respectively, all in Hz. A filter with several resonances can be constructed by cascading several such second-order sections (*cascade model*) or by adding several such sections together (*parallel model*). Formant synthesizers typically use the parallel model to synthesize fricatives and stops and the cascade model for all voiced sounds.

Unlike the cascade model, the parallel model requires gains to be specified for each second-order section, which often are chosen proportional to the formant's frequency and inversely proportional to the formant's bandwidth. The cascade model results in an all-pole filter, whereas the parallel model has zeros in addition to poles. Such a combination is shown in Figure 16.3, where R1 through R6 are the resonances 1 to 6 and each one represents a second-order IIR filter like that in Eq. (16.1). RNP represents the nasal resonance, and RNZ is an FIR filter with the nasal zero. A1 through AB are the gains for each filter, used for the parallel model. Switch SW controls whether the cascade model or parallel model is used.

For voiced sounds the excitation model consists of an impulse train driving a low-pass filter RGP and then a bandpass filter created by the parallel of RGZ and RGS. For unvoiced sounds the excitation consists of white noise driving a low-pass filter LPF. The excitation for voiced fricatives is a combination of the two sources. In practice, this mixed excitation is used for all voiced sounds to add some breathiness. Klatt [30] showed that this model could reproduce quite faithfully a natural recording if the parameters had been manually selected.

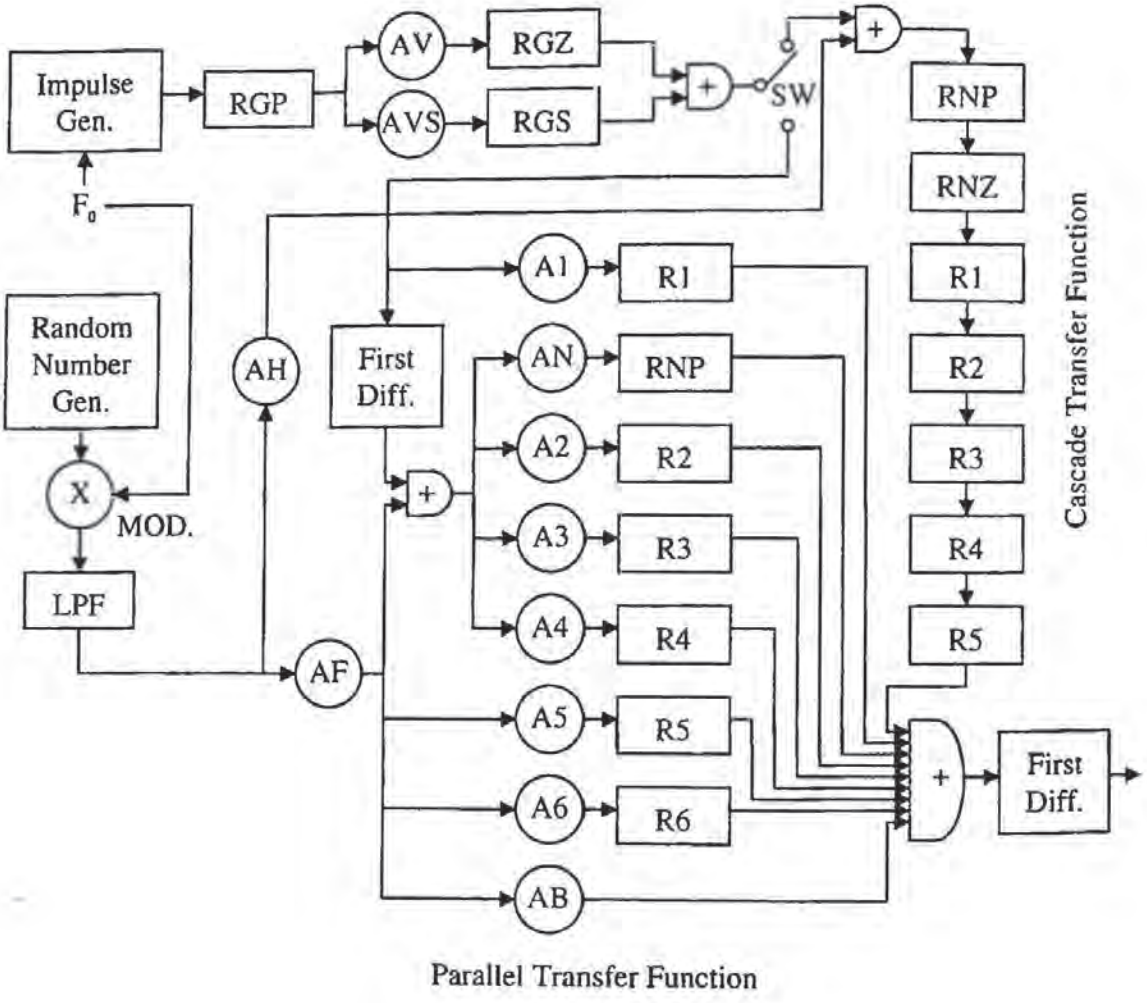


Figure 16.3 Block diagram of the Klatt formant synthesizer (after Allen [4]).

The parameter names and their minimum and maximum values are listed in Table 16.1, where the switch SW can be in voiced (V) or consonant (C) mode. For example, in Figure 16.3, R2 is the resonator corresponding to the second formant, whose center frequency F2 and bandwidth B2 are given in Table 16.1. In addition to the six resonances associated to the six formants, there are other resonances: RGP, RGZ, RGS, RNP, and RNZ. Other source models are also possible [43].

Table 16.1 Parameter values for Klatt's cascade/parallel formant synthesizer with the parameter symbol, full name, minimum, maximum, and typical values (after Allen [4]).

N	Symbol	Name	Min	Max	Typ
1	AV	Amplitude of voicing (dB)	0	80	0
2	AF	Amplitude of frication (dB)	0	80	0
3	AH	Amplitude of aspiration (dB)	0	80	0
4	AVS	Amplitude of sinusoidal voicing (dB)	0	80	0
5	F0	Fundamental frequency (Hz)	0	500	0
6	F1	First formant frequency (Hz)	150	900	500
7	F2	Second formant frequency (Hz)	500	2500	1500
8	F3	Third formant frequency (Hz)	1300	3500	2500
9	F4	Fourth formant frequency (Hz)	2500	4500	3500
10	FNZ	Nasal zero frequency (Hz)	200	700	250
11	AN	Nasal formant amplitude (Hz)	0	80	0
12	A1	First formant amplitude (Hz)	0	80	0
13	A2	Second formant amplitude (Hz)	0	0	0
14	A3	Third formant amplitude (Hz)	0	80	0
15	A4	Fourth formant amplitude (Hz)	0	80	0
16	A5	Fifth formant amplitude (Hz)	0	80	0
17	A6	Sixth formant amplitude (Hz)	0	80	0
18	AB	Bypass path amplitude (Hz)	0	80	0
19	B1	First formant bandwidth (Hz)	40	500	50
20	B2	Second formant bandwidth (Hz)	40	500	70
21	B3	Third formant bandwidth (Hz)	40	500	110
22	SW	Cascade/parallel switch	0	1	0
23	FGP	Glottal resonator 1 frequency (Hz)	0	600	0
24	BGP	Glottal resonator 1 bandwidth (Hz)	100	2000	100
25	FGZ	Glottal zero frequency (Hz)	0	500	1500
26	BGZ	Glottal zero bandwidth (Hz)	100	9000	6000
27	B4	Fourth formant bandwidth (Hz)	100	500	250
28	F5	Fifth formant frequency (Hz)	3500	4900	3850
29	B5	Fifth formant bandwidth (Hz)	150	700	200
30	F6	Sixth formant frequency (Hz)	4000	4999	4900
31	B6	Sixth formant bandwidth (Hz)	200	2000	100
32	FNP	Nasal pole frequency (Hz)	200	500	250
33	BNP	Nasal pole bandwidth (Hz)	50	500	100
34	BNZ	Nasal zero bandwidth (Hz)	50	500	100
35	BGS	Glottal resonator 2 bandwidth (Hz)	100	1000	200
36	SR	Sampling rate (Hz)	500	20000	10000
37	NWS	Number of waveform samples per chunk	1	200	50
38	G0	Overall gain control (dB)	0	80	48
39	NFC	Number of cascaded formants	4	6	5

16.2.2. Formant Generation by Rule

As described in Chapter 2, formants are one of the main features of vowels. Because of the physical limitations of the vocal tract, formants do not change abruptly with time. Rule-based formant synthesizers enforce this by generating continuous values for $f_i[n]$ and $b_i[n]$, typically every 5–10 milliseconds. Continuous values can be implemented through the above structures if a lattice filter is used, because it allows its reflection coefficients to vary at every sample (see Chapter 6). In practice, the values can be fixed within a frame as long as frames are smaller than 5 ms.

Rules on how to generate formant trajectories from a phonetic string are based on the *locus theory* of speech production. The locus theory specifies that formant frequencies within a phoneme tend to reach a stationary value called the *target*. Targets for formant frequencies and bandwidths for a male speaker are shown in Table 16.2 (nonvocalic segments) and Table 16.3 (vocalic segments). This target is reached if either the phoneme is sufficiently long or the previous phoneme's target is close to the current phoneme's target. The maximum slope at which the formants move is dominated by the speed of the articulators, determined by physical constraints. Since each formant is primarily caused by the position of a given articulator, formants caused by the body of the tongue do not vary as rapidly as formants caused by the tip of the tongue or the lips. Thus, rule-based systems store targets for each phoneme as well as maximum allowable slopes and transition times.

For example, a transition between a vowel and a sonorant can follow the rule shown in Figure 16.4 with a_1 being the target of unit 1 and a_2 the target of unit 2. The values of T_{cb} and T_{cf} are 40 and 80 ms, respectively, and $a_b = a_2 + 0.75(a_1 - a_2)$. The time $T_{cb} + T_{cf}$ specifies how rapid the transition is. While linear interpolation could be used, a_b and the ratio T_{cb}/T_{cf} are sometimes used to further refine the shape of the formant transition.

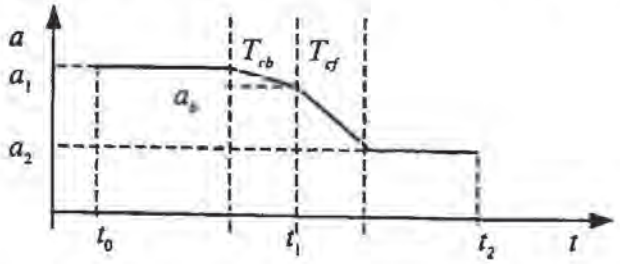


Figure 16.4 Transition between two vowels in a formant synthesizer.

Other rules can allow a discontinuity, for example, when a transition out of an unvoiced segment takes place. To improve naturalness, all these parameters can be made dependent on the immediate phonetic context.

Table 16.2 Targets used in the Klatt synthesizer: formant frequencies and bandwidths for non-vocalic segments of a male speaker. Note that in addition to the phoneme set used in Chapter 2, there are several additional phonetic segments here such as *axp, dx, el, em, en, gp, hx, kp, lx, qq, rx, tq, wh*, that allow more control (after Allen [4]).

	F1	F2	F3	B1	B2	B3
axp	430	1500	2500	120	60	120
b	200	900	2100	65	90	125
ch	300	1700	2400	200	110	270
d	200	1400	2700	70	115	180
dh	300	1150	2700	60	95	185
dx	200	1600	2700	120	140	250
el	450	800	2850	65	60	80
em	200	900	2100	120	60	70
en	200	1600	2700	120	70	110
ff	400	1130	2100	225	120	175
g	250	1600	1900	70	145	190
gp	200	1950	2800	120	140	250
h	450	1450	2450	300	160	300
hx	450	1450	2450	200	120	200
j	200	1700	2400	50	110	270
k	350	1600	1900	280	220	250
kp	300	1950	2800	150	140	250
l	330	1050	2800	50	100	280
lx	450	800	2850	65	60	80
m	480	1050	2100	40	175	120
ng	480	1600	2050	160	150	100
n	480	1400	2700	40	300	260
p	300	900	2100	300	190	185
qq	400	1400	2450	120	140	250
r	330	1060	1380	70	100	120
rx	460	1260	1560	60	60	70
sh	400	1650	2400	200	110	280
sil	400	1400	2400	120	140	250
s	400	1400	2700	200	95	220
th	400	1150	2700	225	95	200
tq	200	1400	2700	120	140	250
t	300	1400	2700	300	180	220
v	300	1130	2100	55	95	125
wh	330	600	2100	150	60	60
w	285	610	2150	50	80	60
y	240	2070	3020	40	250	500
zh	300	1650	2400	220	140	250
z	300	1400	2700	70	85	190

Table 16.3 Targets used in the Klatt synthesizer: formant frequencies and bandwidths for vocalic segments of a male speaker. Note that in addition to the phoneme set used in Chapter 2, there are several additional phonetic segments here such as *axr, exr, ix, ixr, oxr, uxr, yu* that allow more control (after Allen [4]).

	F1	F2	F3	B1	B2	B3
aa	700	1220	2600	130	70	160
ae	620	1660	2430	70	130	300
ah	620	1220	2550	80	50	140
ao	600	990	2570	90	100	80
aw	640	1230	2550	80	70	110
ax	550	1260	2470	80	50	140
axr	680	1170	2380	60	60	110
ay	660	1200	2550	100	120	200
eh	530	1680	2500	60	90	200
er	470	1270	1540	100	60	110
exr	460	1650	2400	60	80	140
ey	480	1720	2520	70	100	200
ih	400	1800	2670	50	100	140
ix	420	1680	2520	50	100	140
ixr	320	1900	2900	70	80	120
iy	310	2200	2960	50	200	400
ow	540	1100	2300	80	70	70
oxr	550	820	2200	60	60	60
oy	550	960	2400	80	120	160
uh	450	1100	2350	80	100	80
uw	350	1250	2200	65	110	140
uxr	360	800	2000	60	60	80
yu	290	1900	2600	70	160	220

Klatt showed that for a given natural utterance, he could manually obtain a sequence of formant tracks $f_i[n]$ and $b_i[n]$, such that the synthesized utterance not only had good quality but also sounded very similar to the original. This shows that the formant synthesizer of Section 16.2.1 appears to be sufficient for generation. On the other hand, when the formant tracks are obtained automatically through rules such as that of Figure 16.4 and Table 16.2 and Table 16.3, the output speech does not sound that natural, and the voice does not resemble the voice of the original recording.

Formant synthesis is very flexible because it can generate intelligible speech with relatively few parameters (about 40). The use of context-dependent rules can improve the quality of the synthesizer at the expense of a great deal of manual tuning. The synthesized speech is, by design, smooth, although it may not resemble any given speaker and may not sound very natural.

Because of their flexibility, formant synthesizers can often generate many different voices and effects. While not as flexible, voice effects can also be produced in concatenative speech systems (see Section 16.5.3).

16.2.3. Data-Driven Formant Generation

While, in general, formant synthesis assumes the formant model of Section 16.2.1 driven by parameter values generated by rules, as in Section 16.2.2, data-driven methods to generate the formant values have also been proposed [3]. An HMM running in generation mode emits three formant frequencies and their bandwidths every 10 ms, and these values are used in a cascade formant synthesizer similar to that described in Section 16.2.1. Like the speech recognition counterparts, this HMM has many decision-tree context-dependent triphones and three states per triphone. A Gaussian distribution per state is used in this work. The baseline system uses a six-dimensional vector that includes the first three formant frequencies and their bandwidths. Initially it is assumed that the input to the synthesizer includes, in addition to the duration of each phoneme, the duration of each state. In this case, the maximum likelihood formant track is a sequence of the state means and, therefore, is discontinuous at state boundaries.

The key to obtaining a smooth formant track is to augment the feature vector with the corresponding delta formants and bandwidths (the difference between the feature at time t and that feature at time $t - 1$) to complete a twelve-dimensional vector. The maximum likelihood solution now entails solving a tridiagonal set of linear equations (see the discussion on statistical formant tracking in Chapter 6). The resulting formant track is smooth, as it balances formant values that are close to the state means with delta values that are also within the state means. In addition, the synthesized speech resembles that of the donor speaker. More details on the analysis and model training can be found in the formant tracking section of Chapter 6.

16.2.4. Articulatory Synthesis

Articulatory synthesis is another model that has been used to synthesize speech by rule, by using parameters that model the mechanical motions of the articulators and the resulting distributions of volume velocity and sound pressure in the lungs, larynx, and vocal and nasal tracts. Because the human speech production articulators do not have that many degrees of freedom, articulatory models often use as few as 15 parameters to drive a formant synthesizer.

The relationship between articulators and acoustics is many-to-one [17]. For example, a ventriloquist can produce speech sounds with very different articulator positions than those of normal speech. The *speech inversion* problem is therefore ill posed. By using the assumption that the articulators do not change rapidly over time, it is possible, however, to estimate

the vocal-tract area from formant frequencies [37]. In [8] the model uses five articulatory parameters: area of lip opening, constriction formed by the tongue blade, opening to the nasal cavities, average glottal area, and rate of active expansion or contraction of the vocal tract volume behind a constriction. These five parameters are augmented with the first four formant frequencies and F0.

Those area parameters can be obtained from real speech through X-rays and magnetic resonance imaging (MRI), though positioning such sensors in the vocal tract alters the way speech is produced (such as the sensors in the lips) and impedes completely natural sounds. The relationship between articulatory parameters and acoustic values has typically been done using a nonlinear mapping such as a neural network or a codebook.

While one day this may be the best way to synthesize speech, the state-of-the-art in articulatory synthesis does not generate speech with quality comparable to that of formant or concatenative systems.

16.3. CONCATENATIVE SPEECH SYNTHESIS

While state-of-the-art synthesis by rule is quite intelligible, it sounds unnatural, because it is very difficult to capture all the nuances of natural speech in a small set of manually derived rules. In *concatenative synthesis*, a speech segment is synthesized by simply playing back a waveform with matching phoneme string. An utterance is synthesized by concatenating together several speech fragments. The beauty of this approach is that unlike synthesis-by-rule, it requires neither rules nor manual tuning. Moreover, each segment is completely natural, so we should expect very natural output.

Unfortunately, this is equivalent to assembling an automobile with parts of different colors: each part is very good yet there is a color discontinuity from part to part that makes the whole automobile unacceptable. Speech segments are greatly affected by coarticulation [42], so if we concatenate two speech segments that were not adjacent to each other, there can be spectral or prosodic discontinuities. Spectral discontinuities occur when the formants at the concatenation point do not match. Prosodic discontinuities occur when the pitch at the concatenation point does not match. A listener rates as poor any synthetic speech that contains large discontinuities, even if each segment is very natural.

Thus, when designing a concatenative speech synthesis system we need to address the following issues:

1. What type of speech segment to use? We can use diphones, syllables, phonemes, words, phrases, etc.
2. How to design the acoustic inventory, or set of speech segments, from a set of recordings? This includes excising the speech segments from the set of recordings as well as deciding how many are necessary. This is similar to the training problem in speech recognition.
3. How to select the best string of speech segments from a given library of segments, and given a phonetic string and its prosody? There may be several

strings of speech segments that produce the same phonetic string and prosody. This is similar to the search problem in speech recognition.

4. How to alter the prosody of a speech segment to best match the desired output prosody. This is the topic of Section 16.4.

Generally, these concatenative systems suffer from great variability in quality: often they can offer excellent quality in one sentence and terrible quality in the next one. If enough good units are available, a given test utterance can sound almost as good as a recorded utterance. However, if several discontinuities occur, the synthesized utterance can have very poor quality. While synthesizing arbitrary text is still a challenge with these techniques, for restrictive domains this approach can yield excellent quality. We examine all these issues in the following sections.

We define *unit* as an abstract representation of a speech segment, such as its phonetic label, whereas we use *instance* as a speech segment from an utterance that belongs to the same unit. Thus, a system can keep several instances of a given unit to select among them to better reduce the discontinuities at the boundaries. This abstract representation consists of the unit's phonetic transcription at the minimum, in such a way that the concatenation of a set of units matches the target phonetic string. In addition to the phonetic string, this representation can often include prosodic information.

16.3.1. Choice of Unit

A number of units have been used in the field, including context-independent phonemes, diphones, context-dependent phonemes, subphonetic units, syllables, words, and phrases. A compilation of unit types for English is shown in Table 16.4 with their coverage in Figure 16.5.

Table 16.4 Unit types in English assuming a phone set of 42 phonemes. Longer units produce higher quality at the expense of more storage. The number of units is generally below the absolute maximum in theory, i.e., out of the $42^3 = 74{,}088$ possible triphones, only about 30,000 occur in practice.

Unit length	Unit type	#Units	Quality
Short	Phoneme	42	Low
	Diphone	~1500	
	Triphone	~30K	
	Demisyllable	~2000	
	Syllable	~15K	
	Word	100K–1.5M	
	Phrase	∞	High
Long	Sentence	∞	

The issues in choosing appropriate units for synthesis are similar to those in choosing units for speech recognition (described in Chapter 9):

- The unit should lead to *low concatenation distortion.* A simple way of minimizing this distortion is to have fewer concatenations and thus use long units such as words, phrases or even sentences. But since some concatenations are unavoidable, we also want to use units that naturally lead to "small" discontinuities at the concatenation points. For example, it has been observed that spectral discontinuities at vowels are much more noticeable than at fricatives, or that a discontinuity within a syllable is more perceptually noticeable than a discontinuity across syllable boundaries, and similarly for within-word and across-word discontinuities [55]. Having several instances per unit is an alternative to long units that allows the choice of instances with low concatenation distortion.
- The unit should lead to *low prosodic distortion.* While it is not crucial to have units with the same prosody as the desired target, replacing a unit with a rising pitch with another with a falling pitch may result in an unnatural sentence. Altering the pitch and/or duration of a unit is possible (see Section 16.4) at the expense of additional distortion.
- The unit should be *generalizable*, if unrestricted text-to-speech is required. If we choose words or phrases as our units, we cannot synthesize arbitrary speech from text, because it's almost guaranteed that the text will contain words not in our inventory. As an example, the use of arbitrarily long units in such a way that no concatenation between voiced sounds occurs by cutting at obstruents results in low concatenation distortion but it is shown [47] that over 180,000 such units would be needed to cover 75% of a random corpus. The longer the speech segments are, the more of them we need to be able to synthesize speech from arbitrary text. This generalization property is not needed if closed-domain synthesis is desired.
- The unit should be *trainable.* Our training data should be sufficient to estimate all our units. Since the training data is usually limited, having fewer units leads to better trainability in general. So the use of words, phrases, or sentences may be prohibitive other than for closed-domain synthesis. The other units in Table 16.4 can be considered trainable depending on the limitations on the size of our acoustic inventory.

A practical challenge is how to balance these selection criteria. In this section we compare a number of units and point out their strengths and weaknesses.

16.3.1.1. Context-Independent Phonemes

The most straightforward unit is the phoneme. Having one instance of each phoneme, independent of the neighboring phonetic context, is very generalizable, since it allows us to gen-

erate every word/sentence. It is also very trainable and we could have a system that is very compact. For a language with N phonemes, say $N = 42$ for English, we would need only N unit instances. The problem is that using *context-independent* phones results in many audible discontinuities. Such a system is not intelligible.

16.3.1.2. Diphones

A type of subword unit that has been extensively used is the so-called *dyad* or *diphone* [41]. A diphone `s-ih` includes from the middle of the `s` phoneme to the middle of the `ih` phoneme, so diphones are, on the average, one phoneme long. The word `hello /hh ax l ow/` can be mapped into the diphone sequence: `/sil-hh/, /hh-ax/, /ax-l/, /l-ow/, /ow-sil/`. If our language has N phonemes, there are potentially N^2 diphones. In practice, many such diphones never occur in the language, so that a smaller number is sufficient. For example, the phonetic alphabet of Chapter 2 has 42 phonemes for English, and only about 1300 diphones are needed. Diphone units were among the first type of unit used in concatenative systems because they yield fairly good quality. While diphones retain the transitional information, there can be large distortions due to the difference in spectra between the stationary parts of two units obtained from different contexts. For example, there is no guarantee that the spectra of `/ax-l/` and `/l-ow/` will match at the junction point, since the instance `/ax-l/` could have been excised from a very different right context than `/ow/` or the instance `/l-ow/` could have been excised from a very different left context than `/ax/`.

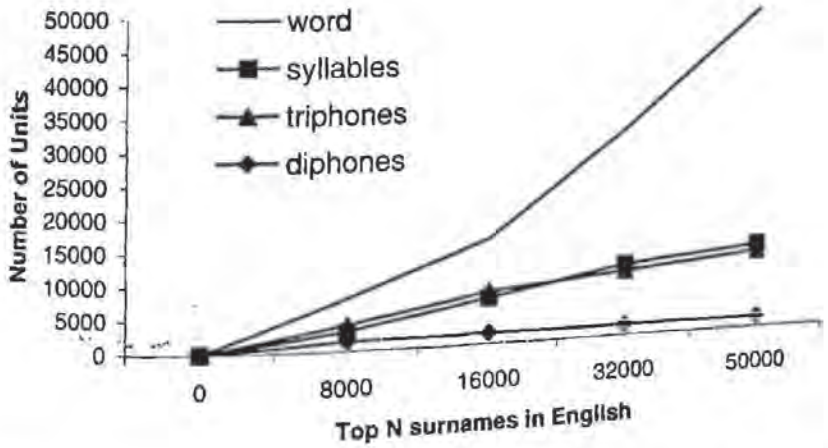


Figure 16.5 Coverage with different number of units displays the number of units of different types required to generate the top N surnames in the United States [34].

For this reason, many practical diphone systems are not purely diphone based: they do not store transitions between fricatives, or between fricatives and stops, while they store

longer units that have a high level of coarticulation [48]. If only one representative of a dyad is available, there are pitch discontinuities. Prosody modification techniques, such as those described in Section 16.4, can be applied to correct this problem. Otherwise many instances of each diphone are needed for good prosodic coverage. Diphones are trainable, generalizable and offer better quality than context-independent phones.

16.3.1.3. Context-Dependent Phoneme

Another subword unit used in the literature [24] is the *context-dependent* phoneme. If the context is limited to the immediate left and right phonemes, the unit is known as *triphone*. As in speech recognition, not all N^3 need to be stored, because not all combinations will occur in practice. For English, typically there can be in excess of 25,000 triphones: 12,000 within-word triphones and another 12,000 across-word triphones. Because of the increased number of units, more contextual variations can be accommodated this way. Drawing from experience in speech recognition, we know that many different contexts have a similar effect on the phoneme; thus, several triphones can be clustered together into a smaller number of *generalized triphones*, typically between 500 and 3000. All the clustering procedures described in Chapter 9 can be used here as well. In particular, decision-tree clustered phones have been successfully used. Because a larger number of units can be used, discontinuities can be smaller than in the case of diphones while making the best use of the available data. In addition to only considering the immediate left and right phonetic context, we could also add stress for the current phoneme and its left and right context, word-dependent phones (where phones in particular words are considered distinct context-dependent phones), quinphones (where two immediate left and right phones are used), and different prosodic patterns (pitch ranges and/or durations). As in speech recognition, clustered context-dependent triphones are trainable and generalizable.

Traversing the tree for a given phoneme is equivalent to following the answers for the branching nodes from root to leaves, which determines the clusters for similar context-dependent phones. The decision trees are generated automatically from the analysis database to obtain minimum within-unit distortion (or entropy) for each split. Therefore, one must be able to acquire a large inventory of context-dependent phone HMMs with a decent coverage of the contexts one wishes to model. All the context-dependent phone units can be well replaced by any other units within the same cluster. This method generalizes to contexts not seen in the training data, because the decision tree uses questions involving broad phonetic categories of neighboring contexts, yet provides detailed models for contexts that are represented in the database. Given the assumption that these clustering decision trees should be consistent across different speakers, the use of ample speaker-independent databases instead of limited speaker-dependent databases allows us to model more contexts as well as deeper trees to generate a high-quality TTS voice. These techniques also facilitate the creation of acoustic inventories with a scalable number of units that trade off size with quality. Thus, we can use questions (about the immediate left/right phonetic contexts, stress, pitch, duration,

word, etc.) in the decision-tree clustering methods of Chapter 4 to reduce all the possible combinations to a manageable number.

16.3.1.4. Subphonetic Unit

Subphonetic units, or senones, have also been used with some success [13]. Typically, each phoneme can be divided into three states, which are determined by running a speech recognition system in forced alignment mode. These states can also be context dependent and can also be clustered using decision trees like the context-dependent phonemes. The HMM state has proved to be more effective than the context-dependent phone in speech recognition, also trainable and generalizable, but for synthesis it means having more concatenations and thus possibly more discontinuities. If multiple instances per subphonetic unit are used, higher quality can be obtained.

A *half phone* goes either from the middle of a phone to the boundary between phones or from the boundary between phones to the middle of the phone. This unit offers more flexibility than a phone and a diphone and has been shown useful in systems that use multiple instances of the unit [7].

16.3.1.5. Syllable

It has been observed that discontinuities across syllables stand out more than discontinuities within syllables [55], so syllables are natural units. There are more than 10,000 syllables in English, depending on the exact definition of syllable, so even a context-independent syllable system needs to store at least as many if one instance per syllable is needed for full generalizability. There will still be spectral discontinuities, though hopefully not too noticeable. More than one instance per unit may be needed to account for varying acoustic contexts or varying prosodic patterns, particularly if no waveform modification is to be used.

16.3.1.6. Word and Phrase

The unit can be as large as a word or even a phrase. While using these long units can increase naturalness significantly, generalizability and trainability are poor, so that it is difficult to have all the instances desired to synthesize an output utterance. One advantage of using a word or longer unit over its decomposition in phonemes, as in the above units, is that there is no dependence on a phonetically transcribed dictionary. It is possible that the phoneme string associated to a word by our dictionary is not correct, or not fluent enough, so that using a whole-word model can solve this problem. Of course, the system may have a combination of all units: a set of the most frequent words, sentences, or phrases for best quality some percentage of the time, and some smaller units for full generalizability and trainability.

16.3.2. Optimal Unit String: The Decoding Process

The goal of the decoding process is to choose the optimal string of units for a given phonetic string that best matches the desired prosody. Sometimes there is only one possible string, so that this process is trivial, but in general there are several strings of units that result in the same phonetic string yet some of them sound better than others. The goal is to come up with an objective function that approximates this sound quality that allows us to select the best string. The quality of a unit string is typically dominated by spectral and pitch discontinuities at unit boundaries. Discontinuities can occur because of:

1. *Differences in phonetic contexts.* A speech unit was obtained from a different phonetic context than that of the target unit.
2. *Incorrect segmentation.* Such segmentation errors can cause spectral discontinuities even if they had the same phonetic context.
3. *Acoustic variability.* Units can have the same phonetic context and be properly segmented, but variability from one repetition to the next can cause small discontinuities. A unit spoken in fast speech is generally different from another in slow or normal speech. Different recording conditions (amplitude, microphone, sound card) can also cause spectral discontinuities.
4. *Different prosody.* Pitch discontinuity across unit boundaries is also a cause for degradation.

The severity of such discontinuities generally decreases as the number of units increases. More importantly, the prosody of the concatenation has, in general, no resemblance with the prosody specified by the TTS front-end unless we have several instances of each unit, each with a different prosody, or use a prosody modification algorithm (see Section 16.4).

16.3.2.1. Objective Function

Our goal is to come up with a numeric measurement for a concatenation of speech segments that correlates well with how well they sound. To do that we define unit cost and transition cost between two units.

Let θ be a speech segment with phonetic transcription $p = p(\theta)$. Let $\Theta = \{\theta_1, \theta_2, \cdots, \theta_N\}$ be a concatenation of N speech segments whose combined phonetic transcription is $P = \{p_1, p_2, \cdots, p_N\}$. P is a string of M phonemes, and since each segment has at least one phoneme, it holds that $M \geq N$.

For example, the phonetic string $P =$ "*hh ax l ow*" corresponding to the word *hello* has $M = 4$ phonemes and can be decomposed in $N = 4$ segments $\Theta_1 = \{\theta_1, \theta_2, \theta_3, \theta_4\}$, where $p(\theta_1) = /hh/$, $p(\theta_2) = /ax/$, $p(\theta_3) = /l/$, $p(\theta_4) = /ow/$, each segment being a phoneme. Or it can be decomposed into $N = 2$ segments $\Theta_2 = \{\theta_5, \theta_6\}$, where $p(\theta_5) = /hh\ ax/$,

$p(\theta_6)$ = /l ow/, so that each segment has two phonemes. There are 8 possible such decompositions for this example (in general there are 2^{M-1} possible decompositions[1]).

The distortion or cost function between the segment concatenation Θ and the target T can be expressed as a sum of the corresponding unit costs and transition costs [27, 46] as follows:

$$d(\Theta,T)=\sum_{j=1}^{N} d_u(\theta_j,T)+\sum_{j=1}^{N-1} d_t(\theta_j,\theta_{j+1}) \tag{16.2}$$

where $d_u(\theta_j,T)$ is the *unit cost* of using speech segment θ_j within target T and $d_t(\theta_j,\theta_{j+1})$ is the transition cost in concatenating speech segments θ_j and θ_{j+1}. The optimal speech segment sequence of units $\hat{\Theta}$ can be found as the one that minimizes the overall cost

$$\hat{\Theta}=\arg\min_{\Theta} d(\Theta,T) \tag{16.3}$$

over sequences with all possible numbers of units. Transition and unit costs are described in Sections 16.3.2.2 through 16.3.2.5.

Let's analyze the second term in the sum of Eq. (16.2), also shown in Figure 16.6. If all transition costs were identical, the word string with fewest units would have lowest distortion. In practice transition costs are different and, thus, the string with fewest units is not necessarily the best, though there is clearly a positive correlation.

When a large number of speech segments are available, finding the segment sequence with lowest cost is a search problem like those analyzed in Chapter 12. Often a Viterbi algorithm is needed to make this efficient.

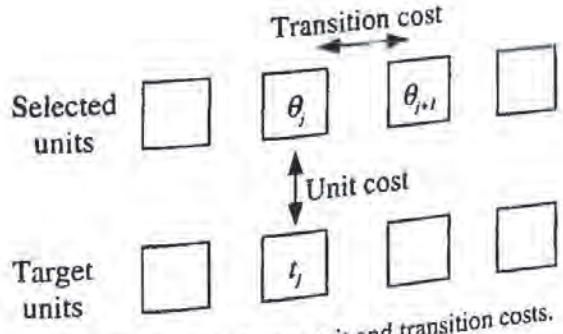


Figure 16.6 Tradeoff between unit and transition costs.

[1] This assumes that one instance per unit is available. If there are several instances per unit, the number of decompositions grows exponentially.

The art in this procedure is in the exact definition of both transition and unit costs, for which no standard has been defined that works best to date. In Sections 16.3.2.2 and 16.3.2.3 we present an approach for which both transition and unit costs are empirically set after perceptual experiments. Such a system is easy to build, and study of those costs gives insight into the perceptual effects.

Costs obtained using a data-driven criterion are described in Sections 16.3.2.4 and 16.3.2.5. While more complicated than that of empirical costs, this method addresses the shortcomings of the previous method. Finally, we need to estimate some weights to combine the different costs for spectrum and prosody, which can be done empirically or by regression [26].

16.3.2.2. Empirical Transition Cost

If spoken in succession, two speech segments have a zero transition cost. But, when they are excised from separate utterances, their concatenation can have varying degrees of naturalness. The transition cost incorporates two types of continuity measures: coarticulatory and prosodic.

An approximation to the prosodic continuity measure is to make it proportional to the absolute difference of the F0 or log F0 at the boundaries, if the boundary is voiced for both units. If we use the prosody modification techniques of Section 16.4, this cost could be set to a small value to reflect the fact that prosody modification is not a perfect process. More sophisticated cost functions can be used to account for prosody mismatches [10].

Regarding the coarticulatory effect, it has been empirically observed that a concatenation within a syllable is more perceptible than when the concatenation is at the syllable boundary. Yi [55] proposed an empirical cost matrix for the concatenation of two speech segments when that concatenation occurs within a syllable (Table 16.5) or at a syllable boundary (Table 16.6). Phonemes are grouped by manner of articulation: vowel/semivowels, fricatives, stops, and nasals. The rows represent the left side of the transition and the columns represent the right side, and NA represents a case that does not occur. These costs reflect perceptual ratings by human listeners to unit concatenations between different phonemes. Values of 10, 2000, 5000, 7500, and 10,000 were used to indicate different degrees of goodness from very good to very bad concatenations.

Table 16.5 Cost matrix for intrasyllable concatenations (after Yi [55]). The rows represent the left side of the transition and the columns represent the right side, and NA represents a case that does not occur.

	vowel	semivowel	nasal	obstruent	/h/
vowel	10,000	10,000	7500	10	NA
semivowel	10,000	7500	7500	10	NA
nasal	5000	10	NA	10	NA
/h/	5000	NA	NA	NA	NA
obstruent	10	10	10	10,000	NA

Table 16.6 Cost matrix for intersyllable concatenations (after Yi [55]). The rows represent the left side of the transition and the columns represent the right side, and NA represents a case that does not occur.

	vowel	semivowel	nasal	obstruent	/h/	silence
vowel	NA	7500	5000	10	5000	10
semivowel	7500	7500	2000	10	10	10
nasal	2000	10	10	10	10	10
obstruent	10	10	10	5000	10	10
/h/	NA	NA	NA	NA	NA	NA
silence	10	10	10	10	10	10

16.3.2.3. Empirical Unit Cost

The unit cost is generally a combination of the coarticulation cost and the prosodic cost. Prosodic mismatches can be made proportional to the F0 difference between the candidate unit and the target unit or set to a fixed low value if the prosody modification techniques of Section 16.4 are used.

A way of determining the cost associated with replacing a phonetic context with another was proposed by Yi [55], who empirically set cost matrices for phone classes by listening to concatenations where such contexts were replaced. These ad hoc values also bring some sense of where the coarticulation problems are. Replacing a vowel or semivowel by another with a context that has a different place of articulation or nasalization results in audible discontinuities. The rows represent the context class of the target phoneme and the columns represent the context class of the proposed unit. Table 16.7, Table 16.8, Table 16.9, Table 16.10, Table 16.11, and Table 16.12 include an empirical set of costs for such mismatches between the target's context and a candidate unit's context for the case of vowel/semivowels, fricatives, stops, and nasals. These costs reflect human listeners' perceptual ratings of speech units with an incorrect phonetic context. Values of 10, 100, 500, and 1000 were used to indicate very good, good, bad, and very bad units. These values are chosen to match the values for transition costs of Section 16.3.2.2.

Table 16.7 Unit coarticulation cost matrix (after Yi [55]) for left and right context replacements for vowels and semivowels.

	labial	alv/den/pal	velar	m	n	ng	front	back	none
labial	10	1000	1000	1000	1000	1000	1000	1000	1000
alv/den/pal	1000	10	1000	1000	1000	1000	1000	1000	1000
velar	1000	1000	10	1000	1000	1000	1000	1000	1000
m	1000	1000	1000	10	1000	1000	1000	1000	1000
n	1000	1000	1000	1000	10	1000	1000	1000	1000
ng	1000	1000	1000	1000	1000	10	1000	1000	1000
front	1000	1000	1000	1000	1000	1000	10	1000	1000
back	1000	1000	1000	1000	1000	1000	1000	10	1000
none	1000	1000	1000	1000	1000	1000	1000	1000	10

Table 16.8 Unit coarticulation cost matrix (after Yi [55]) for left and right context replacements for fricatives.

	retroflex	round	sonorant	other
retroflex	10	100	100	100
round	100	10	100	100
sonorant	100	100	10	100
other	100	100	100	10

Table 16.9 Unit coarticulation cost matrix (after Yi [55]) for left context replacements for stops.

	front	back	retroflex	round	other
front	10	10	10	10	10
back	10	10	10	10	10
retroflex	10	10	10	10	10
round	10	10	10	10	10
other	500	500	500	500	10

Table 16.10 Unit coarticulation cost matrix (after Yi [55]) for right context replacements for stops.

	front	back	retroflex	round	schwa	other
front	10	100	100	100	500	100
back	100	10	100	100	500	100
retroflex	100	100	10	100	500	100
round	100	100	100	10	500	100
schwa	500	500	500	500	10	500
other	100	100	100	100	500	10

Table 16.11 Unit coarticulation cost matrix (after Yi [55]) for left context replacements for nasals.

	obstruent	sonorant
obstruent	10	1000
sonorant	1000	10

Table 16.12 Unit coarticulation cost matrix (after Yi [55]) for right context replacements for nasals.

	voiced	unvoiced	sonorant
voiced	10	100	1000
unvoiced	100	10	1000
sonorant	1000	1000	10

16.3.2.4. Data-Driven Transition Cost

The empirical transition costs of Section 16.3.2.2 do not necessarily mean that a spectral discontinuity will take place, only that one is likely, and that if it occurs within a syllable it will have a larger perceptual effect than if it occurs across syllable boundaries. While that method can result in a good system, the cost is done independently of whether there is a true spectral discontinuity or not. Thus, it has been also proposed to use a measurement of the spectral discontinuity directly. This is often estimated as:

$$d_t(\theta_i,\theta_j)=\left|\mathbf{x}_i(l(\theta_i)-1)-\mathbf{x}_j(0)\right|^2 \tag{16.4}$$

the magnitude squared of the difference between the cepstrum at the last frame of θ_i and the first frame of θ_j. The quantity $l(\theta_i)$ denotes the number of frames of speech segment θ_i, and $\mathbf{x}_i(k)$ the cepstrum of segment θ_i at frame k.

This technique can effectively measure a spectral discontinuity in a region with slowly varying spectrum, but it can fail when one of the segments is a nasal, for example, for which a sharp spectral transition is expected and desired. A better way of measuring this discontinuity is shown in Figure 16.7, in which we measure the cepstral distance in an overlap region:[2] the last frame of segment 1 and the first frame before the beginning of segment 2:

$$d_t(\theta_i,\theta_j)=\left|\mathbf{x}_i(l(\theta_i)-1)-\mathbf{x}_j(-1)\right|^2 \tag{16.5}$$

When many speech segments are considered, a large number of cepstral distances need to be computed, which in turn may result in a slow process. To speed it up an approximation can be made where all possible cepstral vectors at the boundaries are vector quantized first, so that the distances between all codebook entries can be precomputed and stored in a table.

[2] This means extra frames need to be stored.

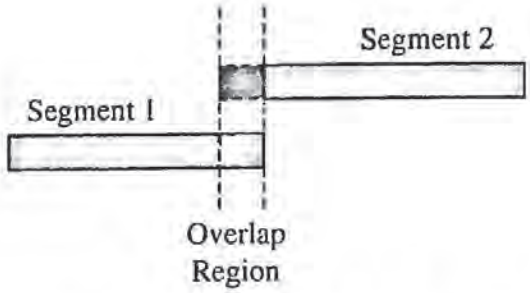


Figure 16.7 Measurement of the spectral discontinuity in the overlap region. The dark gray area is the speech region that precedes segment 2 and does not form part of segment 2. This area should match the last part of the segment 1.

A spectral discontinuity across, say, fricatives is perceptually not as important as if it happens across vowels [48]. For this reason, the cepstral distance described above does not correlate well with perceptual distances. To solve this problem, it is possible to combine both methods, for example by weighting the spectral/cepstral distance by different values.

Even if no spectral discontinuity is present, a phase discontinuity may take place. The pitch periodicity may be lost at the boundary. This can be generally solved by fine adjustment of the boundary using a correlation approach as described in Section 16.4. You need to keep in mind that such methods are not perfect.

16.3.2.5. Data-Driven Unit Cost

Spectral discontinuities across concatenations are often the result of using a speech segment with a different phonetic context than the target. One possibility is, then, to consider only speech segments where the phonetic contexts to the left and right match exactly. For example, if we use a phoneme as the basic speech segment, a perfect match would require on the order of at least 25,000 different segments. In this case, the coarticulation unit cost is zero if the target and candidate segment have the same phonetic context and infinite otherwise. When longer segments are desired, this number explodes exponentially. The problem with this approach is that it severely reduces the number of potential speech segments that can be used.

Generalized triphones, as described in Section 16.3.1.3, are used in [24]. In this approach, if the speech segments have the same generalized triphone contexts as the target utterance, the unit cost is zero, otherwise the cost is infinite. The technique allows us to use many more possible speech segments than the case above, yet it eliminates those speech segments that presumably have context mismatches that in turn lead to unnatural concatenations. When using a large training database, it was found that bringing the number of triphones from 25,000 down to about 2000 did not adversely impact the quality, whereas some degradation was perceived when using only 500 phoneme-length segments. Thus, this technique allows us to reduce the size of the speech segment inventory without severely degrading the voice quality.

If we set the number of decision-tree clustered context-dependent phonemes to be large, there will be fewer choices of long speech segments that match. For instance, in a system with 2000 generalized triphones, the phonetic context of the last phoneme of a long segment and the context of the target phoneme may be clustered together, whereas in a 3000-generalized-triphone system, both contexts may not be clustered together, so that the long segment cannot be used. This would be one example where using a larger number of generalized triphones hurts speech naturalness because the database of speech segments is limited. This problem could have been avoided if we didn't have to match generalized triphones and instead allowed context substitutions, yet penalized them with a corresponding cost. In the framework of decision-tree clustered context-dependent phonemes, this cost can be computed as the increase in entropy when those contexts are merged, using the methods described in Chapter 9. The larger the increase in entropy, the larger the penalty is when doing that context substitution between the candidate segment and the target segment. This approach gives more flexibility in the number of speech segments to be considered. In this case, there is a nonzero unit coarticulation cost associated with replacing one phonetic context with another.

Speech segments that have low HMM probability can be discarded, as they are probably not representative enough for that unit. Moreover, we can eliminate outliers: those units that have parameters too far away from the mean. Eliminating pitch outliers helps if prosody modification is to be done, as modifying pitch by more than a factor of 2 typically yields a decrease of quality, and by keeping units with average pitch, this is less likely to occur. Eliminating duration or amplitude outliers may signal an incorrect segmentation or a bad transcription [13].

16.3.3. Unit Inventory Design

The minimal procedure to obtain an acoustic inventory for a concatenative speech synthesizer consists of simply recording a number of utterances from a single speaker and labeling them with the corresponding text.

Since recording is often done in several sessions, it is important to maintain the recording conditions constant to avoid spectral or amplitude discontinuities caused by changes in recording conditions. The same microphone, room, and sound card should be used throughout all sessions [49].

Not all donor speakers are created equal. The choice of donor speaker can have a significant effect in voice quality (up to 0.3 MOS points on a 5-MOS scale) [7, 51, 52].

We can obtain higher-quality concatenative synthesis if the text read by the target speaker is representative of the text to appear in our application. This way we will be able to use longer units, and few concatenations will be needed.

Then the waveforms have to be segmented into phonemes, which is generally done with a speech recognition system operating in forced-alignment mode. Phonetic transcription, including alternate pronunciations, is generated automatically from text by the phonetic analysis module of Chapter 14. A large part of the inventory preparation includes checking

correspondence between the text and corresponding waveform. Possible transcription errors may be flagged by phonemes whose durations are too far away from the mean (outliers) [13, 24].

Once we have the segmented and labeled recordings, we can use them as our inventory, or create smaller inventories as subsets that trade off memory size with quality [21, 25]. A database with a large number of utterances is generally required to obtain high-quality synthesis. It is noteworthy to analyze whether we can reduce the size of our database while obtaining similar synthesis quality on a given set of utterances. To do this, we can measure the cost incurred when we use a subset of the units in the database to synthesize our training database. A greedy algorithm can be used that at each stage eliminates the speech unit that increases the total distortion the least, repeating the approach until the desired size is achieved. This is an iterative analysis-by-synthesis algorithm.

The above procedure can also be used to find the set of units that have lowest cost in synthesizing a given text. For efficiency, instead of a large training text, we could use representative information from such text corpus, like the word trigrams with their corresponding counts, as an approximation.

In concatenative systems, you need to store a large number of speech segments, which could be compressed using any of the speech coding techniques described in Chapter 7. Since many such coders encode a frame of speech based on the previous one, you need to store this context for every segment you want to encode if you are to use such systems.

16.4. PROSODIC MODIFICATION OF SPEECH

One problem of segment concatenation is that it doesn't generalize well to contexts not included in the training process, partly because prosodic variability is very large. There are techniques that allow us to modify the prosody of a unit to match the target prosody. These prosody-modification techniques degrade the quality of the synthetic speech, though the benefits are often greater than the distortion introduced by using them because of the added flexibility.

The objective of prosodic modification is to change the amplitude, duration, and pitch of a speech segment. Amplitude modification can be easily accomplished by direct multiplication, but duration and pitch changes are not so straightforward.

We first present OLA and SOLA, two algorithms to change the duration of a speech segment. Then we introduce PSOLA, a variant of the above that allows for pitch modification as well.

16.4.1. Synchronous Overlap and Add (SOLA)

Time-scale modification of speech is very useful, particularly voice compression, as it allows a user to listen to a voice mail or taped lecture in a fraction of the time taken by the original segment user to listen to information The overlap-and-add (OLA) technique [12]

shown in Figure 16.8 shows the analysis and synthesis windows used in the time compression. Given a Hanning window of length $2N$ and a compression factor of f, the analysis windows are spaced fN. Each analysis window multiplies the analysis signal, and at synthesis time they are overlapped and added together. The synthesis windows are spaced N samples apart. The use of windows such as Hanning allows perfect reconstruction when f equals 1.

In Figure 16.8, some of the signal appearance has been lost; note particularly some irregular pitch periods. To solve this problem, the synchronous overlap-and-add (SOLA) [45] allows for a flexible positioning of the analysis window by searching the location of the analysis window i around fNi in such a way that the overlap region had maximum correlation. The SOLA algorithm produces high-quality time compression. A mathematical formulation of PSOLA, an extension of both OLA and SOLA, is presented in Section 16.4.2.

While typically compression algorithms operate at a uniform rate, they have also been used in a nonuniform rate to take into account human perception, so that rapid transitions are compressed only slightly, steady sounds are compressed more, and pauses are compressed the most. It's reported in [11], that while uniform time compression can achieve a factor of 2.5 at most without degradation in intelligibility, nonuniform compression allows up to an average compression factor of 4.

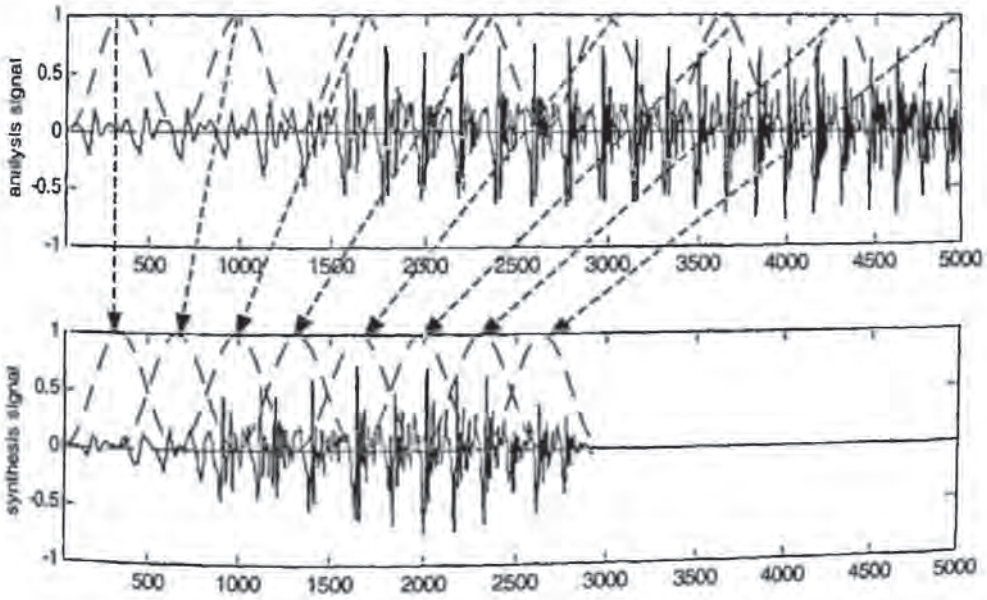


Figure 16.8 Overlap-and-add (OLA) method for time compression. Hanning windows of length $2N$, $N = 330$, are used to multiply the analysis signal, and resulting windowed signals are added. The analysis windows, spaced $2N$ samples, and the analysis signal $x[n]$ are shown on the top. The synthesis windows, spaced N samples apart, and the synthesis signal $y[n]$ are shown below. Time compression is uniform with a factor of 2. Pitch periodicity is somewhat lost, particularly around the fourth window.

16.4.2. Pitch Synchronous Overlap and Add (PSOLA)

Both OLA and SOLA do duration modification but cannot do pitch modification. On the other hand, they operate without knowledge of the signal's pitch. The most widely used method to do pitch modification is called pitch synchronous overlap and add (PSOLA) [38, 39], though to do so it requires knowledge of the signal's pitch. This process is illustrated in Figure 16.9.

Let's assume that our input signal $x[n]$ is voiced, so that it can be expressed as a function of pitch cycles $x_i[n]$

$$x[n] = \sum_{i=-\infty}^{\infty} x_i[n - t_a[i]] \tag{16.6}$$

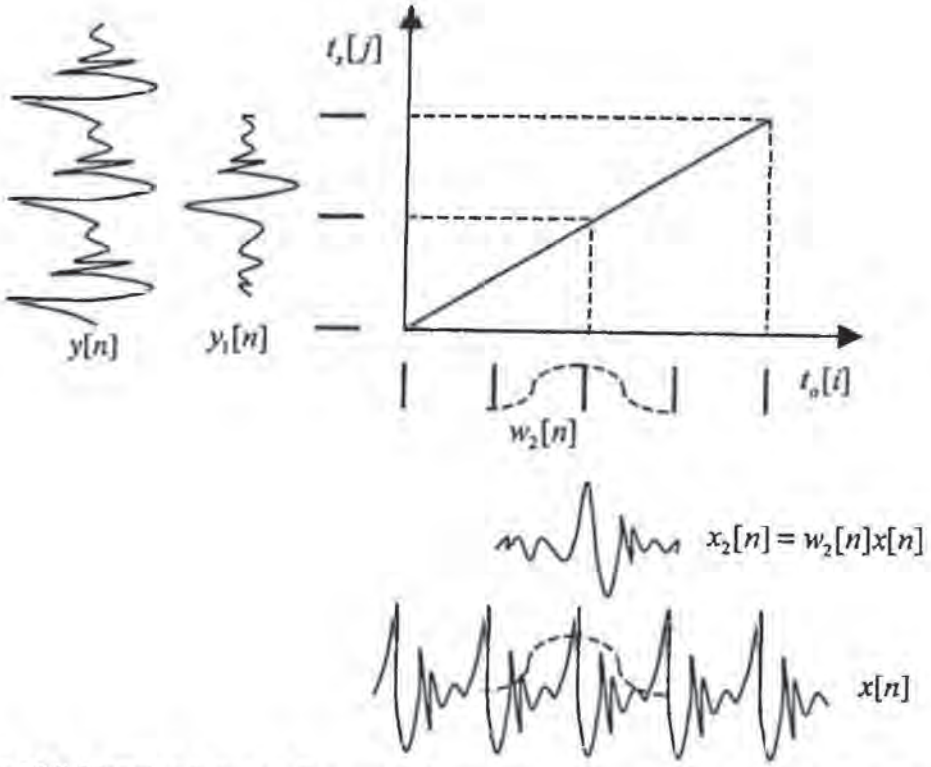


Figure 16.9 Mapping between five analysis epochs $t_a[i]$ and three synthesis epochs $t_s[j]$. Duration has been shortened by 40% and pitch period increased by 60%. Pitch cycle $x_2[n]$ is the product of the analysis window $w_2[n]$, in dotted line, with the analysis signal $x[n]$, which is aligned with analysis epochs $t_a[i]$. In this case, synthesis pitch cycle $y_1[n]$ equals $x_2[n]$ and also $y_0[n] = x_0[n]$ and $y_2[n] = x_3[n]$. Pitch is constant over time in this case.

where $t_a[i]$ are the epochs of the signal, so that the difference between adjacent epochs $P_a[i] = t_a[i] - t_a[i-1]$ is the pitch period at time $t_a[i]$ in samples. The pitch cycle is a windowed version of the input

$$x_i[n] = w_i[n]x[n] \tag{16.7}$$

which requires the windows $w_i[n]$ to meet the following condition:

$$\sum_{i=-\infty}^{\infty} w_i[n - t_a[i]] = 1 \tag{16.8}$$

which can be accomplished with a Hanning window, or a trapezoidal window that spans two pitch periods.

Our goal is to synthesize a signal $y[n]$, which has the same spectral characteristics as $x[n]$ but with a different pitch and/or duration. To do this, we replace the analysis epoch sequence $t_a[i]$ with the synthesis epochs $t_s[j]$, and the analysis pitch cycles $x_i[n]$ with the synthesis pitch cycles $y_j[n]$:

$$y[n] = \sum_{j=-\infty}^{\infty} y_j[n - t_s[j]] \tag{16.9}$$

The synthesis epochs are computed so as to meet a specified duration and pitch contour, as shown in Figure 16.9. This is equivalent to an impulse train with variable spacing driving a time-varying filter $x_i[n]$ which is known for $t = t_a[i]$, as shown in Figure 16.10. The synthesis pitch cycle $y_j[n]$ is obtained via a mapping from the closest corresponding analysis pitch cycle $x_i[n]$. In the following sections we detail how to calculate the synthesis epochs and the synthesis pitch-cycle waveforms.

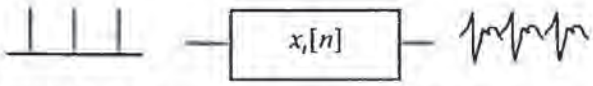


Figure 16.10 PSOLA technique as an impulse train driving a time-varying filter.

The term overlap-and-add derives from the fact that we use overlapping windows that we add together. The pitch-synchronous aspect comes from the fact that the windows are spaced a pitch period apart and are two pitch periods long. As you can see from Figure 16.9, the synthesis waveform has a larger pitch period than the analysis waveform and is shorter in duration.

For unvoiced speech, a set of epochs that are uniformly spaced works well in practice, as long as the spacing is smaller than 10 ms. If the segment needs to be stretched in such a way that these characteristic waveforms are repeated, an artificial periodicity would appear. To avoid this, the characteristic waveform that was to be repeated is flipped in time [38].

This approach is remarkably simple, yet it leads to high-quality prosody modification, as long as the voiced/unvoiced decision is correct and the epoch sequence is accurate.

To do prosody modification, the PSOLA approach requires keeping the waveform of the speech segment and its corresponding set of epochs, or time marks. As you can see from Eq. (16.6), if no prosody modification is done, the original signal is recovered exactly.

16.4.3. Spectral Behavior of PSOLA

Let's analyze why this simple technique works and how. To do that let's consider the case of a speech signal $x[n]$ that is exactly periodic with period T_0 and can be created by passing an impulse train through a filter with impulse response $s[n]$:

$$x[n] = s[n] * \sum_{i=-\infty}^{\infty} \delta[n - iT_0] = \sum_{i=-\infty}^{\infty} s[n - iT_0] \tag{16.10}$$

If we know the impulse response $s[n]$, then we could change the pitch by changing T_0. The problem is how to estimate it from $x[n]$. Let's assume we want to build an estimate $\tilde{s}[n]$ by multiplying $x[n]$ by a window $w[n]$:

$$\tilde{s}[n] = w[n]x[n] \tag{16.11}$$

The Fourier transform of $x[n]$ in Eq. (16.10) is given by

$$X(\omega) = \frac{2\pi}{T_0} S(\omega) \sum_{k=0}^{T_0-1} \delta(\omega - k\omega_0) = \frac{2\pi}{T_0} \sum_{k=0}^{T_0-1} S(k\omega_0)\delta(\omega - k\omega_0) \tag{16.12}$$

where $\omega_0 = 2\pi / T_0$. The Fourier transform of $\tilde{s}[n]$ can be obtained using Eqs. (16.11) and (16.12):

$$\tilde{S}(\omega) = \frac{1}{2\pi} W(\omega) * X(\omega) = \sum_{k=0}^{T_0-1} S(k\omega_0) \frac{W(\omega - k\omega_0)}{T_0} \tag{16.13}$$

If the window $w[n]$ is pitch synchronous, a rectangular window with length T_0 or a Hanning window with length $2T_0$, for example, then the above estimate is exact at the harmonics, i.e., $\tilde{S}(k\omega_0) = S(k\omega_0)$, because the window leakage terms are zero at the harmonics. In-between harmonics, $\tilde{S}(\omega)$ is an interpolation using $W(\omega)$, the transfer function of the window. If we use a rectangular window, the values of $S(\omega)$ in between $S(k\omega_0)$ and $S((k+1)\omega_0)$ are not determined only by those two harmonics, because the leakage from the other harmonics is not negligible. The use of a Hanning window drastically attenuates this leakage, so the estimate of the spectral envelope is better. This is what PSOLA is doing: getting an estimate of the spectral envelope by using a pitch-synchronous window.

Since it is mathematically impossible to recover $S(\omega)$ for a periodic signal, it is reasonable to fill in the remaining values by interpolation with the main lobes of the transform of the window. This approach works particularly well if the harmonics form a dense sam-

pling of the spectral envelope, which is the case for male speakers. For female speakers, where the harmonics may be spaced far apart, the spectral envelope estimated by interpolating across harmonics could be far different from the real envelope.

16.4.4. Synthesis Epoch Calculation

In practice, we want to generate a set of synthesis epochs $t_s[j]$ given a target pitch period $P_s(t)$. If the desired pitch period $P_s(t) = P$ is constant, then the synthesis epochs are given by $t_s[j] = jP$.

In general the desired pitch period $P_s(t)$ is a function of time. Intuitively, we could compute $t_s[j+1]$ in terms of the previous epoch $t_s[j]$ and the pitch period at that time:

$$t_s[j+1] - t_s[j] = P_s(t_s[j]) \tag{16.14}$$

though this is an approximation, which happens to work well if $P_s(t)$ changes slowly over time.

Now we derive an exact equation, which also can help us understand pitch-scale and time-scale modifications of the next few sections. Epoch $t_s[j+1]$ can be computed so that the distance between adjacent epochs $t_s[j+1] - t_s[j]$ equals the average pitch period in the region $t_s[j] \le t < t_s[j+1]$ between them (see Figure 16.11). This can be done by the following expression

$$t_s[j+1] - t_s[j] = \frac{1}{t_s[j+1] - t_s[j]} \int_{t_s[j]}^{t_s[j+1]} P_s(t)\,dt \tag{16.15}$$

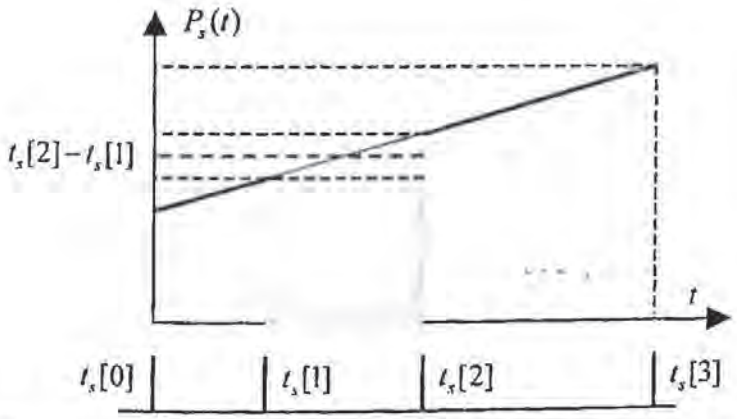


Figure 16.11 The desired pitch period $P_s(t)$ is a linearly increasing function of time such that the pitch period is doubled by the end of the segment. The four synthesis epochs $t_s[j]$ are computed to satisfy Eq. (16.15). In particular, $t_s[2]$ is computed such that $t_s[2] - t_s[1]$ equals the average pitch period in that region. Note that the growing spacing between epochs indicates that pitch is growing over time.

It is useful to consider the case of $P_s(t)$ being linear with t in that interval:

$$P_s(t) = P_s\left(t_s[j]\right) + b\left(t - t_s[j]\right) \tag{16.16}$$

so that the integral in Eq. (16.15) is given by

$$\int_{t_s[j]}^{t_s[j+1]} P_s(t)dt = \delta_j\left[P(t_s[i]) + b\frac{\delta_j}{2}\right] \tag{16.17}$$

where we have defined δ_j as

$$\delta_j = t_s[j+1] - t_s[j] \tag{16.18}$$

Inserting Eqs. (16.17) and (16.18) into Eq. (16.15), we obtain

$$\delta_j = P(t_s[i]) + b\frac{\delta_j}{2} \tag{16.19}$$

which, using Eq. (16.18), gives a solution for epoch $t_s[j+1]$ as

$$t_s[j+1] - t_s[j] = \delta_j = \frac{P_s(t_s[j])}{(1 - b/2)} \tag{16.20}$$

from the previous epoch $t_s[j]$, the target pitch at that epoch $P_s(t_s[j])$, and the slope b. We see that Eq. (16.14) is a good approximation to Eq. (16.20) if the slope b is small.

Evaluating Eq. (16.16) for $t_s[j+1]$ results in an expression for $P_s(t_s[j+1])$

$$P_s(t_s[j+1]) = P_s\left(t_s[j]\right) + b\left(t_s[j+1] - t_s[j]\right) \tag{16.21}$$

Equations (16.20) and (16.21) can be used iteratively. It is important to note that Eq. (16.20) requires $b < 2$ in order to obtain meaningful results, which fortunately is always the case in practice.

When synthesizing excitations for speech synthesis, it is convenient to specify the synthesis pitch period $P_s(t)$ as a piecewise linear function of time. In this case, Eq. (16.20) is still valid as long as $t_s[j+1]$ falls within the same linear segment. Otherwise, the integral in Eq. (16.17) has two components, and a second-order equation needs to be solved to obtain $t_s[j+1]$.

16.4.5. Pitch-Scale Modification Epoch Calculation

Sometimes, instead of generating an epoch sequence given by a function $P_s(t)$, we want to modify the epoch sequence of an analysis signal with epochs $t_a[i]$ by changing its pitch while maintaining its duration intact. This is called *pitch-scale* modification. To obtain the corresponding synthesis epochs, let's assume that the pitch period $P_a(t)$ of the analysis waveform at time t is constant and equals the difference between both epochs

$$P_a(t) = t_a[i+1] - t_a[i] \tag{16.22}$$

as seen in Figure 16.12.

The pitch period of the synthesis waveform $P_s(t)$ at the same time t now falls in between epochs j and $j+1$

$$t_s[j] \le t < t_s[j+1] \tag{16.23}$$

with $t_s[j]$ being the time instant of the j epoch of the synthesis waveform. Now, let's define a relationship between analysis and synthesis pitch periods

$$P_s(t) = \beta(t) P_a(t) \tag{16.24}$$

where $\beta(t)$ reflects the pitch-scale modification factor, which, in general, is a function of time. Following the derivation in Section 16.4.4, we compute the synthesis epoch $t_s[j+1]$ so that

$$t_s[j+1] - t_s[j] = \frac{1}{t_s[j+1] - t_s[j]} \int_{t_s[j]}^{t_s[j+1]} \beta(t) P_a(t)\, dt \tag{16.25}$$

which reflects the fact that the synthesis pitch period at time t is the average pitch period of the analysis waveform times the pitch-scale modification factor. Since $\beta(t)P_a(t)$ is piecewise linear, we can use the results of Section 16.4.4 to solve for $t_s[j+1]$. In general, it needs to be solved recursively, which results in a second-order equation if $\beta(t)$ is a constant or a linear function of t.

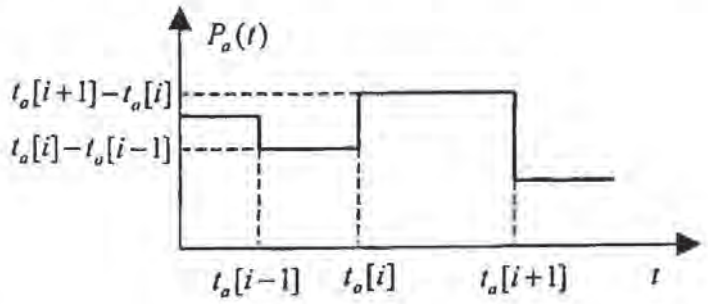


Figure 16.12 Pitch period of the analysis waveform as a function of time. It is a piecewise constant function of time.

16.4.6. Time-Scale Modification Epoch Calculation

Time-scale modification of speech involves changing the duration of a speech segment while maintaining its pitch intact. This can be realized by defining a mapping $t_s = D(t_a)$, a time-warping function, between the original signal and the modified signal. It is useful to define the duration modification rate $\alpha(t)$ from which the mapping function can be derived:

$$D(t) = \int_0^t \alpha(\tau) d\tau \tag{16.26}$$

Let's now assume that the duration modification rate $\alpha(t) = \alpha$ is constant, so that the mapping $D(t)$ in Eq. (16.26) is linear. If $\alpha > 1$, we are slowing down the speech, whereas if $\alpha < 1$, we are speeding it up. Let's consider time t in between epochs i and $i + 1$ so that $t_a[i] \le t < t_a[i+1]$:

$$\begin{aligned} &D(t_a[0]) = 0 \\ &D(t) = D(t_a[i]) + \alpha(t - t_a[i]) \end{aligned} \tag{16.27}$$

So that the relationship between analysis and synthesis pitch periods is given by

$$P_s(D(t)) = P_a(t) \tag{16.28}$$

To solve this it is useful to define a stream of virtual time instants $t'_a[j]$ in the analysis signal related to the synthesis time instants by

$$t_s[j] = D(t'_a[j]) = \alpha t'_a[j] \tag{16.29}$$

as shown in Figure 16.13.

Now we try to determine $t_s[j+1]$ such that $t_s[j+1] - t_s[j]$ is equal to the average pitch period in the original time signal between $t'_a[j]$ and $t'_a[j+1]$:

$$t_s[j+1] - t_s[j] = \frac{1}{t'_a[j+1] - t'_a[j]} \int_{t'_a[j]}^{t'_a[j+1]} P_a(t) dt \tag{16.30}$$

which, using Eq. (16.29), results in

$$t_s[j+1] - t_s[j] = \frac{\alpha}{t_s[j+1] - t_s[j]} \int_{t_s[j]/\alpha}^{t_s[j+1]/\alpha} P_a(t) dt \tag{16.31}$$

which again results in a second-order equation if $P_a(t)$ is piecewise constant or linear in t.

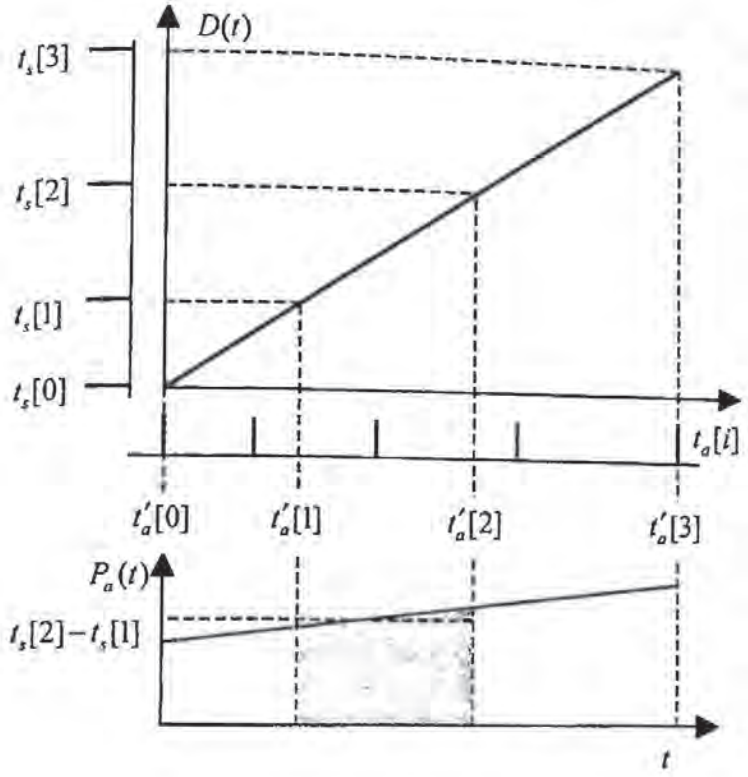


Figure 16.13 Time-scale modification of speech. The five analysis epochs $t_a[i]$ are shown in the x-axis and the four synthesis epochs $t_s[i]$ in the ordinate. Duration is shortened by 25% while maintaining the same pitch period. The corresponding virtual analysis epochs $t'_a[i]$ are obtained through the mapping $D(t)$, a linear transformation with $\alpha = 0.75$.

16.4.7. Pitch-Scale Time-Scale Epoch Calculation

The case of both pitch-scale and time-scale modification results in a combination of Eqs. (16.25) and (16.31):

$$t_s[j+1]-t_s[j]=\frac{\alpha}{t_s[j+1]-t_s[j]}\int_{t_s[j]/\alpha}^{t_s[j+1]/\alpha}\beta(t)P_a(t)dt \tag{16.32}$$

which again results in a second-order equation if $\beta(t)P_a(t)$ is piecewise constant or linear in t.

16.4.8. Waveform Mapping

The synthesis pitch waveforms can be computed through linear interpolation. Suppose that $t_a[i] \le t'_a[j] < t_a[i+1]$, then $y_j[n]$ is given by

$$y_j[n]=(1-\gamma_j)x_i[n]+\gamma_j x_{i+1}[n] \tag{16.33}$$

where γ_j is given by

$$\gamma_j=\frac{t'_a[j]-t_a[i]}{t_a[i+1]-t_a[i]} \tag{16.34}$$

Using this interpolation for voiced sounds results in smooth speech. For unvoiced speech, this interpolation results in a decrease of the amount of aspiration. Since smoothness is not a problem in those cases, the interpolation formula above is not used for unvoiced frames. A simplification of this linear interpolation consists of rounding γ_j to 0 or 1 and, thus, selecting the closest frame.

16.4.9. Epoch Detection

In the PSOLA approach, the analysis epochs $t_a[i]$ were assumed known. In practice this is not the case and we need to estimate them from the speech signal. There can be errors if the pitch period is not correctly estimated, which results in a rough, noisy voice quality. But estimating the epochs is not a trivial task, and this is the most sensitive part of achieving prosody modification in PSOLA.

Most pitch trackers attempt to determine F0 and not the epochs. From the $t_a[i]$ sequence it is easy to determine the pitch, since $P(t)=t_a[i+1]-t_a[i]$ for $t_a[i]<t<t_a[i+1]$. But from the pitch $P(t)$ the epoch placement is not uniquely determined, since the time origin is unspecified.

Common pitch tracking errors, such as pitch doubling, pitch halving, or errors in voiced/unvoiced decisions, result in rough speech. While manual pitch marking can result in accurate pitch marks, it is time consuming and error prone as well, so automatic methods have received a great deal of attention.

A method that attains very high accuracy has been proposed through the use of an *electroglottograph* (EGG) [32]. It consists of a pair of electrodes strapped around the neck at both sides of the larynx that measures the impedance of the larynx. Such a device, also called *laryngograph*, delivers a periodic signal when the vocal cords are vibrating and no signal otherwise. The pitch shape of a laryngograph signal is fairly stationary, which makes it relatively easy to determine the epochs from it (see Figure 16.14).

High-quality epoch extraction can be achieved by performing peak picking on the derivative of the laryngograph signal. Often, the derivative operation is accomplished by a first-order preemphasis filter $H[z]=1-\alpha z^{-1}$, with α being close to 1 (0.95 is a good choice).

In practice, the signal is preprocessed to filter out the low frequencies (lower than 100 Hz) and high frequencies (higher than 4 kHz). This can be done with rectangular window filters that are quite efficient and easy to implement. There is a significant amount of energy outside this band that does not contribute to epoch detection, yet it can complicate the process, as can be seen in Figure 16.14, so this bandpass filtering is quite important.

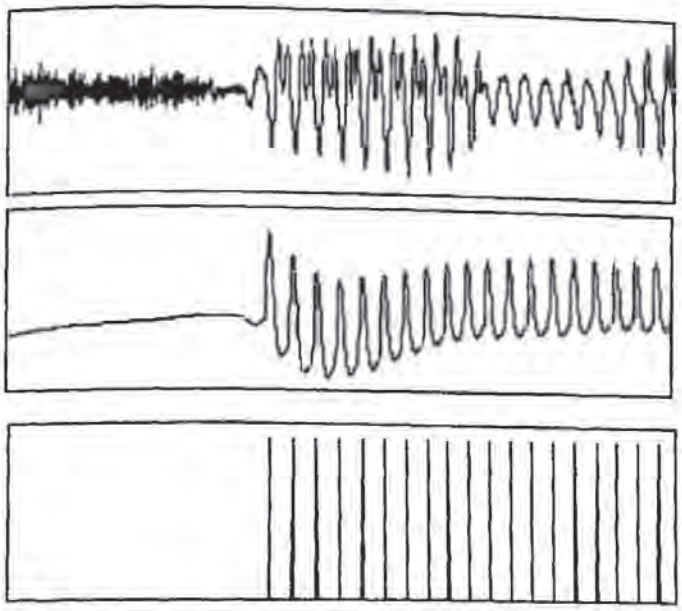


Figure 16.14 Speech signal, laryngograph signal, and its corresponding epochs.

The preemphasized signal exhibits peaks that are found by thresholding. The quality of this epoch detector has been evaluated on recordings from two female and four male speakers, and the voiced/unvoiced decision errors are lower than 1%. This is definitely acceptable for our prosody-modification algorithms. The quality of prosody modification with the epochs computed by this method can vastly exceed the quality achieved when standard pitch trackers (as described in Chapter 6) are used on the original speech signal [2].

16.4.10. Problems with PSOLA

The PSOLA approach is very effective in changing the pitch and duration of a speech segment if the epochs are determined accurately. Even assuming there are no pitch tracking errors, there can be problems when concatenating different segments:

- *Phase mismatches.* Even if the pitch period is accurately estimated, mismatches in the positioning of the epochs in the analysis signal can cause glitches in the output, as can be seen in Figure 16.15. The MBROLA [15] technique, an attempt to overcome phase mismatches, uses the time-domain PSOLA method for prosody modification, but the pitch cycles have been preprocessed so that they have a fixed phase. The advantage is that the spectral smoothing can be done by directly interpolating the pitch cycles in the time domain without adding any extra complexity. Since MBROLA sets the phase to a constant, the algorithm is more robust to phase errors in the epoch detection. Unfortunately, setting the phases constant incurs the added perceived noise described before.

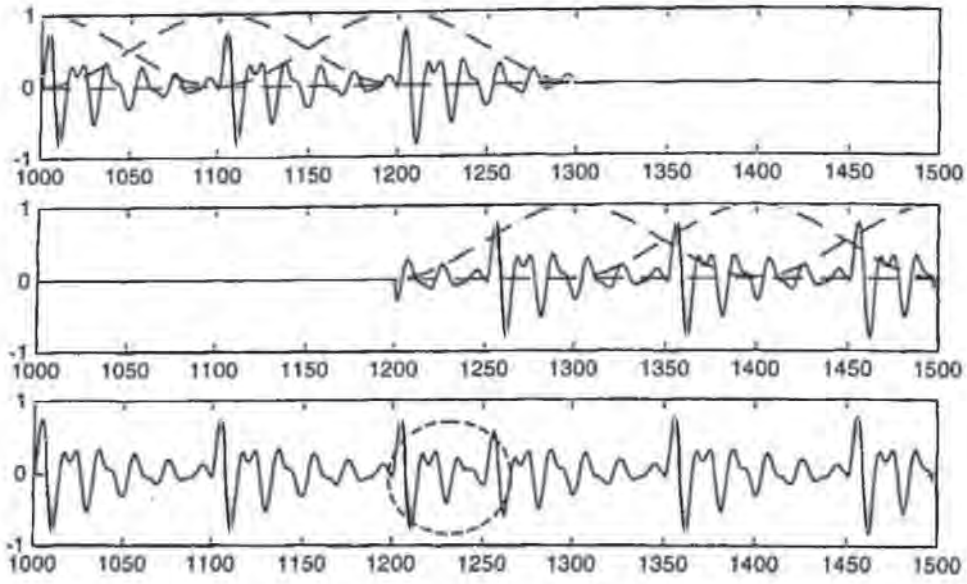


Figure 16.15 Phase mismatches in unit concatenation. Waveforms are identical, but windows are not centered on the same relative positions within periods.

- *Pitch mismatches.* These occur even if there are no pitch or phase errors during the analysis phase. As shown in Section 16.4.3, if two speech segments have the same spectral envelope but different pitch, the estimated spectral envelopes are not the same, and, thus, a discontinuity occurs (see Figure 16.16). In addition, pitch and timbre are not independent. Even when producing the same sound in the same phonetic context, a vastly different pitch will likely result in a different spectral envelope. This effect is particularly accentuated in the case of opera singers, who move their formants around somewhat so that the harmonics fall near the formant values and thus produce higher output.
- *Amplitude mismatch.* A mismatch in amplitude across different units can be corrected with an appropriate amplification, but it is not straightforward to compute such a factor. More importantly, the timbre of the sound will likely change with different levels of loudness.
- *Buzzy voiced fricatives.* The PSOLA approach doesn't handle well voiced fricatives that are stretched considerably because of added buzziness (repeating frames induces periodicity at the high frequency that wasn't present in the original signal) or attenuation of the aspirated component (if frames are interpolated).

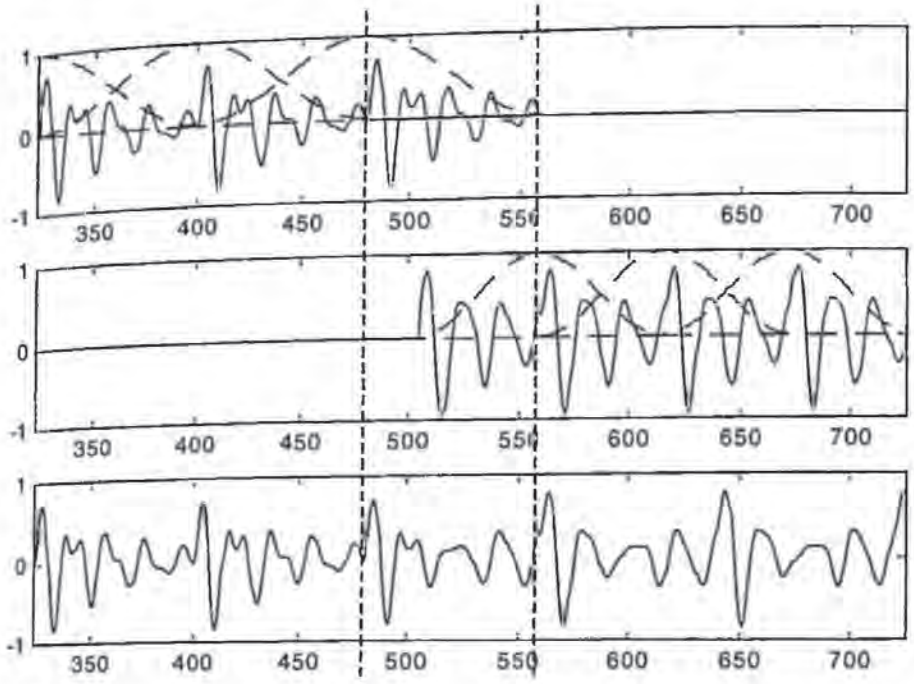


Figure 16.16 Pitch mismatches in unit concatenation. Two synthetic vowels were generated with a pitch of 138 Hz (top) and 197 Hz (middle) and exactly the same transfer function. There is no pitch tracking error, and windows are positioned coherently (no phase mismatch). The pitch of the second wave is changed through PSOLA to match the pitch of the first wave. There is a discontinuity in the resulting waveform and its spectrum (see Section 16.4.3), which is an artifact of the way the PSOLA approach estimates the spectral envelope.

16.5. SOURCE-FILTER MODELS FOR PROSODY MODIFICATION

The largest problem in concatenative synthesis occurs because of spectral discontinuities at unit boundaries. The methods described in Section 16.3 significantly reduce this problem but do not eliminate it. While PSOLA can do high-quality prosody modification on speech segments, it doesn't address these spectral discontinuities occurring at unit boundaries. It would be useful to come up with a technique that allows us to smooth these spectral discontinuities. In addition, PSOLA introduces buzziness for overstretched voiced fricatives. In the following sections we describe a number of techniques that have been proposed to cope with these problems and that are based on source-filter models.

The use of source-filter models allow us to modify the source and filter separately and thus maintain more control over the resulting synthesized signal. In Section 16.5.1 we study an extension of PSOLA that allows filter modification as well for smoothing purposes. Section 16.5.2 describes mixed excitation models that also allow for improved voiced fricatives. Finally, Section 16.5.3 studies a number of voice effects that can be achieved with a source-filter model.

16.5.1. Prosody Modification of the LPC Residual

A method known as LP-PSOLA that has been proposed to allow smoothing in the spectral domain is to do PSOLA on LPC residual. This approach, thus, implicitly uses the LPC spectrum as the spectral envelope instead of the spectral envelope interpolated from the harmonics (see Section 16.4.3) when doing F0 modification. If the LPC spectrum is a better fit to the spectral envelope, this approach should reduce the spectral discontinuities due to different pitch values at the unit boundaries. LP-PSOLA reduces the *bandwidth widening*. In practice, however, this hasn't proven to offer a significant improvement in quality, possibly because the spectral discontinuities due to coarticulation dominate the overall quality.

The main advantage of this approach is that it allows us to smooth the LPC parameters around a unit boundary and thus obtain smoother speech. Since smoothing the LPC parameters directly may lead to unstable frames, other equivalent representations, such as line spectral frequencies, reflection coefficients, log-area ratios, or autocorrelation coefficients, are used instead. The use of a long window for smoothing may blur sharp spectral changes that occur in natural speech. In practice, a window of 20–50 ms centered around the boundary has been proven useful.

While time-domain PSOLA has low computational complexity, its use in a concatenative speech synthesizer generally requires a large acoustic inventory. In some applications this is unacceptable, and it needs to be compressed using any of the coding techniques described in Chapter 7. You need to keep in mind that to use such encoders you need to store the coder's memory so that the first frame of the unit can be accurately encoded. The combined decompression and prosody modification is not as computationally efficient as time-domain PSOLA alone, so that the LP-PSOLA approach may offer an effective tradeoff, given that many speech coders encode the LPC parameters anyway.

16.5.2. Mixed Excitation Models

The block diagram of PSOLA shown in Figure 16.10 for voiced sounds also works for unvoiced sounds by choosing arbitrary epochs. The time-varying filter of Figure 16.10 can be kept in its time-domain form or in the frequency domain $X_t[k]$ by taking the FFT of $x_t[n]$.

It has been empirically shown that for unvoiced frames, the phase of $X_t[k]$ is unimportant as long as it is random. Thus, we can pass white noise through a filter with magnitude response $|X_t[k]|$ and obtain perceptually indistinguishable results. This reduced representation is shown in Figure 16.17. Moreover, it has been shown that the magnitude spectrum

does not need to be encoded accurately, because it doesn't affect the synthesized speech. The only potential problem with this model occurs when voiced frames are incorrectly classified as unvoiced.

Maintaining the phase of $X_t[k]$ is perceptually important for voiced sounds. If it is set to 0, two audible distortions appear: the reconstructed speech exhibits a noisy quality, and voiced fricatives sound buzzy.

The perceived noise may come from the fact that a listener who hears a formant, because of its amplitude spectrum, also expects the 180° phase shift associated with a complex pole. In fact, it is not the absolute phase, but the fact that if the formant frequency/bandwidth changes with time, there is a phase difference over time. If such a phase is not present, scene analysis done in the auditory system may match this to noise. This effect can be greatly attenuated if the phase of the residual in LP-PSOLA is set to 0, possibly because the LPC coefficients carry most of the needed phase information.

The buzziness in voiced fricatives is the result of setting phase coherence not only at low frequencies but also at high frequencies, where the aspiration component dominates. This is the result of treating the signal as voiced, when it has both a voiced and an unvoiced component. In fact, most voiced sounds contain some aperiodic component, particularly at high frequencies. The amount of aspiration present in a sound is called *breathiness*. Female speech tends to be more breathy than male speech [29]. Mixed-excitation models, such as those in Figure 16.18, are then proposed to more accurately represent speech.

Such a model is very similar to the waveform-interpolation coding approach of Chapter 7, and, hence, we can leverage much of what was described there regarding the estimation of $x_t'[n]$ and $x_t''[n]$. This approach allows us to integrate compression with prosody modification.

The harmonic-plus-noise [50] model decomposes the speech signal $s(t)$ as a sum of a random component $s_r(t)$ and a harmonic component $s_p(t)$

$$s(t) = s_r(t) + s_p(t) \tag{16.35}$$

where $s_p(t)$ uses the sinusoidal model described in Chapter 7:

$$s_p(t) = \sum_{k=1}^{K(t)} A_k(t)\cos(k\theta(t) + \phi_k(t)) \tag{16.36}$$

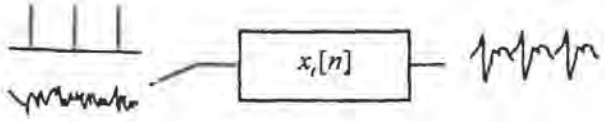


Figure 16.17 Speech synthesis model with white noise or an impulse train driving a time-varying filter.

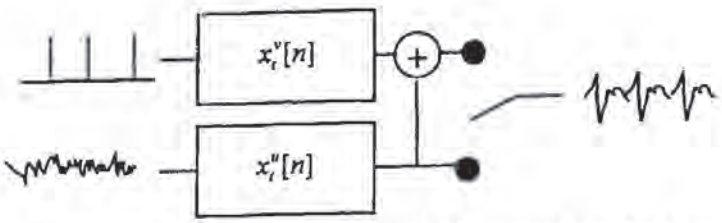


Figure 16.18 Mixed excitation model for speech synthesis.

where $A_k(t)$ and $\phi_k(t)$ are the amplitude and phase at time t of the kth harmonic, and $\theta(t)$ is given by

$$\theta(t) = \int_{-\infty}^{t} \omega_0(l)dl \tag{16.37}$$

16.5.3. Voice Effects

One advantage of using a spectral representation like those described in this section is that several voice effects can be achieved relatively easily, such as whisper, voice conversion, and echo/reverberation.

A whispering effect can be achieved by replacing the voiced component by random noise. Since the power spectrum of the voiced signal is a combination of the vocal tract and the glottal pulse, we would need to remove the spectral roll-off of the glottal pulse. This means that the power spectrum of the noise has to be high-pass in nature. Using white noise results in unnatural speech.

Voice conversion can be accomplished by altering the power spectrum [6, 28]. A warping transformation of the frequency scale can be achieved by shifting the LSF or the LPC roots, if using an LPC approach, or a warping curve if using an FFT representation.

Adding a controlled number of delayed and attenuated echoes can enhance an otherwise *dry* signal. If the delay is longer, it can simulate the room acoustics of a large hall.

16.6. Evaluation of TTS Systems

How do we determine whether one TTS system is better than another? Being able to evaluate TTS systems allows a customer to select the best system for his or her application. TTS evaluation is also important for developers of such systems to set some numerical goals in their design. As in any evaluation, we need to define a metric, which generally is dependent on the particular application for which the customer wants the TTS system. Such a metric consists of one or several variables of a system that are measured. Gibbon et al. [19] present a good summary of techniques used in evaluation of TTS systems.

Here we present a taxonomy of a TTS evaluation:

- *Glass-box vs. black-box evaluation.* There are two types of evaluation of TTS systems according to whether we evaluate the whole system or just one of its components: black-box and glass-box. A black-box evaluation treats the TTS system as a black box and evaluates the system in the context of a real-world application. Thus, a system may do very well on a flight reservation application but poorly on an e-mail reading application. In a glass-box evaluation, we attempt to obtain diagnostics by evaluating the different components that make up a TTS system.
- *Laboratory vs. field.* We can also conduct the study in a laboratory or in the field. While the former is generally easier to do, the latter is generally more accurate.
- *Symbolic vs. acoustic level.* In general, TTS evaluation is normally done by analyzing the output waveform, the so-called acoustic level. Glass-box evaluation at the symbolic level is useful for the text analysis and phonetic module, for example.
- *Human vs. automated.* There are two fundamentally distinct ways of evaluating speech synthesizers, according to how a given attribute of the system is estimated. One is to use human subjects; the other to automate the evaluation process. Both types have some issues in common and a number of dimensions of systematic variation. But the fundamental distinction is one of cost. In system development, and particularly in research on high-quality systems, it can be prohibitively expensive to run continuously a collection of human assessments of every algorithmic change or idea. Though human-subject checkpoints are needed throughout the development process, human testing is of greatest importance for the integrated, functionally complete system in the target field setting. At all earlier stages of development, automated testing should be substituted for human-subject testing wherever possible. The hope is that someday TTS research can be conducted as ASR research is today: algorithms are checked for accuracy or performance improvements automatically in the lab, while human subjects are mainly used when the final integrated system is deployed for field testing. This allows for rapid progress in the basic algorithms contributing to accuracy on any given dimension.
- *Judgment vs. functional testing.* Judgment tests are those that measure the TTS system in the context of the application where it is used, such as what percentage of the time users hang up an IVR system. System *A* may be more appropriate than system *B* for a banking application where most of the speech consists of numerical values, and system *B* may be better than system *A* for reading e-mail over the phone. Nonetheless, it is useful to use functional tests that measure task-independent variables of a TTS system, since such tests allow an easier comparison among different systems, albeit a nonoptimal one.

Since a human listener is the consumer of a TTS system, tests have been designed to determine the following characteristics of synthesized speech: intelligibility, overall quality, naturalness, suitability for a given task, and pleasantness. In addition, testing has been used for ranking and comparing a number of competing speech synthesizers, and for comparing synthetic with natural speech.

- *Global vs. analytic assessment.* The tests can measure such global aspects as overall quality, naturalness, and acceptability. Analytic tests can measure the rate, the clarity of vowels and consonants, the appropriateness of stresses and accents, pleasantness of voice quality, and tempo. Functional tests have been designed to test the intelligibility of individual sounds (phoneme monitoring), of combinations of sounds (syllable monitoring), and of whole words (word monitoring) in isolation as well as in various types of context.

It should be noted that all the above tests focus on segments, words, and sentences. This is a historical artifact, and as the field evolves, we should see an emphasis on testing of higher-order units. The diagnostic categories mentioned above can be used as a basis for developing tests of systems that take other structure into account. Such systems might include document-to-speech, concept-to-speech, and simulated conversation or dialog. A good system will reflect document and paragraph structure in the pausing and rhythm. Concept-to-speech systems claim to bring fuller knowledge of the intended use of information to bear in message generation and synthesis. Simulated dialog systems, or human-computer dialog systems, have to mimic a more spontaneous style, which is a subtle quality to evaluate. The tricky issue with higher-order units is the difficulty of simple choice or transcription-oriented measures. To develop tests of higher-order synthesizers, the word and sentence metrics can be applied to components and the overall system until reasonable intelligibility can be verified. Then tests of the special issues raised by higher-order systems can be conducted. Appropriate measures might be MOS overall ratings, preference between systems, summarization/gist transcription with subjective scoring, and task-based measures such as following directions. With task-based testing of higher-order units, both the correctness of direction-following and the time to completion, an indirect measure of intelligibility, pleasantness, and fatigue, can be recorded.

Furthermore, speech perception is not simply auditory. As discussed in Chapter 2, the McGurk effect [36] shows that perception of a speech sound is heavily influenced by visual cues. Synthetic speech is thus perceived with higher quality when a talking head is added as a visual cue [9, 18].

Glass-box evaluation of the text analysis and phonetic analysis modules, requiring evaluation at the *symbolic* level, is done in Chapter 14. A glass-box evaluation of the prosody module is presented in Chapter 15. In this section we include glass-box evaluation of the synthesis module, as well as a black-box evaluation of the whole system.

16.6.1. Intelligibility Tests

A critical measurement of a TTS system is whether or not human listeners can understand the text read by the system. Tests that measure this are called *intelligibility tests*. In this section we describe the Diagnostic Rhyme Test, the Modified Rhyme Test, the Phonetically Balanced word list test, the Haskins Syntactic Sentence Test, and the Semantically Unpredictable Sentence Test. The first three were described in a procedure approved by the American National Standards Institute [5].

Among the best known and most mature of these tests is the Diagnostic Rhyme Test (DRT) proposed by Voiers [54], which provides for diagnostic and comparative evaluation of the intelligibility of single initial consonants. The test runs twice through the list of 96 rhyming pairs shown in Table 16.13. The test consists of identification choice between two alternative English (or target-language) words, differing by a single phonetic feature in the initial consonant. For English the test includes contrasts among easily confusable paired consonant sounds such as *veal/feel, meat/beat, fence/pence, cheep/keep, weed/reed*, and *hit/fit*. In the test, both *veal* and *feel* are presented with the response alternatives *veal* and *feel*. Six contrasts are represented, namely voicing, nasality, sustention, sibilation, graveness, and compactness. Each contrast is included 32 times in the test, combined with 8 different vowels. The percentage of right answers is used as an indicator of speech synthesizer intelligibility. The tests use a minimum of five talkers and five listeners; larger subject groups reduce the margin of error. Even for high-quality speech coders, 100% *correct* responses are rarely achieved, so synthesizer results should be interpreted generously.

Table 16.13 The 192 stimulus words of the Diagnostic Rhyme Test (DRT).

Voicing		Nasality		Sustenation		Sibilation		Graveness		Compactness	
veal	feel	meat	beat	vee	bee	zee	thee	weed	reed	yield	wield
bean	peen	need	deed	sheet	cheat	cheep	keep	peak	teak	key	tea
gin	chin	mitt	bit	vill	bill	jilt	gilt	bid	did	hit	fit
dint	tint	nip	dip	thick	tick	sing	thing	fin	thin	gill	dill
zoo	sue	moot	boot	foo	pooh	juice	goose	moon	noon	coop	poop
dune	tune	news	dues	shoes	choose	chew	coo	pool	tool	you	rue
vole	foal	moan	bone	those	doze	joe	go	bowl	dole	ghost	boast
goat	coat	note	dote	though	dough	sole	thole	fore	thor	show	so
zed	said	mend	bend	then	den	jest	guest	met	net	keg	peg
dense	tense	neck	deck	fence	pence	chair	care	pent	tent	yen	wren
vast	fast	mad	bad	than	dan	jab	gab	bank	dank	gat	bat
gaff	calf	nab	dab	shad	chad	sank	thank	fad	thad	shag	sag
vault	fault	moss	boss	thong	tong	jaws	gauze	fought	thought	yawl	wall
daunt	taunt	gnaw	daw	shaw	chaw	saw	thaw	bong	dong	caught	thought
jock	chock	mom	bomb	von	bon	jot	got	wad	rod	hop	fop
bond	pond	knock	dock	vox	box	chop	cop	pot	tot	got	dot

A variant of this is the Modified Rhyme Test (MRT) proposed by House [22], which also uses a 300-entry word list for subjective intelligibility testing. The modified Rhyme Test (shown in Table 16.14) uses 50 six-word lists of rhyming or similar-sounding monosyllabic English words, e.g., *went, sent, bent, dent, tent, rent.* Each word is basically Consonant-Vowel-Consonant (with a few consonant clusters), and the six words in each list differ only in the initial or final consonant sound(s). Listeners are asked to identify which of the words was spoken by the synthesizer (closed response), or in some cases to enter any word they thought they heard (open response). A carrier sentence, such as "Would you write <test word> now," is usually used for greater naturalness in stimulus presentation. Listener responses can be scored as the number of words heard correctly; or the frequency of confusions of particular consonant sounds. This can be viewed as *intelligibility* of the synthesizer.

Though this is a nice isolation of one property, and as such is particularly appropriate for diagnostic use, it is not intended to substitute for fuller evaluation under more realistic listening conditions involving whole sentences. Segmental intelligibility is somewhat overestimated in these tests, because all the alternatives are real words and the subjects can adjust their perception to match the closest word. A typical human voice gives an MRT score of about 99%, with that of TTS systems generally ranging from 70% to 95%.

Table 16.14 The 300 stimulus words of the Modified Rhyme Test (MRT).

went	sent	bent	dent	tent	rent	same	name	game	tame	came	fame
hold	cold	told	fold	sold	gold	peel	reel	feel	eel	keel	heel
pat	pad	pan	path	pack	pass	hark	dark	mark	bark	park	lark
lane	lay	late	lake	lace	lame	heave	hear	heat	heal	heap	heath
kit	bit	fit	hit	wit	sit	cup	cut	cud	cuff	cuss	cud
must	bust	gust	rust	dust	just	thaw	law	raw	paw	jaw	saw
teak	team	teal	teach	tear	tease	pen	hen	men	then	den	ten
din	dill	dim	dig	dip	did	puff	puck	pub	pus	pup	pun
bed	led	fed	red	wed	shed	bean	beach	beat	beak	bead	beam
pin	sin	tin	fin	din	win	heat	neat	feat	seat	meat	beat
dug	dung	duck	dud	dub	dun	dip	sip	hip	tip	lip	rip
sum	sun	sung	sup	sub	sud	kill	kin	kit	kick	king	kid
seep	seen	seethe	seek	seem	seed	hang	sang	bang	rang	fang	gang
not	tot	got	pot	hot	lot	took	cook	look	hook	shook	book
vest	test	rest	best	west	nest	mass	math	map	mat	man	mad
pig	pill	pin	pip	pit	pick	ray	raze	rate	rave	rake	race
back	bath	bad	bass	bat	ban	save	same	sale	sane	sake	safe
way	may	say	pay	day	gay	fill	kill	will	hill	till	bill
pig	big	dig	wig	rig	fig	sill	sick	sip	sing	sit	sin
pale	pace	page	pane	pay	pave	bale	gale	sale	tale	pale	male
cane	case	cape	cake	came	cave	wick	sick	kick	lick	pick	tick
shop	mop	cop	top	hop	pop	peace	peas	peak	peach	peat	peal
coil	oil	soil	toil	boil	foil	bun	bus	but	bug	buck	buff
tan	tang	tap	tack	tam	tab	sag	sat	sass	sack	sad	sap
fit	fib	fizz	fill	fig	fin	fun	sun	bun	gun	run	nun

The set of twenty phonetically balanced (PB) word lists was developed during World War II and has been used very widely since then in subjective intelligibility testing. In Table 16.15 we include the first four PB word lists [20]. The words in each list are presented in a new, random order each time the list is used, each spoken in the same carrier sentence. The PB intelligibility test requires more training of listeners and talkers than other subjective tests and is particularly sensitive to SNR: a relatively small change causes a large change in the intelligibility score.

Tests using the *Haskins Syntactic Sentences* [40] go somewhat farther toward more realistic and holistic stimuli. This test set consists of 100 semantically unpredictable sentences of the form *The <Adjective> <Noun1> <Verb> the <Noun2>*, such as *"The old farm cost the blood,"* using high-frequency words. Compared with the rhyme tests, contextual predictability based on meaning is largely lacking, the longer speech streams are more realistic, and more coarticulation is present. Intelligibility is indicated by percentage of words correct.

Another test minimizing predictability is *Semantically Unpredictable Sentences* [23], with test sets for Dutch, English, French, German, Italian, and Swedish. A short template of syntactic categories provides a frame, into which words are randomly slotted from the lexicon. For example, the template *<Subject> <Verb> <Adverbial>* might appear as *"The chair ate through the green honesty."* Fifty sentences (10 per syntactic template) are considered adequate to test a synthesizer. Open transcription is requested, and *sentences correct* is used to score a synthesizer's intelligibility. Other such tests exist, and some include systematic variation of prosody on particular words or phrases as well.

The Harvard Psychoacoustic Sentences [16] is a set of 100 meaningful, syntactically varied, phonetically balanced sentences, such as *"Add salt before you fry the egg,"* requiring an *open response identification,* instead of a multiple-choice test.

Table 16.15 Phonetically balanced word lists.

List 1	are, bad, bar, bask, box, cane, cleanse, clove, crash, creed, death, deed, dike, dish, end, feast, fern, folk, ford, fraud, fuss, grove, heap, hid, hive, hunt, is, mange, no, nook, not, pan, pants, pest, pile, plush, rag, rat, ride, rise, rub, slip, smile, strife, such, then, there, toe, use, wheat
List 2	awe, bait, bean, blush, bought, bounce, bud, charge, cloud, corpse, dab, earl, else, fate, five, frog, gill, gloss, hire, hit, hock, job, log, moose, mute, nab, need, niece, nut, our, perk, pick, pit, quart, rap, rib, scythe, shoe, sludge, snuff, start, suck, tan, tang, them, trash, vamp, vast, ways, wish
List 3	ache, air, bald, barb, bead, cape, cast, check, class, crave, crime, deck, dig, dill, drop, fame, far, fig, flush, gnaw, hurl, jam, law, leave, lush, muck, neck, nest, oak, path, please, pulse, rate, rouse, shout, sit, size, sob, sped, stag, take, thrash, toil, trip, turf, vow, wedge, wharf, who, why
List 4	bath, beast, bee, blonde, budge, bus, bush, cloak, course, court, dodge, dupe, earn, eel, fin, float, frown, hatch, heed, hiss, hot, how, kite, merge, lush, neat, new, oils, or, peck, pert, pinch, pod, race, rack, rave, raw, rut, sage, scab, shed, shin, sketch, slap, sour, starve, strap, test, tick, touch

16.6.2. Overall Quality Tests

While a TTS system has to be intelligible, this does not guarantee user acceptance, because its quality may be far from that of a human speaker. In this section we describe the Mean Opinion Score and the Absolute Category Ratings.

Human-subject judgment testing for TTS can adapt methods from speech-coding evaluation (see Chapter 7). With speech coders, *Mean Opinion Score* (MOS) is administered by asking 10 to 30 listeners to rate several sentences of coded speech on a scale of 1 to 5 (1 = Bad, 2 = Poor, 3 = Fair, 4 = Good, 5 = Excellent). The scores are averaged, resulting in an overall MOS rating for the coder. This kind of methodology can be applied to speech synthesizers as well. Of course, as with any human subject test, it is essential to carefully design the listening situation and carefully select the subject population, controlling for education, experience, physiological disorders, dialect, etc. As with any statistically interpreted test, the standard analyses of score distributions, standard deviation, and confidence intervals must be performed. The range of quality in coder evaluations by MOS are shown in Table 16.16.

Since we are making the analogy to coders, certain ironies can be noted. Note the lowest-range descriptor for coder evaluation: *synthetic*. In using MOS for synthesis testing, output is being evaluated by implicit reference to real human speech, and the upper range in the coder MOS interpretations above (3.5–4.5) is probably not applicable to the output of most TTS systems. Even a *good* TTS system might fare poorly on such a coder MOS evaluation. Therefore, the MOS interpretive scale, when applied to synthesis, cannot be absolute as the above coding-based interpretive table would imply. Furthermore, subjects participating in MOS-like tests of synthesizers should be made aware of the special nature of the speech (synthetic) and adjust their expectations accordingly. Finally, no matter how carefully the test is designed and administered, it is difficult to correlate, compare, and scale such measures. Nevertheless, MOS tests are perhaps suited to relative ranking of various synthesizers. The 1-to-5 scale is categorical, but similar judgment tests can be run in *magnitude* mode, with the strength of the quality judgment being indicated along a continuous scale, such as a moving slider bar.

Table 16.16 Mean opinion score (MOS) ratings and typical interpretations.

MOS Scores	Quality	Comments
4.0–4.5	Toll/Network	Near-transparent, "in-person" quality
3.5–4.0	Communications	Natural, highly intelligible, adequate for telecommunications, changes and degradation of quality very noticeable
2.5–3.5	Synthetic	Usually intelligible, can be unnatural, loss of speaker recognizability, inadequate levels of naturalness

Table 16.17 Listening Quality Scale.

Quality of the Speech	Score
Excellent	5
Good	4
Fair	3
Poor	2
Bad	1

The International Telecommunication Union (ITU) has attempted to specify some standards for assessing synthetic speech, including spliced digitized words and phrases, typically with the expectation of delivery over the phone. The *Absolute Category Rating* (ACR) system recommended by ITU P.800 offers instructions to be given to subjects for making category judgments in MOS-style tests of the type discussed here. The first is the *Listening Quality Scale*, shown in Table 16.17, and the second the *Listening Effort Scale* shown in Table 16.18.

It is sometimes possible to get subjects to pay particular attention to various particular features of the utterance, which may be called *analytic* as opposed to *global* listening. The desired features generally have to be described somehow, and these descriptions can be a bit vague. Thus, standard measures of reliability and validity, as well as result normalization, must be applied. Typical descriptors for important factors in analytic listening might be: *smoothness*, *naturalness*, *pleasantness*, *clarity*, *appropriateness*, etc., each tied to a particular underlying target quality identified by the system designers. For example, smoothness might be a descriptor used when new algorithms for segment concatenation and blending are being evaluated in a concatenative system. Naturalness might be the quality descriptor when a formant-based system has been made more natural by incorporation of a richer glottal source function. Some elements of the speech can be more directly identified to the subject in familiar terms. For example, pleasantness might be a way of targeting the pitch contours for attention, or the subject could be specifically asked to rank the pitch contours per se, in terms of naturalness, pleasantness, etc. Appropriateness might be a way of getting at judgments of accentuation: e.g., a stimulus that was accented as "... *birthday PARTY*" might be judged less appropriate, in a neutral semantic context, than one that was perceived as "... *BIRTHDAY party*." But no matter how the attributes are described, in human-subject MOS-style testing there cannot be a clear and consistent separation of effects.

Table 16.18 Listening Effort Scale.

Effort Required to Understand the Meanings of Sentences	Score
Complete relaxation possible; no effort required	5
Attention necessary; no appreciable effort required	4
Moderate effort required	3
Considerable effort required	2
No meaning understood with any feasible effort	1

16.6.3. Preference Tests

Normalized MOS scores for different TTS systems can be obtained without any direct preference judgments. If direct comparisons are desired, especially for systems that are informally judged to be fairly close in quality, another ITU recommendation, the *Comparison Category Rating* (CCR) method, may be used. In this method, listeners are presented with a pair of speech samples on each trial. The order of the *system A system B* samples is chosen at random for each trial. On half of the trials, the *system A* sample is followed by the *system B* sample. On the remaining trials, the order is reversed. Listeners use the instructions in Table 16.19 to judge the quality of the second sample relative to that of the first. Sometimes the granularity can be reduced as much as simply "*prefer A/prefer B.*"

Assuming (A,B) is the reference presentation order, scores for the (B,A) presentations may be normalized by reversing their signs (e.g., –1 in B,A order becomes 1, etc.). Subsequently, standard statistical summarizations may be performed, like the one described in Chapter 3.

Table 16.19 Preference ratings between two systems. The quality of the second utterance is compared to the quality of the first by means of 7 categories. Sometimes only better, same, or worse are used.

3	Much Better
2	Better
1	Slightly Better
0	About the Same
-1	Slightly Worse
-2	Worse
-3	Much Worse

16.6.4. Functional Tests

Functional testing places the human subject in the position of carrying out some task related to, or triggered by, the speech. This can simulate a full field deployment, with a usercentric task, or can be more of a laboratory situation, with a testcentric task. In the laboratory situation, various kinds of tasks have been proposed. In *analytic* mode, functional testing can enforce isolation of the features to be attended to in the structure of the test stimuli themselves. This can lead to a more precise form of result than the MOS judgment approach. There have been a wide variety of proposals and experiments of this type.

One of the well-known facts in TTS evaluation is that the quality of a system is dominated by the quality of its worst component. While it may be argued that it is impossible to separate the effects of the front-end analysis and back-end synthesis, it is convenient to do so to gain a better understanding of each component. An attempt to study the quality of the speech synthesis module has been done via the use of natural instead of synthetic prosody. This way, it is presumed that the prosody module is doing the best possible job, and that any

problem is then due to a deficient speech synthesis. The natural pitch contour can be obtained with a pitch tracker (or using a laryngograph signal), and the natural durations can be obtained either through manual segmentation or through the use of a speech recognition system used in forced-alignment mode. Plumpe and Meredith [44] conducted a preference test between original recordings and waveforms created when one of the modules of a concatenative TTS system used synthetically generated values instead of the natural values. The results indicated that using synthetic pitch instead of natural pitch was the cause of largest degradation according to listeners, and, thus, that pitch generation was the largest bottleneck in the system. The pitch-generation module was followed by the spectral discontinuities at the concatenation points, with duration being the least damaging.

Some functional tests are much more creative than simple transcription, however. They could, in theory, border on related areas, such as memory testing, involving summarizing passages, or following synthesized directions, such as a route on a map. The ultimate test of synthesis, in conjunction with all other language interface components, is said to be the Turing test [53]. In this amusing scenario, a human being is placed into conversation with a computational agent, represented vocally for our purposes, perhaps over the telephone. As Turing put it: "It is proposed that a machine may be deemed intelligent, if it can act in such a manner that a human cannot distinguish the machine from another human merely by asking questions via a mechanical link." Turing predicted that in the future "an average interrogator will not have more than a 70 percent chance of making the right identification, human or computer on the other end, after five minutes of questioning" in this game. A little reflection might raise objections to this procedure as a check on speech output quality per se, since some highly intelligent people have speech disabilities, but the basic idea should be clear, and it remains a Holy Grail for the artificial intelligence field generally. Of course, no automated or laboratory test can substitute for a real-world trial with paying customers.

16.6.5. Automated Tests

The tests described above always involved the use of human subjects and are the best tests that can be used to evaluate a TTS system. Unfortunately, they are time consuming and expensive to conduct. This limits their application to an infrequent use, which can hardly have any diagnostic value. Automated *objective* tests usually involve establishing a test corpus of correctly tagged examples of the tested phenomena, which can be automatically checked. This style of testing is particularly appropriate when working with isolated components of the TTS system, for diagnosis or regression testing (glass-box testing). It is not particularly productive to discuss such testing in the abstract, as the test features must closely track each system's design and implementation. Nevertheless, a few typical areas for testing can be noted. In general, tests are simultaneously testing the linguistic model and content as well as the software implementation of a system, so whenever a discrepancy arises, both possible sources of error must be considered.

Several automated tests for text analysis and letter-to-sound conversion are presented in Chapter 14. A number of automated tests for prosody are discussed in Chapter 15. Here we touch on automated tests for the synthesis module.

The ITU has created the P.861 proposal for estimating perceptual scores using automated signal-based measurements. The P.861 specifies a particular technique known as *Perceptual Speech Quality Measurement* (PSQM). In this method, for each analysis frame, various quantified measures based on the time signal, the power spectrum, the Bark power spectrum, the excitation spectrum, the compressed loudness spectrum, etc. of both the reference and the test signal can be computed. In some cases the PSQM score can be converted to an estimated MOS score, with interpretations similar to those of Table 16.16. At present such methods are limited primarily to analysis of telephone-quality speech (300–3400 Hz bandwidth), to be compared with closely related reference utterances. This method could perhaps be adapted to stand in for human judgments during system development of new versions of modules, say glottal source functions in a formant synthesizer, comparing the resulting synthetic speech to a standard reference system's output on a given test sample.

16.7. HISTORICAL PERSPECTIVE AND FURTHER READING[3]

In 1779 in St. Petersburg, Russian Professor Christian Kratzenstein explained physiological differences between five long vowels (*/a/*, */e/*, */i/*, */o/*, and */u/*) and made apparatus to produce them artificially. He constructed acoustic resonators similar to the human vocal tract and activated the resonators with vibrating reeds as in music instruments. Von Kempelen (1734–1804) proposed in 1791 in Vienna a mechanical speaking machine that could produce not just vowels but whole words and sentences (see Figure 16.19). While working with his speaking machine, he demonstrated a speaking chess-playing machine. Unfortunately, the main mechanism of the machine was a concealed, legless chess-player expert. Therefore, his real speaking machine was not taken as seriously as it should have been. In 1846, Joseph Faber developed a synthesizer, called speech organ, that had more control of pitch to the extent it could sing *God Save the Queen* in a performance in London.

The first electrical synthesis device was introduced by Stewart in 1922 [4]. The device had a buzzer as excitation and two resonant circuits to model the acoustic resonances of the vocal tract and was able to generate single static vowel sounds with the first two formants. In 1932 Japanese researchers Obata and Teshima added a third formant for more intelligible vowels.

Homer Dudley of Bell Laboratories demonstrated at the 1939 New York World's Fair the *Voder*, the first electrical speech synthesizer, which was human-controlled. The operator worked at a keyboard, with a wrist bar to control the voicing parameter and a pedal for pitch control (see Figure 16.20 and Figure 16.21), and it was able to synthesize continuous speech. The *Pattern Playback* is an early talking machine that was built by Franklin S. Cooper and his colleagues at Haskins Laboratories in the late 1940s. This device synthesized sound by passing light through spectrograms that in turn modulated an oscillator with a fixed F0 of 120 Hz and 50 harmonics.

[3] Chapter 6 includes a historical perspective on representation of speech signals that is intimately tied to speech synthesis.

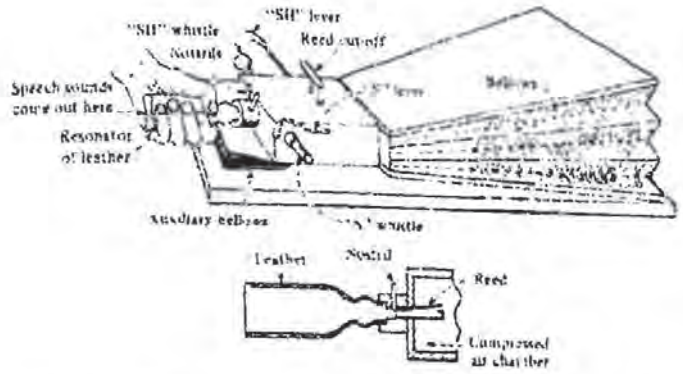


Figure 16.19 Wheatstone's reconstruction of von Kempelen's speaking machine [14] (after Flanagan [17]).

The first analog parallel formant synthesizer, the *Parametric Artificial Talker* (PAT), was developed in 1953 by Walter Lawrence of the Signals Research and Development Establishment of the British Government. Gunnar Fant of the KTH in Sweden developed an analog cascade formant synthesizer, the OVE II. Both Lawrence and Fant showed in 1962 that by manually tuning the parameters, a natural sentence could be reproduced reasonably faithfully. Acoustic analog synthesizers were also known as terminal analogs, resonance-synthesizers. John Holmes tuned by hand the parameters of his formant synthesizer so well that the average listener could not tell the difference between the synthesized sentence "*I enjoy the simple life*" and the natural one [31].



Figure 16.20 The Voder developed by Homer Dudley of Bell Labs at the 1939 World's Fair in New York. The operator worked at a keyboard, with a wrist bar to control the voicing parameter and a pedal for pitch control.

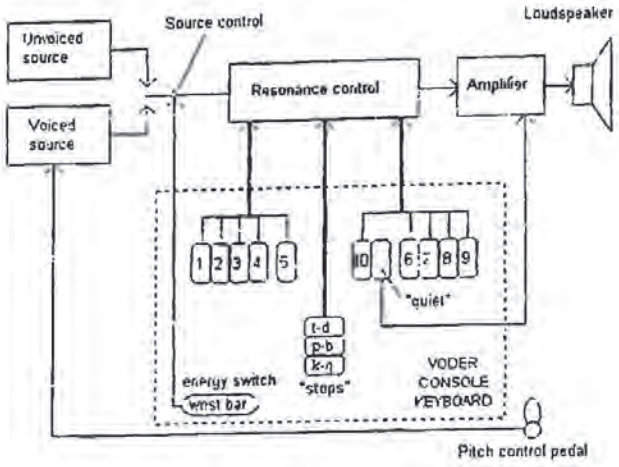


Figure 16.21 Block diagram of the Voder by Homer Dudley, 1939 (after Flanagan [17]).

The first articulatory synthesizer, the DAVO, was developed in 1958 by George Rosen at M.I.T. Cecil Coker designed rules to control a low-dimensionality articulatory model in 1968. Paul Mermelstein and James Flanagan from Bell Labs also used articulatory synthesis in 1976. Articulatory synthesis, however, never took off, because formant synthesis was better understood at the time.

The advent of the digital computer prompted John Kelly and Louis Gerstman to create in 1961 the first phonemic-synthesis-by-rule program. John Holmes and his colleagues Ignatius Mattingly and John Shearme developed a rule program for a formant synthesizer at JSRU in England. The first full text-to-speech system was developed by Noriko Umeda in 1968 at the Electrotechnical Laboratory of Japan. It was based on an articulatory model and included a syntactic analysis module with sophisticated heuristics. The speech was quite intelligible, but monotonous and far from the quality of present systems.

In 1976, Raymond Kurzweil developed a unique reading machine for the blind, a computer-based device that read printed pages aloud. It was an 80-pound device that shot a beam of light across each printed page, converting the reflected light into digital data that was transformed by a computer into synthetic speech. It made reading of all printed material possible for blind people, whose reading has previously been limited to material translated into Braille. The work of Dennis Klatt of MIT had a large influence in the field. In 1979 together with Jonathan Allen and Sheri Hunnicut he developed the MITalk system. Two years later Klatt introduced his famous Klattalk system, which used a new sophisticated voicing source.

The early 1980s marked the beginning of commercial TTS systems. The Klattalk system was the basis of Telesensory Systems' Prose-2000 commercial TTS system in 1982. It also formed the basis for Digital Equipment Corporation's DECtalk commercial system in 1983, probably the most widely used TTS system of the twentieth century. The Infovox TTS system, the first multilanguage formant synthesizer, was developed in Sweden by Rolf Carlson, Bjorn Granstrom, and Sheri Hunnicutt in 1982, and it was a descendant of Gunnar Fant's OVE system. The first integrated circuit for speech synthesis was probably the Votrax chip, which consisted of cascade formant synthesizer and simple low-pass smoothing

circuits. In 1978 Richard Gagnon introduced an inexpensive Votrax-based Type-n-Talk system. The first work in concatenative speech synthesis was done in 1968 by Rex Dixon and David Maxey, where diphones were parametrized with formant frequencies and then concatenated. In 1977, Joe Olive and his colleagues at Bell Labs [41] concatenated linear-prediction diphones. In 1982 Street Electronics introduced the Echo system, a diphone concatenation synthesizer which was based on a newer version of the same chip as in the Speak-n-Spell toy introduced by Texas Instruments in 1980.

Concatenative systems started to gain momentum in 1985 with the development of the PSOLA prosody modification technique by France Telecom's Charpentier and Moulines. PSOLA increased the text coverage of concatenative systems by allowing diphones to have their prosody modified. The hybrid Harmonic/Stochastic (H/S) model of Abrantes [1] has also been successfully used for prosody modification. The foundation of corpus-based concatenative systems was developed by a team of researchers at ATR in Japan in the early 1990s [10, 27]. The use of a large database of long units was also pioneered by researchers at AcuVoice Inc. Other corpus-based systems have made use of HMMs to automatically segment speech databases, as well as to serve as units in concatenative synthesis [13, 24]. Microsoft integrated a concatenative TTS [24] in Windows 2000.

For more detailed description of speech synthesis development and history see, for example, [31] and [17] and references in these. A number of audio clips are available in Klatt [31] showing the progress through the early years. You can hear samples at the Smithsonian's Speech Synthesis History Project [35]. A Web site with comparison of recent TTS systems can be found at [33].

REFERENCES

[1] Abrantes, A.J., J.S. Marques, and I.M. Trancoso. "Hybrid Sinusoidal Modeling of Speech without Voicing Decision," *Proc. Eurospeech*, 1991, Genoa, Italy, pp. 231-234.

[2] Acero, A., "Source-Filter Models for Time-Scale Pitch-Scale Modification of Speech," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, Seattle, WA, pp. 881-884.

[3] Acero, A., "Formant Analysis and Synthesis Using Hidden Markov Models," *Eurospeech*, 1999, Budapest, pp. 1047-1050.

[4] Allen, J., M.S. Hunnicutt, and D.H. Klatt. *From Text to Speech: The MITalk System*, 1987, Cambridge, UK, University Press.

[5] ANSI, *Method for Measuring the Intelligibility of Speech Over Communication Systems*, 1989, American National Standards Institute.

[6] Arslan, L.M. and D. Talkin, "Speaker Transformation Using Sentence HMM Based Alignments and Detailed Prosody Modification," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1998. Seattle, pp. 289-292.

[7] Beutnagel, M., *et al.*, "The AT&T Next-Gen TTS System," *Joint Meeting of ASA*, 1999, Berlin, pp. 15-19.

[8] Bickley, C.A., K.N. Stevens, and D.R. Williams, "A Framework for Synthesis of Segments Based on Pseudoarticulatory Parameters" in *Progress in Speech Synthesis*, 1997, New York, Springer-Verlag, pp. 211-220.

[9] Bregler, C., M. Covell, and M. Slaney, "Video Rewrite: Driving Visual Speech with Audio," *ACM Siggraph*, 1997, Los Angeles, pp. 353-360.

[10] Campbell, W.N. and A.W. Black, "Prosody and the Selection of Source Units for Concatenative Synthesis" in *Progress in Speech Synthesis*, J.V. Santen, *et al.*, eds., 1996, pp. 279-292, Springer Verlag.

[11] Covell, M., M. Withgott, and M. Slaney, "Mach1: Nouniform Time-Scale Modification of Speech," *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, Seattle, WA, pp. 349-352.

[12] Crochiere, R., "A Weighted Overlap-Add Method of Short Time Fourier Analysis/Synthesis," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1980, **28**(2), pp. 99-102.

[13] Donovan, R. and P. Woodland, "Improvements in an HMM-based Speech Synthesizer," *Proc. of the EuroSpeech Conf.*, 1995, Madrid, pp. 573-576.

[14] Dudley, H. and T.H. Tarnoczy, "The Speaking Machine of Wolfgang von Kempelen," *Journal of the Acoustical Society of America*, 1950, **22**, pp. 151-166.

[15] Dutoit, T., *An Introduction to Text-to-Speech Synthesis*, 1997, Kluwer Academic Publishers.

[16] Egan, J., "Articulation Testing Methods," *Laryngoscope*, 1948, **58**, pp. 955-991.

[17] Flanagan, J., *Speech Analysis, Synthesis and Perception*, 1972, New York, Springer-Verlag.

[18] Galanes, F.G., *et al.*, "Generation of Lip-Synched Synthetic Faces from Phonetically Clustered Face Movement Data," *AVSP*, 1998, Terrigal, Australia.

[19] Gibbon, D., R. Moore, and R. Winski, *Handbook of Standards and Resources for Spoken Language Systems*, 1997, Berlin & New York, Walter de Gruyter Publishers.

[20] Goldstein, M., "Classification of Methods Used for Assesment of Text-to-Speech Systems According to the Demands Placed on the Listener," *Speech Communication*, 1995, **16**, pp. 225-244.

[21] Hon, H., *et al.*, "Automatic Generation of Synthesis Units for Trainable Text-to-Speech Systems," *Int. Conf. on Acoustics, Signal and Speech Processing*, 1998, Seattle, WA, pp. 293-296.

[22] House, A., *et al.*, "Articulation Testing Methods: Consonantal Differentiation with a Closed Response Set," *Journal of the Acoustical Society of America*, 1965, **37**, pp. 158-166.

[23] Howard-Jones, P., *SOAP, Speech Output Assessment Package*, 1992, ESPRIT SAM-UCL-042.

[24] Huang, X., *et al.*, "Whistler: A Trainable Text-to-Speech System," *Int. Conf. on Spoken Language Processing*, 1996, Philadephia, pp. 2387-2390.

[25] Huang, X., *et al.*, "Recent Improvements on Microsoft's Trainable Text-To-Speech System - Whistler," *Int. Conf. on Acoustics, Signal and Speech Processing*, 1997, Munich, Germany, pp. 959-962.

[26] Hunt, A.J. and A.W. Black, "Unit Selection in a Concatenative Speech Synthesis System Using a Large Speech Database," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1996, Atlanta, pp. 373-376.

[27] Iwahashi, N., N. Kaiki, and Y. Sagisaka, "Concatenation Speech Synthesis by Minimum Distortion Criteria," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Francisco, pp. 65-68.

[28] Kain, A. and M. Macon, "Text-to-Speech Voice Adaptation from Sparse Training Data," *Int. Conf. on Spoken Language Systems*, 1998, Sydney, Australia, pp. 2847-2850.

[29] Klatt, D. and L. Klatt, "Analysis, Synthesis and Perception of Voice Quality Variations among Female and Male Talkers," *Journal of the Acoustical Society of America*, 1990, **87**, pp. 737-793.

[30] Klatt, D.H., "Software for a Cascade/Parallel Formant Synthesizer," *Journal of Acoustical Society of America*, 1980, **67**, pp. 971-995.

[31] Klatt, D.H., "Review of Text to Speech Conversion for English," *Journal of the Acoustical Society of America*, 1987, **82**, pp. 737-793.

[32] Krishnamurthy, A.K. and D.G. Childers, "Two Channel Speech Analysis," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1986, **34**, pp. 730-743.

[33] LDC, *Interactive Speech Synthesizer Comparison Site*, 2000, http://morph.ldc.upenn.edu.

[34] Maachi, M., "Coverage of Names," *Journal of the Acoustical Society of America*, 1993, **94**(3), pp. 1842.

[35] Maxey, H., *Smithsonian Speech Synthesis History Project*, 2000, http://www.mindspring.com/~dmaxey/ssshp/.

[36] McGurk, H. and J. MacDonald, "Hearing Lips and Seeing Voices," *Nature*, 1976, **264**, pp. 746-748.

[37] Mermelstein, P. and M.R. Schroeder, "Determination of Smoothed Cross-Sectional Area Functions of the Vocal Tract from Formant Frequencies," *Fifth Int. Congress on Acoustics*, 1965.

[38] Moulines, E. and F. Charpentier, "Pitch-Synchronous Waveform Processing Techniques for Text-to-Speech Synthesis Using Diphones," *Speech Communication*, 1990, **9**(5), pp. 453-467.

[39] Moulines, E. and W. Verhelst, "Prosodic Modifications of Speech" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds. 1995, Elsevier, pp. 519-555.

[40] Nye, P. and J. Gaitenby, *The Intelligibility of Synthetic Monosyllabic Words in Short, Syntactically Normal Sentences*, 1974, Haskins Laboratories.

[41] Olive, J., "Rule Synthesis of Speech from Dyadic Units," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1977, Hartford, CT, pp. 568-570.

[42] Olive, J.P., A. Greenwood, and J.S. Coleman, *Acoustics of American English Speech: a Dynamic Approach*, 1993, New York, Springer-Verlag.

[43] Oliveira, L., "Estimation of Source Parameters by Frequency Analysis," *Proc. of the Eurospeech Conf.*, 1993, Berlin, pp. 99-102.

[44] Plumpe, M. and S. Meredith, "Which Is More Important in a Concatenative Text-to-Speech System: Pitch, Duration, or Spectral Discontinuity," *Third ESCA/COCOSDA Int. Workshop on Speech Synthesis*, 1998, Jenolan Caves, Australia, pp. 231-235.

[45] Roucos, S. and A. Wilgus, "High Quality Time-Scale Modification of Speech," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1985, pp. 493-496.

[46] Sagisaka, Y. and N. Iwahashi, "Objective Optimization in Algorithms for Text-to-Speech Synthesis" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds., 1995, pp. 685-706, Elsevier.

[47] Santen, J.V., "Combinatorial Issue in Text-to-Speech Synthesis," *Proc. of the Eurospeech Conf.*, 1997, Rhodes, Greece, pp. 2511-2514.

[48] Sproat, R., *Multilingual Text-To-Speech Synthesis: The Bell Labs Approach*, 1998, Dordrecht, Kluwer Academic Publishers.

[49] Stylianou, Y., "Assessment and Correction of Voice Quality Variabilities in Large Speech Databases for Concatenative Speech Synthesis," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1999, Phoenix, AZ, pp. 377-380.

[50] Stylianou, Y., J. Laroche, and E. Moulines, "High Quality Speech Modification Based on a Harmonic + Noise Model," *Proc. Eurospeech*, 1995, Madrid.

[51] Syrdal, A., A. Conkie, and Y. Stylianou, "Exploration of Acoustic Correlates in Speaker Selection for Concatenative Synthesis," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 2743-2746.

[52] Syrdal, A., *et al.*, "Voice Selection for Speech Synthesis," *Journal of the Acoustical Society of America*, 1997, **102**, pp. 3191.

[53] Turing, A.M., "Computing Machinery and Intelligence," *Mind*, 1950, **LIX**(236), pp. 433-460.

[54] Voiers, W., A. Sharpley, and C. Hehmsoth, *Research on Diagnostic Evaluation of Speech Intelligibility*, 1975, Air Force Cambridge Research Laboratories, Bedford, MA.

[55] Yi, J., *Natural Sounding Speech Synthesis Using Variable-Length Units*, Masters' Thesis in EECS Dept. 1998, MIT, Cambridge, MA.

PART V

SPOKEN LANGUAGE SYSTEMS

CHAPTER 17

Spoken Language Understanding

Formal methods for describing sentences are discussed in Chapter 11. While the context-free grammars and *n*-gram models have mathematically well-understood formulations and bounded processing complexity, they are only partial aids in interpreting semantic meaning of the sentences. Suppose a recognizer correctly transcribes a series of spoken words into the written form—the system still has no idea what to do, because there is often no direct mapping between a sequence of words (or the syntactic structure of the sentence) and the functions that the system provides. The problem can also be approached from the opposite direction, i.e., solving the recognition problem itself may require semantic analysis, or domain and language knowledge for perplexity reduction.

What is meant by *meaning* or *understanding*? We could define it operationally: understanding is when a computer we interact with understands our desires and delivers the goods. Or we could define it propositionally: the computer has an accurate and unambiguous representation of *who did what to whom* corresponding to a real-world situation. In practice, the concept of understanding is situation dependent, and both conceptions above have their places. Meaning is often a constellation that emerges from a conversational environment.

There are four main interacting areas in *spoken language understanding* (SLU) systems from which meaning arises:

- **Intent**: goals of listener and speaker in the interaction
- **Context**: the pressures, opportunities, interruptions, etc. of the interaction scene and communication media
- **Content**: the propositional or literal content of each utterance and the discourse as a whole
- **Assumptions**: what each participant can assume about other participants' mental state, abilities, limitations, etc.

In this chapter we take a functional view of SLU systems, where the basic principle is to link linguistic expressions to concrete real-world entities. Currently, only with systems that are restricted to limited domains can understanding be attempted in practice. The domain restrictions allow the creation of specific, highly restricted language models and fully interpretable semantic descriptions that enable high accuracy and usability. Such systems are in contrast to speech recognition approaches that use large dictionaries, but make relatively *loose* or probabilistic predictions of word sequences for general dictation/transcription.

The need for spoken language understanding is double-edged. We generally want more than a string of word choices as a system's output. Instead, we want some interpretation of the word string that helps in accomplishing complex tasks. At the same time, being able to determine *what makes sense in context*, what is more or less likely as a speaker's input, could make a major contribution toward improving speech recognition word accuracy and search efficiency. SLU systems that combine the semantic precision of grammars with the probabilistic coverage of statistical language models can guide recognition and simultaneously control interpretation.

Figure 1.4 in Chapter 1 illustrates a basic SLU system architecture. The SLU problem can be broadly viewed as yet another pattern recognition problem. Namely, given a speech input **X**, the objective of the system is to arrive at actions **A** (including dialog messages and necessary operations) so that the cost of choosing **A** is minimized. Assuming uniform cost, the optimal solution, known as the maximum a posteriori (MAP) decision, can be expressed as

$$\begin{aligned} \mathbf{A}^* &= \arg\max_{\mathbf{A}} P(\mathbf{A} \mid \mathbf{X}, S_{n-1}) \\ &\approx \arg\max_{\mathbf{A}, S_n} P(\mathbf{A} \mid S_n) \sum_{\mathbf{F}} P(S_n \mid \mathbf{F}, S_{n-1}) P(\mathbf{F} \mid \mathbf{X}, S_{n-1}) \end{aligned} \tag{17.1}$$

where **F** denotes semantic interpretation of **X** and S_n, the discourse semantics for the n^{th} dialog turn.

Based on the formulation in Eq. (17.1), a dialog system consists of three pattern recognition components:

- **Semantic parser**—use semantic model $P(\mathbf{F} \mid \mathbf{X}, S_{n-1})$ to convert **X** into a collection of semantic objects **F**. This component is often further decomposed into *speech recognition* module (converting speech signal **X** into textual sen-

tence **W**) and *sentence interpretation* module (parsing sentence **W** into semantic objects **F**). Since the collection of semantic objects **F** is in the linguistic level, it is often referred to as surface semantics.

- **Discourse analysis**—use discourse model $P(S_n \mid \mathbf{F}, S_{n-1})$ to derive new dialog context S_n based on the per-turn semantic parse **F** and the previous context S_{n-1}.
- **Dialog manager**—iterate through the possible actions and pick the most suitable one. The quantitative measures governing operations for dialog manager is called the *behavior model*, $P(\mathbf{A} \mid S_n)$.

The pattern recognition framework can be generalized to multimodal systems as well. For input other than speech signal, you only need to replace the input **X** in the semantic parser with input from an associated modality, e.g., **X** could be input from keyboard typing, mouse clicking, pen input, video, etc. As long as the new semantic parser (replacing speech recognizer and sentence interpretation modules in Figure 1.4) can convert it into appropriate semantic representation, the rest of the system can be identical. Similarly, for different output modality, you just need to replace *message generation* and *text-to-speech* modules with a new rendering mechanism.

In this chapter we first describe the characteristics of spoken languages in comparison with written languages. The structure of dialog is discussed in Section 17.2. Understanding is the most fundamental issue in the field of *artificial intelligence*. The kernel of understanding lies on the representation of semantics (knowledge). Several state-of-the-art semantic representation schemes are discussed in Section 17.3. Based on the architecture of SLU systems illustrated in Chapter 1 (Figure 1.4), major modules are discussed in detail, with the Dr. Who SLU system serving as an example to illustrate important issues.

17.1. WRITTEN VS. SPOKEN LANGUAGES

To construct SLU systems, we need to understand the characteristics of spoken languages. It is worth thinking about possible differences between spoken and written use of language that could be relevant to developing spoken language systems. The following is a typical example of two-agent, task-oriented dialog in action:

Sys: Flight reservation service, how can I help you?
User: One ticket to Honolulu, please
Sys: Anchorage to Honolulu, when would you like to leave?
User: Next Thursday
Sys: Next Tuesday, the 30th of November; and at what time?
User: No, Thursday, December 2nd, late in the evening, and make it first class.
Sys: OK, December 2nd United flight 291, first class. Will you need a car or hotel?
User: No.

17.1.1. Style

In both spoken and written forms, a communicative setting is established. Both forms involve participants. In the case of written language, we normally expect passivity on the part of the addressee(s), though with e-mail bulletin boards, Web chat rooms, and the like, this assumption can be challenged. The communicative event emerges from personal characteristics of the participants—their mood, goals, and interests. Communication depends both on the actual world knowledge and shared knowledge of the participants and on their *beliefs* about one another's knowledge. Communication can be influenced by the setting in which it takes place, whether in spoken or written mode. Also, different subchannels of supportive communication, such as visual aids, gesture, etc., may be available.

A number of grammatical and stylistic attributes have been found to distinguish conversational from written forms. Biber's analysis [8] distinguishes not only a dimension of modality, but also formality; for example a panel discussion is a relatively formal, yet spoken, modality. Some typical features for which distinctions can be measured include the number of passives, the number of pronouns, the use of contractions, and the use of nominalized forms.[1] An example of the grammatical and stylistic difference continuum that Biber uses is illustrated in Figure 17.1. The variation can be measured along multiple orthogonal scales for different genres. In the SLU case, style can be orthogonal to the modality (dialog or dictation, spoken or written). A crossover case is speech dictation used to create a written document that may never be orally rendered again.

Fortunately, much of the disjuncture between spoken and written forms in grammatical style and lexical choice can be handled by training task-specific and modality-specific language models for the recognizer. For this, only the data need vary, not necessarily the modeling methods. In Figure 17.1, the right-hand side is toward the spoken style, while the

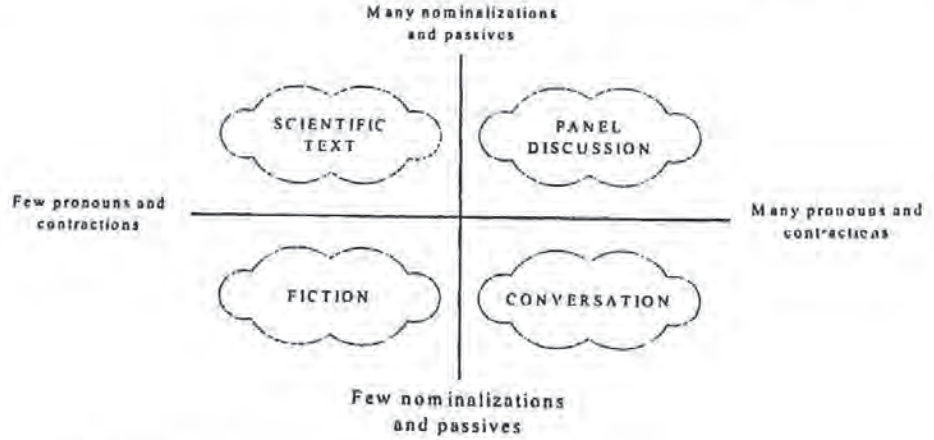


Figure 17.1 Dimensions of written vs. spoken language variation.

[1] Nominalization is a stylistic device whereby a main verb is converted to a noun. For example, *The dean rejected the application unexpectedly* may become: *The rejection of the application by the dean was unexpected.*

left-hand side is toward the written one. The difference in styles is best illustrated by the fact that the statistical n-gram trained from newspaper text exhibits a very high perplexity when evaluated on conversational Air Travel Information Service (ATIS) texts.

17.1.2. Disfluency

Another issue for spoken language processing is *disfluency*. Spoken dialogs show a large set of problems such as interruptions, corrections, filled pauses, ungrammatical sentences, ellipses, and unconnected phrases. These challenges are unique to spontaneous spoken input and represent a possible further degradation of speech recognizer performance, as current systems often rely on acoustic models trained from read speech, and language models trained on written text corpora. When speech input is used as dictation for document creation, of course, the models would presumably be most appropriate.

There are a number of types of disfluencies in human-human dialog, and, possibly to a lesser extent, in human-computer dialog as well. The more common types of nonlinguistic disfluencies are listed below:

- Filled pauses: *um*
- Repetitions: *the – the*
- Repairs: *on Thursday – on Friday*
- False Starts: I like – what I always get is ...

Early work in discourse led to the determination that discourses are divided into *discourse segments*, much as sentences are divided into phrases [18]. In the experiments of [46], CART methods (see Chapter 4) were used to predict occurrence and location of each of the above types of disfluency. A tree was trained from labeled corpora for each type, and the resulting system classified each interword boundary as having no disfluency or one or more of the above types. The feature types used to derive the classification questions included duration of vocalic regions and pauses, fundamental frequency and its derivatives, signal-to-noise ratios, and distance of the boundary from silence pauses. The basic classification task consisted in selecting each of the four disfluency types listed above (D), given the list of prosodic features ($\mathbf{X}$), by computing the maximum of $P(D|\mathbf{X})$. When decision trees were used to supplement the language-model scoring of hypothesis word strings, performance improved.

A number of intriguing regularities were also observed in this work. For example, it was noted that the marked (less common) pronunciation of *the* - */dh iy/* was often used just prior to a production problem, e.g., a disfluent silent pause. Also, it has been noted that the leftmost word of a major phrase or clause is likely to be repeated, as in their example, "*I'll I'll do what I can.*" Continued research on disfluencies may contribute an important secondary knowledge source to supplement text-based language models and *read speech* acoustic models in the future.

17.1.3. Communicative Prosody

Prosodic attributes of utterances, such as fundamental frequency and timing (cf. Chapter 15), are crucial cues for detecting disfluency. However, prosody can be deliberately manipulated by speakers for deep communicative purposes as well. The speaker may intentionally or subconsciously manipulate the fundamental frequency, timing, and other aspects of voice quality to communicate attitude and emotion. If a conversational interface is equipped to recognize and interpret prosodic effects, these can be taken into account for understanding.

In addition to serving as a disfluency detector, as described above, prosodic analysis modules could aid recognition of:

- Utterance type—declarative, *yes-no* questions, *wh*-question, etc.
- Speech act type—directive, commissive, expressive, representative, declarative, etc. Different speech acts will be described in Section 17.2.2.
- Speaker's attentional state.
- Speaker's attitude toward his/her utterance(s).
- Speaker's attitude to system presentations.
- Speaker's mood or emotional state.

Consider the simple utterance *OK*. This may be used along a range of attitudes and meanings, from *bored contempt*, to *enthusiastic agreement*, to *questioning* and *uncertainty*. The interpretation will depend on both the dialog state context of expectations-to-date and the prosody. Generally, a higher relative F0 in a wider range correlates with submission, involvement, questioning, and uncertainty, while a lower relative F0 in a narrower range correlates with dominance, detachment, assertion, and certainty. Even though acknowledgement words such as *yeah* and *ok* are potentially ambiguous among: true agreement; intention of the listener to initiate a new turn; and simple passive encouragement from listener to speaker, the system may rely on a longer duration and greater pitch excursion of a lexical item such as *yeah* or *ok* to hypothesize genuine agreement with a speaker statement, as opposed to mere acknowledgement.

In addition to correlating with speech acts, F0 and timing can be used to demarcate utterance and turn segments. For example, certain boundary pitch movements and phonemic lengthening systematically signal termination of clauses. In general, a fall to the very bottom of a speaker's range, in a prepausal location, coincides with a clause or sentence boundary. A sharp upturn preceding a significant silence gives an impression of incompletion, perhaps signaling a yes-no question, or may signal an intention by the speaker to carry on with further information, as in the case of list intonation.

The disfluent and prosodic characteristics of the conversational speech are in general very distinct from those of read speech. Thus, we often refer conversational speech as *spontaneous* speech.

17.2. DIALOG STRUCTURE

The analysis methods discussed in Chapter 11 are focused on single sentences. They are steps along the way, helping to map vague and ambiguous natural language constructions into precise *logical forms* of propositions. In reality, however, the communicative function of language is not a simple, uncomplicated assembly of discrete logical propositions derived from sentences in a one-to-one fashion. In *discourse*, each sentence or utterance contributes to a larger abstract information structure that the user is attempting to construct. Sometimes feedback is directly available to the user or can be inferred. These considerations take us beyond the process of mapping of isolated utterances into logically structured propositions (with simple truth-values).

A set of principles, known collectively as the cooperative principle, is introduced by Grice [9]. It consists of a set of conversational maxims, the violation of which may lead to a breakdown in communication.

GRICE'S MAXIMS

Quantity: speaker tries to be as informative as possible, and gives only as much information as needed
Quality: speaker tries to be truthful, and does not give information that is false or that is not supported by evidence
Relevance: speaker tries to be relevant, and says things that are pertinent to the discussion
Manner: speaker tries to be as clear, as brief, and as orderly as possible, and avoids obscurity and ambiguity

In general, there are five main domains of operation that must be modeled for intelligent conversation systems, although all these areas are linked:

- **Linguistic forms**: all the knowledge a human-computer dialog system requires to perform semantic and syntactic analysis and generation of actual utterances.
- **Intentional state**: goals related to both the task (*Show me all flights ...)*, and the dialog process itself (*Please repeat ...*) of the users.
- **Attentional state**: the set of entities at any point in time that can be felicitously discussed and referred to, i.e., the main topic of any stage of interaction.
- **World knowledge**: common sense knowledge and inference. Examples include temporal and spatial concepts and the relation of these to linguistic forms.
- **Task knowledge**: all information relevant to achieving the user's goal in a complete, correct, and efficient fashion.

Human-computer dialog is multiagent communication. Each agent has to form a notion of the other's beliefs, desires, and knowledge, all of which underlie their intentions, plans, and actions. In a limited application, deep inference may not be possible, and the system may have more or less *hardwired* assumptions about the user, the interaction, and the flow of action. An interaction may be controlled by the system's own rigid schedule of information acquisition. In the research community, such a dialog system—always leading the interaction flow control and not allowing the user to digress—is called *system initiative*. On the other hand, a dialog system is called *user initiative* if it always lets the user decide what to do next. It is often more natural, however, to allow for *mixed initiative* systems, where interaction starts with a user's query or command and the system attempts to derive, via inference or further questioning of the user, all information needed to understand and process a complete transaction. When the user knows clearly what he wants and the system has no trouble catching up, the user is in the driver's seat. However, when the system detects that the user is in a state of confusion, or when it has trouble getting user's intention, the machine will offer guidance or negotiate with the user to steer the dialog back on track.

Whether it is system-initiative, user-initiative, or mixed-initiative, however, the fundamental structure of dialog consists of initiative-response pairs as indicated in Figure 17.2. The *Initiatives* (I) are often issued by users while the *Responses* (R) are issued by the system. As shown in Section 17.2.2, there are many types of Initiatives and Responses and there may also be higher-order structure subsuming a number of I/R pairs in a dialog.

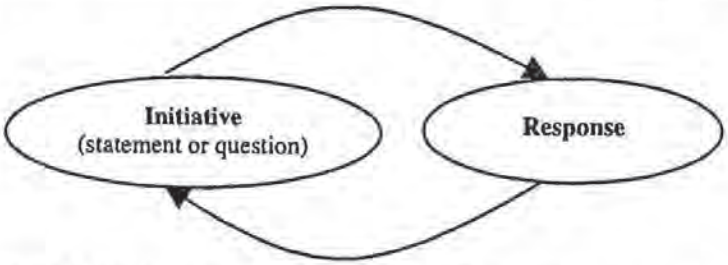


Figure 17.2 The fundamental structure of dialog: initiative and response.

17.2.1. Units of Dialog

The words uttered in a dialog are the surface manifestation of a complex underlying layer of participants' shared interaction knowledge and desires, even when one participant is a computer simulation. It is natural to assume that the *sentence* is a clear and simple chunking unit for dialog, by analogy with written communication. However, since sentences are artificially delimited in written text, researchers in dialog communication usually speak of the *utterance* as the basic unit. An Initiative or Response could consist of one or more utterances. The utterance, however, is not necessarily trivial to define.

It is tempting to posit an equivalence of the notion utterance with *turn*, i.e., an uninterrupted stream of speech from one participant in a dialog. This formulation makes it easy to segment dialog data into utterance units—they are just each speaker's turns. The downside

is that this kind of *utterance* possibly spans grammatical units that really do have some rough correspondence to traditional sentences (predicate-argument structures), and to which much of the hard-won gains in natural language processing would apply fairly directly. Thus, the use of *turn* as synonymous with utterance unit is probably too broad, though the *turn* may be independently useful for higher-level segmentation.

Turns are building blocks for constructing a common task-oriented understanding among participants. This process is called *grounding*, a set of discourse strategies by which dialog actors (humans in most current research) attempt to achieve a common understanding, and come to feel confident of the other participants' understanding. In other words, conversational partners are finding or establishing *common ground.*

Turns may have their own typology. For example, a *speaking turn* conveys new information, while a *back-channel turn* is limited to acknowledgement or encouragement, such as *OK, really*?, etc. The turns themselves consist of linguistic substructures, such as sentences, clauses, and phrases. If we assume that turns can be segmented, by grammatical and/or prosodic criteria, into utterances, we can then begin to explore distinct types of utterances, their properties, and their communicative functions.

Finally, dialogs are not flat streams of unrelated turns or utterances. The utterances that make up a dialog have higher-order affiliations with one another. A discourse segment would thus consist of groups of related utterances organized around a common dialog subtask, perhaps spanning turns.

17.2.2. Dialog (Speech) Acts

In simpler applications, the amount and sophistication of world knowledge can be kept to a minimum, and attentional state can be modeled simply as the complete set of task-specific entities. A layer of structure has therefore been sought to link linguistic forms with task knowledge or operations in a theoretically appropriate fashion, which also yields an implicit understanding of intentional state. This is necessary because the function of utterances in discourse cannot be predicted strictly on the basis of their surface grammatical form. The layer of structure that can abstract away from linguistic details and can map well to formulation of goals is called *dialog acts* [42]. Dialog acts are also often referred to as *speech acts* that group infinite families of surface utterances into abstract functional classes. They are traditionally classified into five broad categories:

- **Directive**: The speaker wants the listener to do something.
- **Commissive**: The speaker indicates that he or she will do something in the future.
- **Expressive**: The speaker expresses his or her feelings or emotional response.
- **Representative**: The speaker expresses his or her belief about the truth of a proposition.
- **Declarative**: Speaker's utterance causes a change in external, nonlinguistic situation.

Table 17.1 A simple dialog analyzed with dialog acts.

Utterance	Form	Function
Do you have the butter?	Y/N-question	REQUEST-ACT
Sure. (passes butter)	statement	COMMIT-TO-ACTION-ACT

While this analysis is somewhat coarse, speech act theory has influenced all current work on human-computer dialog, except the very simplest and most rigid systems. Because dialog functions can be realized with a bewildering variety of linguistic forms, researchers have posited systems of functional abstractions. Speech acts are functional abstractions over variation in utterance form and content. Declare, request, accept, contradict, withdraw, acknowledge, confirm, and assert are all examples of speech acts—things we are attempting to do with speech. An example of dialog acts and their relation to syntactic form is shown in the two-turn dialog in Table 17.1.

The relation between speech acts and linguistic forms (utterances) is a many-to-many mapping. That is, a single linguistic form, such as *OK*, could realize a large number of speech acts, such as *request for acknowledgment* or *confirm*, etc. Likewise, a single speech act, such as *agreement*, could be realized by a variety of linguistic forms, such as *ok, yes, you bet*, etc. In a particular application, special task-specific speech acts may be used to supplement the universal inventory.

Tagging of dialog utterance data with speech-act labels can add useful information for training models. There are a number of ways that dialog-act analysis could be useful:

- *Speech recognition*: Given a history, we can predict the most likely dialog act type for the *next* utterance, so that specialized language models may be applied.
- *Spoken language understanding*: Given a history, and a transcription/parse of the current utterance, we can identify the user's intentions, so that the system can respond appropriately.
- *Semantic authoring*: It is tedious for each team designing or customizing a new application area for SLU to have to wrack their brains for all the ways a given generic function, such as *request* or *confirm*, might be realized linguistically. Libraries of speech acts (form-to-function mappings) may reduce the work in new-domain adaptation of systems.

An example of a practical dialog tagging system that could be the foundation of speech-act analysis is the *Dialog Act Markup in Several Layers* (DAMSL) system [14], which has been used and adapted for a variety of projects. This is a system for annotating dialog transcriptions with speech-act labels and corresponding structural elements. The structuring is based on a loose hierarchy of: discourse segment, turn, utterance, and speech act. The tags applied to utterances fall into three basic categories:

- **Communicative Status**: records whether the utterance is intelligible and whether it was successfully completed. It is mainly used to flag problematic utterances that should be used for data modeling only with caution—Uninterpretable, Abandoned, or Self-talk. *Uninterpretable* is self-explanatory. *Abandoned* marks utterances that were broken off without, crucially, adding any information to the dialog. *Self-talk* is a note that, while an utterance may contain useful information, it did not appear to be intentionally communicated. Self-talk can be considered reliable only when the annotator is working from speech data.
- **Information Level**: a characterization of the semantic content of the utterance. This is used to specify the kind of information the utterance mainly conveys. It includes *Task* (Doing the task), *Task-management* (Talking about the task), *Communication-management* (Maintaining the communication), and *Other-level*. Task utterances relate directly to the business of the transaction and move it toward completion. Task-management utterances ask or tell about the task, explain it perhaps, but do not materially move it forward. Communication-management utterances are about the dialog process and capabilities. The Other level is for unclear cases.
- **The Forward/Backward Looking Function**: how the current utterance constrains the future/previous beliefs and actions of the participants and affects the discourse. Forward Looking functions introduce new information or otherwise move the dialog or task completion forward, while Backward Looking Functions are tied to an antecedent, a prior utterance which they respond to or complete. This distinction is the DAMSL reflection of the common observation that dialogs have a tendency to consist of Initiation/Response pairs. The core of the system is the set of particular act types. The core Forward/backward Looking tags are listed in Table 17.2 and Table 17.3.

Table 17.2 Forward looking tags.

Forward Looking Tags	Example
assert	I always fly first class.
reassert	No, as I said, I always fly first class.
action-directive	Book me a flight to Chicago.
open-option	There's a red-eye flight tonight ...
info-request	What time is it?, Tell me the time.
offer	I can meet at 3 if you're free.
commit	I'll come to your party.
conventional opening	May I help you?
conventional closing	Goodbye.
explicit-performative	Thank you, I apologize.
exclamation	Ouch! Darn!

Table 17.3 Backward looking tags.

Backward Looking Tags	Example
accept	(Will you come?) Yes. [and/or, I'll be there at 10.]
accept-part	(Will you come with your wife?) I'll come, she may be busy.
reject	(Will you come?) No.
reject-part	(Want fries and a shake with that burger?) Just the hamburger and fries, please.
maybe	Maybe.
signal-nonunderstanding	What did you say?
acknowledgment	OK.
answer	(Can I fly nonstop from Anchorage to Kabul?) No.

Multiple tags may appear on any given utterance. In the example shown in Figure 17.3, B's utterance is coded as opening the option of buying (from B), asserting the existence of the sofas, and functioning as an offer or solicitation.

Action-directive	`A: Let's buy the living room furniture first.`
Open-option/Assert/Offer	`B: OK, I have a red sofa for $150 or a blue one for $200`

Figure 17.3 A tagged dialog fragment.

The DAMSL system is actually more complex than the example demonstrated above, since subsets of the tags are grouped into mutually exclusive options for a given general speech function. For example, there is a general *Agreement* function, under which the *accept*, *accept-part*, *reject*, and *reject-part* tags are grouped as mutually exclusive options. Above the level of those groupings, however, a single utterance can receive multiple nonexclusive tags. For example, as illustrated in Figure 17.4, the assistant may respond with a countersuggestion (a kind of action-directive) that rejects part of the original command.

Action-directive	`utt1 oper: Take the train to Avon via Bath`
Action-directive/Reject-part(utt1)	`utt2 asst: Go via Corning instead.`

Figure 17.4 A tagged dialog fragment, showing that utterances can be tagged with multiple nonexclusive tags.

The prototypical dialog turn unit in simple applications would be the I/R pair *info-request/answer*, as in the interaction shown in Figure 17.5 between an operator (planner) and an assistant regarding railroad transport scheduling [1].

The example in Figure 17.5 illustrates a dialog for a railway-scheduling task. The turns are numbered T1–T4, the utterances within turns are also numbered sequentially, and the speaker identity alternates between oper: and asst:. The tagging is incomplete, because, for example, within the *ansl* sequence, each utterance is performing a function, asserting, acknowledging, etc. The example in Figure 17.6 is a more completely annotated fragment, omitting turn numbers.

```
info-req    T1 utt1 oper:   where are the engines?
ansl        T2 utt2 asst:   there's an engine at Avon
|           T3 utt3 oper:   okay
|           T4 utt4 asst:   and we need
               utt5 asst:   I mean there's another in Corning
```

Figure 17.5 A tagged dialog fragment in railroad transport scheduling.

```
info-req/assert      utt1 oper:  and it's gonna take us also an
                                 hour to load boxcars right
ans/accept(utt1)     utt2 asst:  right
assert               utt3 oper:  and it's gonna take us also an
                                 hour to load boxcars
accept(utt1)         utt4 asst:  right
```

Figure 17.6 A tagged dialog fragment, showing backward-looking utterances.

The example in Figure 17.6 shows backward-looking utterances, where the relevant antecedent in the dialog is shown (in parentheses) as part of the dialog coding.

More elaborate variants of DAMSL have been developed that extend the basic system presented here. Consider, for example, the SWITCHBOARD Shallow-Discourse-Function Annotation SWBD-DAMSL [27]. This project used a shallow discourse tag set of 42 basic tags (frequent composed tags from the large set of possible multitags) to tag 1155 5-minute conversations, comprising 205,000 utterances and 1.4 million words, from the SWITCHBOARD corpus of telephone conversations. Distributed by the Linguistic Data Consortium[2] [28], SWITCHBOARD is a corpus of spontaneous conversations that addresses the growing need for large multispeaker databases of telephone bandwidth speech. The corpus contains 2430 conversations averaging 6 minutes in length—in other words, over 240 hours of recorded speech, and about 3 million words of text, spoken by over 500 speakers of both genders from every major dialect of American English.

More detailed tags are added to DAMSL to create SWBD-DAMSL, most of which are elaborations of existing DAMSL broad categories. For example, where DAMSL has the simple category *answer*, SWBD-DAMSL has: *yes answer*, *no answer*, *affirmative non-yes*

[2] http://www.ldc.upenn.edu

answer, negative non-no answers, other answers, no plus expansion, yes plus expansion, statement expanding y/n answer, expansions of y/n answers, and *dispreferred answer.* SWBD-DAMSL is intended for the annotation and learning of structure in human-human dialog, and could be considered overkill as a basis for describing or constructing grammars for most limited-domain human-computer interactions of today. But the more sophisticated agent-based services of the future will need to assume ever-greater linguistic sophistication along these lines.

One fact noted by the SWBD-DAMSL researchers, which may not apply directly to task-directed human-computer interactions but which casts interesting light on human communication patterns, is that out of 1115 conversations studied, simple nonopinion statements and brief acknowledgements together constituted 55% of the conversational material! If statements of opinion (including simple stuff like *I think it's great!*), expressions of agreement (*That's right!*), turn breakoffs and no-content utterances (*So...*), and appreciative acknowledgements (*I can imagine.*) are added to this base, 80% of the utterances are accounted for. This relative poverty of types may bode well for future attempts to annotate and predict utterance function automatically. The DAMSL scheme is challenging to apply automatically, because it relies on complex linguistic and pragmatic judgments of the trained annotators.

17.2.3. Dialog Control

The system's view of how the dialog should proceed is embodied in its management strategy. Strategy is closely connected to the concept of *initiative* in dialog, meaning basically who is controlling the interaction. Different dialog initiatives are defined in Section 17.2. Initiative can be seen as a continuum from system controlled to user controlled. As background for the dialog management discussion, some important steps along this continuum can be identified:

- *System directs*—The system retains complete dialog control throughout the interaction. The system chooses the content and sequence of all subgoals and initiates any dialog necessary to obtain completion of information from the user for each transaction. This style is often referred as *system initiative.*
- *System guides*—The system may initiate dialog and may maintain a general plan, but the sequence of information acquisition from the user may be flexible, and system subgoals and plans may be modified in response to the user's input. This style is often referred as *mixed initiative.*
- *System inform*—The user directs the dialog and the system responds as helpfully as possible, which may include presentation of relevant data not specifically requested by the user but which the system believes could be helpful. This style also belongs to *mixed initiative,* though users control most of the dialog flows.
- *System accepts*—This is the typical human-computer interaction in traditional systems (whether it is a GUI-, command-line-, or natural language-based sys-

tem). The system interprets each command without any attempted inference of a deeper user plan, or recommendation of any suitable course of action. This style is referred as *user initiative*.

17.3. SEMANTIC REPRESENTATION

Most SLU systems require an internal representation of *meaning* that lends itself to computer processing. In other words, we need a way of representing semantic entities, which are used at every possible step. In general, an SLU system needs to deal with two types of semantic entities. The first type is *physical entities*, which correspond to the real-world entities. Such representation is often referred as knowledge representation in the field of artificial intelligence. The second type is *functional entities*, which correspond to a way of unambiguously representing the meaning or structure of situations, events, and concepts that can be expressed in natural language. Such representations are often similar to the *logical form* introduced in Chapter 2. Processing may include operations such as determining similarity or identity of events or entities, inference from a state of affairs to its logical consequences, and so on. Here, we briefly review some general properties of the common semantic representation frameworks.

17.3.1. Semantic Frames

Semantic objects are used to represent real world entities. Here, we assume that the domain knowledge conforms to a relational or objected-oriented database, of which the schema is clearly defined. We use the term *entity* to refer to a data item in the domain (a row in a database table), or a function (command or query) that can be fulfilled in the domain. A column in the database table is called an entity attribute, and each database table is given an entity type. Through a small subset of its attributes, an entity can be realized linguistically in many fashions. We call each of them a *semantic class*. For example, a person can be referred to in terms of her full name (*Angela*), a pronoun anaphora (*her*), or her relationships to others (*Christina's manager*). In this case, one can derive three semantic classes for the entity type.

Semantic classes can be viewed as a type definition to denote the objects and describe the relations that hold among them. One of the most popular representations for semantic classes is the *semantic frame* [31]—a type of representation in which a semantic class (concept) is defined by an entity and relations represented by a number of attributes (or slots) with certain values (the attributes are filled in for each instance). Thus, frames are also known as *slot-and-filler* structures.

We could, for example, define a generalized frame for the concept *dog*, with attributes that must be filled in for each particular instance of a particular dog. A type definition for the concept dog appears in Figure 17.7. Many different notational systems have been used for frames [51]. For these introductory examples, we use a simple declarative notation that should be fairly intuitive.

```
[DOG:]-
  [SUPERTYPE]->[mammal]
  [NAME]->()
  [BREED]->()
  [LOC]->()
  [Color]->()
```

Figure 17.7 A semantic frame representation for *dog*.

When we need to describe a particular dog, say *Lassie*, we create an *instance definition*, as shown in Figure 17.8. The knowledge base supporting a typical dialog system consists of a set of type definitions, perhaps arranged in an inheritance hierarchy, and a set of instances.

```
[DOG:]-
  [NAME]->(Lassie)
  [BREED]->(Collie)
  [LOC]->()
  [Color]-()
```

Figure 17.8 An instance of semantic frame *dog*.

Fillers in semantic frames can be attained by attachment of inheritance, procedures or default. Attributes in frame can typically be inherited, as the Lassie instance inherits mammalian properties from the DOG type definition. In some cases, properties of a particular dog may be dynamic. Sometimes *attached procedures* are used to fill dynamic slots. For example, the location of a dog may be variable, and if the dog has a *Global Positioning System* (GPS) chip in its collar, the LOC property could be continually updated by reference to the GPS calculations. Furthermore, procedures of the type `when-needed` or `when-filled` can also be attached to slots. Finally, some default value could provide a typical value for a slot when the information for that slot is not yet available. For example, it might be appropriate to set the default color for *dog* frame as white when such information is not available. For frames without a default-value slot, it is natural to define *mandatory* slots (slots' values must be filled) and optional slots (slots could have null value). For the *dog* frame, it is reasonable to assume the NAME slot should be mandatory while the COLOR slot can be optional.

Often *descriptions* can be attached to slots to establish constraints within or between frames. Description may have connectives, co-referential (description attached to a slot are attached to another) and declarative conditions. For example, the *return-date* slot of a *round-trip* itinerary frame must be no earlier than the *departure-date* slot, and this constraint can be specified by descriptions in both slots. Descriptions can also be inherited and are often implemented by a special procedure (different from the slot-filling procedure) attached to the slot.

The main motivation for having multiple semantic classes for each entity type is to better encapsulate the language, semantic, and behavior models based on the domain knowledge. While the entity relationships capture the domain knowledge, the semantic class hier-

archy represents how knowledge can be expressed in the semantics of a language and thus can cover linguistic variation. The concept of semantic objects/classes is similar to that of objects/classes in modern *object-oriented programming*. The semantic classes in Dr. Who [59, 60] are a good illustration of borrowing some important concepts from object-oriented programming to enhance the effectiveness and efficiency of using semantic objects/classes to represent domain knowledge and linguistic expressions.

The semantic grammar used in the Dr. Who Project [58] contains the definitions of semantic classes that refer to real-world or functional entities. A semantic class is defined as a semantic frame containing a set of slots that need to be filled with terminal (verbatim) words or with recursive semantic class objects. For example, *ByRel* is a semantic class that has the type PERSON. The semantic grammar specifies that it has two slots—one has to be filled with an object of a semantic class having the type PERSON, and the other has to be filled with an object of a semantic class having the type P_RELATION. On the other hand, the syntax grammar for this semantic class is specified by the <cfg> tags. Within <cfg> tags, several production rules can be specified to provide linguistic constraints (orders) of possible expressions for this semantic class. The syntactic aspect of semantic classes will be described further in Section 17.4.1.

17.3.1.1. Type Abstraction

As described above, a physical *entity* is an element in the real world that an application has to deal with and wishes to expose to the user via natural language. Since a physical entity can be referred to in many different ways, different semantic classes may have the same type. In Figure 17.9, a person can be referred to in terms of his name (*Peter*) or his relation to another person (*Peter's manager*); therefore, both semantic classes ByName and ByRel can share the same type, PERSON.

Semantic classes are designed to separate the essential attributes of a semantic object from its physical realizations. A semantic class may refer to an entity, and the entity is called the type of the semantic class. The attributes of a semantic class can, in turn, be semantic classes themselves. The concept behind semantic classes is identical to the mechanism known as type abstraction commonly employed in software engineering using a strongly typed programming language. Semantic class can be recursive, as demonstrated in Figure 17.9; a ByRel semantic class of type PERSON contains an attribute of PERSON type. Since the entities can be *nested*, i.e., a database column can in turn refer to another table, an attribute in the semantic class can also be an entity type. From an understanding point of view, a semantic class is an abstraction of the collection of semantic objects that have the same attributes and usually can be expressed, and hence be understood, in similar manners. Under this view, a semantic object is just an instantiation.

Another argument for type abstraction is that the multitude of semantic objects is usually a result of the numerous ways and perspectives that can be used to describe a physical entity. Quite often in an understanding system it is more important to correctly identify the entity of interest than to capture the mechanism that describes it. For instance, one may refer to a person by his name, job function, relations to others, or, in a multimodal environment,

by pointing to his photo on a display. All these references lead to semantic objects that are apparently distinct yet should be associated with the same physical entity. Accordingly, it is often useful to segregate the conceptual manifestation and its realizations into different layers of abstraction so that the semantic objects can be better organized and managed. Type abstraction allows the discourse sentence interpretation module to perform robust parsing, since sentence fragments can be parsed into its semantic class type that can be filled into slots with the same correspondent semantic type, as discussed in Section 17.4.1.

```
<!-- semantic class definition for ByRel that has type PER-
SON -->
<class type="PERSON" name="ByRel">
    <slot type="PERSON" name="person"/>
    <slot type="P_RELATION" name="p_relation"/>
    <cfg>
          <prod> [person] [p_relation] </prod>
          <prod> [p_relation] of [person] </prod>
    </cfg>
</class>
<!-- semantic class definition for ByName that has type PERSON too -->
<class type="PERSON" name="ByName">
    <slot type="FIRSTNAME" name="firstname"/>
    <slot type="LASTNAME" name="lastname"/>
    <cfg>
          <prod> [firstname] [lastname] </prod>
          <prod> [firstname] </prod>
          <prod> [lastname] </prod>
    </cfg>
</class>
<!-- semantic class definition for FIRSTNAME and LASTNAME -->
<verbatim type="FIRSTNAME"
    <cfg>
          <prod> john | john's | peter ... </prod>
    </cfg>
</verbatim >
<verbatim type="FIRSTNAME"
    <cfg>
          <prod> smith | smith's | shaw ... </prod>
    </cfg>
</verbatim >
<!-- semantic class definition for P_RELATION -->
<verbatim type="P_RELATION"
    <cfg>
          <prod> manager | father | mother | ... </prod>
    </cfg>
</verbatim >
```

Figure 17.9 The semantic classes of type PERSON as implemented in Dr. Who.

Finally, type abstraction provides a unified framework for resolving *relative expressions* in the discourse analysis module. Type matching often serves to impose strong constraints between real-world entities and relative expressions. The resolution of relative expressions is discussed in Section 17.5.

17.3.1.2. Property Inheritance

Introducing inheritance into the semantic class hierarchy further augments the multilayer abstraction mentioned above. Class A is said to inherit or be derived from class B if class A possesses all the attributes of class B. In this case, class A is called the derived class and class B the base class. Inheritance is a mechanism to propagate knowledge and properties through the structural relationships of semantic classes. It is crucial for many types of intelligent behavior, such as deducing presumed facts from general knowledge and assuming default values in lieu of explicit and specific facts.

Perhaps the strongest motivation to employ inheritance is to facilitate the multilayer abstraction mentioned above. Very often, a base class is constructed with the general properties of a type of semantic objects, and a collection of more specific classes are derived from the base class to support the various embodiments of the underlying type of the semantic objects. For example, a semantic class hierarchy for the reference to a person can have the methods (e.g., by name, job function) and the media (e.g., speech, handwriting) of reference as the first layer of derived classes. One can then cross-match the viable means (e.g., by name via speech, by name via handwriting) and develop the second layer of derived classes for use in the real applications.

17.3.1.3. Functionality Encapsulation

The goal of abstraction is to reduce the complexity in describing the world—in this case, the semantic objects and their relations. One can inspect the quality of abstraction by examining the extent to which the constructs, i.e., semantic classes, are self-contained, and how proliferating they have to become in order to account for novel scenarios. Studies in data structure and software engineering propose the notion of encapsulation, which suggest that individual attributes have local rather than global impacts. This principle also serves as a guideline in designing the semantic class.

Semantic class encapsulation can be elaborated in two aspects: syntactic and semantic. The syntactic encapsulation refers to the constraint that each attribute in a semantic class can only have relations to others from the same class. The collection for these relations is called the *semantic grammar*, which specifies how a semantic object of this type can be identified. In Figure 17.9, the tag <CFG> specifies how the semantic class can be referred to syntactically via a context-free grammar (CFG). For the class ByRel, the specified syntax indicates that expressions like *Peter's manager* and *manager of Peter* are legal references to semantic class ByName. The semantic encapsulation, on the other hand, dictates the actions and the discourse context under which they may be taken by a semantic class. This is discussed further in Section 17.5.

As described in 17.2.2, it is a nontrivial task to determine the types of speech acts. The semantic frame is an abstraction of the speech acts, the domain knowledge, and sometimes even the application logic. Once we have this rich semantic representation, how to parse spoken utterances into the semantic frames becomes the critical task. Nonetheless, the combination of semantic frames and the semantic parser alleviates the need for a dedicated module for determining speech acts.

Semantic frames and associated robust parsing (described in Section 17.4.1) have been widely used in spoken language understanding. For detailed description of semantic classes and frames, you can refer to [58, 65].

17.3.2. Conceptual Graphs

The semantic-representation requirement has led to development of a proposal to standardize the logical form that may form the basis of the internal semantics and semantic interchange of natural language systems, including dialog processing, information retrieval, and machine translation. The proposal is based on conceptual graphs derived from Charles Sanders Peirce [38] and the various types of semantic networks used in artificial intelligence research.

The *conceptual graph* (CG) proposal [53] specifies the syntax and semantics of conceptual graphs as well as formats for graphical and character-based representation and machine-based exchange. In the terms of the proposed standard, a conceptual graph (CG) is an abstract representation for logic with nodes called *concepts* and *conceptual relations*, linked together by arcs. In the graphical representation of a CG, concepts are represented by rectangles, and conceptual relations are represented by circles or ovals. The ordinary phrasing for the association of relations (circles) to concepts (rectangles) is *has a(n)* for arrows pointing toward the circle and *is a(n)* for arrows pointing away.

Figure 17.11 illustrates a conceptual graph for the sentence *Eric is flying to Boston by airplane*. The mnemonic meaning of the arrows is: *Fly* has an agent who is a person, *Eric*, and a destination *Boston*. The proposal also specifies a linear form, as shown in Figure 17.10. In the form, concepts are in square brackets and conceptual relations are in parentheses. The hyphen means that relations of a given concept continue on subsequent lines, as shown in Figure 17.11.

```
[Fly] -
     (Agent)->[Person: Eric]
      (Dest)->[City: Boston]
      (Inst)->[Airplane]
```

Figure 17.10 A linear form representation of *Fly* has an agent who is a person, *Eric*, and a destination *Boston*.

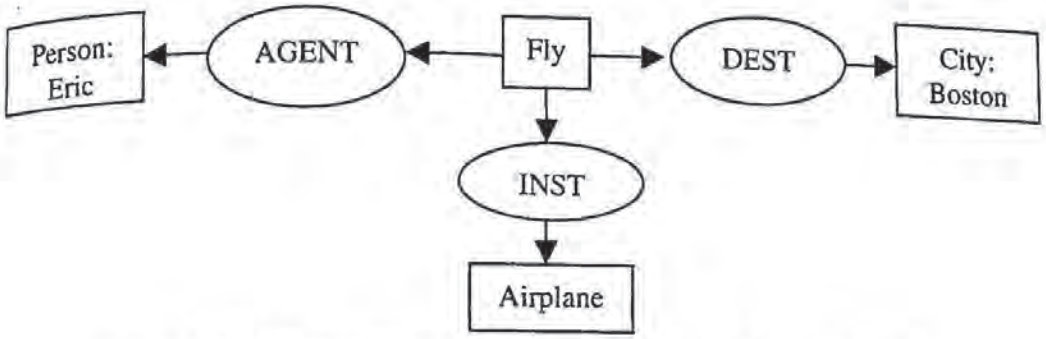


Figure 17.11 CG display form for *Eric is flying to Boston by airplane* [53].

Each concept has a type and a (possibly empty) referent. An empty referent means that at least one, unspecified example of the type is assumed to exist somewhere (an existential quantifier). So, in Figure 17.10, the type is present, but the referent is left unspecified. In an application, the referent can be completed by referring to a train-schedule database and inserting a particular instance of a scheduled train departure time, location, and number. The *valence* of a relation is the number of required concepts that it links. For example, as shown in Figure 17.12, the relation *between* would be a conceptual relation of valence 3, because typically (something/somebody) is *between* one (something/somebody) and another (something/somebody), as in the familiar English idiom "*somebody is between a rock and a hard place*" (meaning, *to be in great difficulty*). This corresponds to the linear form, as shown in Figure 17.13.

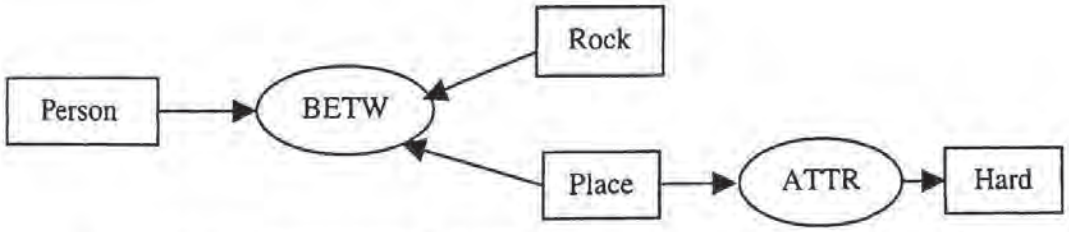


Figure 17.12 CG display form for *a person is between a rock and a hard place* [53].

```
[Person]<-(Betw) -
        <-1-[Rock]
        <-2-[Place]->(Attr)->[Hard]
```

Figure 17.13 A linear form representation of *A person is between a rock and a hard place.*

17.4. SENTENCE INTERPRETATION

We follow the convention of most modern SLU systems—treating semantic parser as a two-step pattern recognition problem (speech recognition followed by sentence interpretation). This convention has the advantage of modular design of SLU systems. Thus, the same SLU

system can be used for text input. However, a unified semantic parser [50, 62] may achieve better accuracy, because no hard decision needs to be made before picking the optimal semantic representation.

The heart and soul of the *sentence interpretation* module is how to convert (translate) a user's query (sentence) into the semantic representation. In other words, one has to fill the semantic slots with information derived from the content (words) in the sentence. In this section we describe two popular approaches to accomplish this task. Although they can be perceived as pattern matching methods, they differ in the matching mechanism.

17.4.1. Robust Parsing

Due to the nested nature of semantic classes, the semantic representation **F** in Eq. (17.1) can itself be a tree of semantic objects. A user's utterance may consist of disjoint fragments that may make sense at the discourse level. For instance, in the context of setting up a meeting, the utterance "*Peter Duke at a quarter to two*" can be parsed into two semantic objects: a person and the meeting time. Therefore, the sentence interpretation module must deal with sentence fragments.

The analysis of spoken language is a more challenging task than the analysis of written text. The major issues that come to play in parsing spontaneous speech are speech disfluencies, the looser notion of grammaticality that is characteristic of spoken language, and the lack of clearly marked sentence boundaries. The contamination of the input with errors of a speech recognizer can further exacerbate these problems. Most natural language parsing algorithms are designed to analyze grammatical input. These algorithms are designed to detect ungrammatical input at the earliest possible opportunity and to reject any input that is found to be ungrammatical in even the slightest way. This property, which requires the parser to make a complete and absolute distinction between grammatical and ungrammatical input, makes such formal parsers fragile for spontaneous speech, where completely grammatical input is the exception more than the rule. This is why a robust parser is needed.

In Chapter 11, context-free grammars (CFG) can be written to analyze the structure of entire sentences. It is natural to extend CFG as a pattern matching vehicle. For example, a question such as "*Where would you like to go*?" might be used to solicit a response from a user, who might respond, "*I would like to fly to Boston.*" The following grammar might be used to parse the response:

```
S → NP VP
NP → N
VP → VCluster PP
VCluster → would like to V
V → go | fly
PP → prep NP
N → Boston | I
Prep → to
```

The resulting phrase structure, characterizing the entire sentence, would be:

```
[S [NP [N I ] ] [VP [VCluster would like to [V fly ] ] [PP
[prep to ] [NP [N Boston]]]]]]
```

This structure in turn can provide the foundation for subsequent semantic analysis. Thus, the grammar is adequate for the example response and can be easily extended to cover more city names by expanding the N → rule, i.e., by enlarging the lexicon. It has some deficiencies, however. Some of the problems are purely formal or logical in nature, such as the fact that "*Boston would like to go to I*" can be equally parsed. These flaws can be addressed with formal fixes (e.g., a more refined category system), but, in any case, they are not crucial for the practical system designer, because pathological examples are rare in real life. The deeper problem is how to deal with legitimate, natural variations.

The user might respond with any of the following:

```
To Boston
I'm going to Boston.
Well, I want to start in New York and get to Boston by the
day after tomorrow.
I'm in a big hurry; I've got a meeting in Boston.
OK, um, wait a second... OK, I think I've gotta head for Bos-
ton.
```

The above sentences incorporate different kinds of variation for which a *sentence coverage* grammar typically has trouble accounting. For this reason, dialog system designers have gravitated to the idea of *robust parsing*. Robust parsing is the idea of extracting all and only the usable chunks of simple meaning from an utterance, ignoring the rest or treating it as noise or filler. Small grammars can be written that scan a word lattice (see Chapter 13) or a word sequence for just those particular items in which they specialize. For example, a *Destination* grammar, not intended to span an entire utterance, can skim each of the complex utterances above and find the Destination in each case:

```
Destination → Preposition CityName
Preposition → to | for | in
CityName → Boston | ...
```

The noise or *filler* elements might include nonspeech noise (cough, laugh, breath, hesitation), elements of *phatic* communication (greetings, polite constructions), irrelevant comments, unnecessary detail, etc. As a user becomes accustomed to the limited yet practical domain of a system's operations, it is expected that variant phrasings would diminish, since they take longer to utter and contribute very little, though disfluencies would always be an issue.

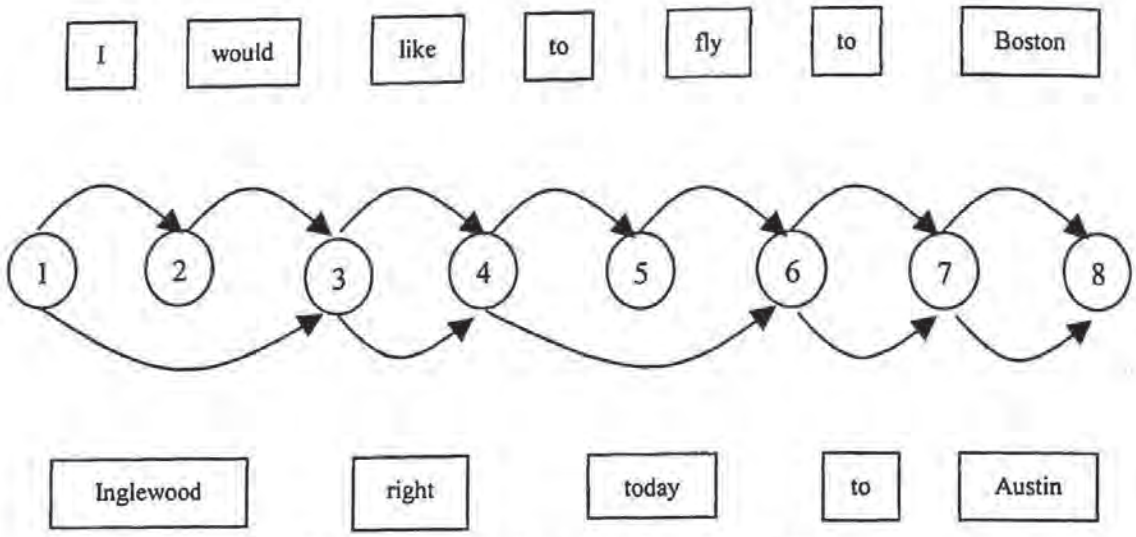


Figure 17.14 Word graph for hypotheses [61].

The original word graph or lattice from the speech recognizer might consist of nodes, representing points in time, and edges representing word hypotheses and acoustic scores for a given span in the utterance. Figure 17.14 illustrates a sample of word graph for the example "*I would like to fly to Boston*" with competing hypotheses. Using the *Destination* grammar on the word graph in Figure 17.14 will skip the earlier parts of the possible sentence hypotheses and identify the short fragment from node 6 to node 8 as a destination. If only the *Destination* grammar were active, a new view of the word graph would result in Figure 17.15.

This example shows that potential and legitimate ambiguities can persist even with flexible grammars of this type, but the key potential meanings have been identified. A robust parser that is capable of handling the example needs to solve the following three problems:

- *Chunking*: appropriate segmentation of text into syntactically meaningful units;
- *Disambiguation:* selecting the unique semantically and pragmatically correct analysis from the potentially large number of syntactically legitimate ones returned; and
- *Undergeneration:* dealing with cases of input outside the system's lexical or syntactic coverage.

Grammars developed for spontaneous speech should concentrate on describing the structure of the meaningful clauses and sentences that are embedded in the spoken utterance. The goal of the parser is to facilitate the extraction of these meaningful clauses from the utterance, while disregarding the surrounding disfluencies. We use the semantic grammar in the Dr. Who SLU engine [61] to illustrate how this works.

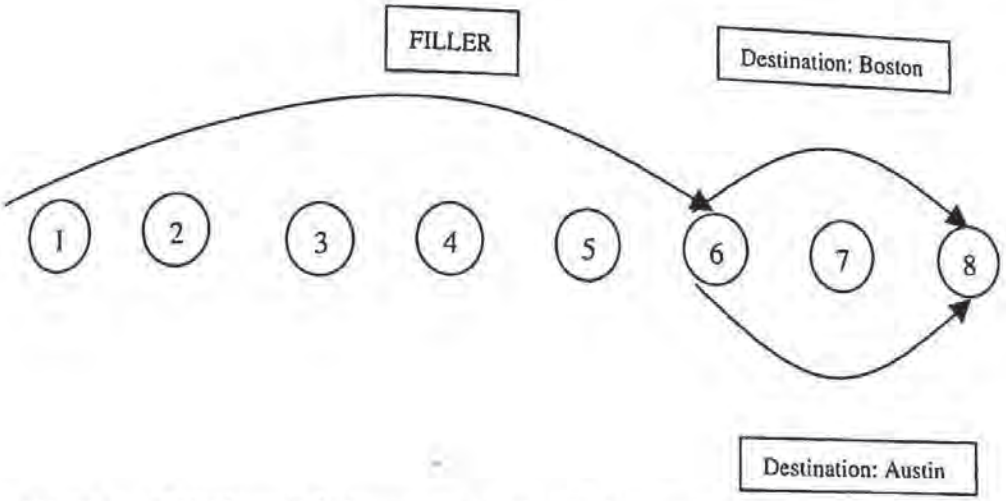


Figure 17.15 Word graph for hypotheses if only the *Destination* grammar is active [61].

As shown in Figure 17.9, a Dr. Who semantic class mostly contains a set of slots that need to be filled with terminal words (verbatim) or with recursive nonterminal semantic classes. Strictly speaking, this semantic class grammar can hardly be called a grammar, since it is primarily used to define the conceptual relations among Dr. Who entities rather than the language expressions that are used to refer to the entities. The syntactic expression is specified by optional CFGs associated with each semantic class. In general, the syntactic grammars need to deal with three kinds of variation in surface linguistic form:

1. *Variation within a slot*—When CFG is missing in the definition of semantic classes, the grammar could allow flexible assembly of an expression. For example, if the <cfg> tags in Figure 17.9 are omitted, any sequence that contains a word of a P_RELATION typed class and a word of a PERSON typed class can be an expression referring to a semantic object of *ByRel* such as *John's father, father of John,* or even *John loves his father*. Thus, CFGs are often specified within the semantic slot to provide linguistic constraints without over-generating.
2. *Variation in the order of frame presentation*—Many systems [64, 66] employ an island-driven robust parsing strategy where the slots in the semantic frames are filled by language fragments from parsing. Parsing of the slots is order independent. Thus utterances "*Schedule a meeting with John at 3 PM*" and "*Schedule a meeting at 3 PM with John*" can be processed without problems.
3. *Disfluencies and irrelevancies*—Disfluencies and irrelevancies are unavoidable for spoken language input. The system has to deal with real utterances such as "*I'd really like to know whether a meeting by 3 PM would be at all possible for John.*"