ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/4222661

Improving Protein Function Prediction using the Hierarchical Structure of the Gene Ontology

Conference Paper · December 2005

DOI: 10.1109/CIBCB.2005.1594940 · Source: IEEE Xplore

CITATIONS
111

READS
192

5 authors, including:

Duane Szafron
University of Alberta
257 PUBLICATIONS **5,974** CITATIONS

SEE PROFILE

Russ Greiner
University of Alberta
505 PUBLICATIONS **25,105** CITATIONS

SEE PROFILE

All content following this page was uploaded by Russ Greiner on 13 January 2014.

The user has requested enhancement of the downloaded file.

IPR2025-01398
CrowdStrike EX1027 Page 1

Improving Protein Function Prediction using the Hierarchical Structure of the Gene Ontology

Roman Eisner, Brett Poulin, Duane Szafron, Paul Lu, Russ Greiner
Department of Computing Science
University of Alberta
Edmonton, AB, T6G 2E8 CANADA
{eisner, poulin, duane, paullu, greiner}@cs.ualberta.ca

***Abstract*—High performance and accurate protein function prediction is an important problem in molecular biology. Many contemporary ontologies, such as Gene Ontology (GO), have a hierarchical structure that can be exploited to improve the prediction accuracy, and lower the computational cost, of protein function prediction. We leverage the hierarchical structure of the ontology in two ways. First, we present a method of creating hierarchy-aware training sets for machine-learned classifiers and we show that, in the case of GO molecular function, it is the most accurate method compared to not considering the hierarchy during training. Second, we use the hierarchy to reduce the computational cost of classification. We also introduce a sound methodology for evaluating hierarchical classifiers using global cross-validation. Biologists often use sequence similarity (e.g. BLAST) to identify a "nearest neighbor" sequence and use the database annotations of this neighbor to predict protein function. In these cases, we use the hierarchy to improve accuracy by a small amount. When no similar sequences can be found (which is true for up to 40% of some common proteomes), our technique can improve accuracy by a more significant amount. Although this paper focuses on a specific important application—protein function prediction for the GO hierarchy—the techniques may be applied to any classification problem over a hierarchical ontology.**

I. Introduction

The rate at which sequencing methods are producing genomic and proteomic data is far outpacing the rate at which these biological systems are being experimentally annotated. In response, there has been much research in automated protein function annotation, designed to provide biologists with the most likely functions that proteins perform [20]. These methods should be highly accurate to be useful, and they should be high-throughput so they can be used for a large amount of data. Here, we present a computational tool for predicting the Gene Ontology (GO) [25] molecular function of proteins from sequence data.

A major concern in automated systems is the ontology used for prediction. The ontology is a set of terms describing the problem domain in a standardized way. In predicting molecular function, the ontology is a set of terms that describe the possible functions that proteins perform in the cell. Although non-hierarchical, flat ontologies such as GeneQuiz [8] and others are suitable for describing the general function of proteins, a more sophisticated approach is useful for describing more specific functions of proteins. Furthermore, different experiments to verify the function of proteins provide different levels of detail about that protein's functions, which leaves many proteins with incomplete or overly general annotations. Hierarchical ontologies are an effective way of addressing both of these issues.

In ontologies such as EC [2], SCOP [3] and GO, both general and specific knowledge is represented in a hierarchical structure where general terms are represented by nodes near the root of the ontology and specific terms are represented by nodes near the leaves of the ontology. The hierarchy defines an inheritance (is-a) relationship between the term nodes, where each term is a special case of its parent terms. That is, any term is-a special case of each of its ancestor terms, where an ancestor is any term along the path from the term to the root of the hierarchy.

In the GO hierarchy, the inheritance relationship is called the *true path rule* [25]. For example, the link from *ion binding* to *metal ion binding* means that every protein associated with the term *metal ion binding* inherits the *ion binding* term (i.e., metal ion binding is-a special case of ion binding). As the knowledge of a particular protein is refined, new terms from the ontology can be added to it. Some of these new terms are newly discovered functions. In this case, the new term appears in the graph as a descendent node of a previously known, more general node. The true path rule plays an important role in our hierarchy-aware methodology for training machine-learned classifiers and predicting protein function. Without the true path rule, predictions within the hierarchy would be inconsistent.

GO describes three aspects of protein annotation: *cellular component*, *biological process*, and *molecular function*. However, our work is focused on only the molecular function aspect of GO. We present Classification in a Hierarchy Under Gene Ontology (CHUGO), a system that exploits the hierarchical structure of the GO to make faster and more accurate predictions of protein function. CHUGO assigns each protein to one or more of a specified set of 406 GO functional classes, spanning a large spectrum of functionality. By contrast, the EC ontology is concerned with only enzymatic proteins, and the SCOP only deals with protein structures.

Many proteins have multiple functions. For example, the average number of experimentally verified functions for the proteins in the Gene Ontology Annotation project (GOA) [1] is 1.35. One standard machine learning approach in such cases is to build a series of *local* binary predictors that predict

"yes" or "no" for each term in the ontology. We argue that if the ontology forms a hierarchy, the speed and accuracy of local predictors can be improved by exploiting the relationship between the terms in the ontology.

This work makes contributions in three main areas:

1. *Evaluation methodology:* In ontologies where multiple labels are possible, precision, recall, and cross-validation are concepts that must be re-visited, especially given a hierarchical ontology. We define and examine a hierarchy-aware evaluation methodology (Section 2).
2. *Training set design:* Structured ontologies are encoded with important information about relationships between terms, and are a way of representing incomplete data. We present a novel and effective methodology that exploits the inherent structure of a hierarchical ontology (Section 3).
3. *Accurate and efficient protein function prediction*: The structure of a hierarchical ontology can also be exploited at prediction time to improve predictive performance, and lower computational costs (Section 4). We also predict a large number of terms – 406 molecular functions.

A. Related Work

There is a large body of research seeking ways to predict the function of a given protein. One approach is to use the protein's 3-dimensional structure to predict that protein's function [22]. Since this approach requires that the protein's structure be solved (or at least predicted accurately) it provides very limited coverage across all of the proteins in any particular organism. A second set of techniques uses documents describing proteins to predict the functions of these proteins [19]. This technique also has extremely limited coverage on a per-organism basis. Coverage can be improved by using a technique based on finding similar sequences using BLAST [16]. Such techniques use annotations from similar sequences to predict the function of the unknown protein (also called Nearest Neighbor). However, we demonstrate (Section 4) that such approaches do not work well for predicting GO functions when the most similar sequences found by BLAST are above the 10^{-3} E-value threshold. This often limits the coverage of these predictors to only about 60% of the proteins in a proteome.

King *et al* [13] presents another approach, one that is only based on the existing annotations of proteins. The system looks at existing annotations of the query protein, and predicts annotations that often correlate with the existing predictions. This is done because the authors correctly observe that protein annotations are often incomplete. They describe protein annotations as incomplete because "...there are genes whose attributes are not yet all known, and because there is literature that has not yet been digested by the database curators" [13]. However, in Section 3 we argue that even experimentally determined annotations fresh from the lab may be incomplete, since they may be overly general – and will become more specific in the future. The methodology presented by King *et al* also suffers from the fact that functions that are correlated may not always occur together. In CHUGO, each function is predicted individually, and correlations occur as a natural result of the prediction process.

Our goal is to predict a protein's function starting from sequence data. Protfun [12] is another example of such an approach. It uses local sequence properties, such as predicted post-translational modifications, sorting signals and properties computed from amino acid composition. However, Protfun does not exploit knowledge of the hierarchy, and the set of 14 GO terms that it predicts is relatively small. In addition to using local sequence properties, we also use the annotations from similar sequences in Swiss-Prot found by BLAST, when they are available.

The machine learning literature has described attempts to utilize the structure of a hierarchical ontology to improve predictive accuracy. Kiritchenko *et al* [19] used the hierarchy to increase the number of training instances at each node, by first making the training data more consistent with the ontology. We extend this work by investigating different degrees of consistency with the hierarchy when creating our predictors.

Chakrabarti *et al* [5] and King *et al* [13] both used the structure of the underlying ontology to build a Bayesian network. Similarly, the structure of the ontology has also been exploited to create a hierarchical mixture of experts model [11]. Although our system does not use more models as complex as these, they could be combined with our approach to potentially increase predictive performance.

Some results [6] have shown that a statistical technique known as *shrinkage* can be used to set the parameters in a hierarchy of predictors. Here, the ontology is exploited as prior knowledge to understand which classes are related, and thus, which parameters should have similar settings.

Other hierarchies have been exploited to change the formulation of Support Vector Machines [15], and Association Rules [7]. In principle, this approach could be combined with our own, since the two techniques are independent.

Greiner *et al* observed that a top-down decision model in hierarchical classification could have poor results since all predictors along the path to the true label must agree [4]. Also, other methods of training individual term predictors were mentioned but not explored. Our methods of training set design allow for a more inclusive classifier in which the top-down model is more feasible, and thus we can reduce computational complexity without a resulting loss of precision and recall.

II. Evaluation Methodology

A. Scoring Predictions in a Hierarchy

When annotating proteins with GO terms, each protein can be assigned multiple terms for two reasons. First, due to the hierarchical nature of GO, a protein annotated with a function is implicitly annotated with all of the function's parents, up to

the root node. For example, if a protein's function is *metal ion binding*, then it is implicitly also a *binding* protein due to the true path rule. Second, a protein could have multiple functional domains or react to more than one molecule. For example, the protein JIP1_MOUSE is annotated with *kinesin binding* and *protein kinase binding*. Neither of these terms is a direct specialization of the other, so there is no ancestor/descendent relationship between these terms in GO.

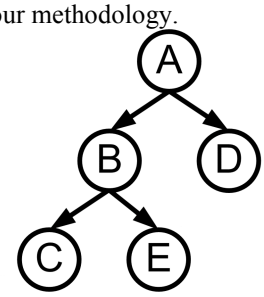
The fact that each protein can have more than one GO term causes problems for the traditional machine learning measures of precision, recall, accuracy, and sensitivity. For classifiers, it is easy to evaluate a binary "correct" or "incorrect" prediction. But, intuitively, predictions that are "close" to the oracle label should score "better" than predictions that are in an unrelated part of the hierarchy.

Our evaluation methodology is simple, intuitive, and consistent with the true path rule. First, if a protein is assigned a label, then it is also assigned the label of all of its ancestors up to the root node. This simple but important consequence of the hierarchy is easily overlooked. If the protein has another predicted label in an unrelated part of the hierarchy, those labels and their propagations are also added to the set of labels. Only after propagation are the formulas for precision and recall applied to find hierarchical precision and recall.

Failure to follow the true path rule (e.g. by propagating the labels via the is-a relationship to the root node), leads to inaccurate predictions and distorted evaluation metrics. Incorporating propagation into the evaluation of predictions allows for a graduated scoring system where distance in the ontology is intrinsically taken into account. Hierarchical precision and recall reflect how close, conceptually, predictions are to the correct labels in the ontology.

For example, consider a term hierarchy where A is the parent of B which is the parent of C (Fig. 1). Assume that protein P_1 is labeled {B} and protein P_2 is labeled {C}. By the true path rule, the labeling really should be {A, B} for P_1, and {A, B, C} for P_2 after propagation (shown by circles in the table within Fig. 1). Assume that for protein P_1 we predict the class to be {C}, which is different than the truth. By the true path rule, we then assign P_1 the labels {A, B, C} (shown by x's in the table within Fig. 1). Similarly, for P_2 we predict the label to be {B} and propagate to get {A, B}. Both of the initial labels for P_1 and P_2 are different than the oracle, which misleadingly suggests a poor prediction. But, hierarchical precision and recall allow for an evaluation scheme which is more in tune with intuition.

Intuitively, despite the differences with the oracle, our prediction for P_1 should have perfect recall, since it correctly recalled that P_1 has terms {A, B}. But the precision is only 2/3 since only 2 out of the 3 predicted labels were correct, which is an intuitively sound penalty for the imperfect prediction. Similarly, our prediction for P_2 should have perfect precision, since every predicted term is correct, but recall is only 2/3 since only 2 out of 3 correct labels were recalled. This example shows that predicting too "high" in the hierarchy (e.g. P_2) reduces recall, but does not affect precision and that predicting too "low" in the hierarchy (e.g. P_1) reduces precision, but does not affect recall. Lastly, predictions that are in a different part of the hierarchy altogether (e.g. predicting P_1 is in D, is not shown in the example) will have neither high precision nor high recall. The ability to handle "close" predictions and altogether wrong predictions are important aspects of our methodology.



Instance	A	B	C	Precision	Recall
P_1	(x)	(x)	x	2/3	2/2
P_2	(x)	(x)	◯	2/2	2/3

x = Predicted
◯ = Labeled

Fig.1. A simple evaluation example. Protein P_1 is labeled as "B" but predicted as "C", and protein P_2 is labeled as "C" but predicted as "B".

The original, formal presentation of our approach to multi-class precision and recall was introduced by Poulin [17]. In this paper, we extend it to hierarchies by applying the true path rule. Independently, the same approach was used by Kiritchenko *et al.* [19], also in the domain of GO, and a formal definition was published later [21]. A similar scoring metric which considers the hierarchy was also presented by Wu *et al* [52], but it is a single measure, and thus lacks the intuitive value of precision and recall.

B. Evaluation using Global Fold Selection

To approximate how well a classifier will perform when presented with new data, cross-validation is often used. For all of our experiments, we use 5-fold cross-validation. In 5-fold cross-validation, the data set is first randomly split into 5 parts of equal size. Then, for 5 iterations, one fold is withheld as the test set, and the remaining 4 folds are used as the training set to create a classifier. It is important that random fold generation ensures that the same ratio of positive and negative training instances is placed in each fold. For example, if there are 100 positive training instances and 900 negative training instances, each fold should have 20 positive training instances and 180 negative training instances. We refer to this property as *fold balance*. Fold balance ensures that each classifier trained during cross-validation behaves as closely as possible to the final classifier created by training on all of the folds. This assumes that the training instances have the same distribution of positive and negative training instances as the unknown query data on which the classifier will be invoked. Of course the folds may not be perfectly balanced due to integer division, but they should be as close as possible.

Precision, recall and F-measure are computed for the classifiers created for each fold. The statistics for the 5 folds are averaged to give an accurate representation of the predictive performance of the classifier on future instances. Statistics from cross-validation are a good indicator of how well a classification system will predict new data due to the fact that no information from a test fold is used to build the

predictor that will be applied to that test fold. We refer to this principle as *fold isolation*. An alternate set of statistics can be computed by using all training data to build a classifier and then applying this classifier to each training instance in turn and counting re-substitution errors. Statistics based on re-substitution do not exhibit any isolation, since test data is used in making predictions on test data. Therefore these statistics are not useful in predicting statistics for future instances.

Hierarchical classification causes problems for cross-validation since in this case, it is difficult to maintain both fold balance and fold isolation, in the presence of the true path rule. First, we assert that fold isolation requires that the same folds be used for all node classifiers across the hierarchy. We will refer to this as *global fold selection*. The following example shows the difficulty of having folds that satisfy both. Assume a protein P is in fold 1 for node B and in fold 2 for its parent node A, and assume we are evaluating the classifiers when the test set is fold 2. It is possible for the classifier at node B to predict positive for protein P and the classifier at node A to predict negative for protein P. However, the true path rule will propagate the positive prediction from node B to parent node A. This means that protein P was used as a training instance to make a classifier for node B and this classifier was used to propagate a positive prediction for protein P to class A. Using a protein in a training set to make a prediction about the same protein in a test set violates the fold isolation rule. Therefore a single global fold split must be used for all nodes.

Unfortunately, global fold selection makes it difficult (and in some cases impossible) to ensure fold balance. To address this problem, we randomly generate 5,000 candidate splits of the data set, and choose the split that has the best fold balance. Assume that at node N_i there are P_i positive training instances out of T_i total training instances of which P_{ij} positive training instances and T_{ij} total training instances are in fold F_j. We define the deviation D_{ij} of fold F_j at node N_i to be $((P_{ij}/T_{ij}) - (P_i/T_i)) / (P_i/T_i)$. For example, if at node N_1 there are 10 positive training instances out of 90 total training instances and in fold F_2 for N_1 there are 4 positive training instances out of 40, then the deviation of fold 2 for node 1 is $D_{12} = (4/40 - 10/90) /(10/90) = -0.1$. For each split we compute the standard deviation of all folds for all nodes:

$$\sigma = \sqrt{\frac{\sum_i \sum_j D_{ij} \times D_{ij}}{NumberofFolds \times NumberofNodes - 1}}$$

The best fold balance occurs for the split whose standard deviation is the smallest (0.064) and the worst fold balance occurs for the split whose standard deviation is the largest (0.079). This suggests that although perfect fold balance cannot be guaranteed, global folds are feasible using a simple randomized approach.

III. Training Set Design

A. Data Set

Our data set consists of 14,362 proteins annotated with their GO function terms. To create this data set, we first downloaded the Gene Ontology from the GO website [25], and then obtained our candidate set of proteins by examining all proteins in the UniProt database [14]. We looked up each of these proteins in the Gene Ontology Annotation project (GOA) [1]. We wanted to build a conservative training set based only on experimentally determined annotations, to avoid any bias in our data set due to other electronic annotation systems. The GOA project includes annotation codes with each of their GO annotations. A protein was retained in the training set if its GOA indicated it was not derived from another electronic annotation system. We included annotation codes IDA, IEP, IGI, IMP, IPI, RCA, TAS, and excluded codes IC, IEA, ISS, NAS, ND - since the latter are either annotated using electronic means, or are ambiguous in their origin. We used the August 28, 2004 version of the GO molecular function ontology, and the August 11, 2004 version of the GOA mapping file. For protein sequence data we used release 27 of TrEMBL, and release 44 of Swiss-Prot (which together comprise UniProt release 2.0).

Originally, there were 7,399 nodes in the GO molecular function hierarchy. However, to create reasonably accurate local GO term predictors, we required a sufficient number of positive training instances. Therefore, we only considered those GO terms that have at least 20 proteins annotated at or below them in the hierarchy. This decreased the size of the ontology to 406 nodes. Note that CHUGO will only make a prediction into one or more of these 406 categories.

B. Term Predictors

CHUGO uses hierarchy-aware 'local' predictors to predict GO function. In hierarchical prediction, local predictors are those that predict a single GO term. In contrast, BLAST Nearest Neighbor (BLAST-NN) (that is, using the top BLAST hit's annotations as the predictions for a query instance) is a global predictor, as a single call to BLAST can assign a protein to one or more nodes. In contrast, CHUGO's local predictors train a binary classifier to predict each term separately in the GO hierarchy, using a variety of supervised learning techniques. Supervised learning is a two-stage process where proteins whose GO terms are known are used to *train* (that is, create) a classifier, and then the classifier is used to make predictions on proteins whose GO terms are not known. During training, each classifier must be presented with positive examples (those that represent the node), and negative examples (those that represent instances outside of the node). Since we are able to choose our training data as we wish, we evaluate a spectrum of methods to construct the training data for each node.

In hierarchical classification, all resulting predictions must obey the true path rule. Therefore, as a post-processing step, we propagate all positive predictions upwards in the ontology. Assuming that we have a perfect classification system, there are two major situations where local predictors in a hierarchy could perform perfectly on test data. These two styles of local predictors act differently when presented with a query protein, but both have the potential to result in perfect accuracy.

First, if a protein is to be assigned GO node N_i, every node in the hierarchy will return a negative prediction, except for node N_i that returns a positive prediction. During evaluation, this prediction is propagated, and evaluates at 100% precision and recall, if the prediction is correct. In this case, we call these classifiers *exclusive classifiers*, since they exclude every instance except for those that belong exactly at node N_i (and not those that are more general or more specific). Exclusive classifiers are traditionally used for non-hierachical, flat ontologies, however, in theory, a system of exclusive classifiers can perform perfectly in a hierarchy if each local predictor predicts perfectly. But, as we will show, they perform poorly on Gene Ontology function prediction.

Second, given a protein that belongs at node N_i, the term predictor for N_i predicts positive, and so do all of N_i's ancestors in the hierarchy. Here, the classification would evaluate at 100% precision and recall as well. In this case, we call these classifiers *inclusive classifiers*.

Between the extremes, from exclusive to inclusive, there is a grey area of classifier design where we sporadically find positive and negative predictions between N_i and the root node. Since it is possible for both exclusive and inclusive designs to have perfect precision and recall, there is no *a priori* reason to choose between inclusive or exclusive classifiers. Thus, we will evaluate 4 schemes to construct training data that range from exclusive to inclusive in their nature. Ultimately, we conclude that the inclusive schemes are preferable, for a variety of reasons.

For a particular GO node N_i, the "exclusive" approach to training set construction (Table I) would use all proteins explicitly labeled with the term N_i as the positive training set and all proteins not explicitly labeled with N_i as the negative training set.

One argument against this approach is that it does not consider the hierarchy in training set construction. However, the descendent nodes of N_i are not good negative instances for the predictor at N_i. Therefore we could decide to exclude these instances from the set of negative examples. This is labeled as "less exclusive" in Table I.

Next we could observe that all descendents of N_i are not only poor negative examples, but they could be used as positive training examples, due to the nature of the is-a relationship in the GO hierarchy. This has previously been done in [21], but we wish to quantify the difference in each of these schemes for constructing training data. This method is called "less inclusive" in Table I.

Finally, to be most consistent with the hierarchy, we could observe that those proteins that are annotated as ancestors of N_i could in fact be instances of N_i. According to the GO specifications, proteins are annotated with the most specific function term for which an experiment has been performed. For example, if a protein is annotated with term *ion binding*, it is due to an experiment that confirmed this. However, this does not mean that the protein does not belong to one of the child classes: *anion binding*, *cation binding* or *metal ion binding*, just that no experiment has determined that it belongs to one of these subclasses. Since it is common for future experiments to supply more specialized terms, it could be dangerous to include proteins annotated with ancestor terms in constructing a negative training set for a term. Therefore, our final training set rule disallows any proteins labeled *ion binding* in the negative training sets of predictors for the three child terms. One could also argue that future experiments could add any arbitrary new term to a protein, so that no negative training instances can be used with confidence. This is a good point, and it applies anytime a classification task can have multiple positive answers and negative (experimental) evidence is not available, not recorded, or incomplete. However, we need negative training instances and at least we guard against the more common case of more specific annotations following less specific ones. We term this final approach an *inclusive classifier*, and it is shown in Table I in the row marked "inclusive".

TABLE I
A VARIETY OF TRAINING STRATEGIES CAN BE USED TO CONSTRUCT A PREDICTOR FOR NODE A IN THE HIERARCHY.

Method	Positive Examples	Negative Examples	Not Used
Exclusive	A	Not A	-
Less Exclusive	A	Not [A + Descendents(A)]	Descendents(A)
Less Inclusive	A + Descendents(A)	Not [A + Descendents(A)]	-
Inclusive	A + Descendents(A)	Not [A + Descendents(A) + Ancestors(A)]	Ancestors(A)

We evaluated these four training methods. For simplicity, we chose to evaluate the training methods using a single local predictor technology for this experiment. We used Proteome Analyst [18] features in conjunction with Support Vector Machines (more details in Section 4). Table II summarizes the results of cross-validation for each of the 4 training set construction methodologies. The precision of all 4 techniques is comparable, but there are significant differences in recall and F-measure (The F-measure in each row of the table also has a 95% confidence interval of less than 0.01, which is not shown).

TABLE II
CROSS-VALIDATION PERFORMANCE ON TRAINING STRATEGIES FOR EACH NODE PREDICTOR USING PA FEATURES. THE BEST VALUES IN EACH COLUMN ARE BOLDFACE. RESULTS SHOWN USE THE PA-SVM LOCAL PREDICTORS.

Method	Precision	Recall	F-Measure	Exceptions per Protein
Exclusive	0.758	0.326	0.456	1.517
Less Exclusive	**0.774**	0.401	0.528	1.736
Less Inclusive	0.772	0.634	0.696	**0.049**
Inclusive	0.754	**0.648**	**0.697**	0.087

Our goal is to maximize precision, recall, and F-measure. The column "exceptions per protein" in Table II describes how often we see a positive prediction for a node predictor, and a negative prediction for that node's ancestor term(s) – violations of the true path rule. By their nature, exclusive

classifiers are more likely to have many exceptions, while inclusive classifiers are likely to have few. The data in Table II matches this intuition. But, as previously discussed, the evaluation methodology requires that we first propagate positive predictions upward in the ontology, which ameliorates the effect of exceptions, so our test is fair to all four strategies. In fact, the differences between techniques can be explained via differences in the size of the positive training set (exclusive classifiers have fewer positive training instances and more negative training instances), and noise in the data used for training.

We can see that as our classifiers become more and more inclusive, recall and F-measure are increased. The first reason is that we are increasing the number of positive training examples (going from less exclusive to less inclusive), so the predictors become better at recognizing those proteins that should belong at or below each node. The second reason classifiers perform better is that noise is removed from the training sets. That is, as we become more and more inclusive we are not using instances that are intuitively positive in the negative training set, when going from exclusive to less exclusive. Also, by excluding ambiguously labeled instances from the negative training set, when going from less inclusive to inclusive classifiers, noise is further removed.

As we make our classifiers more inclusive, there is a higher chance that a false negative at a node will be offset by a true positive prediction at a descendent node, and the false negative will be overruled by the true path rule. In a sense, inclusive classifiers reinforce each other along the path in a hierarchy, whereas in a system of exclusive classifiers, we are relying on one predictor to make the correct call for each assigned label.

The inclusive strategy has the best overall recall and F-measure and we recommend it as the best overall strategy. Admittedly, our cross-validation experiment shows that excluding ancestors from the negative training instances set only has a small advantage over the less inclusive strategy. However, we speculate that the actual advantages of excluding ancestors are greater than shown by this experiment. We believe that the nature of cross-validation tests, and the fact that an absence of a label in the GO hierarchy does not necessarily mean a label is wrong, leads to lower quantitative results for what is, arguably, a more accurate predictor in the future, where more information is obtained about the training data. Specifically, proteins may not be annotated with all the labels that are appropriate. As discussed earlier, a missing experiment results in a missing label, but the label may be correct. For example, a protein that is annotated as an *ion binding* protein, but in actuality is a *anion binding* protein (which is a child of *ion binding* in the hierarchy), would non-intuitively give us a better score when we predict it as NOT *anion binding*. This is because "not anion binding" matches the annotations, which we consider as the correct answer during cross-validation, but "not anion binding" may not match reality. Our predictor may answer "anion binding" because of legitimate, machine-learned similarities between the protein and other proteins in the *anion binding* set. A future experiment may show that the protein is indeed "anion binding". In fact, the overall goal is to have predictors that can predict labels that are not currently known. This systemic side-effect of taking the annotation as "complete truth" (even when it is not complete) is a difficult issue to measure and address. However, we note that our hierarchy-aware definitions for recall, precision, and F-measure minimize these side-effects, and we plan on investigating the issue as future work.

IV. Predicting Protein Function

Biologists strive to understand the function of a protein. Ultimately, a laboratory experiment is needed to confirm the function of a protein, but a computational prediction can be useful both in itself and in suggesting an appropriate experiment.

A variety of computational techniques have been studied, ranging from sequence comparison, to machine learning, to structure analysis and simulation. As well, the techniques range from local predictors to global predictors, either of which may or may not exploit knowledge of the hierarchy.

We argue (and demonstrate) that the hierarchical structure of ontologies such as GO should be exploited. First, not exploiting the hierarchical structure of GO can violate the true path rule, which implies inconsistent predictions. Second, as already discussed, the construction of training sets for machine-learned classifiers benefits from a hierarchy. In general, it can be difficult to accumulate a sufficient number of positive and negative training instances, and we can exploit the hierarchy to address this problem. Third, the performance cost of using machine-learned classifiers can be controlled when the hierarchy is considered.

Sequence comparison is commonly used to give biologists an initial idea of what a protein's role in the cell is. The most commonly used method of sequence comparison is BLAST [1]. A typical use case of BLAST would be running a new protein against a trusted database (such as Swiss-Prot), and then manually reviewing the annotations for the similar proteins found by BLAST. Assuming that the top BLAST hits are homologous to the query protein, the user may then decide that it is likely that their query protein is similar in function, and then proceed with further experiments based on this data. However, proteins that are not similar to well-studied proteins will not return a good BLAST result (measured in terms of *E-value*), so the biologist is faced with either looking at proteins that are very disparate from their protein of interest, looking for other sequence information, or proceeding with "wet lab" experiments without any initial idea of the protein's function. Due to the ubiquitous use of BLAST, we compare CHUGO to BLAST in terms of predictive accuracy, coverage, and computational cost, even though BLAST itself does not take into account hierarchy information.

During cross validation, BLAST is run for each of the test proteins against the current fold's training set of proteins. A BLAST hit is a match for this test protein against the current training set. Each BLAST hit is scored with an *E-value* that represents the similarity of the two protein sequences. As the

E-value increases, the less similar the match is to the query protein. Our experiments have found that the BLAST predictor performs the best when we set our threshold for accepting BLAST hits at an *E-value* of 10^{-3}. The sequences that do not have a BLAST hit with *E-value* $\leq 10^{-3}$ are proteins that are quite different from well-studied proteins. These two cases - when the protein has high sequence similarity to our training set, and when it does not - will be examined separately.

A. Proteins with one or more good BLAST Hits

During cross-validation, 89% (12,725 out of 14,362) of the proteins in our data set had a good BLAST hit. Our goal is to try to exploit the hierarchical structure of GO to make the predictions for these proteins more accurate.

Our predictors at each node are created using an ensemble classifier. The ensemble is composed of several classifiers:

1. SVM with PFAM as features
2. SVM with Proteome Analyst (PA) features
3. Probabilistic Suffix Trees (PSTs)
4. BLAST.

We use PFAM [10] matches within each sequence as features for a Support Vector Machine (SVM) classifier. We also use SwissProt annotation-based features, extracted by Proteome Analyst [18], with an SVM classifier. Probabilistic Suffix Trees (PSTs) have been used to predict high level GO functions with some success in the past [17]. PSTs are trained for each GO term, and a brute force search through their parameter space (window length, prune depth, and pseudocount) is performed for each predictor to optimize their accuracy.

TABLE III

PERFORMANCE ON PROTEINS FOR WHICH BLAST RETURNS A GOOD HIT (EVALUE ≤ 0.001 DURING CROSS-VALIDATION. BLAST-NN ASSIGNS THE FUNCTIONS OF THE MOST SIMILAR PROTEIN TO THE QUERY PROTEIN. VOTING IS OUR ENSEMBLE CLASSIFIER

Method	Precision	Recall	Average Cost
BLAST-NN	77%	78%	1
Ensemble Voting	77%	80%	1219

The ensemble classifier is a simple voting scheme, where 2 or more predictors assigning a term to a protein will predict that GO term. The results of using a voting scheme are shown in Table III, compared to a BLAST Nearest Neighbor search.

However, to achieve these results, the predictors for each node in the ontology must be computed for each query sequence. This results in a far greater computational cost than running BLAST alone. The cost in Table III is a rough estimate at the average cost of running each prediction method, per protein. Each node predictor is assigned a cost of 1, and BLAST is assigned a constant cost of 1 since it is a global predictor. Thus, the cost of 1,219 for voting over the ontology is derived by the formula *(Number of Nodes) × (Number of Predictors at Each Node) + (Cost of BLAST)*. Since we use three predictors at each node (PA-SVM, PFAM-SVM, and PSTs in a voting ensemble) this value is $406 \times 3 + 1 = 1{,}219$. We do not add a cost of calculating the result of voting since this is a trivial computation. Although the costs of these predictors would in fact vary, this method will give us an initial idea of how computationally intensive each approach is.

BLAST is a global predictor of molecular function. This means its computational complexity does not depend on the size of the predicted ontology (it only depends on the size of the database that we are searching against). As Table III shows, the results of a BLAST search are quite accurate when highly similar sequences are found. Furthermore, the information gained from running a BLAST search is very important since it is both accurate, and computationally efficient compared to running many local predictors at each node. We can use the results of BLAST to give us a set of candidate nodes to run our local predictors on, which will decrease computational cost from running all node predictors.

If we examine the annotations that BLAST-NN returns, we can see that when the annotations are incorrect, they tend to be near the actual annotations (Fig. 2). We can exploit this fact in two ways. First, if there is more than one good BLAST hit, we can use these additional hits as guidelines for prediction. Second, since we know the structure of the ontology during prediction, we can look at classes that are nearby (in terms of path length within the ontology) to the annotations found by BLAST-NN. In both methods, the results of the BLAST search are used as a seed to begin searching. As will be shown later, searching from the root downward is a viable option, however, it is more costly.

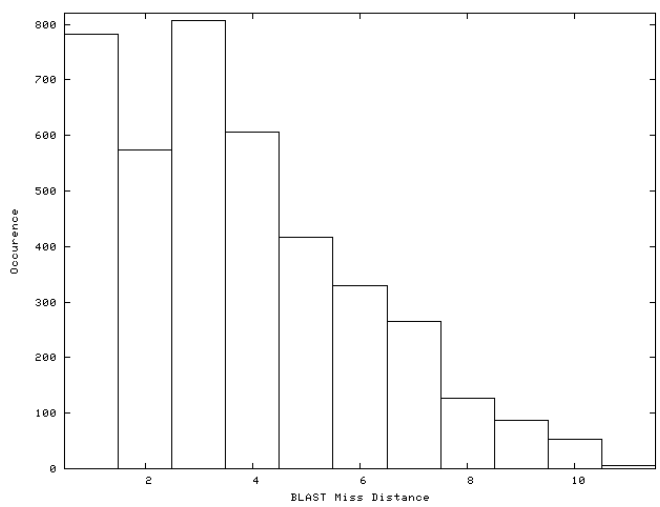


Fig. 2. When BLAST-NN does not return the correct nodes, the nodes that are returned tend to be close to the correct answer.

The first option, called *B-N-Union*, is to look at more than one BLAST hit's annotation. N is the number of hits whose annotations are used. For example, if the top 2 BLAST hits for a given protein are used to generate a union set of annotations, then N=2. Intuitively, if we take the union of more than one BLAST hit's annotations, we will get a wider range of predictions. This should increase recall, and decrease precision. However, since this set is smaller than the entire ontology, and the recall is high, we can run our predictors on this set of labels and get good performance while keeping computational cost low.

The second option, called *SearchN*, is to exploit the annotations returned by BLAST and search in the neighborhood of the top BLAST hit's annotations. N is the

graph distance from the seed annotations in which we add nodes to our set of candidate nodes. In this case, we exploit the structure of the GO hierarchy to know which terms are close by. Since BLAST tends to come close to the actual predictions when it is wrong (Fig. 2), we know that there is a high chance of the correct labels being within those that BLAST-NN returns, or nearby. In this case we can start with the set of GO terms that BLAST-NN returns and add the terms in the neighborhood to the set of candidate terms. As in the first method, recall will be increased by this method, and the node predictor is used as a validation to compensate for this by removing false positives. Also, similar to the first proposed method, the computational runtime is less than running the validating predictors for all nodes.

The advantage of the SearchN and B-N-Union approaches is that they decrease the computational cost from running all of the node predictors to running only predictors for likely GO nodes. An interesting side effect is that constraining our candidate nodes for our predictors raises precision since we never consider those GO terms that are unlikely to be assigned to an instance. An important note is that regardless of which method is computationally cheaper for finding the set of candidate GO term predictors to run, the neighborhood search method must be used when only a single good BLAST hit is found (since there are no other good hits to union). In total, 1,259 proteins had a single good BLAST hit, 944 had 2, and 10,522 had 3 or more. A comparison of the methods is shown in Table IV.

TABLE IV
COMPARING METHODS OF USING BLAST TO FIND CANDIDATE LABELS, AND THEN VALIDATING THESE SETS USING OUR PREDICTORS.

Candidate Method	Precision	Recall	Average Cost
B-1-Union	81%	75%	16
B-2-Union	79%	78%	20
B-3-Union	78%	79%	22
B-10-Union	77%	80%	32
Search1	80%	77%	82
Search2	78%	78%	221
Search3	78%	79%	430

Although the B-2-Union and Search-3 method produce the same precision and recall, the Search-3 method is very costly. Therefore, we recommend using B-2-Union whenever possible (when there are enough good BLAST hits) but use Search-3 when only a single BLAST hit is found.

Although using a simple voting technique for our predictors is not very sophisticated, it works quite well in practice. As a comparison, we used SVM to learn the weights on each predictor, given all of the prediction data (which is a violation of fold isolation). As the predictive performance in this case was almost the same as using a simple voting ensemble, we did not pursue weighting functions further.

B. Proteins with no good BLAST Hit

The case when there are no BLAST hits is the most challenging, and arguably most important scenario. The more novel the protein or organism, the less likely a protein will appear in well-curated and well-annotated databases. In particular, 11% (1,637) of the proteins in our data set did not have a good BLAST hit during cross-validation. These are sequences that are very disparate from those that have been studied, and thus, BLAST will not be able to find a good similar sequence in the database of experimentally verified proteins. One option the user has is to simply accept the top BLAST result, regardless of its *E-value*. However, this method proves to be quite inaccurate, as shown in Table V. Since our predictors of protein function model the sequences in a variety of ways, rather than simply looking for similarities between the sequences directly, our predictors can make predictions on a wider range of protein sequences. For these results, the PFAM-SVM and PA-SVM predictors prove to be the best combination for our ensemble classifiers (since BLAST and PSTs do not perform well on these sequences).

TABLE V
PERFORMANCE ON PROTEINS THAT DO NOT RETURN A GOOD BLAST HIT DURING CROSS-VALIDATION.

Method	Precision	Recall	Average Cost
BLAST NN (Any E-Value)	19%	20%	1
CHUGO Local Voting	55%	32%	812
CHUGO Top-Down Local Voting	56%	32%	58

Each of these methods (accepting a lower quality BLAST hit, or using our local predictors) increase the coverage of just using a BLAST predictor and only accepting good BLAST hits. However, our local predictor strategies (i.e., Local Voting and Top-Down Local Voting) result in a large increase in precision and recall.

Running the voting classifiers for each of the nodes in the hierarchy is expensive. However, as the low exceptions in Table II show, our predictions tend to be consistent with each other. In other words, when a label is predicted as positive for an instance, it is likely that the parent term in the ontology was predicted as positive as well. This is because our classifiers are as inclusive as possible. Due to the fact that we are using an inclusive scheme for training our classifiers, we can do a top-down search on the hierarchy without any significant loss in predictive performance. In fact, as Table IV shows, we actually see an increase in precision when using a top-down model, since we do not consider some nodes that are very unlikely to be positively labeled.

It should be noted that this scheme would not be possible with an exclusive classifier (even if it performed perfectly), due to the inconsistent predictions that it would produce. Therefore, inclusive classifiers have the added advantage that a top-down search will decrease computational cost without the penalty to accuracy that we would see in the case of exclusive classifiers.

There are nodes in the ontology that are fundamentally difficult to predict using sequence-based methods, especially on proteins that are dissimilar to the set of experimentally annotated proteins. By keeping these nodes from the ontology, we can raise the overall predictive performance of CHUGO.

To evaluate a smaller ontology, we need a single measure of how well we are performing on a set of proteins. We use the hierarchical F-measure [21] as an overall measure of our classifier's performance, with $\beta = 1$, which means that precision and recall are weighted as equally important.

$$F - measure = \frac{(\beta^2 + 1) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}}$$

We apply an algorithm that starts with no nodes in the ontology, and add the node that increases our hierarchical F-measure the most. We continue on, until all 406 nodes have been added. Fig. 3 depicts the scores from applying the algorithm to CHUGO and to BLAST independently. Although this algorithm can reach local maxima, it is a good way to compare how well the two predictors perform when excluding hard to predict GO terms. Furthermore, we can see that CHUGO consistently has a higher F-measure than BLAST-NN, even when the ontologies are pruned in each prediction system's favour.

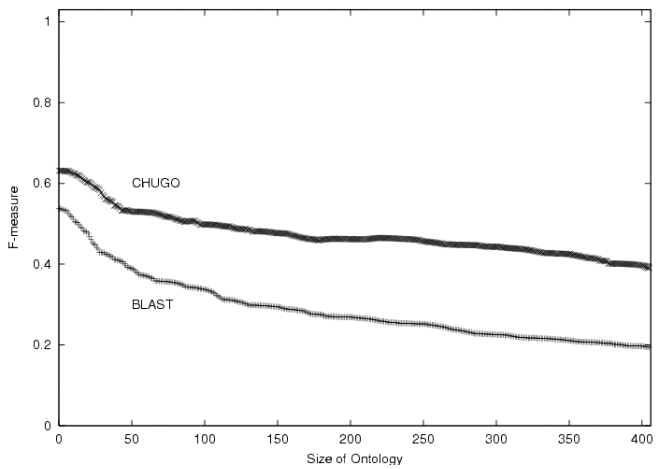


Fig. 3. CHUGO produces better classifiers for dissimilar proteins even when the ontology is pruned in each predictor's favor.

V. Organism Coverage

Knowing how often a good BLAST hit is found during cross-validation is useful, but ultimately a predictive system will be used on unknown proteins, possibly in newly sequenced organisms. It is therefore important to know how often each of the cases described in the preceding section would occur in reality. To approximate how often future unknown sequences would not result in a good BLAST hit, we run a BLAST query for each sequence in two model organisms against our entire data set of GOA experimentally annotated proteins (Table VI).

TABLE VI
NOT FINDING A GOOD BLAST RESULTS AGAINST EXPERIMENTAL DATA ACCOUNTS FOR A LARGE PERCENTAGE OF SEQUENCED PROTEOMES.

Organism	Good BLAST Hit	No Good BLAST Hit
D. melanogaster	60%	40%
S. cerevisae	62%	38%

Within an entire proteome, the number of proteins that do not find a good BLAST hit against our experimental data set is much higher than we have found during cross-validation. This shows an increased importance in the case of no good BLAST hits found, as there are more proteins that are not similar to studied proteins in an entire proteome. This is most likely because there are many more proteins in these proteomes that have not been studied, whose function is unknown, and have no well-studied homologues. The effects would be magnified when examining an organism that is not well studied (relative to those shown in Table VI).

VI. Discussion

Since each of the methods presented (including BLAST) are in fact sequence-based, their performance on disparate proteins declines. When dealing with these proteins, which are far from well-studied proteins (in terms of sequence similarity, and thus homology), the biologists considering them would likely appreciate any leads they can get before beginning lengthy experiments on them. Although our predictors may not be extremely reliable on these disparate sequences, they do allow for a large increase in predictive performance over simply using BLAST Nearest Neighbor. CHUGO is therefore a way of pushing the boundaries of sequence analysis, and ultimately a way of speeding up the process of protein annotation in general.

In the future we would like to find an even more efficient way of using our local predictor technology when there are no good BLAST hits. Accepting a lower quality hit does not appear to be a good way of finding candidate terms to compute, but perhaps there are other global predictors that we could use to seed the search in the hierarchy.

Another sequence-based approach we could take is to use secondary structure predictors to get features for our SVMs. Secondary structure predictors are quite accurate, quick to compute, and have proven to be correlated with some protein functions [12]. We would like to pursue this and other sequence-based features [12].

To help us determine the most efficient classifier, we would also like to have a better approximation of each predictor's cost, rather than assigning a constant value for each. Although this is a good way of getting an initial idea of the total cost of different prediction methods, a more accurate measure would be desirable.

Finally, in the case when a good BLAST hit is found, there are ways to improve our predictors. One obvious direction would be to find better ways to tune the many variables (parameters for each machine learning technology, feature selection, kernels, combinations of predictors).

VII. Conclusion

High-throughput and accurate protein function prediction is important to closing the gap between sequencing data and biological experimental data. Ontologies such GO help to alleviate this problem by providing standardized, hierarchical vocabularies with which to define protein functions. We have

shown three novel methods to exploit this hierarchical nature of GO to increase predictive performance, while retaining efficiency. We utilize the hierarchy to increase the accuracy of our term predictors, to lower the computational cost of running all of these term predictors, and to make our predictions more accurate by adhering to the true path rule.

The fact that many annotations in Gene Ontology are incomplete is partially alleviated through the use of a hierarchical ontology. Our method of building inclusive classifiers is a way of exploiting this hierarchical structure, and dealing with incomplete information. The methods we have presented are therefore applicable to many other domains where there is incomplete data, and a standardized, hierarchical ontology, such as document classification, medical diagnosis, and others.

ACKNOWLEDGMENT

We would like to thank the Protein Engineering Networks of Centres of Excellence (PENCE), the Alberta Ingenuity Centre for Machine Learning (AICML), the Alberta Science and Research Authority (ASRA), the Natural Sciences and Engineering Research Council of Canada (NSERC), Silicon Graphics, Inc., and Sun Microsystems for supporting this research.

REFERENCES

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "A basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[2] C. Li, editor. "Biochemical nomenclature and related documents", Portland Press, second edition, 1992.

[3] Murzin A. G., Brenner S. E., Hubbard T., Chothia C. "SCOP: a structural classification of proteins database for the investigation of sequences and structures". *J. Mol. Biol*. 247, pp. 536-540, 1995.

[4] R. Greiner, A. Grove, D. Schuurmans. "On learning hierarchical classifications". Available from http://citeseer.ist.psu.edu/38202.html, 1997.

[5] S. Chakrabarti, B. E. Dom, R. Agrawal, P. Raghaven. "Using taxonomy, discriminants, and signatures for navigating in text databases", *Proceedings of VLDB 97*, pp. 446-455, 1997.

[6] A. K. McCallum, R. Rosenfeld, T.M. Mitchell, A. Y. Ng. "Improving text classification by shrinkage in a hierarchy of classes". *15th International Conference on Machine Learning*, pp. 359-367, 1998.

[7] K. Wang, S. Zhou, S. C. Liew. "Building Hierarchical Classifiers Using Class Proximity", *Proceedings of the 25th International Conference on Very Large Databases*, pp. 363-374, 1999.

[8] M. Andrade, N. Brown, C. Leroy, S. Hoersch, A. de Daruvar, C. Reich, A. Franchini, J. Tamames, A. Valencia, C. Ouzounis, and C. Sander, "Automated genome sequence analysis and annotation," *Bioinformatics*, vol. 15, pp. 391–412, 1999.

[9] M. Ashburner et al. "Gene Ontology: Tool for the Unification of Biology". *Nature Genetics*, 25(1):25-29, 2000.

[10] A. Bateman et al. "The Pfam protein families database", *Nucleic Acids Research*, 30(1) pp. 276-280, 2002.

[11] M. E. Ruiz, P. Srinivasan, "Hierarchical text categorization using neural networks", *Information Retrieval*, Volume 5, No. 1, pp. 87-118, 2002.

[12] L. J. Jensen, H. Staerfeldt, and S. Brunak, "Prediction of human protein function according to gene ontology categories." *Bioinformatics*, vol. 19, pp. 635–642, 2003.

[13] O. D. King, R. E. Foulger, S. S. Dwight, J. V. White, and F. P. Roth. Predicting gene function from patterns of annotation. *Genome Research*, 13, pp. 896–904, 2003.

[14] R. Apweiler, et al. "UniProt: the universal protein knowledgebase". *Nucleic Acids Research*, 32 pp. D115-D119, 2004.

[15] O. Dekel, J. Keshet, Y. Singer. "Large margin hierarchical classification". *The 21st ICML*, pp. 209-216, 2004.

[16] A. Vinayagam, R. Konig, J. Moormann, F. Schubert, R. Eils, K. Glatting, and S. Suhai, "Applying support vector machines for gene ontology based gene function prediction," *BMC Bioinformatics*, vol. 5, 2004.

[17] B. Poulin, "Sequence-based protein function prediction," Master's thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, 2004.

[18] D. Szafron, P. Lu, R. Greiner, D. S. Wishart, B. Poulin, R. Eisner, Z. Lu, J. Anvik, C. Macdonell, A. Fyshe, and D. Meeuwis, "Proteome Analyst: Custom Predictions with Explanations in a Web-based Tool for High-throughput Proteome Annotations", *Nucleic Acids Research*, Volume 32, July 2004, pp. W365-W371.

[19] S. Kiritchenko, S. Matwin, and A. F. Famili, "Hierarchical text categorization as a tool of associating genes with gene ontology codes," in *Proc. of the Second European Workshop on Data Mining and Text Mining for Bioinformatics*, Pisa, Italy, 2004, pp. 26–30.

[20] "The Automated Function Prediction Special Interest Group Meeting", ISMB 2005, http://ffas.burnham.org/AFP

[21] S. Kiritchenko, S. Matwin, and F. Famili, "Functional annotation of genes using hierarchical text categorization," *in Proc. of the BioLINK SIG: Linking Literature, Information and Knowledge for Biology* (held at ISMB-05), Detroit, USA, 2005.

[22] D. Pal and D. Eisenberg, "Inference of protein function from protein structure," *Structure*, vol. 13, no. 1, pp. 121–130, 2005.

[23] H. Wu, Z. Su, F. Mao, V. Olman, and Y. Xu. Prediction of functional modules based on comparative genome analysis and Gene Ontology application. *Nucleic Acids Research*, 33(9), pp. 2822–2837, 2005.

[24] "Gene Ontology Annotation @ ebi," http://www.ebi.ac.uk/GOA/, 2005.

[25] "the Gene Ontology website," http://www.geneontology.org/, 2005.

View publication stats