Information Sciences 181 (2011) 2512–2528

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Learning-based concept-hierarchy refinement through exploiting topology, content and social information

Tsung-Ting Kuo [1], Shou-De Lin *

National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei 10617, Taiwan

ARTICLE INFO

Article history:
Received 1 January 2010
Received in revised form 11 November 2010
Accepted 11 February 2011
Available online 19 February 2011

Keywords:
Concept hierarchy
Data mining
Machine learning
Content indexing methods
Concept learning
Recommendation

ABSTRACT

Concept hierarchies, such as the ACM Computing Classification Scheme and InterPro Protein Sequence Classification, are widely used in categorization and indexing applications. In the Internet and Web 2.0 era, new concepts and terms are emerging on an almost daily basis, so it is essential that such hierarchies maintain up-to-date records of concepts. This paper proposes a mechanism to identify the most suitable position to insert new terms into an existing concept hierarchy. The problem is challenging because there are hundreds or even thousands of candidate positions for insertion. Furthermore, usually there is no training instance available for an insertion; nor is it practical to assume the availability of a detailed description of the target concept, except in the hierarchy itself. To resolve the problem, we exploit the topology, content and social information, and apply a learning approach to identify the underlying construction criteria of the concept hierarchy. We utilize three metrics (namely, accuracy, taxonomic closeness, and ranking) to evaluate the proposed learning-based approach on the ACM CCS, the DOAJ and the InterPro datasets to evaluate the proposed learning-based approach. The results demonstrate that, in all three metrics, our approach outperforms similarity-based approaches, such as the Normalized Google Distance, by a significant margin. Finally, we propose a level-based recommendation scheme as a novel application of our system. The source code, dataset, and other related resources are available at http://www.csie.ntu.edu.tw/~d97944007/refinement/.

1. Introduction

A concept hierarchy is a tree-based data structure in which higher level nodes represent more general concepts that are connected by relationships like "is-a", "part-of", "include" or some other precedence relations. In this paper, we adopt the conventional definition of a concept hierarchy [7,21] as *a taxonomy or tree of concepts* (*i.e., there is no cycle in it*). Generally, the term *concept* can be any phrase in any language. Concept hierarchies are utilized in various domains, such as knowledge classification, library categorization, web content directories, and product catalogues [4]. For example, the ACM Computing Classification Scheme (CCS) [1] is based on taxonomy of computer science concepts. There are four levels in the 1991 version of CCS (CCS91), as shown in Fig. 1.

Currently, concept hierarchies are refined manually, which is demanding and time-consuming. The ACM CCS, for example, was last updated in 1998 (CCS98), which was seven years after the release of the previous version (CCS91). In fact, since it was first published in 1964, the ACM CCS has only been updated five times: 1982, 1983, 1987, 1991, and 1998. For rapidly

* Corresponding author. Tel.: +886 2 33664888x333; fax: +886 2 33664898.
E-mail addresses: d97944007@csie.ntu.edu.tw (T.-T. Kuo), sdlin@csie.ntu.edu.tw (S.-D. Lin).
[1] Tel.: +886 972 313 873.

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.02.006

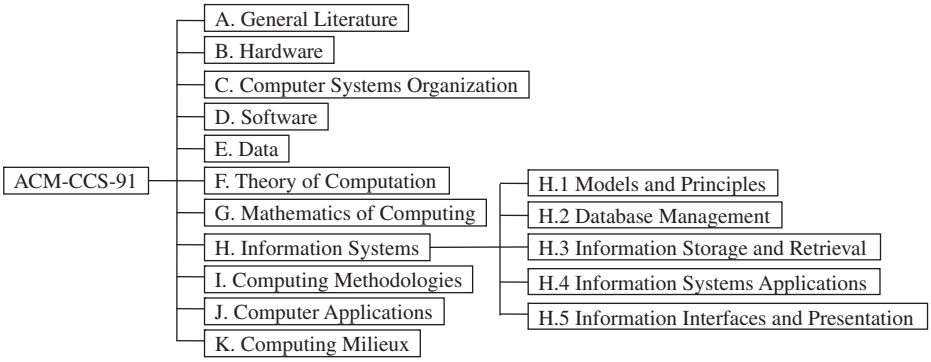

Fig. 1. The components of the CCS91 concept hierarchy. Due to space limitation, only part of the CCS91 is shown.

changing domains like computer science, infrequent manual updating is not sufficient to keep pace with the rapid emergence of new terms or satisfy users' requirements. Therefore, we believe it is essential to design an automatic concept hierarchy refinement framework that can keep concept hierarchies up-to-date.

In this paper, we focus on the following concept hierarchy refinement problem: *given a concept hierarchy and a set of new concepts, how can we identify the most suitable position to insert a new concept into the original concept hierarchy*? Note that we assume the concept can be inserted as the child node of any node (not just leaf nodes). For example, in Fig. 1, we might want to insert a new sub-domain of information system under node H, but not on any of the leaf nodes (i.e., H.1, H.2, . . . , H.5). However, inserting a new concept into a suitable position is not a trivial task, because the number of candidate positions to be inserted is exactly the same as the total number of nodes in the hierarchy. For instance, in the ACM CCS91 there are 1101 candidate positions for insertion, which means that a random assignment can only achieve 0.09% in accuracy.

In a hierarchy, the concept node may contain very little information, e.g., only the concept's name; hence, the conventional document-based hierarchy classification method [4,25] is not very effective in this situation due to the lack of features. Unsupervised approaches, such as similarity-based methods, may be able to solve the problem of insufficient information by inserting a new term based on its similarity (e.g., content-based point-wise mutual information) to the other concept terms. However, the major drawback of similarity-based approaches is that they cannot learn the underlying criteria that were exploited during the construction of the hierarchy. In fact, similarity is the only criterion they consider; therefore, it is not easy for them to determine whether a new concept should be inserted as a sibling, a child or even one of the parents of its closest nodes. Our approach is motivated by the fact that the creators of the hierarchy usually consider some underlying construction criteria, and we believe that learning those criteria could improve the accuracy of the refinement process. For example, a concept hierarchy for animals (Fig. 2) usually assumes there are "is-a" relationships between the concepts, whereas a component-hierarchy of an automobile (Fig. 3) usually assumes there are "part-of" relationships between the concepts. In Fig. 2, a similarity-based approach might find that *Eagle* is the most similar concept to *Owl*, but it would have to decide whether to insert *Eagle* as a sibling or a child of *Owl*. Meanwhile, in Fig. 3, a similarity-based approach might assign *Steering Wheel* to *Wheel/ Tire* or *Steering* categories because their surface forms are similar; however, it should be inserted in *Control System* (under *Equipment*). Clearly, it is difficult for a similarity-based approach to insert new concepts into concept hierarchies with "part-of" relationships. These examples demonstrate why it is essential to learn the original classification criteria when solving concept hierarchy refinement problems.

The above analysis suggests that a learning-based (or supervised) approach might be more suitable for concept hierarchy refinement. One plausible way would be to treat refinement problems as hierarchical or multi-class classification problems. In other words, given a sufficient number of training instances, we can train classifiers to determine the positions of newly acquired terms. However, to realize this goal, we need a certain number of training samples for each class (or node). For example, given 10 training samples for each of the 1101 candidate positions in the ACM CCS dataset, we would need

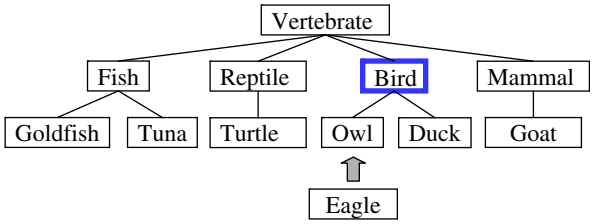

Fig. 2. Using a similarity-based method to insert *Eagle* into an animal concept hierarchy. The ground truth in this example is *Bird*. The similarity-based method is appropriate for such concept hierarchies because they are based on "*is-a*" relationships.

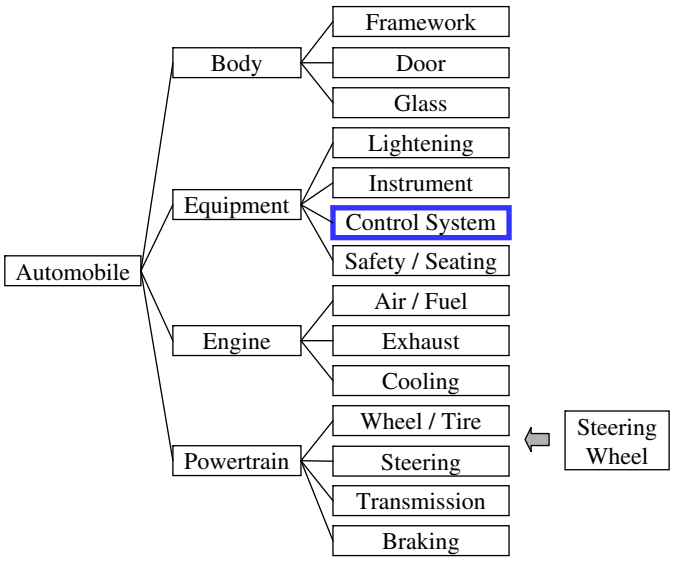

Fig. 3. Using a similarity-based method to insert *Steering Wheel* into an automobile concept hierarchy. The ground truth in this example is *Control System*. The similarity-based method might not be appropriate for such concept hierarchies because they are based on "*part-of*" relationships.

approximately 11,010 training instances. Of course, the major problem with the above design is that such training concepts are not usually available. If there were sufficient concepts for training, they would have been included in the original hierarchy.

To address this issue, we propose a novel learning framework for hierarchy refinement. In the training phase, instead of training a multi-classifier system, we train a single binary classifier to learn whether two nodes have a parent–child relationship in the hierarchy. Training samples are generated automatically by pairing any two nodes in the existing hierarchy and assigning a positive or negative label to the pair. In the insertion (or testing) phase, we pair a new term with each node in the hierarchy and insert it in the position assigned the highest probability by the classifier. We also propose deriving features for learning by using topology, content and social information acquired internally (e.g., from the hierarchy) and externally (e.g., by searching the Web).

The proposed framework includes a hierarchy refinement recommender system that provides level-by-level suggestions to users during the refinement process. Our experiment results demonstrate the efficacy of the recommendation system. Compared to manual refinement methods, the system only needs 3/4 to 1/7 of the time to perform the refinement task.

The major contributions of this paper are as follows:

(1) We propose a framework that models the hierarchy-refinement problem as a classification task in which the lack of the training samples is not a concern. The framework outperforms unsupervised, similarity-based approaches because it learns a hierarchy's underlying classification criteria.
(2) To determine the existence of parent-child relationships, we exploit three types of information: (i) topology information, which is generated based on the structure of the original concept hierarchy; (ii) content information, most of which is acquired by searching the Web; and (iii) social information, which is gathered by using certain bipartite relationships obtained externally.
(3) We evaluate the proposed method on the ACM CCS, the Directory of Open Access Journals (DOAJ) [9], and the InterPro Protein Sequence Classification (InterPro) [17] dataset. In addition, we compare our method's performance with that of several similarity-based approaches. For the evaluation, we utilize two existing metrics, *accuracy* and *taxonomic closeness*, and propose a *rank-based area under curve (rank-based AUC)* metric to assess the effectiveness of the system. The results show that, on the above three datasets, our method outperforms the compared methods in terms of all three metrics.
(4) The proposed framework can be used as an effective and efficient recommender mechanism to help users refine a hierarchy. Our experiment results show that, compared to manual refinement methods, the proposed framework only needs 3/4 to 1/7 of the time to perform the refinement task.

The remainder of the paper is organized as follows. In the next section, we describe the proposed concept refinement method; and in Section 3, we introduce the recommender mechanism. In Section 4, we present the experiment results; and in Section 5, we discuss the results in detail. Section 6 contains a literature survey. Then, in Section 7, we provide some concluding remarks.

2. Related work

Several studies have focused on concept hierarchy generation, where the objective is to create complete concept hierarchies from scratch. For example, Attribute-Oriented Induction [15], Conceptual Abstraction Hierarchy [16], and Self-Organizing Maps [24] derive concept hierarchies for categorical datasets. Some studies utilize clustering techniques, such as hierarchical clustering [7,19–21] and conceptual clustering [11], to build concept hierarchies. Another well-known approach, Formal Concept Analysis [13], which generates concept lattices automatically, is used in various application domains [3,18,29,31]. In this paper, we focus on a different problem where we assume that a manually built concept hierarchy already exists and cannot be altered.

The hierarchical classification problem [10], on the other hand, involves classifying documents into the leaf nodes of a concept hierarchy. That is, hierarchical classification methods try to classify documents into concept hierarchies. In contrast, our objective is to insert new concepts into an original hierarchy. From a technical point of view, our approach tries to learn the relationship between the parent–child nodes, whereas hierarchical classification learns which category a particular document belongs to.

Ontology evolution is a related research area that deals with the problem of incorporating new information into an existing ontology [12]. Some studies focus on the evolution of concept hierarchies (or taxonomies). For example, ReTAX+ [22], a system for revising taxonomies, uses pre-defined rules to resolve inconsistencies when a new concept is added to a hierarchy. The method, which is semi-automatic, requires domain users to specify the dominant attributes and choose a refinement strategy. CleanONTO [27], is a related method is that evaluates and refines taxonomies by using semantic paths extracted from WordNet. However, WordNet only provides semantic information about "is-a" relationships. It has difficulty learning other types of relationships such as "part-of" relationships. To the best of our knowledge, our work is the first attempt to model the general ontology evolution problem by using a supervised framework. The proposed system does not require handcrafted rules or domain users' help, and it can be applied to all types of ontology.

3. Concept refinement method

We use k to denote a term or phrase that represents a concept, and K to denote a set of concepts. A parent-child pair $[k_p, k_c]$ indicates that k_p is the parent node of k_c. $P(K)$ is a set of parent-child pairs such that for each $[k_p, k_c] \in P(K)$, $k_p, k_c \in K$. A concept hierarchy can be defined as a set of concepts together with their relations: $H = \{K, P(K)\}$. We say that a concept hierarchy H_i subsumes H_j if $K_j \subseteq K_i$ and $P(K_j) \subseteq P(K_i)$. The concept hierarchy refinement problem is defined as follows: Given an original concept hierarchy $H_O = \{K_O, P(K_O)\}$ and a new concept set K_N, how can we generate a new concept hierarchy $H_N = \{K_O \cup K_N, P(K_O \cup K_N)\}$ such that H_N subsumes H_O?

3.1. Outline of our approach

Our supervised learning framework for the concept hierarchy refinement problem is illustrated in Fig. 4. In the training phase, we disassemble the original concept hierarchy H_O into a set of parent–child pairs, and use them to learn the hierarchy's underlying classification criteria. In the insertion phase, we pair each new concept $k_n \in K_N$ as a child node with each concept node, and use a trained classifier to determine the likelihood of each pair. Finally, we insert k_n beneath the concept node whose pairing with k_n yields the highest classification score, and generate new concept hierarchy H_N. The training and insertion phases are described in detail in the next two sub-sections, and we explain how the features for classification are generated in Section 2.4.

3.2. Training

The training samples are created automatically from the original concept hierarchy H_O. We want to determine the feasibility of inserting a new concept into an existing concept. Therefore, it is reasonable to train the classifier with information about the insertions that are considered valid and those that are deemed invalid.

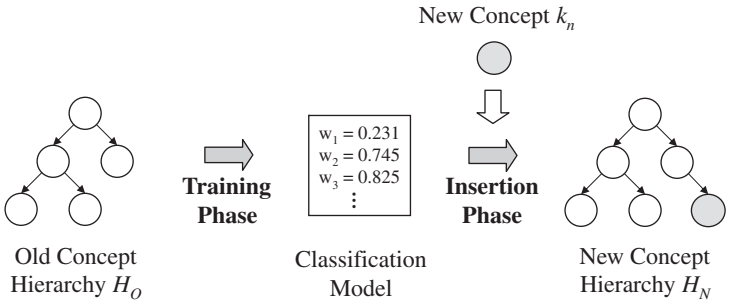

Fig. 4. The framework of the proposed concept hierarchy refinement method.

If we pair all the $n = |K_O|$ nodes in the hierarchy with each other, there could be *Permutation* $(n, 2) = n * (n-1)$ possible combinations of ordered pairs. However, since there are only $(n-1)$ edges in the network, we can assume there are only $(n-1)$ valid pairs (or positive training examples). The remaining $n * (n-1) - (n-1) = (n-1)^2$ pairs are invalid or negative pairs. Fig. 5 shows an example of the training process. To learn the underlying criteria of H_O, which has 5 concepts $\{k_1, k_2, k_3, k_4, k_5\}$, we use all possible parent–child pairs in H_O to compile the training dataset. There are *Permutation* $(5, 2) = 20$ possible pairs. If any one of the possible pairs exists in H_O such as $[k_1, k_2]$, then the class label of that pair is +1 (the gray labels in Fig. 5). Otherwise, the class label is -1 (the white labels in the figure). Since H_O is a tree, there are $5 - 1 = 4$ positive tuples, and $(5 * 4) - 4 = 16$ negative tuples. Next, we construct three features, i.e., the topology, content and social features (which we describe in Section 2.4) for the 20 training data pairs to train a binary classifier.

3.3. Insertion

In this phase, the model tries to determine the most appropriate insertion position in H_O for a new concept k_n, based on the classification model learned previously. Note that each insertion creates a parent–child pair, and the number of possible positions for insertion is equivalent to the total number of concepts in the hierarchy.

First, we create the insertion dataset $P_{insertion}(k_n \cup K_O) = \{[k_i, k_n] \mid k_i \in K_O\}$ for k_n. That is, we generate test pairs by appending k_n as child node to all $k_i \in K_O$ as possible parent nodes. Therefore, we generate $|K_O|$ tuple in the insertion dataset, where $|K_O|$ represents the number of elements in K_O. Next, we construct the features for each pair by using the methods described in the next subsection. Then, we utilize the classification model generated during the training step to estimate the likelihood of each plausible insertion position of k_n. Finally, we select a pair $[k_{max}, k_n]$ with the highest likelihood score, and insert k_n into k_{max}. Fig. 6 shows the insertion phase when a new concept k_6 is inserted into k_3 in the concept hierarchy H_O in Fig. 5.

3.4. Feature construction

To train a classifier, we need to construct a set of representative features for each training sample, as shown in Table 1. The training set consists of pairs of parent–child concepts. Note that the child node (i.e., the new concept to be inserted) of each parent–child pair $[k_p, k_c]$ is identified in the insertion phase; therefore, we do not have to encode any individual information about k_c in the training phase. As a result, based on the sources of information, it is possible to exploit two types of knowledge to construct the features: the information about k_p itself (denoted as "parent" in Table 1), and the information about the relationship between k_p and k_c (denoted as "relation" in the table). In a Bayesian learning scenario, the parent features carry the prior information, while the relation features are useful for learning the likelihood function. Orthogonally, we can divide the features into three groups (i.e., topology, content and social) based on the information that each feature possesses. Table 1 shows these two dimensions of feature-categorization. Next, we use the pair $[k_{info}, k_{user}] = [$*information systems*, *user machine systems*$]$ as an example to demonstrate the features.

First, the topology information of the parent node k_p is acquired directly from the structure of H_O. There are three topology features, namely, the level (or depth in the hierarchy) of k_p, the number of siblings of k_p (including k_p itself), and the number of children of k_p.

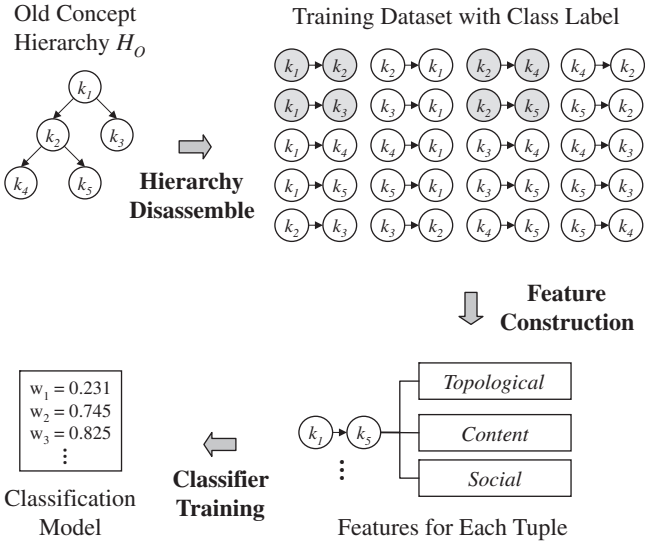

Fig. 5. An example of the training phase: we train a classification model from a concept hierarchy $H_O = \{\{k_1, k_2, k_3, k_4, k_5\}, \{[k_1, k_2], [k_1, k_3], [k_2, k_4], [k_2, k_5]\}$.

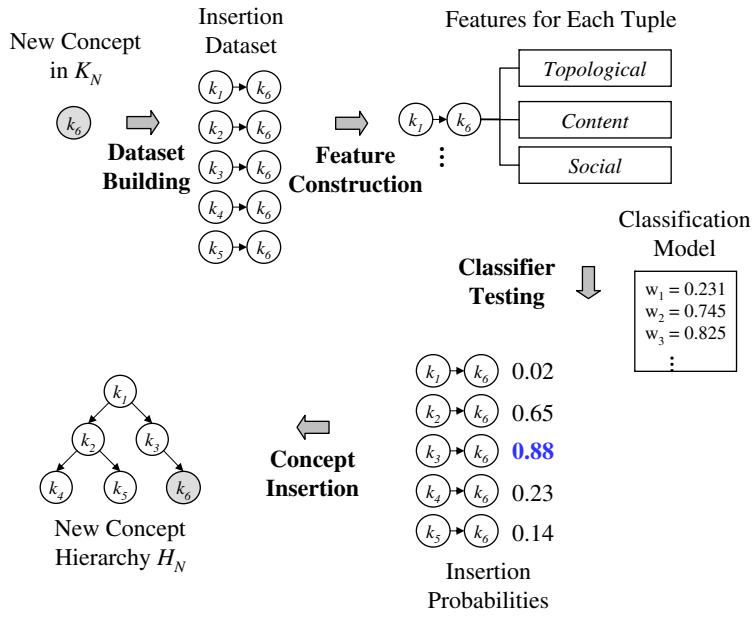

Fig. 6. An example of the insertion phase: we use the trained classification model to insert a new keyword k_6 into k_3 in the original concept hierarchy H_O to form the new hierarchy H_N.

Table 1
Supervised learning features by category.

Group	Feature	Parent	Relation
Topology	Level of parent	O	
	Siblings count of parent	O	
	Children count of parent	O	
Content	Frequency of parent	O	
	Partial name matching		O
	Co-occurrence on Web pages		O
	Jaccard similarity		O
	NGD similarity		O
Social	Number of coauthor		O
	Number of coauthor sequence		O

Second, to generate the content features, we compare the concept terms directly, or use the concept terms as keywords to search the Web. There are five types of content features:

- *Frequency of parent* $C(k_p)$. This is the logarithmic value of the web page count returned by a search engine when querying with the parent concept k_p. For example, if the query "*information systems*" returns 100,000,000 search results, $C(k_{info}) = log\,(100{,}000{,}000) = 8$. The intuition behind this feature is that a new term is more likely to be inserted into a popular concept than an unpopular one.
- *Partial name matching*. Let $W(k_p)$ be the set of words that constitute k_p and $W(k_c)$ be the set of words that constitute k_c. Then, we can define the *partial name matching* feature as $|W(k_p) \cap W(k_c)|$. For example, *Partial_Name_Matching* $(k_{info}, k_{user}) = |W(k_{info}) \cap W(k_{user})| = 1$ (i.e., the word *systems* is the same).
- *Co-occurrence on Web Pages* $C(k_p, k_c)$. This is the logarithmic value of the web page count returned by a search engine while using "kp" + "k_c" as the query. For example, if the query *"information systems"* + *"user machine systems"* returns 10,000 page counts, $C(k_{info}, k_{user}) = \log(10{,}000) = 4$.
- *Jaccard similarity*. We define the Jaccard similarity feature as $C(k_p, k_c)/(C(k_p) * C(k_c))$,*where* $C(k_p)$, $C(k_c)$ *and* $C(k_p, k_c)$ are defined as above. Following the previous example, if $C(k_{info}) = 8$ and $C(k_{user}) = 5$, then the Jaccard_Similarity $(kinfo, k_{user}) = 4/(8 * 5) = 0.10$.
- *NGD Similarity*. The Normalized Google Distance (NGD) [5] similarity feature is defined as follows:

$$NGD\_Similarity(k_p, k_c) = 1 - \frac{\max(C(k_p), C(k_c)) - C(k_p, k_c)}{N - \min(C(k_P), C(k_c))}.$$

N is the logarithmic estimated total number of searchable web pages. The value does not affect the result if it is large enough [5]. In our experiment, we assume that $N = 12$. For example, $NGD\_Similarity\ (k_{info}, k_{user}) = 1 - (\max(8, 5) - 4)/(12 - \min(8, 5)) = 0.43$.

Finally, the social features are generated based on the relationships between the concept-related objects. In other words, two concepts are correlated if they are linked through some objects. In our experiment, we select authors as the objects and authorship as the relationship. There are two types of social features:

- *Number of social connections* (or *number of co-authors* in our experiment). Let $A(k_p)$ be the set of the top-r relevant objects (e.g., individuals or web pages) to the parent concept k_p, and let $A(k_c)$ be the set of the top-r relevant objects to the child concept k_c. Both sets of objects can be generated by searching the Web or a publicly available database, such as Microsoft Academic Search. Given $A(k_p)$ and $A(k_c)$, we can construct a concept-object bipartite graph in which the links represent a relationship, e.g., authorship. Then, we define the *Number of Social Connections* feature as the number of distinct paths from one concept to another in the bipartite social graph. For example, if the top-three relevant objects of k_{info} are $\{a_1, a_2, a_4\}$, and the top-three relevant objects of k_{user} are $\{a_1, a_3, a_5\}$, we can build the bipartite graph shown in Fig. 7. Then, the *Number_of_Social_Connections* $(k_{info}, k_{user}) = 1$ since there is one path from k_{info} to k_{user} (indicated by the bold lines in Fig. 7).
- *Number of sequential social connections* (or *number of coauthors sequence* in our experiment). We extend the concept of social connections feature to construct the *Number of sequential social connections* feature. First, we build the same concept-object bipartite graph using $A(k_p)$ and $A(k_c)$. Then, for each object in $A(k_p)$ and $A(k_c)$, we retrieve the related objects (e.g., the co-authors) to construct an enriched graph. Finally based on the graph, we compute the total number of paths from k_p to k_c. An example is shown in Fig. 8, where a_6, a_7, a_8, a_9 are co-authors extended from the neighbor authors of those concepts; and the *Number_of_Sequential_Social_Connections* $(k_{info}, k_{user}) = 2$ (represented by the bold lines and dashed lines in the figure). Note that such relational information is usually available from some public source, such as Wikipedia. In Section 4, we explain how different features perform to provide some insight into their usefulness.

To summarize, we construct three groups of features: *topology* (the level of the parent, the sibling count of the parent, and the children count of the parent); *content* (frequency of the parent, partial name matching, co-occurrence on web pages, the Jaccard similarity, and the NGD similarity); and *social* (the number of social connections and the number of sequential social connections).

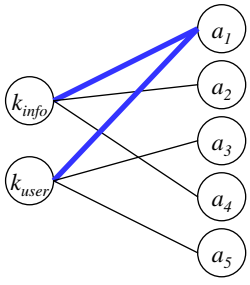

Fig. 7. An example of a concept-object bipartite graph used to generate the number of social connections feature. There is one path $(k_{info} - a_1 - k_{user})$ from k_{info} to k_{user}; thus, the *Number_of_Social_Connections*$(k_{info}, k_{user}) = 1$.

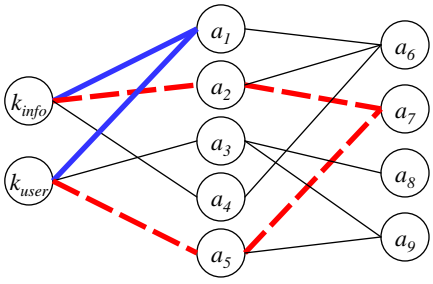

Fig. 8. An example of a graph used to generate the number of sequential social connections feature. There are two paths $(k_{info} - a_1 - k_{user}$ and $k_{info} - a_2 - a_7 - a_5 - k_{user})$ from k_{info} to k_{user}; thus, the *Number_of_Sequencial_Social_Connections*$(k_{info}, k_{user}) = 2$.

3.5. Feature enrichment using neighboring nodes

In the previous section, we only considered the parent node (i.e., the node to be inserted in the hierarchy) and its relationship to the new concepts. However, concepts that are close to each other in the hierarchy are deemed to be more relevant. Thus, it is reasonable to assume that the concept to be inserted should also have certain relevance to the nodes in the neighborhood of the target parent node. Therefore, we enrich the feature set by finding the relationship of a concept k_c to its potential parent k_p's relatives, namely k_p's parent (i.e., k_c's grandparent), k_p's children (i.e., k_c's siblings), and k_p's siblings. Then, we can create the following features by using the new term k_c and k_p's relatives (for example, if we consider inserting a new node k_c into k_2 in Fig. 5):

- Relational features between k_c and its grandparent (e.g., $[k_1, k_c]$).
- Maximum/minimum/average relational features for k_c and its siblings (e.g., $[k_4, k_c]$ and $[k_5, k_c]$).
- Maximum/minimum/average relational features for k_c and k_p's sibling (e.g., $[k_3, k_c]$).

Note that the numbers of siblings are not fixed. Hence, to maintain a fixed size feature vector for learning, we need to aggregate their values by taking the maximum, minimum or average scores. In addition, we only enrich the content and social features because the enrichment process itself is inherently topological.

4. A level-based recommender mechanism for refinement

A concept hierarchy like ACM CCS may contain hundreds or even thousands of possible positions for insertion. Therefore it might not be practical to expect a system to identify the exact position every time. Our experiment results show that our system improves the insertion accuracy significantly from 0.09% (random guess) to 34%; however, such accuracy might not be sufficient for auto-insertion. Nevertheless, in this section we propose an application for our system that enables users to save time in identifying the correct positions for insertion.

We exploit the proposed framework to develop a level-based ranking system that recommends insertion positions level-by-level. That is, starting from the top, the system suggests the order that the positions in each level should be utilized. The rationale for this approach is that when users want to insert new concepts into a hierarchy, they do not need to search the whole hierarchy node-by-node. Instead, they can identify the correct position level-by-level in a top-down fashion. Given a new concept k_n, in each level v there is a node (in a set of possible candidate positions K_v) that is the most appropriate for k_n. Thus, in each level, the user examines each pair of concepts (k_n, k_v) to learn whether they should be connected, where $k_v \in K_v$. Once a match is found, the user moves to the next (lower) level to identify the child nodes that could be associated with k_n. The process continues until there are no more matches for k_n, after which its position can be determined. An example of the above process is shown in Fig. 9. If a user wants to insert a new concept k into the hierarchy, he/she starts at the root x, and examines the child concepts of x one by one from y_1 to y_3, if necessary. In this example, while examining the relationship, the user realizes that k should be inserted somewhere in concept y_3 (Fig. 9(b)). There is no need to examine the remaining nodes in that level, so the user can go to next (i.e., lower) level to check the children of y_3 (z_1 and z_2 in this case). The process is repeated (Fig. 9(c)) until the correct position for insertion is confirmed in 9(d).

Given that the nodes in K_v are examined in random order, the total time, T, needed to complete the insertion process can be estimated as $T = t * d * b/2$, where t is the average time required to examine one pair, b is the average branching factor, and d is the depth of k_n in the hierarchy. Since the nodes in K_v are ordered randomly, users are expected to examine $b/2$ nodes in each level. In this case, the users do not have to examine all nodes in the hierarchy to determine the most appropriate position for insertion. They only have to examine $d * b$ nodes at most.

However, in domains like molecular biology, evaluating each parent-child pair usually takes a substantial amount of time (i.e., t is large) and financial resources. For example, in Gene Ontology [28], it is usually necessary to search a large amount of literature to confirm a parent–child connection. To construct a transcription network as small as the Genetic Regulatory

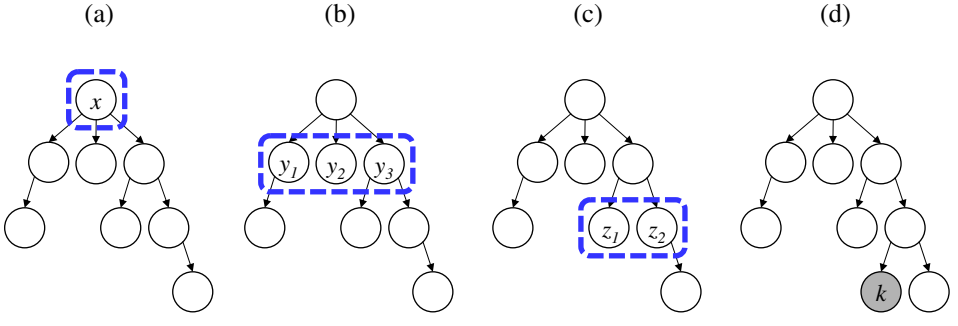

Fig. 9. Inserting a new concept k into an existing hierarchy without the rank recommender; (a)–(d) represent steps 1–4, respectively.

Network (GRN) for Embryos (Fig. 10), it usually necessary to perform a series of biological experiments [8,26,30]. The researchers responsible for creating and maintaining the PRINTS database, which identifies protein families, observed that "creating and annotating family discriminators is time consuming", and "the database is still small relative to the number of protein families that exist, largely because the detailed documentation of entries is extremely time-consuming" [2]. Because the cost of scrutinizing each pair can be very high, reducing the number of examinations is critical. To solve this kind of problem, we provide a recommender that can rank the candidates in K_v successfully based on the likelihood that they have a parent-child connection to k_n. In Section 4.7, we show that for InterPro dataset, the recommender ranks the gold standard in the top 1/7 of the list; therefore, we only need to exam $d * b/7$ pairs. An ideal recommender would completely remove b from the equation as $T = t * d$, and is generally beneficial for tasks with large b (e.g., in the InterPro dataset that we used for evaluation, the largest branching factor is 74).

To provide ranked recommendations to users, we can simply utilize the insertion probability (or likelihood score) described in Section 2.3. The probability allows us to recommend the rank for each level easily. Fig. 11 shows the insertion process with ranked recommendations for the example in Fig. 9. Again, we try to insert k into the hierarchy in a top-down fashion. Using the recommendations as a guide, we can evaluate each level according to its rank. Therefore, in Fig. 11(b) we can find the correct position in 2 steps (the rank of the matched concept is 2). Similarly, in Fig. 11(c) we can find the correct position in the first step. This example demonstrates that the insertion process requires less time if the recommended rank is reliable. In Section 4.7, we evaluate the effectiveness of the recommender system based on how many nodes have to be traversed under different algorithms.

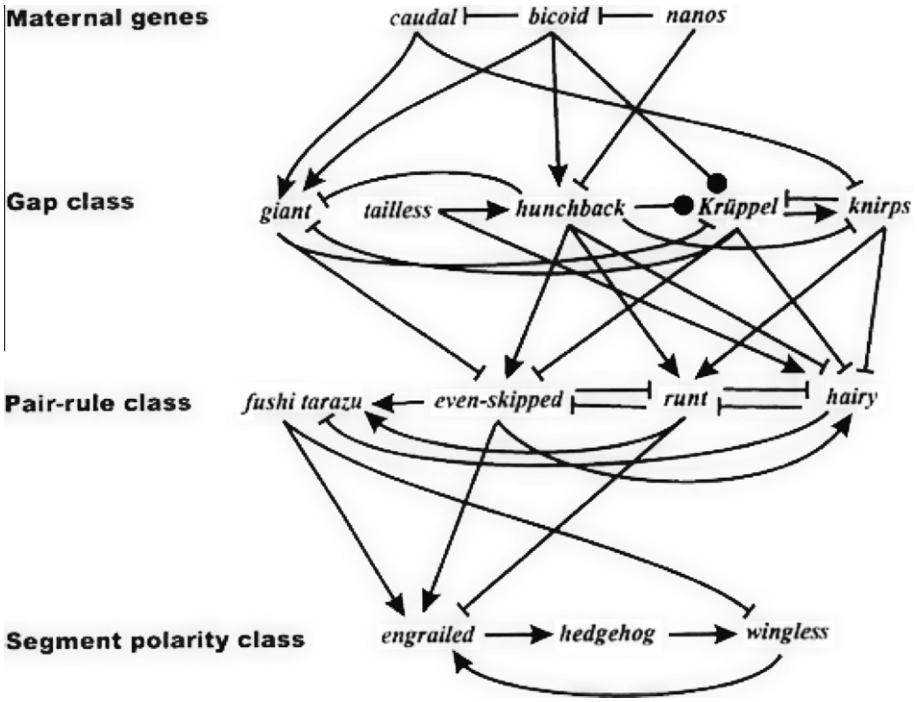

Fig. 10. The Genetic Regulatory Network for Embryo transcription networks.

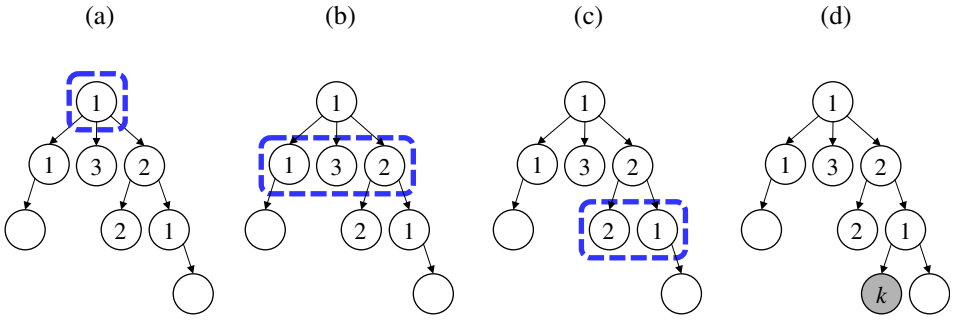

Fig. 11. Inserting a new concept k into an existing hierarchy with the rank recommender in the proposed system; (a)–(d) represent steps 1–4, respectively.

5. Evaluations

The objective of the experiments is to verify the following four hypotheses about the concept hierarchy refinement problem.

(1) Learning-based methods can improve on the results derived by non-learning based methods.
(2) Topology, content and social features make diverse contributions when refining a concept hierarchy.
(3) Enriching features by using information about neighboring nodes can improve the accuracy.
(4) A learning-based concept hierarchy refinement framework can be used as an effective and efficient recommender mechanism to help users refine a hierarchy.

For hypothesis (1), we compare similarity-based methods with learning-based methods; for (2), we compare the performance of learning-based methods using different groups of features; for (3), we compare the results of learning-based methods with/without feature enrichment; and for (4), we compare the methods to determine which one is the most efficient in terms of average traversed nodes per insertion.

5.1. Datasets

To evaluate our framework, we used three datasets: ACM CCS, DOAJ, and InterPro. For ACM CCS, we used ACM CCS91 as the original concept hierarchy H_O in the training phase. After data cleaning and preprocessing (e.g., removing duplicate entries), we extracted $|K_O| = 1101$ from CCS91. Thus, the training dataset size = $|K_O| * (|K_O| - 1) = 1{,}211{,}100$. The height of the CCS91 tree is 4, and the average branching factor is 4.97. In the insertion phase, we utilized ACM CCS98 as the gold standard for evaluation. We wanted to observe how well our system could simulate the terms that users worked on in the 1990s in order to add new concepts to the original CCS91. Thus, concepts in CCS98 that were not in CCS91 were treated as new concepts to be added. As a result, there were $|K_N| = 244$ new concepts to be inserted. Our system had to select an appropriate position from the 1101 possible positions available for insertion. Of the 244 new concepts, 3.7% had to be inserted into leaf nodes and the rest were connected below non-leaf nodes.

For the second dataset, DOAJ, we also utilized the ACM CCS91 dataset to build a classification model in the training phase. However, in the insertion phase, we used keywords listed in the 153 computer science journals in DOAJ. There are 209 keywords in the journals. We believe the keywords represent important evolving concepts in the computer science field. Therefore, we tried to add them to CCS91 to evaluate the effectiveness of our system. After removing duplicate concepts and those that were too general, there were $|K_N| = 83$ new concepts. We asked users to annotate the ideal insertion positions manually. Of the 83 concepts, 3.6% had to be inserted into the leaf nodes.

For the third dataset, InterPro, we extracted the largest protein hierarchy tree (i.e., "GPCR, rhodopsin-like superfamily"). Of the 254 nodes in the tree, we randomly removed $|K_N| = 24$ concepts for use in the insertion phase; the remaining 230 concepts were used as the original concept hierarchy H_O. Thus, the training dataset size = $|K_O| * (|K_O| - 1) = 52{,}670$. The height of the InterPro tree is 5, and the average branching factor is 4.49. All 24 new concepts had to be inserted into non-leaf nodes.

5.2. Resources and baseline methods

In this experiment, we exploited the Naïve Bayes (NB) classifier. We tried various classifiers, such as Support Vector Machine, Random Forest, Logistic Regression, AdaBoost, and combinations of them, but none of them performed significantly better than NB. Hence, we only consider the results of the NB classifier.

We utilized Yahoo! as the standard search engine to derive content features; Google produced similar results. To generate the coauthors and coauthor sequence graphs, we used Microsoft Academic Search; Google Scholar yielded similar results. The system code is written in Java, and the system runs on a PC with an AMD Opteron 2350 2.0 GHz Quad-core CPU and a 32GB RAM.

We used similarity-based methods for comparison. That is, given certain similarity metric (as have been exploited in Section 2.4) to generate the features (e.g., NGD similarity), we inserted each new concept into the most similar node in the original concept hierarchy. To integrate all the similarity measures, we applied the 1-Norm average of the features as one of the methods for comparison. We also tried a 2-Norm average and obtained very similar results.

5.3. Measurements

Since there is no consensus on evaluation metrics for this problem, we use the accuracy, taxonomic closeness, and ranking metrics to evaluate the performance of our proposed method in the experiments. The accuracy is the most straightforward metric to calculate. An insertion is considered correct if the predicted parent node is the same as the gold standard or the user-annotated solution. Thus, the accuracy is defined as follows:

$$accuracy = \frac{number\_of\_correct\_insertions}{number\_of\_new\_keywords}.$$

The denominator is the total number of new concepts in the experiment. It is equal 244 concepts in the ACM CCS dataset, 83 in the DOAJ dataset, and 24 in the InterPro dataset.

The drawback with the accuracy measure is that it cannot distinguish between missed insertions that are close to the target and those that differ by a large margin. However, the second metric, taxonomic closeness, resolves the problem. The reason is that even if an insertion is incorrect, we consider that a closer prediction is better than one that differs significantly. To evaluate the taxonomic closeness, we apply the concept of taxonomy similarity [6,14,23]: Let *top* be the root concept, and $\delta(a, b)$ be the edge distance of the two concepts a and b in a concept hierarchy. The least common super-concept $LCS(a, b)$ is defined as

$$LCS(a,b) = c \text{ such that } (\delta(a,c) + \delta(b,c) + \delta(top,c)) \text{ is minimized}.$$

Then, the taxonomy similarity between two nodes a and b is defined as

$$T_{sim}(a,b) = \frac{\delta(top,c)+1}{\delta(a,c)+\delta(b,c)+\delta(top,c)+1}.$$

Let us consider the intuition behind this measurement. For two concepts (a, b) in a concept hierarchy, the shortest taxonomic distance between them is $\delta(a, c) + \delta(b, c)$, where $c = LCS(a, b)$. However, even with the same taxonomic distance, the similarity of two "more general" concepts should be lower than the similarity of two "more specific" concepts. For example, in the animal concept hierarchy in Fig. 2, the distance of (*Fish*, *Mammal*) is 2, which is equal to the distance of (*Owl*, *Duck*). However, intuitively, the similarity of the former pair is lower than that of the latter pair. In this sense, the "generality" is considered by adding $\delta(top, c)$ to the formula.

The taxonomic distance, which is a precise number in the range [0–1], represents the similarity of a given concept pair (a, b). When $T_{sim} = 1$, the two concepts (a, b) are exactly the same. Consider the concept hierarchy shown in Fig. 12. Let a be the actual parent of the new concept and b be its predicted parent; then the taxonomy similarity $T_{sim}(a, b) = (1 + 1)/(1 + 3 + 1 + 1) = 0.33$. We use the average T_{sim} to evaluate the overall taxonomic closeness of the predicted and actual insertion positions.

Although the taxonomic closeness shows how close the predicted position is to the target position, it does not really reveal the system's ability to identify the ideal solution. If we view our system as a filter, the top p% positions of candidates ranked by the system are provided to the user for a final check. In other words, if the system can rank the new concepts in the top 10% of all 1100 possible choices, we can simply provide the user with the top-110 candidates returned by the classifier. Then, the user can choose the most appropriate candidates. As result, the system can save as much as 90% of the time required by humans to identify the best insertion position. Therefore, we evaluate the ranking based on the accuracy of the top-h ranked predictions. That is, we regard a prediction as correct if the actual insertion position is ranked among the top-h predictions. By moving h, we can draw a *ranking-based accuracy curve*, with the ranking threshold h as the horizontal axis and the accuracy as the vertical axis. The idea is similar to an ROC curve, although the ROC curve is not as meaningful in our case because there is only one positive instance. The area under a ranking-based accuracy curve is defined as the *rank-based AUC*. If the rank-based AUC is high, it implies that the actual answer has higher probability of being included in the top ranked positions. We adopt rank-based AUC because, even if two methods have similar accuracy or average taxonomy similarity, the method that ranks the actual answer higher is deemed to be better.

5.4. Results for the ACM CCS dataset

In the experiments, we compare the performance of random assignment, similarity-based, and learning-based methods. Similarity-based methods determine which node in the hierarchy is the most similar to the new concept, and insert the concept as a child of that node. To be fair, we utilize similarity measures based on the topology, content and social information features because we exploited them in the proposed learning framework. The measures include all the following features, which are also shown in Table 1: level of the parent (Level), sibling count of the parent (Sibling), children count of the parent (Children), frequency of the parent (Frequency), partial name matching (Name), co-occurrence on web pages (Page), Jaccard

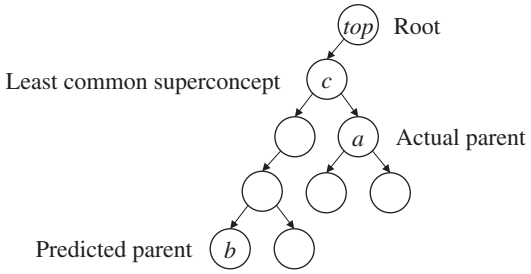

Fig. 12. An example of a concept hierarchy for computing the taxonomy similarity. In this hierarchy, the taxonomy similarity $T_{sim}(a, b) = 0.33$.

similarity (Jaccard), NGD similarity (NGD), number of coauthors (Coauthor), and number of coauthor sequences (Sequence). We also combine these baseline methods by taking the mean (1-Norm) of the decision values of the compared methods. For learning-based methods we exploit a different combination of features, namely, topology features only, content features only, content features with enrichment, social features only, social features with enrichment, all features without enrichment, and all features with enrichment.

The results are shown in Table 2, including the degree of accuracy, average T_{sim}, and rank-based AUC. The proposed learning-based method with all features outperforms baseline methods, and the proposed method performs even better with enriched features.

For the accuracy metric, the mean combination of baseline methods (1-Norm) (20.08%) outperforms all other baseline methods. This is reasonable because the measure combines information acquired from different sources. In learning-based methods, content features generally outperform social and topological features. With all the features and enrichment, our learning-based methods achieved 34.84% accuracy, which was the best score overall in the experiments. The results indicate that, in this difficult multiple-choice problem (i.e., the accuracy reaches only 1/1101 = 0.09% based on random guess), our system can answer one out of three questions correctly. A similar situation holds for the average taxonomy similarity metric, where a learning-based method with all features and enrichment can achieve 0.51 T_{sim} on average, which is 0.11 better than the best baseline method. For rank-based AUC, the learning-based method with all features and enrichment also outperforms the other methods with a score of 97.28%, which is 4.58% better than the best baseline method.

Fig. 13 shows the ranking-based accuracy curves for four methods: random assignment, the combined means of the baseline methods (1-Norm), the learning-based method with all features, and the learning-based method with all features and enrichment. We select the 1-Norm combination method for comparison with the proposed system because, overall, it outperforms the other baseline methods (Table 2). As shown in Fig. 13, when we set the ranking threshold h = 100, the 1-Norm method achieves 73% accuracy; however, with enrichment, the proposed system yields 92% accuracy. In other words, our system enables users to complete an insertion task much faster (1101/100 = 11 times faster) than the baseline method with reasonable accuracy (92%) Similarly, when h = 50, users can finish the task 22 times faster with a slightly lower accuracy rate (87%).

5.5. Results for the DOAJ dataset

The results of the evaluation on the DOAJ dataset are listed in Table 3. Similar to the experiments on ACM CCS, the proposed learning-based method with all enriched features outperforms the other methods in terms of all three metrics. Generally, the 1-Norm method outperforms the other baseline methods. In the learning-based methods, "social features with enrichment" yields better accuracy than the other features. With all features and enrichment, our learning-based methods can achieve 19.28% accuracy, which is the best score overall. Likewise, for the average taxonomy similarity metric, the learning-based methods with all features and enrichment can achieve 0.36 T_{sim} on average. For rank-based AUC, learning-based methods with all features and enrichment yield the best performance (95.87%).

5.6. Results for the InterPro dataset

The results for the InterPro dataset are shown in Table 4. It is noteworthy that, in learning-based methods, the "social features with enrichment" feature performs better than the other metrics (accuracy = 54.17%, average T_{sim} = 0.71, and

Table 2
Experiment results for the ACM CCS dataset.

Method			Accuracy (%)	Average T_{sim}	Rank-based AUC (%)
Random			0.09	0.13	50.05
Similarity-based (baseline)	Topology	Level	0.00	0.11	5.06
		Siblings	0.01	0.15	62.28
		Children	0.00	0.17	86.74
	Content	Frequency	0.00	0.13	60.73
		Name	5.74	0.29	37.63
		Page	6.56	0.25	85.30
		Jaccard	13.93	0.36	87.40
		NGD	9.02	0.36	85.07
	Social	Coauthor	4.92	0.23	21.55
		Sequence	7.79	0.27	28.26
	Combined	1-Norm	20.08	0.40	92.70
Learning-based	Topology	Topology	0.00	0.17	88.31
	Content	Content	14.04	0.36	88.86
		Content-enriched	32.34	0.49	96.57
	Social	Social	11.06	0.34	47.49
		Social-enriched	23.36	0.40	74.09
	Combined	All	29.51	0.47	96.49
		All-enriched	**34.84**	**0.51**	**97.28**

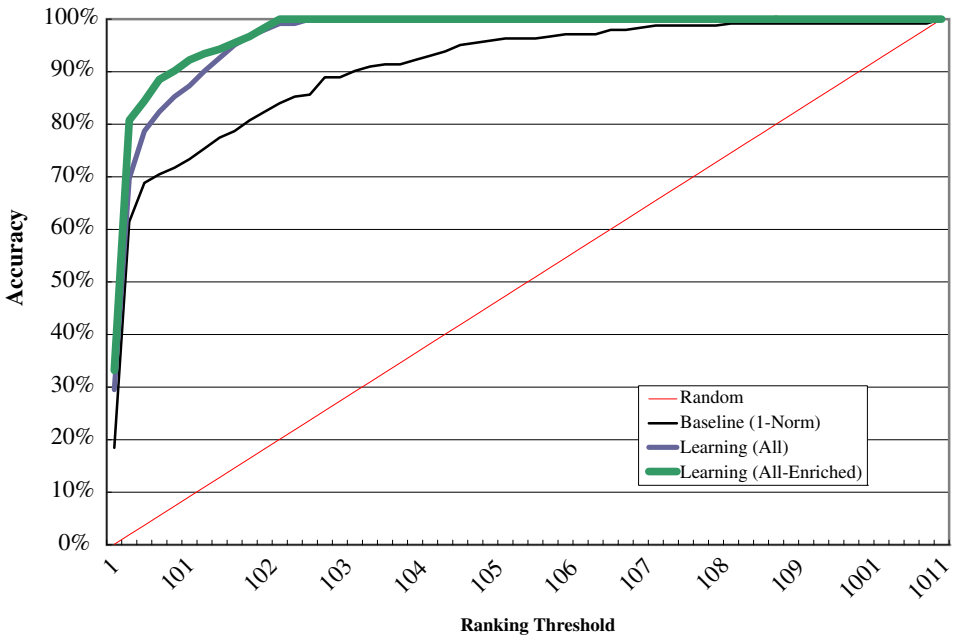

Fig. 13. Ranking-based accuracy curves for the ACM CCS dataset.

Table 3
Experiment results for the DOAJ dataset.

Method			Accuracy (%)	Average T_{sim}	Rank-based AUC (%)
Random			0.09	0.13	50.05
Similarity-based (baseline)	Topology	Level	0.00	0.13	4.42
		Siblings	0.00	0.18	61.71
		Children	0.05	0.22	87.41
	Content	Frequency	0.00	0.13	63.87
		Name	2.41	0.23	29.82
		Page	4.82	0.22	74.75
		Jaccard	6.02	0.24	77.33
		NGD	4.82	0.23	83.35
	Social	Coauthor	2.41	0.21	9.95
		Sequence	2.41	0.21	14.69
	Combined	1-Norm	8.43	0.28	85.49
Learning-based	Topology	Topology	0.00	0.18	89.15
	Content	Content	3.61	0.25	78.81
		Content-enriched	9.64	0.26	93.92
	Social	Social	8.43	0.27	35.87
		Social-enriched	16.87	0.35	65.05
	Combined	All	12.05	0.33	95.02
		All-enriched	**19.28**	**0.36**	**95.87**

rank-based AUC = 99.02%). Among the baseline methods, "children count of parent topology" outperforms the other metrics (accuracy = 12.50%, average T_{sim} = 0.46, and rank-based AUC = 90.13%).

5.7. Evaluation of the level-based refinement recommender mechanism

To evaluate the level-based recommendation system described in Section 3, we propose an intuitive metric: the traversed length of an insertion. The traversed length is the total number of nodes that must be traversed during the insertion process. As mentioned in Section 3, for some tasks, such as biology hierarchy refinement, investigating whether there is a parent–child relationship between two nodes (usually genes or proteins) takes a great deal of time. Therefore, it is essential that the total number of pair-wise investigations should be reduced. The traversed length measures the number of investigations exactly. As shown in Fig. 11, the traversed length given the recommendation (discussed in Section 3) is 1 + 2 + 1 = 4.

Table 5 shows the results of applying the average traversed length on the three datasets. For the ACM CCS and DOAJ datasets, the combined feature with enrichment yields the best performance; and for the InterPro dataset, the social feature with

Table 4
Experiment results for the InterPro dataset.

Method			Accuracy (%)	Average T_{sim}	Rank-based AUC (%)
Random			0.43	0.29	54.40
Similarity-Based (Baseline)	Topology	Level	0.00	0.24	8.61
		Siblings	4.17	0.36	56.20
		Children	12.50	0.46	90.13
	Content	Frequency	0.00	0.30	50.33
		Name	0.00	0.34	14.78
		Page	0.00	0.44	43.59
		Jaccard	0.00	0.43	46.09
		NGD	0.00	0.39	51.20
	Social	Coauthor	0.00	0.26	0.87
		Sequence	0.00	0.27	0.87
	Combined	1-Norm	0.00	0.38	63.35
Learning-based	Topology	Topology	12.50	0.46	86.90
	Content	Content	12.50	0.46	50.36
		Content-enriched	25.00	0.54	97.52
	Social	Social	0.00	0.17	25.16
		Social-enriched	**54.17**	**0.71**	**99.02**
	Combined	All	12.50	0.46	89.82
		All-enriched	41.67	0.64	97.97

Table 5
Results of the recommendation application using the traversed length on the three datasets.

Method			ACM CCS	DOAJ	InterPro
Random			11.20	10.60	33.54
Similarity-Based (baseline)	Topology	Level	11.82	11.39	35.13
		Siblings	11.82	11.39	35.13
		Children	11.12	9.18	17.35
	Content	Frequency	11.61	11.86	27.98
		Name	10.43	10.16	36.56
		Page	7.95	9.99	29.96
		Jaccard	7.39	9.50	34.67
		NGD	7.54	7.98	34.79
	Social	Coauthor	11.56	10.98	35.19
		Sequence	11.45	10.92	35.29
	Combined	1-Norm	7.75	8.82	29.75
Learning-based	Topology	Topology	10.80	10.29	18.85
	Content	Content	7.63	9.15	31.60
		Content-enriched	6.44	8.62	6.33
	Social	Social	10.22	9.94	34.37
		Social-enriched	8.50	8.60	**4.75**
	Combined	All	7.35	8.72	19.83
		All-enriched	**5.84**	**7.87**	5.79

enrichment outperforms the other methods. Compared to random guess, the saving in time is 5.84/11.20 = 52.14% on the ACM CCS dataset. The result indicates that, with the recommender mechanism, the refinement task can be completed in about half the time. For the DOAJ and InterPro datasets, the time required to complete the same task are reduced to about 3/4 and 1/7, respectively.

We observe that the traversed length is highly dependent on the structure of the concept hierarchy. The structural statistics of training data for the three datasets are shown in Table 6. The training data for ACM CCS and DOAJ is the same, and the structure of the datasets is quite balanced. However, the structure of the InterPro dataset is very unbalanced. Because the

Table 6
Structural statistics of the training data for the three datasets.

Training data	Total nodes	Tree height	Average branching factor	Node in each level				
				Level 1	Level 2	Level 3	Level 4	Level 5
ACM CCS/DOAJ	1101	4	4.97	11	58	274	758	
InterPro	229	5	4.49	3*	74	142	9	1

* Although the total number of level-1-nodes in InterPro is 3, two of them are single nodes without a child. Therefore the degree of the remaining node is 74.

branching factor of the first node is 74, it takes much longer to identify the right position for insertion in the first level. Our experiment results demonstrate that for harder tasks, such as those in biology domain, the proposed recommender mechanism can improve the performance even more significantly (i.e., it only needs 1/7 of the time) than easier tasks, such as those in computer science domain.

5.8. Time complexity

The time complexity of the training phase is $O(f * |K_O|^2 + C_{training})$, where f is the number of features selected, and $C_{training}$ is the time required to train the chosen classifier ($C_{training} = |K_O|$ in NB classifier); $O(f * |K_O|^2)$ denotes the time needed to compile the training dataset. As a result, the training time complexity equals $O(|K_O|^2)$. In our experiments, the packages we select (Weka [32] for an NB classifier) are relatively fast, so training the ACM CCS dataset (1,211,100 tuples) takes no more than an hour.

The time complexity of the insertion phase is $O(|K_N| * (f * |K_O| + C_{testing}))$, where $C_{testing}$ is the time required to obtain the probability scores for the test instances used by the classifier ($C_{testing} = |K_O|$ in NB classifier). As a result, the insertion time complexity equals $O(|K_N| * |K_O|)$. Insertion is much faster than training in general.

6. Discussion

In this section, we discuss our findings for the four hypotheses detailed in Section 4.

(1) *Learning-based methods can improve on the results derived by non-learning based methods.* As shown in Tables 2–4, because learning-based methods use different types of features (topology, content, and social information features, and a combination of them), they generally perform better than non-learning methods. In Table 7, we compare the performance of the methods on the ACM CCS, DOAJ and InterPro datasets.

(2) *Topology, content and social features make diverse contributions when refining a concept hierarchy.* The topology feature yields low accuracy and average taxonomy similarity scores, but it performs fairly well in terms of the rank-based AUC (i.e., its performance is comparable to that of the content feature). This implies that, although the topology information is not sufficient to identify the most appropriate position, it can filter a substantial number of unlikely candidates and thereby improve the ranking of the correct candidate. Content features seem to be more effective than the other features in the ACM CCS and InterPro dataset compared to the DOAJ dataset. We believe this is because the features are more sensitive to the relationship between the training and testing data. In the experiment, we use ACM CCS91 as training data for the ACM CCS and DOAJ datasets. For test data from the same source, such as ACM CCS98, content features achieve better results; however, they are less informative in a different source like DOAJ. We draw the same conclusion from the results for the InterPro dataset. Finally, in the ACM CCS and DOAJ datasets, social features do not perform well individually in rank-based AUC, but they achieve relatively good results in terms of accuracy and average taxonomy similarity. This may be because social features are obtained from external resources like the Web, so it is reasonable to assume that obtaining information for certain concepts is relatively easy. For concepts that were originally ranked close to the top, the additional social information allows us to move them slightly closer to the top position, which improves the accuracy and taxonomy similarity. However, the ranks of concepts that lack social information can be lowered, which impacts the overall rank-based AUC score.

(3) *Enriching features by using information about neighboring nodes can improve the accuracy.* This provides an alternative way to incorporate topological information into the learning process. In particular, enriching features can provide a significant improvement for the InterPro dataset. This may be because the hierarchical structure of InterPro dataset is not well balanced (as mentioned in Section 4.7); therefore, learning the structure and neighbor information becomes important.

Table 7
Comparison of the performance of the similarity-based and learning-based methods on the three datasets using the topology, content, and social information features and combined features.

Method		Accuracy (%)			Average T_{sim}			Rank-based AUC (%)		
		ACM	DOAJ	InterPro	ACM	DOAJ	InterPro	ACM	DOAJ	InterPro
Topology	Similarity	0.01	0.05	12.50	0.17	0.22	0.46	86.74	87.41	90.13
	Learning	0.00	0.00	12.50	0.17	0.18	0.46	88.31	89.15	86.90
Content	Similarity	13.93	6.02	0.00	0.36	0.24	0.44	87.40	83.35	51.20
	Learning	32.34	9.64	25.00	0.49	0.26	0.54	96.57	93.92	97.52
Social	Similarity	7.79	2.41	0.00	0.27	0.21	0.27	28.26	14.69	0.87
	Learning	11.06	16.87	**54.17**	0.34	0.35	**0.71**	47.49	65.05	**99.02**
Combined	Similarity	20.08	8.43	0.00	0.40	0.28	0.38	92.70	85.49	63.35
	Learning	**34.84**	**19.28**	41.67	**0.51**	**0.36**	0.64	**97.28**	**95.87**	97.97

(4) *A learning-based concept hierarchy refinement framework can be used as an effective and efficient recommender mechanism to help users refine a hierarchy.* The results show that, compared to a random-ranked system, our recommender (using a combination of features) can reduce the amount of required time by between 25% and 86%.

Finally, we tried the following operations, but found that they were not feasible:

(1) We tried several post-processing adjustment methods. After computing the probabilities of each possible insertion position, we assigned each node a new probability based on the probabilities of its neighboring nodes. For example, we averaged a node's probability with the probabilities of all its child nodes. The objective is to utilize the information about neighboring nodes. The results show that adding neighbor information in the post-processing stage is less effective than incorporating it during training/testing phase for feature enrichment.
(2) We also tried level-wise training, which makes classification decisions level-by-level. That is, we began by classifying a new concept into one of the first-level nodes, and then classified it into one of the second-level nodes which are children nodes of the selected first-level node. We found that learning each level independently did not improve the performance, probably because dividing the training data into smaller portions led to overfitting.

7. Conclusion

Because of the lack of training samples, supervised methods have rarely been proposed for the ontology refinement problem. The major contribution of this paper is that we explore the idea of decomposing an original hierarchy into a set of training pairs. The decomposed pairs enable us to train a binary classifier that can be exploited in the insertion phase to determine the best position for a new concept. As a result, we do not need to acquire manually annotated training samples. Furthermore, we can explore the topology information with the content and social information features to obtain better results. We consider that the proposed method is domain and language independent because most of the features can be generated from the Web; thus, specific knowledge about the query terms is not required. To evaluate the method, we utilize two existing evaluation metrics and propose a rank-based AUC metric. We also propose a practical application for our system as a recommender mechanism that interacts with users to refine a concept hierarchy. We believe that recommender is useful for tasks that require more resources during evaluation, such as ontology refinement tasks in the molecular biology domain.

References

[1] ACM Computing Classification Scheme (CCS). http://www.acm.org/about/class (last accessed at 13.10.10).
[2] T.K. Attwood, The PRINTS database: a resource for identification of protein families, Briefings in Bioinformatics 3 (3) (2002) 252–263.
[3] R. Belohlavek, E. Sigmund, J. Zacpal, Evaluation of IPAQ questionnaires supported by formal concept analysis, Information Sciences 181 (10) (2011) 1774–1786.
[4] K. P. Chitrapura, R. Krishnapuram, S. Joshi, Method and apparatus for populating a predefined concept hierarchy or other hierarchical set of classified data items by minimizing system entrophy, US Patent, United States, 2008.
[5] R.L. Cilibrasi, P.M.B. Vitányi, The Google similarity distance, IEEE Transactions on Knowledge and Data Engineering (TKDE) 19 (3) (2007) 370–383.
[6] P. Cimiano, G.U. Ladwig, S. Staab, Gimme' The context: context-driven automatic semantic annotation with C-PANKOW, in: Proceedings of the 14th International World Wide Web Conference (WWW), 2005, pp. 332–341.
[7] P. Cimiano, S. Staab, Learning concept hierarchies from text with a guided hierarchical clustering algorithm, in: Proceedings of the Workshop on Learning and Extending Lexical Ontologies with Machine Learning Methods at the 22nd International Conference on Machine Learning (ICML), 2005.
[8] A. Cornish-Bowden, M. L. Cárdenas, Control of metabolic processes. NATO ASI Series A, vol. 191, New York, 1990.
[9] Directory of Open Access Journals (DOAJ), http://www.doaj.org (last accessed at 13.10.10).
[10] S. Dumais, H. Chen, Hierarchical classification of web content, in: Proceedings of the 23rd Annual International Conference on Research and Development in Information Retrieval (SIGIR), 2000, pp. 256–263.
[11] D.H. Fisher, Knowledge acquisition via incremental conceptual clustering, Machine Learning (1987) 139–172.
[12] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, G. Antoniou, Ontology change: classification and survey, Knowledge Engineering Review 23 (2) (2008) 117–152.
[13] B. Ganter, R. Wille, Formal Concept Analysis, Springer, Berlin, 1999.
[14] U. Hahn, K. Schnattinger, Towards text knowledge engineering, in: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI), 1998, pp. 524–531.
[15] J. Han, Y. Fu, Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases, in: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI), 1994, pp. 157–168.
[16] S. Huh, J. Lee, Providing approximate answers using a knowledge abstraction database, Journal of Database Management 12 (2) (2001) 14–24.
[17] S. Hunter, R. Apweiler, T.K. Attwood, A. Bairoch, A. Bateman, D. Binns, P. Bork, U. Das, L. Daugherty, L. Duquenne, R.D. Finn, J. Gough, D. Haft, N. Hulo, D. Kahn, E. Kelly, A. Laugraud, I. Letunic, D. Lonsdale, R. Lopez, M. Madera, J. Maslen, C. McAnulla, J. McDowall, J. Mistry, A. Mitchell, N. Mulder, D. Natale, C. Orengo, A.F. Quinn, J.D. Selengut, C.J.A. Sigrist, M. Thimma, P.D. Thomas, F. Valentin, D. Wilson, C.H. Wu, C. Yeats, InterPro: the integrative protein signature database, Nucleic Acids Research 37 (Database Issue) (2009) D211–D215.
[18] Mehdi Kaytoue, Sergei O. Kuznetsov, Amedeo Napoli, Sébastien Duplessis, Mining gene expression data with pattern structures in formal concept analysis. Information Sciences 181 (10) (2011) 1989–2001.
[19] H.-C. Kuo, J.-P. Huang, Building a concept hierarchy from a distance matrix, in: Proceedings of the 14th International Conference on Intelligent Information Systems, 2005, pp. 87–95.
[20] H.-C. Kuo, H.-C. Lai, J.-P. Huang, Building a concept hierarchy automatically and its measuring, in: Proceedings of the 7th International Conference on Machine Learning and Cybernetics (ICMLC), 2008, pp. 3975–3978.
[21] H.-C. Kuo, T.-H. Tsai, J.-P. Huang, Building a concept hierarchy by hierarchical clustering with join/merge decision, in: Proceedings of the 9th Joint Conference on Information Sciences (JCIS), 2006.
[22] S.C.J. Lam, D. Sleeman, W. Vasconselos, ReTAX+: a cooperative taxonomy revision tool, in: Proceedings of the 24th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI-2004), 2004, pp. 64–77.
[23] A. Maedche, V. Pekar, S. Staab, Ontology Learning Part One – On Discovering Taxonomic Relations from the Web, Springer, 2002.

[24] D. Merkl, A. Rauber, Uncovering the hierarchical structure of text archives by using an unsupervised neural network with adaptive architecture, in: Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2000, pp. 384–395.
[25] M. Sanderson, B. Croft, Deriving concept hierarchies from text, in: Proceedings of the 22nd Annual International Conference on Research and Development in Information Retrieval (SIGIR), 1999, pp. 206–213.
[26] M.A. Savageau, Biochemical Systems Analysis: A Study of Function and Design in Molecular Biology, Addison Wesley, Reading, Mass, 1976.
[27] D. Sleeman, Q. Reul, CleanONTO: evaluating taxonomic relationships in ontologies, in: Proceedings of the 4th International Evaluation of Ontologies (EON) Workshop on Evaluation of Ontologies for the Web, 2006.
[28] The Gene Ontology Consortium, Gene ontology: tool for the unification of biology, Nature Genetics 25(1) (2000) 25–29.
[29] Francisco J. Valverde-Albacete, Carmen Peláez-Moreno, Extending conceptualisation modes for generalised formal concept analysis, Information Sciences (2010).
[30] E.O. Voit, Computational Analysis of Biochemical Systems: A Practical Guide for Biochemists and Molecular Biologists, Cambridge University Press, Cambridge, UK, 2000.
[31] Lidong Wang, Xiaodong Liu, Jiannong Cao, A new algebraic structure for formal concept analysis, Information Sciences 180 (24) (2010) 4865–4876.
[32] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann, 2005.