A Hybrid Approach for Tag Hierarchy Construction

Chapter · January 2018
DOI: 10.1007/978-3-319-90421-4_4

5 authors, including:

Shangwen Wang
National University of Defense Technology
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Tao Wang
National University of Defense Technology
47 PUBLICATIONS **151** CITATIONS
SEE PROFILE

Xiaoguang Mao
National University of Defense Technology
68 PUBLICATIONS **293** CITATIONS
SEE PROFILE

Yue Yu
National University of Defense Technology
29 PUBLICATIONS **260** CITATIONS
SEE PROFILE

A Hybrid Approach for Tag Hierarchy Construction

Shangwen Wang, Tao Wang, Xiaoguang Mao, Gang Yin, and Yue Yu

National University of Defense Technology, Changsha, China

shang_wen_wang@163.com
{taowang2005,xgmao,yingang,yuyue}@nudt.edu.cn

Abstract. Open source resources are playing a more and more important role in software engineering for reuse. However, the dramatically increasing scale of these resources brings great challenges for their management and location. In this study, we propose a hybrid approach for automatic tag hierarchy construction, which combines the tag co-occurrence relations and domain knowledge to build and optimize the hierarchy. We firstly calculate the generality of each tag in accordance with the co-occurrence relationship with others, and construct the hierarchy based on the generality. Then we leverage the domain knowledge of existing hierarchical categories to perform an optimization and promote the final hierarchy. We select 8064 projects in *Openhub* community and 10703 posts in *StackOverflow* community as the original data and use the information of the *SourceForge* community as the domain knowledge. We conduct extensive experiments and evaluate our approach by utilizing *Wordnet* and *F-measure* method. The results show that our approach exhibits better performance than others with accuracy rate and recall that exceed 90%.

Keywords: Open Source Community; Tag Hierarchy Construction; Domain Knowledge; Optimization

1 Introduction

Open source software is a computer program of freely available source code and has been favored by the majority of developers since its release. The rise in the number of open source software brings together the creativity and wisdom of the entire society to promote software updates and bug fixes, thereby providing much attention to software development and increasing the types of producers participating in the development. Different types of producers come together to form two different open source communities: collaborative development community and knowledge sharing community. Software reuse technology [1], which utilizes others' code to perfect one's own program, is developed by this open source movement.

However, the increasing number of open source software has brought difficulty for traditional methods in organizing and managing large amounts of software resources. Organizing and locating open source software effectively and improving the efficiency of software reuse have become major challenges. Begelman et al. [2] proposed a

method to automate the construction of taxonomy by using a tagging mechanism. Since then, this approach has been extensively studied since a reasonable and comprehensive tag hierarchy can organize open source resources well and provide convenience for software reuse. In 2012, Wang et al. [3] inferred term taxonomy by leveraging collaborative tagging. At the same time, an approach that can automatically derive a domain-dependent taxonomy from a set of keyword phrases was proposed by Liu [4].

This study presents a method for automatically constructing a tag hierarchy and optimizing it with domain knowledge. First, we calculate the generality of each tag on the basis of the co-occurrence relationships between tags. We then construct hierarchy in accordance with the generalities by selecting the most suitable father node for each tag except the one with largest generality. For each tag *p,* its father node *q* must satisfy two conditions: 1) *q* possesses a larger generality than *p*; 2) among the tags of generality larger than *p*, *q* possesses the largest degree of correlation with *p*.

Domain knowledge refers to some existing human-created hierarchies. In the last step, we take the construction of the tag hierarchy into the domain knowledge for testing. The proposed algorithm uses the "reverse" idea. In particular, the hierarchy construction of manual construction is compared with the result of previously automated construction, the incorrect side of the automatic construction result is corrected, and the relationship that exists in the domain knowledge but does not exist in the automated hierarchy construction is added to the results. After these steps, we finally obtain our results.

Our experimental dataset is taken from one open source community, Openhub, and a programming knowledge sharing community, Stackoverflow, to avoid the error caused by a single data source. After calculating tag generalities, we choose 342 tags to build hierarchy, and all the tag generalities are greater than 1,000. SourceForge open source community hosts more than 600,000 projects and has established a software taxonomy hierarchy by hand. Thus, its classification is plausible. In our experiment, we use the taxonomy hierarchy in SourceForge as the domain knowledge for optimization. Experimental results show that our method can construct tag hierarchy efficiently and accurately, and the optimized hierarchical structure accuracy can reach more than 90%, which surpasses that of the previous methods. The main contributions of this study are described as follows:

- We take advantage of the co-occurrence relationship between tags, calculate the software tag generalities by digging the inherent relationship between them, and build tag hierarchy in an unsupervised mode.
- We propose an optimization algorithm based on domain knowledge to optimize the results of automated building by using the manually established taxonomic hierarchies in some communities. We obtain improved taxonomy results by merging the information from two different hierarchies.
- We conduct extensive experiments. We select 342 tags at the top of the generality ranking, all of which possess a generality greater than 1,000. We conduct various types of experiments to prove the superiority of our approach.

The rest of the paper is organized as follows. Section 2 introduces some works related to tag hierarchy construction. We describe our method in detail in Section 3. We

introduce the design of our experiments in Section 4 and provide results and discussions in Section 5. Section 6 elaborates the conclusions and comes up with a plan for our future work.

2 Related Work

Taxonomies constructed with general tags have been widely investigated. A key step in the existing methods of taxonomy construction is to calculate the generality for each tag. This step can be achieved by two types of technology as follows.

One is to use set theory techniques. Among these works, each resource is considered a distinct data item with their textual contents ignored, and each tag presents the collection of items it annotates. For example, Heymann et al. [5] came up with a simple but effective way to learn a tag taxonomy. Heymann modeled each tag as m-d vector with m documents it annotates and used the cosine similarities between tag vectors to generate the tag similarity graph. He then demonstrated that the social network notion of graph centrality seems to be a valid way to calculate generality. Sanderson and Croft [6] compared the size of image collection in which two tags occur to determine the affiliation relation between them. Schmitz [7] developed Heymann's model to control highly idiosyncratic vocabulary frequency limits to improve the quality of the results. Liu et al. [8] used association rule mining. This method takes each tagged resource as a transaction and tags as items. The method is governed by the following rule: "for a specific unknown resource X, if tag A appears, then tag B will probably appear," that is, "tag B subsumes tag A." The natural possibility of inclusion is naturally modeled on the confidence and support of the corresponding rules. On the basis of the inclusion probabilities between each pair of tags, they calculated the overall general rating of each tag by using a random walk. Finally, they built taxonomies in a top-down fashion. The two methods exhibit a common flaw, that is, they only distinguish one resource from another but do not exploit tagged web documents.

Another way is based on *LDA (Latent Dirichlet Allocation)* model [9]. Tang et al. [10] designed a tag–topic model based on this classic model. They assumed each tag possesses multiple submeanings, which are also called topics. Tags with similar high distributions on multiple topics indicate a high probability that the tag is a normal tag, whereas a tag with a high distribution on only one specific topic indicates that the tag may possess a specific meaning. Wang [11] suggested that a document annotated by the same tag can be considered the interpretation of this tag. He said we can combine these documents into a new document, learn the subject distribution of standard LDA models from the basic corpus, and measure the generality on the basis of "surprise" theory [12] plus an intuitive law that, "given an anticipated tag A, the appearance of a document on a more general tag B will cause less 'surprise' than if A and B are switched." This theory is a slight modification of LDA.

Hierarchy construction can be done in several ways. Liu et al. [4] argued that building taxonomy on the basis of keywords is difficult. They obtained knowledge using short context conceptualization and a general-purpose knowledge base called Probase to distinguish the relation between tags. They retrieved the excerpts by sub-

mitting the query to a commercial search engine and then calculated the generality by combining knowledge and context to obtain the context. Wang et al. [3] measured similarity on the basis of open source community labeling system by combining document collection and text similarity. Brooks and Montanez [17] avoided computing tag generality. They adopted a method that only relies on the similarity or distance between two tags in building a hierarchy. However, their obtained hierarchy lacks supertype–subtype relationships. Li et al. [13] referred to this model, proposed an approach based on agglomerative hierarchical clustering by skipping the error prone step of calculating each tag generality, and called this model *AHCTC (Agglomerative Hierarchical Clustering for Taxonomy Construction)*.

Gu et al. [14] suggested utilizing domain knowledge for optimization. In this method, they introduced the construction into domain knowledge for testing. This work is similar to ours but presents two weak points. First, the above-mentioned authors did not add the tags contained in the domain knowledge but not contained in the construction to be optimized to the final results. Second, they considered the tags contained in the construction to be optimized but not contained in the domain knowledge as incorrect relations and deleted them, thereby possibly affecting the diversity of tags. Fahad et al. [19] suggested another idea for optimization. They ordered tags in descending order by generality and then added them into the hierarchy by choosing the tag of the most co-occurrence frequency and the tag already in the hierarchy to be its father node. They then checked the correctness of the direction of this edge. This method is limited because it simply checks the edge relation rather than finding another suitable father node.

Construction of taxonomy generally has two types: tree and *DAG (Directed Acyclic Graph)*. The author in [15] proposed a tree-based label hierarchy research method, used the Jaccard coefficient to measure the label similarity, and proposed the label hierarchy algorithm of the maximum spanning tree. Marszalek et al. [16] found through observation that finding a suitable segmentation of feature space increasingly becomes difficult with the increase in the number of categories. Therefore, they proposed the idea that the unspecified classification can be extended to classification when the classification boundaries are clear. Accordingly, assigning of each son node can be postponed, thereby resulting in a DAG chart. Finally, a relaxed classification level can be obtained.

3 Hybrid Hierarchy Construction Method

In this section, we introduce our approach from two aspects. We provide an overview of the framework first and then describe each part in detail.

Our goal is to build a reasonable hierarchy for tags which means we need to consider the relationship between tags and the scope of tag to be used. We define two terms as follows:

Definition 1. **Co-occurrence frequency:** For a tag pair (p, q), its co-occurrence frequency is the sum of times it occurs in the tag list of a project.

Definition 2. **Tag generality:** Tag generality is an indicator to measure the limited scope of the tag to be used. Tags of large generality generally possess a large scope to be used.

3.1 Overview of Our Approach

Co-occurrence frequency shows the relevance of two tags and generality reflects the degree of acceptance of the tag. These two characteristics have good use of value for our goal and thus are utilized. Some errors may be contained in the construction for a defect may exist in our approach. We choose to optimize the construction with the guidance from domain knowledge in that it possesses high reliability.

Our approach to build tag hierarchy consists of two parts, namely, unsupervised hierarchy building and hierarchy optimization. The framework of our method is shown in Figure 1.

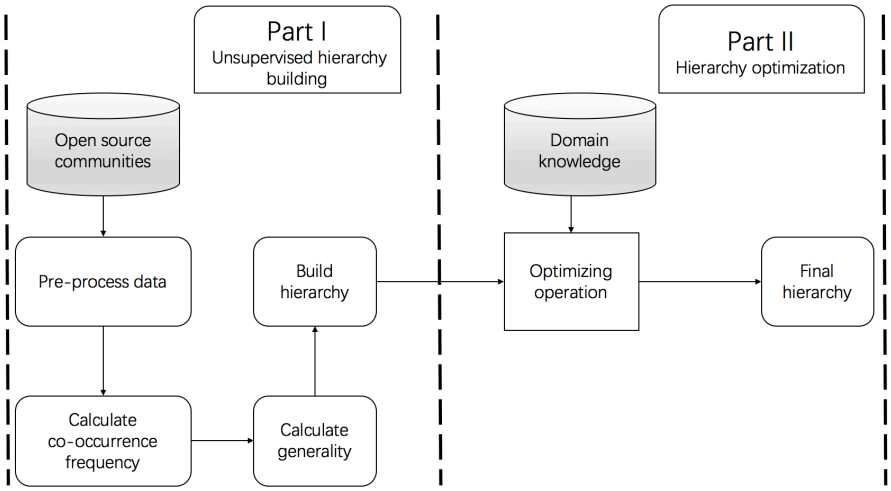

Figure 1 Framework of our method

The figure shows our approach is a two-phase hybrid model. It combines unsupervised hierarchy building through tag co-occurrence relation information with optimization utilizing domain knowledge. These two main phases are as follows:

Unsupervised hierarchy building: The first phase involves four steps, most of which are related to mathematical calculation. First, we extract data from several open source communities and pre-process the original data. Second, we calculate co-occurrence frequency for tag pairs. Third, we calculate generality for each tag. Fourth, we can build this hierarchy using the data calculated from the previous steps. All these steps except the first one are automatically completed.

Hierarchy optimization: In the second phase, we introduce domain knowledge. We combine it with the hierarchy we have built into an algorithm for optimization. The algorithm checks whether the construction possesses incorrect edges by comparing with domain knowledge and fixing the errors if any exists.

3.2 Unsupervised Hierarchy Building

According to [13], one tag can only be connected with the tag that is most relevant to it, and tags with large generalities should be at the upper levels in an ideal hierarchy. Thus, we should calculate co-occurrence frequency for tag pairs and generality for each tag before building. We design our approach for unsupervised hierarchy building on the basis of this idea. Our approach involves the following steps:

Pre-process data: The information we need to construct the hierarchy are the project number and the tags it contains. We obtain a great deal of unpractical information in the detail page from the communities. We pre-process them in the database and retain only two columns, namely, project id and tags, to minimize the size of the data space and ensure efficient data query.

Calculate co-occurrence frequency: In a collaborative development community, each project is tagged with multiple tags that are related to its content as a basis of recommendation to developers. Similarly, in a knowledge sharing community, each post is associated with multiple tags. Co-occurrence between two tags indicates that both tags appear in the tag list of the same project. We define a helper function P(t_1, t_2) to judge whether the two tags, namely, t_1 and t_2, are in the tag list of project P.

$$\mathrm{P}(t_1, t_2) = \begin{cases} 1 & if\ t_1\ and\ t_2\ are\ both\ tags\ of\ P \\ 0 & if\ t_1\ or\ t_2\ is\ not\ the\ tag\ of\ P \end{cases} \quad (1)$$

In Equation (1), t_1 and t_2 denote two specific tags. We calculate the co-occurrence frequency of two tags by Equation (2) as follows:

$$\mathrm{Co}(t_i, t_j) = \sum_{x=1}^{n} P_x(t_i, t_j). \quad (2)$$

In Equation (2), t_i and t_j denote the tags we are calculating. P denotes a project in the project set. We obtain the co-occurrence frequency of each of the two tags by applying the equation to every tag pair.

Calculate generality: The generality of a tag is not only related to the number of tag it co-occurrences. The co-occurrence frequency of this tag to the other tags is important as well. We define Equation (3) to calculate tag generality as follows:

$$\mathrm{G(t)} = \sum_{i=1}^{n} Co(t, t_i). \quad (3)$$

In Equation (3), t denotes the tag we are calculating, and t_i denotes another tag in the tag set. We calculate tag generality through this circulation.

Build hierarchy: The generality of tag reflects the commonality of this tag. On this basis, we consider that a tag with a large generality should be located at a high place in the hierarchy construction. We introduce our method to build tag hierarchy in Algorithm 1.

We construct the hierarchy in accordance with the generalities by selecting a most suitable father node for each tag except the one with largest generality. For each tag p, its father node q must satisfy two conditions: 1) q possesses a larger generality than p; 2) among the tags of generality larger than p, q possesses the largest degree of correlation with p.

Algorithm 1 Tag Hierarchy Construction

Input: T: Set of tags
G: Set of generality for each tag
Co-occurrence: Set of co-occurrence frequency for each two tags
Output: R: Tag hierarchy construction

```
1: for tag t in T do
2:     set tmp ← 0
3:     for tag e in T-{t} do
4:         if G(e) > G(t) then
5:             if Co-occurrence (e, t) > tmp then
6:                 update tmp ← Co-occurrence(e, t)
7:                 record this key-value (tmp, e)
8:             end if
9:         end if
10:    end for
11:    get tag f which is correspond to current tmp
12:    add edge relation f → t to R
13: end for
```

We first traverse the tags on line 1. Line 2 defines a variable *tmp* to record the largest co-occurrence frequency between t and one of the other tags. We then traverse the other tags and compare the generality of the selected tag e with t on line 4. If its generality is larger than that of t, we then check if the co-occurrence frequency between e and t is larger than *tmp*. If this condition is satisfied, we then regard e as a candidate father node for tag t. We modify the value of *tmp* and record this candidate tag similar to lines 6–7. When the inner loop is finished, we obtain tag f that corresponds to current *tmp* in accordance with our records in line 11. For the tag pair (f, t), we add direction from f to t. Finally, we obtain the full hierarchy after the outer loop is completed.

3.3 Hierarchy Optimization

Our construction algorithm presents the following defect. For two tags of great relevance, the one with large generality can be the father node of the other during our building procedure, but they should be on the same level in terms of their semantics (e.g., dvd → cd). As a result, fixing incorrect edges is important. Domain knowledge refers to some existing hierarchy, its artificially built character gives it high reliability. We can use it to trim some unreasonable edges and add some reasonable edges. We obtain these data from open source community and utilize them for optimization.

We design an optimization algorithm based on the "reverse" thinking, that is, the domain knowledge is introduced into the hierarchy construction to be optimized for testing. In accordance with different test results, we conduct different operations on the construction to achieve optimization.

For each tag pair in the domain knowledge, we consider the following conditions:

Algorithm 2 Optimizing Tag Hierarchy

Input: Tagpair1: tag pairs in the domain knowledge
Tagpair2: tag pairs in the construction to be optimized
Tags: all tags that are in the construction to be optimized
Output: R: hierarchy construction after optimizing

```
1: Initialize set Changed ← Φ
2: for tag pair (p, q) in Tagpair1 do
3:     if p not in Tags ∨ q not in Tags then
4:         add edge relation p → q to R
5:     else
6:         if p and q satisfy function isfather (p, q) then
7:             continue
8:         else
9:             add edge relation p → q to R
10:            add q to Changed
11:        end if
12:    end if
13: end for
14: for tag pair (m, n) in Tagpair2 do
15:    if n ∈ Changed then
16:        continue
17:    else
18:        add edge relation m → n to R
19:    end if
20: end for
```

1) If the domain knowledge contains tags that do not appear in the construction to be optimized, we then add this relation to the result directly.
2) If both tags are already in the construction, we then place this tag pair into the construction for checking.

We use function *isfather* to conduct this check. Its parameters are two tags, namely, p and q, and it returns a Boolean value. It uses a recursive method to search if a structure such as p → … → q exists in the construction to be optimized. There are two conditions when the returned value is true: One is tag pair (p, q) is included in the construction to be optimized. Another is the construction to be optimized contains structure like: p → … → t → … → q. The results are described in two conditions:

1) If the result returns true, then this relation is correct. We do nothing under this condition.
2) If the result returns false, then the construction possesses a fault. Thus, tag q must be optimized. We change q's father node to p and record q into a set named *Changed*. In other words, this tag has changed its father node.

Finally, we traverse the construction to be optimized. For each child node in the tag pairs, we check if it has been recorded.

1) If it has not been recorded, then this tap pair possesses a right relation. We then place this relation into the final results.
2) If it has been recorded, then this tag's father node has been changed. Thus, we do not need to do any operation here.

The pseudo-codes of our algorithm are shown in Algorithm 2.

4 Experiment Design

In this section, we describe the research questions, experiment setting, and evaluation metrics in detail.

4.1 Research Questions

In order to check the performance of our optimization algorithm as well as to make a comprehensive evaluation for our approach, we focus on the following research questions:

- **RQ1:** Does domain knowledge promote the tag hierarchy?
- **RQ2:** Does our approach work more accurately than others'?

For RQ1, we compare the hierarchies before and after optimization to observe the differences and make evaluation. For RQ2, we reproduce others' work and make comparison with ours to evaluate the performance.

4.2 Experimental Setting

Data and storage: We collect 8,064 projects from Openhub and 10,703 posts from StackOverflow. The sum of the tags they include is over 40,000. After pre-processing, we only retain two columns of information, namely, id and tags.

Procedure: We use the information previously extracted from open source communities to conduct our experiment. We calculate co-occurrence frequency for tag pairs first and then calculate tag generality for each tag. Next, we select 342 tags of generalities larger than 1,000 to build hierarchy and thus constrain the time consumption of our experiment. We select out some tags that are commonly used in the open source communities and display their generality values in Table 1.

Table 1 Some Important Tags and Their Generality Values

Tags	Generality value
html	9,469
javascript	26,059
mysql	17,876
python	42,914
linux	35,700

SourceForge open source community hosts more than 600,000 projects and has established a software taxonomy hierarchy by hand. Thus, its taxonomy presents a high reliability. In our experiment, we choose hierarchy information from SourceForge community as our domain knowledge for optimization. We obtain the final results by conducting optimization operations.

Experimental environment: All our experiments are conducted under Window7 operation system, with Eclipse programming environment and mysql database for data storage.

4.3 Evaluation Metrics

For the proposed research questions, we use two methodologies for evaluation.

One is to use the WordNet [18] tool. This tool has been widely considered as a gold standard for testing hyponym/hypernym relations between tags. We can find synonyms, vocabularies in the sub-categories, and vocabularies in the higher level for a specific tag. We use this tool to check each edge relation in our hierarchy and record the total number of edges in the hierarchy(t), the number of edges found in WordNet(f), and the number of correct edges checked by WordNet(c). The two evaluation factors, *Edge coverage* and *Agreement with WordNet*, equal to f divided by t and c divided by f, respectively.

Another is to apply F-measure, which was introduced by Mario et al. [20], to compare accuracy and recall. The formulas are as follows:

$$PRC_i = \frac{TP_i}{TP_i + FP_i}\ ;\ REC_i = \frac{TP_i}{TP_i + FN_i}\ ;\ F_i = \frac{2 * PRC_i * REC_i}{PRC_i + REC_i} \qquad (4)$$

For a specific subtree, we manually check each edge relation by comparing with WordNet and record terms should have belonged to this subtree but not by searching synonyms and vocabularies in the sub-categories for tags in this subtree traversely. We calculate three parameters: TP_i is the number of true positives (edge relations that are correct), FP_i is the number of false positives (edge relations that are incorrect, for example, *ftp* is a false positive in http → ftp), and FN_i is the number of false negatives (edge relations that do not belong to this tree but should have belonged, for example, *dns* is a false negative if it is not belong to *web*). Implementing this method to all the subtrees in hierarchy construction will result in Formula (5) as follows:

$$\text{PRC} = \frac{\sum_i TP_i}{\sum_i TP_i + FP_i}\ ;\ REC = \frac{\sum_i TP_i}{\sum_i TP_i + FN_i}\ ;\ F = \frac{2 * PRC * REC}{PRC + REC} \qquad (5)$$

where PRC represents the average precision; REC represents recall; and F, which is a harmonic mean of PRC and REC, provides a way to combine precision and recall in a unique metric. We use this theory to calculate the values of F-measure for each hierarchy to check its performance.

5 Results and Discussions

In this section, we present the experimental results for the research questions we have proposed. We also use a case study to show the superiority of our approach and discuss the threats to validity in detail.

5.1 RQ1: Does domain knowledge promote the tag hierarchy?

In this section, we conduct qualitative and quantitative analyses for hierarchies before and after optimization to check whether domain knowledge promotes the tag hierarchy from content and evaluation metrics.

Hierarchies before and after optimization have 317 and 429 edges respectively, indicating that the content in the hierarchy after optimization is much richer than before. Some errors are corrected and some new tags are added. We choose the subtree of *web* as a case to evaluate this question in detail. Subtrees of *web* before and after optimization are shown in Figures 2 and 3, respectively.

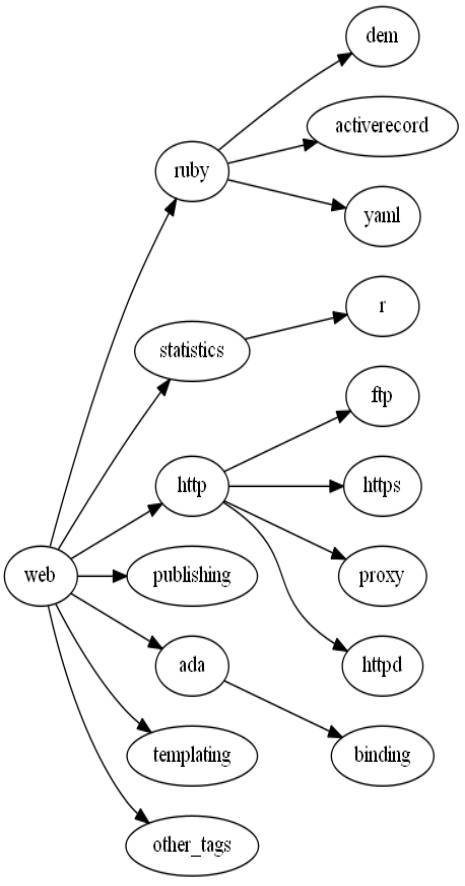

Figure 2 Subtree of *web* before optimization

Figure 3 Subtree of *web* after optimization

Qualitative analysis: The subtree of *web* before optimization possesses four incorrect relation edges, namely, http → ftp, web → statistics, web → publishing (we consider that the father node of *publishing* should be *tomcat*), and web → templating (we consider that the father node of *templating* should be a programming language, such as *jquery*). *Web* is a tag about some terms and technologies about net. Four nodes should appear in this subtree: *html*, *xhtml*, *ssh*, and *dns*.

During the optimization, we modify the incorrect edge http → ftp, add tags *dns* and *ssh* into the hierarchy construction, and enlarge the number of the nodes in the tree, which can be construed as enriching the contents of the tree.

Quantitative analysis: We check each edge in the constructions with the help of Wordnet, and the results are illustrated in Table 2.

Table 2 Edges evaluation against WordNet

	Number of edges found in WordNet	Edge coverage (%)	Agreement with WordNet (%)
Before optimization	317	92.96	82.20
After optimization	429	95.76	91.93

The results show that we obtain a large number of edges in the hierarchy after optimization, indicating that optimization enriches the content of hierarchy. The percentages of edge cover and agreement with WordNet are increased. We then calculate the values of F-measure for the two hierarchies, and the results are illustrated in Table 3.

Table 3 Values of F-measure for hierarchies before and after optimization

	PRC	REC	F-measure
Before optimization	87.10%	79.41%	83.08%
After optimization	95.90%	98.08%	96.98%

The results show that accuracy and recall rate significantly differ before and after optimization. Our optimization improves the accuracy and recall rate for domain knowledge brings new information to this hierarchy, thereby increasing the value of F-measure to a large extent.

The above-mentioned analysis shows that the optimization based on domain knowledge exerts a satisfactory effect. Domain knowledge not only enriches the content of the tag hierarchy but also improves the accuracy. It promotes the tag hierarchy to a large extent.

5.2 RQ2: Does our approach work more accurately than others'?

In this section, we choose Fahad's work [19] and Gu's work [14] for comparison with ours in that both of them have an optimization step as we have mentioned in Section 2. Using our dataset, we reproduce their works. For Fahad's method, we set

the values of *occurrence* to 1,000, *generality* to 2,000, and *min_sim* to 10. For Gu's method, we select tags of generality greater than 5,000 to build construction and use information from SourceForge to optimize. The subtrees containing tag "xml" built by their methods are shown in Figure 4.

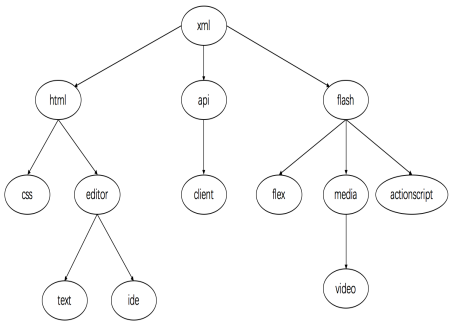

Figure 4a "xml" in Fahad's

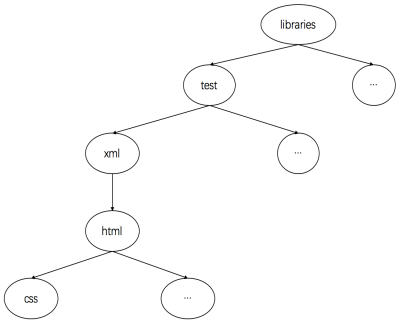

Figure 4b "xml" in Gu's

We check each edge in the hierarchies by Wordnet for comparison with ours. The results are listed in Table 4.

Table 4 Edge comparison results against Wordnet

	Number of edges found in WordNet	Edge coverage (%)	Agreement with WordNet (%)
Fahad's	55	93.22	83.64
Gu's	40	86.96	90.00
Ours	429	95.76	91.93

The results show that our hierarchy possesses far more information than others. Our hierarchy possesses more than 400 edges found in Wordnet whereas others possess around 50. In addition, our edge coverage and agreement with Wordnet are both the highest. We calculate F-measure values for the three hierarchies to further verify our conclusion, and the results are shown in Table 5.

Table 5 Results of applying F-measure to three methods

	PRC	REC	F-measure
Fahad's method	79.60%	71.43%	74.06%
Gu's method	83.33%	66.67%	74.07%
Ours	95.90%	98.08%	96.98%

The results prove the superiority of our approach with accuracy and recall rate reaching approximately 95% whereas others reach only around 80%. As mentioned in Section 2, Gu's optimization reduces many tags from the original construction, and its recall is low as a result. Fahad's optimization plays a little role in finding a suitable parent node for tags. Thus, this method presents a low accuracy rate. Our method presents no apparent flaw, and our F-measure is thus far larger than that of others.

This experiment proves the validity and rationality of our method. Widely used tool and theoretical calculation show that our approach exhibits better performance than others in terms of accuracy.

5.3 Case Study

In this section, we provide a case study to show the superiority of our approach. *Multimedia* is widely used in tag hierarchy as a classification symbol and subtree of *multimedia* in our approach is shown in Figure 5.

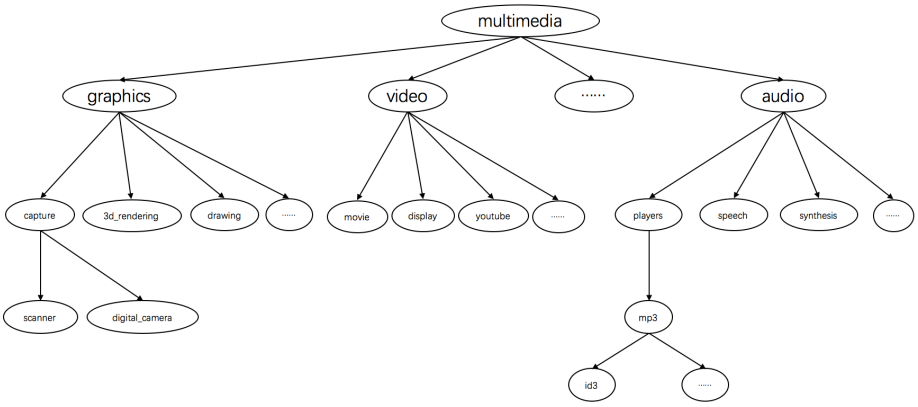

Figure 5 A case study: subtree of *multimedia*

The subtree contains 52 edges in total, which is a large number, and the paths in this hierarchy contain 5 layers at most. *Graphics*, *video*, and *audio* are all commonly used sub-category under *multimedia*, proving that our classification is reasonable. Terminologies widely used under *graphics* (e.g., *capture*, *3d_rendering*, and *drawing*), *video* (e.g., *movie*, *display*, and *youtube*), and *audio* (e.g., *players*, *speech*, and *synthesis*) are all in this construction, indicating that our approach provides a high recall rate with abundant content of tags.

Some errors are noted in this construction (e.g., dvd → cd, mp3 → mp4). These errors may be caused by the defect of our algorithm. For two tags of great relevance, the one with large generality can be the father node of the other during our building procedure, but they should be on the same level in terms of their semantics. We check our domain knowledge dataset and find that those tags in the incorrect edge are excluded in it. Accordingly, we cannot fix them during optimization.

We reach the conclusion that our approach can build a reasonable and comprehensive hierarchy. We can fix all the errors if we obtain a domain knowledge containing sufficient information.

5.4 Threats to Validity

Some threats to the validity of our experiment may affect the results. First, we only choose tags of generality larger than 1,000 to build hierarchy to limit the time consumption of our experiment. As a result, some key tags may be dropped, thereby in-

fluencing our result. Second, we assume domain knowledge is absolutely right in the optimization procedure. Although domain knowledge is artificially established, it may still possess some defects. Thus, the process of enriching the contents of our hierarchy may introduce some errors.

6 Conclusion and Future Work

The increasing amount of information in the open source community has introduced the need for effective information management. In this study, we put forward a hybrid approach to build tag hierarchy. First, we extract data from StackOverflow and the Openhub community, define concepts and computational methods for tag co-occurrence frequency and tag generalization, and propose an unsupervised hierarchical building algorithm based on tag generalization. Next, we design a new set of reverse complementation optimization algorithms that takes the existing taxonomic levels in SourceForge as domain knowledge into the established hierarchy and tests it to arrive at an optimized, accurate hierarchy. We conduct extensive experiments. We compare the hierarchy constructions before and after optimization to show that our optimization exerts great effect. We also compare our work with others to verify its superior performance by utilizing the Wordnet tool and F-measures method.

However, some limitations exist. First, we do not have operations for synonyms. In the subtree of *multimedia*, *movie* and *movies* are son nodes of *video*, but they have the same meaning in semantic. We can merge the two tags into one. The study for plurals and stems has been going on for a long time and several achievements have been realized. If we introduce some relative methods into our approach, we will obtain an accurate result. Second, we simply compare our approach with two state-of-the-art methods to verify the superior performance of our approach. There are many other ways to resolve this problem as this topic has been studied for a long time. In the future, we plan to make a comprehensive comparison with other methods to further verify the performance of our approach.

Acknowledgement. Our approach is publicly-available to support further research on tag hierarchy construction and provide convenience for others to reproduce our experiment: https://github.com/Kaka727/Tag_Hierarchy_Construction_WSW.

Reference

1. Tao W, Huaimin W, Gang Y I N, et al. Hierarchical Categorization of Open Source Software by Online Profiles[J]. IEICE TRANSACTIONS on Information and Systems, 2014, 97(9): 2386-2397.
2. Begelman, G.; Keller, P. & Smadja, F. (2006), Automated Tag Clustering: Improving search and exploration in the tag space, in 'Collaborative Web Tagging Workshop at WWW2006, Edinburgh, Scotland'.
3. Shaowei Wang, David Lo, and Lingxiao Jiang. 2012. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In Proceedings of the

2012 IEEE International Conference on Software Maintenance (ICSM) (ICSM '12). IEEE Computer Society, Washington, DC, USA, 604-607.
4. Xueqing Liu, Yangqiu Song, Shixia Liu, and Haixun Wang. Automatic taxonomy construction from keywords. In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'12). ACM, 2012: 1433-1441.
5. Heymann, P., Garcia-Molina, H.: Collaborative creation of communal hierarchical taxonomies in social tagging systems. Technical report, Computer Science Department, Standford University (April 2006).
6. Sanderson M, Croft B. Deriving concept hierarchies from text[C]//Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 1999: 206-213.
7. Schmitz P. Inducing ontology from flickr tags[C]//Collaborative Web Tagging Workshop at WWW2006, Edinburgh, Scotland. 2006, 50.
8. Liu, K., Fang, B., Zhang, W.: Ontology emergence from folksonomies. In Huang, J., Koudas, N., Jones, G.J.F., Wu, X., Collins-Thompson, K., An, A., eds.: CIKM, ACM (2010) 1109–1118.
9. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. Journal of Machine Learning Research 3 (2003) 993–1022.
10. J. Tang, H. fung Leung, Q. Luo, D. Chen, and J. Gong, Towards ontology learning from folksonomies. in IJCAI, C. Boutilier, Ed., 2009: 2089–2094.
11. W. Wang, P. M. Barnaghi, and A. Bargiela, Probabilistic topic models for learning terminological ontologies. IEEE Trans. Knowl. Data Eng., vol. 22, no. 7, pp. 1028–1040, 2010.
12. Itti, L., Baldi, P.: Bayesian surprise attracts human attention. In: NIPS. (2005)
13. Li, X.; Wang, H.; Yin, G.; Wang, T.; Yang, C.; Yu, Y. & Tang, D. Inducing Taxonomy from Tags: An Agglomerative Hierarchical Clustering Framework., in Shuigeng Zhou; Songmao Zhang & George Karypis, ed., 'ADMA' , Springer: 64-77.
14. Chongming Gu, Gang Yin, Tao Wang, Cheng Yang, Huaimin Wang. A supervised approach for tag hierarchy construction in open source communities. Asia-pacific Symposium on Internetware, 2015 :148-152.
15. P. De Meo, G. Quattrone, and D. Ursino. Exploitation of semantic relationships and hierarchical data structures to support a user in his annotation and browsing activities in folksonomies. Inf. Syst., 34(6):511–535, 2009.
16. Marszałek M, Schmid C. Constructing category hierarchies for visual recognition[M]//Computer Vision–ECCV 2008. Springer Berlin Heidelberg, 2008:479-491.
17. Brooks, C.H., Montanez, N.: Improved annotation of the blogosphere via auto-tagging and hierarchical clustering. In Carr, L., Roure, D.D., Iyengar, A., Goble, C.A., Dahlin, M., eds.: WWW, ACM (2006) 625–632.
18. Miller, G.: WordNet: a lexical database for English. Communications of the ACM 38(11), 39-41 (1995).
19. Fahad Almoqhim, David E. Millard, Nigel Shadbolt: Improving on Popularity as a Proxy for Generality When Building Tag Hierarchies from Folksonomies. International Conference on Social Informatics , 2014, 8851:95-111.
20. Mario Linares-Vásquez, Collin McMillan, Denys Poshyvanyk, Mark Grechanik: On Using Machine Learning to Automatically Classify Software Applications into Domain Categories. 《Empirical Software Engineering》 , 2014 , 19 (3) :582-618