Proceedings of the IASTED International Conference
ADVANCES IN COMPUTER SCIENCE AND TECHNOLOGY
January 23-25, 2006, Puerto Vallarta, Mexico

A WEIGHTED RANKING ALGORITHM FOR FACET-BASED COMPONENT RETRIEVAL SYSTEM

Yong Yang, Weishi Zhang, Xiuguo Zhang, Jinyu Shi
Department of Computer Science and Technology, Dalian Maritime University, Dalian 116026
{dmuyy, teesiv, zhangxg, sjy1967}@dlmu.edu.cn

ABSTRACT
Facet-based component retrieval techniques have been proved to be an effective way for retrieving. These Techniques are widely adopted by component library systems, but they usually simply list out all the retrieval results without any kind of ranking. In our work, we focus on the problem that how to determine the ranks of the components retrieved by user. Factors which can influence the ranking are extracted and identified through the analysis of ER-Diagram of facet-based component library system. In this paper, a mathematical model of weighted ranking algorithm is proposed and the timing of ranks calculation is discussed. Experiment results show that this algorithm greatly improves the efficiency of component retrieval system.

KEY WORDS
Weighted ranking algorithm, matching degree, facet, component retrieval, and component library

1. Motivations

A high efficiency retrieval system for software component library is important for the reuse of software components. The point of high efficiency is not that the time performance in one matching or retrieving process which can be measured by how many seconds or how many milliseconds elapsed, but that the efficiency to make the component consumers be able to find what they need as soon as possible, even though the former is the basis of the latter.

No matter accuracy matching or fuzzy matching, our component retrieval system usually simply lists out all the retrieval results without any kind of ranking, or at least without a systematic ranking. Users have to view the detail information of all the retrieval results one by one to find out which is the best to fit their requirements, or else they have to adjust their query conditions to retrieve again. If there are a large number of components retrieved from the component library, it could be a tough and torturous experience to find a proper component. However, it's a fact that there's a matching degree between the query conditions and retrieval results. The matching degree is just the similarity and relevancy between the query condition and its retrieval results. Only when we rank the retrieval results by the matching degree as the Web search engines can component consumers easily find what they need. They only have to compare the first several retrieval results but not all of them.

According to the discussion above, it's clear that a formula to calculate the matching degree and its corresponding ranking algorithm, which can greatly improve the retrieval efficiency for software component library, are needed. In this paper, we propose a weighted ranking algorithm for facet-based component retrieval system. This algorithm has been implemented in a software component library, called DLCL, and greatly improves the efficiency of the retrieval system.

2. Introduction to Retrieval Methods for Component Library

2.1 Existing Retrieval Methods for Component Library

Retrieval of software components is a core technique of component library. Today there are lots of retrieval methods for software component library. The main are as follows [1, 2]: (1) Specification matching method; (2) AI Based method; (3) Information science method; (4) Hypertext browsing method. As to the four methods, each has its own features and there's no a general formula to calculate the matching degree. For example, specification matching method uses formal specifications to describe the behavior of software components and relies on theorem proving to determine match and mismatch. AI Based method relies on the use of AI planning techniques to automatically search software components in component library. So we have to use different calculating strategies to calculate the matching degree of each retrieval method.

Among the retrieval methods discussed above, information science method is widely used in practice. Information science method usually comprises several different retrieval methods which are attribute-value, enumerated, faceted, and keyword method. Of the four methods, facet-based component retrieval method has been proved to be an effective way for retrieving and has been widely adopted by component library systems. In the following section, we'll discuss the facet-based retrieval method.

2.2 Facet-based Retrieval Method

A component classification is a set of {facet, facet term} pairs, also called descriptors [3]. Reusable software components (RSC) are classified by assigning appropriate facet terms for all applicable facets. The objective in classifying the RSC is to make it possible for all users who might use the RSC to retrieve it as a result of their requests. Faceted classification scheme is an effective way for classifying the components and widely adopted by component library systems.

Correspondingly, there are several retrieving algorithms for faceted classification scheme. Some systems use the traditional database query techniques in facet-based retrieval. Wang YF proposed a tree matching algorithm in his PH.D dissertation [4]. This algorithm maps the component facets into a facet tree and maps the query conditions into a query tree. The matching algorithm deals with the match of the facet tree and query tree and calculates the matching cost. This algorithm bases on the tree matching theories, such as tree embedding, tree inclusion, and tree containment. These three tree matching methods are becoming more and more elastic in order to improve the retrieving recall while maintaining the precision to a certain extent. Matching cost of the tree matching will be calculated to measure the approximate degree between the facet trees of the components and the query tree. The data structure of a tree is represented by a three-tuple: T= (V, E, root (T)), V represents a limited set of vectors, root (T) represents the root of the tree, E represents the set of edges.

3. Weighted Ranking Algorithm for Facet-based Component Retrieval System

There's no a general formula to calculate the matching degree due to the different feature of each retrieval method. Facet-based retrieval method has been widely adopted by existing component library systems, such as REBOOT, Proteus, Asset Library, and JBCL [5]. It has been proved to be an effective method to the retrieval of component library system. And therefore, it makes great sense to propose a component ranking algorithm for facet-based retrieval system.

3.1 ER-Diagram of Software Component Library

The extraction and identification of the influential factors which are used to calculate the matching degree is the first step to establish a mathematical model. To Analyze the ER-Diagram of software component library is an effective way to extract the factors. An ER-diagram of facet-based component library was given below:

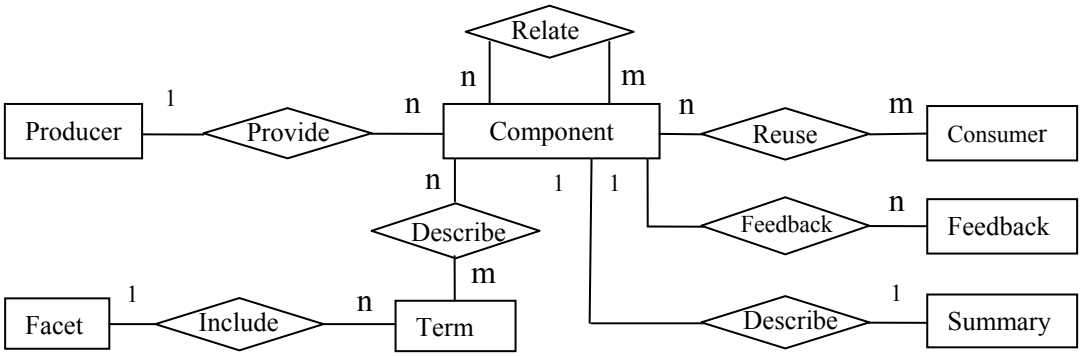

Fig. 1. ER-Diagram of Component Library

Entities list:

- Component: component is the basic and primary entity in component library. Besides the attributes, there are facet-term pairs and information summary to describe a component.
- User Feedback: an opinion, a comment or a score provided by users after they have used a component.
- Component Summary: an information summary to describe a component which enables users to know well the component quickly.
- Facet: facet and its terms are used to classify and represent the components.

3.2 Factors of Weighted Ranking Algorithm

As to a facet-based component library system, facet is the most important method to classify and represent the components. Correspondingly, facet-based retrieval methods, such as facet tree matching method, are important for the component retrieval system. The matching degree between facet tree and query tree is of much importance for ranking. However, matching degree of facet is not the only factor which is able to influence the ranking.

Retrieval system of component library usually has two search modes: simple query and complex query. Simple query just simply uses the traditional database query method to match the Attribute-Valued pairs. In contrast, complex query is a much more effective way which combines several query methods together to match different kinds of component information. And therefore,

we should take other factors into account besides the facet for ranking the retrieval results. According to the analysis of ER-Diagram above, we can extract some other factors which are able to influence the ranking of component retrieval results while using the complex query.

Attributes of component, such as component name, can be used to match the keywords in the query conditions. The matching degree of Attribute-Valued pairs should be an influential factor for ranking.

Summary of component can also be used to match the query conditions. Query conditions usually consist of several keywords. The density, prominence, and position of keywords within the component summary will influence the ranking of components. The keyword density is just the number of occurrences of the keywords within the component summary divided by the total number of words. Keyword prominence is related to the location of keywords in the summary. For example, keywords placed at the beginning of the summary maybe carry more weight than those towards the end of it.

User feedback of a component is very useful for other users who want to use it to evaluate the quality and other features of the component. They can acquire much more objective description and useful information about the component besides the component attributes and summary.

How many times the component information has been visited and how many times the component has been downloaded for reusing should also be taken into account as the factors to calculate the matching degree for ranking. They reflect the popularity and reusability of the component from another aspect.

3.3 Mathematical Model

Retrieval results consist of a collection of components matching the query conditions:

Definition 1: Components (C_1, C_2, ……, C_i, ……, Cn); ($n \in N$, $n \geq 1$)

It makes no sense to discuss the circumstance of empty retrieval results, since that we are going to discuss the ranking of component lists.

Accordingly, each component has a rank value:

Definition 2: Ranks (R_1, R_2, ……, R_i, ……, Rn); ($n \in N$, $n \geq 1$)

The query condition consists of a collection of keywords:

Definition 3: Keywords (K_1, K_2, ……, K_i, ……, K_{n0}); ($n_0 \in N$, $n_0 \geq 1$)

Each component is described by a set of Attribute-Valued pairs:

Definition 4: Attributes (A_1, A_2, ……, A_i, ……, A_{n1}); ($n_1 \in N$, $n_1 \geq 1$)

Besides with Attribute-Valued pairs, components are also classified and represented by a set of facets and their terms:

Definition 5: Facets (F_1, F_2, ……, F_i, ……, F_{n2}); ($n_2 \in N$, $n_2 \geq 1$)

Summary of component information differs from Attribute-Valued pairs. It provides a comprehensive description of a component in context.

Definition 6: Summary (S);

User feedback includes all the comments and feedback to a specific component:

Definition 7: User Feedback (U_1, U_2, ……, U_i, ……, U_{n3}); ($n_3 \in N$, $n_3 \geq 1$)

User feedback must be analyzed and evaluated to a relative number. We use E to represent the Evaluation number of user feedback.

Definition 8: E = Evaluate (User Feedback).

Definition 9: Visited times of a component: Visited times (V);

Definition 10: Downloaded times of a component: Downloaded times (D);

We have listed out all the influential factors above, which constitute a six-tuple:

Factors (A, F, S, U, V, D);

Their influential weights differ from each other according to their feature and importance:

Definition 11: Weights (W_A, W_F, W_S, W_U, W_V, W_D); ($0 \leq W_A$, W_F, W_S, W_U, W_V, $W_D \leq 1$, $W_A+W_F+W_S+W_U+W_V+W_D = 1$)

W_A represents the weight of Attributes; W_F represents the weight of Facets; W_S, W_U, W_V, and W_D also represent the weight of corresponding factor discussed above.

There are several functions for calculating the matching degree of some factors. The core calculating formula of each function relies on its corresponding matching algorithm.

	Functions	Formulas
Summary	F_S (Keywords, Summary)	$\sum_{i=1}^{no} match(Ki, S)$
Facets	F_F (Keywords, Facets)	$\sum_{i=1}^{no}\sum_{j=1}^{n2} match(Ki, Fj)$
Attributes	F_A (Keywords, Attributes)	$\sum_{i=1}^{no}\sum_{j=1}^{n1} match(Ki, Aj)$

Match function of component summary uses the content-based similarity measurement algorithm of the search engine techniques. A Best-First algorithm was proposed by Cho [6]. This algorithm uses a vector space model to calculate the similarity between the keywords and the content. Its formula is given as following:

$$sim(q, p) = \frac{\sum_{k \in q|p} W_{kq} W_{kp}}{\sqrt{\sum_{k \in p} W_{kp}^2 \sum_{k \in q} W_{kq}^2}}$$

The variable q represents the collection of keywords, p represents the content, and W_{kp} represents the importance of k to a specific topic. In our mathematical model, variable q represents the query condition, and the variable p represents the summary of the component information.

Facet-based retrieving method usually adopts the facet tree matching. And therefore, its match function calculates the matching degree between the facet tree of component and the query tree. A formula to calculate the matching cost of tree containment matching was given by Xu [7]:

Q=(V,E,root(Q)), D=(W,F,root(D)) are two unordered label tree, TCostM(Q, D) represents the tree containment matching cost from tree Q to tree D.

$$\mathrm{TCostM}(Q,D)=\min\{\gamma(f)\mid f:Q\Rightarrow D\}$$

$$\gamma(f)=\sum_{v\in domain(f)}\gamma(label(v)\rightarrow label(f(v)))+\sum_{v\in V-domain(f)}\gamma(label(v)\rightarrow\lambda)+\sum_{w\in spectrum(f)-Range(f)}\gamma(\lambda\rightarrow label(v))$$

If f is a tree containment matching from tree Q to tree D, and γ (f) = TCostM(Q, D), then f is the tree containment matching which obtains the minimum matching cost from tree Q to tree D. This definition could be also applied to the containment matching between tree and forest or between forest and forest.

As to the match function of component attributes, we just use the traditional database query methods to deal with it.

E, V, and D are three ranking factors without any relation to query keywords. Even though they are numbers, we could not use them directly for ranking. Functions should be provided to transform them.

	Functions
Evaluation of Feedback	F_E (E)
Visited Times	F_V (V)
Downloaded Times	F_D (D)

According to the discussion above, we finally draw out a very simple formula to calculate the rank for each component:

Rank = $F_A \times W_A + F_F \times W_F + F_S \times W_S + F_E \times W_E + F_V \times W_V + F_D \times W_D$

We can use matrix operation to represent the calculation of rank value for each component in the retrieval results. There are n components and 6 influential factors. $F_{6\times n} \times W_{1\times 6} = R_{1\times n}$:

$$\begin{bmatrix} F_{A1} & F_{F1} & F_{S1} & F_{E1} & F_{V1} & F_{D1} \\ F_{A2} & F_{F2} & F_{S2} & F_{E2} & F_{V2} & F_{D2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ F_{Ai} & F_{Fi} & F_{Si} & F_{Ei} & F_{Vi} & F_{Di} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ F_{An} & F_{Fn} & F_{Sn} & F_{En} & F_{Vn} & F_{Dn} \end{bmatrix} \times \begin{bmatrix} W_A \\ W_F \\ W_S \\ W_E \\ W_V \\ W_D \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_i \\ \vdots \\ R_n \end{bmatrix}$$

We specify the weights with experiential values at the very beginning, and then use the data mining technology to analyze the user logs to dynamically and iteratively adjust those values.

3.4 Timing of Ranks Calculation

It's time for us to determine when to deal with the calculation of ranks, since that we have designed how to deal with it. It's just the matter of the timing of rank calculation. There are two probable times for calculation: calculating after the retrieving process has been finished; calculating during the process of retrieving. Both of the probable times have their own advantages and disadvantages.

If we calculate the rank after the retrieving process has been finished, we can deal with the retrieving and ranking separately. It will be much easier for us to design and maintain the system, since the retrieving and ranking process are independent. However, it costs much more time and space to calculate the rank. It needs a lot of memory space to store a large number of retrieval results temporally before they are ranked, and costs much more time to manage the transmission of data between storage devices and CPU. On the contrary, if we calculate the rank during the process of retrieving, we have to combine the retrieving with the ranking process completely or partly. Undoubtedly, it will be hard for us to implement and maintain the system, but it can greatly improve the time and the space performances.

According to the discussion above, we can choose a proper time to calculate the ranks. Which solution we should choose depends on the requirements of the system. The solution which calculates the ranks during the retrieving process should be adopted if the time and the space performances are rigorously required.

4. Implementation

Our component library system, named DLCL, is implemented with J2EE platform. The mathematical model and its algorithm we discussed above have been implemented in this system with Java. Java is an object-oriented language. Its implementation consists of several core interfaces and classes. The core interfaces, classes and the relationship between them are demonstrated in the UML class diagram:

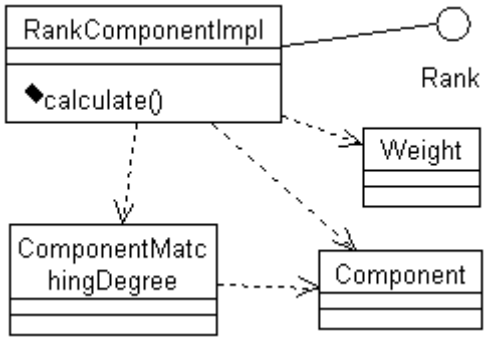

Fig. 2. Class Diagram of Ranking Module

We calculate the ranks during the retrieving process to improve the time and space performances, and therefore, we have to combine the ranking module partly with the retrieving module. In order to lower the coupled degree between these two modules, a callback mechanism was adopted. We define an interface Rank, which consists of only one method to calculate the ranks. RankComponentImpl is a class implementing the interface Rank to calculate the ranks of those components retrieved by users. The method of its concrete object can be executed by the searcher during the retrieving process. Class Component encapsulates those methods which provide the component information. ComponentMatchingDegree is a class providing those methods for calculating the matching degree between the query keywords and component. Each influential factor has its own strategy for calculation. There's also a class Weight, which provides the methods to get the influential weight of each factor.

5. Experiment and its Results

In order to verify the efficiency of the component retrieving system which adoptes our weighted ranking algorithm, we design an experiment and carry out this experiment in our component library system, named DLCL. There are more than 1000 components in this system. Retrieving system of DLCL splits the retrieval results into several pages if there are too many components retrieved, and lists out 10 components per page.

The experiment separates the users into two groups, group 1 and group 2. Each group consists of 10 persons. All the users know about the knowledge of component reuse to a certain extent. Both groups use the facet-based component retrieving method to retrieve the components. Retrieval results of group 1 are listed out without any ranking, however, those of group 2 are ranked by our weighted ranking algorithm.

There are several aspects to measure the efficiency of each group: how many pages they turned; how many times they had to adjust the query condition; and the most important, how many time was elapsed during the whole retrieving process. The experimental results are given in the following table:

	Group 1	Group 2
Average Turned Pages (pages)	2.7	1.4
Average Adjusted Times (times)	2.3	1.1
Average Time elapsed (minutes)	26.6	9.5

The experimental results obviously shows that the efficiency of group 2 is greatly higher than group 1. By applying the weighted ranking algorithm into the retrieving system of DLCL, users needn't turn too many pages to view and compare the component information or to adjust the query condition to improve the query precision. Only to view the first page of retrieval results will be enough most of the time. And therefore, it greatly saves the time and retrieval costs.

6. Related Works

The idea of Component Rank comes from computing fair impact factors of published papers [8]. Google is a web search engine. Its method can be considered as an HTML extension of the method proposed for counting impact of publications, called influence weight in [8]. Google computes the ranks (called PageRanks) for HTML documents in the Internet [9, 10]. In reference [11], the authors present the Component Rank model for ranking software components, and show a system for computing Component Rank. In this model, a collection of software components is represented as a weighted directed graph whose nodes correspond to the components and edges correspond to the usage relations. Similar components are clustered into one node so that effect of simply duplicated nodes is removed. The nodes in the graph are ranked by their weights which are defined as the elements of the eigenvector of an adjacent matrix for the directed graph. A major distinction of Component Rank model in [11] from PageRank and the influence weight in [9, 10] is that Component Rank model explores similarity between components before the weight computation.

In this paper, we also propose a weighted ranking algorithm for component retrieval system. This weighted ranking algorithm uses different calculating strategies according to the feature of facet-based retrieval methods. While in [11], the authors employed only statical use relations.

7. Conclusion

In this paper, a mathematical model of weighted ranking algorithm is proposed and the timing of ranks calculation is discussed. We have applied this ranking algorithm into our component library system, named DLCL. The experiment we carried out shows that this algorithm greatly improves the efficiency of component retrieving system, saving the time and retrieval costs for component reusing.

Acknowledgement

This research is partially supported by the National High Technology Development 863 Program under Grant No. 2004AA116010.

References

[1] Frakes WB, Pole TP, An empirical study of representation methods for reusable software components, *IEEE Transactions on Software Engineering*, 1994, l20(8), pp617-630

[2] H. Mili, R. Rada, W. Wang, K. Strickland, C. Boldyreff, L. Olsen, J. Witt, J. Heger, W. Scherr, and P. Elzer, Practitioner and SoftClass: A Comparative Study of Two Software Reuse Research Projects, *J. Systems and Software*, 1994, 27(5)

[3] NEC Software Engineering Laboratory, NATO Standard for Management of a Reusable Software Component Library, *NATO Communications and Information Systems Agency*, 1991

[4] Wang YF. Research on retrieving reusable components classified in faceted scheme [*Ph.D. Thesis*]. Shanghai: Fudan University, 2002.

[5] Chang JC, et al. Representation and Retrieval of Reusable Software Components [J].*Computer Science*, 1999, 26(5):41-48.

[6] Cho J, Garcia-Molina H, Page L. Efficient Crawling Through URL Ordering [J]. *Computer Networks*, 1998, 30(1~7):161-172

[7] Xu RZ, et al. Research on Matching Algorithm for XML-Based Software Component Query, *Journal of Software*, 2003, 14(7):1195-1202.

[8] G. Pinski and F. Narin. "Citation Influence for Journal Aggregates of Scientific Publications: Theory, with Application to the Literature of Physics". *Information Processing and Management*, 12(5):297.312, 1976.

[9] L. Page, S. Brin, R. Motwani, and T. Winograd. "The PageRank Citation Ranking: Bringing Order to the Web". Technical Report of Stanford Digital Library Technologies Project, 1998. *"http://www-db.stanford.edu/.backrub/ pageranksub.ps"*.

[10] J. Kleinberg. "Authoritative Sources in a Hyperlinked Environment". *Journal of the ACM*, 46(5):604.632, 1999.

[11] Katsuro Inoue, Reishi Yokomori, Hikaru Fujiwara, Tetsuo Yamamoto, Makoto Matsushita, Shinji Kusumoto: Component Rank: Relative Significance Rank for Software Component Search. *ICSE 2003*: 14-24