Vector Space Based on Hierarchical Weighting: A Component Ranking Approach to Component Retrieval

Gui Gui and Paul D. Scott

Department of Computer Science, University of Essex, Colchester ,UK
{ggui, scotp}@essex.ac.uk

Abstract. In this paper, we present an approach to software component ranking intended for use in searching for such components on the internet. The method used introduces a novel method of weighting keywords that takes account of where within the structure of a component the keyword is found. This hierarchical weighting scheme is used in two ranking algorithms: one using summed weights, the other using a vector space model. Experimental comparisons with algorithms using TF-IDF weighting that ignore component structure are described. The results demonstrate consistent superiority of the hierarchical weighting approach.

1 Introduction

The internet is rapidly becoming a major repository of software components. In order to realise the potential of this resource, software developers need effective search engines that can retrieve those components that match their requirements. Conventional search engines are unsuitable for this task because they are oriented towards retrieving documents comprised largely of natural language text. Software components, viewed as text objects, differ from natural language documents in that they have a much more rigid structure but a greatly impoverished vocabulary.

The simplest possible approach is to retrieve all components that contain the keywords that appear in the query. Unfortunately, this typically returns a large number of components of which only a small percentage are actually relevant; that is, only a few of the many components retrieved match the user's requirements. A more sophisticated approach, in which the retrieved components are ranked by relevance, is thus essential.

In this paper, we propose a new ranking algorithm that exploits component structure by combining text information ranking techniques with the hierarchical structure analysis of components. In our proposal, keywords are retrieved from naming information at different hierarchical levels within components and used as index words. A hierarchy weighting (HW) scheme is developed to define the relative importance of keywords found at different levels in the structural hierarchy of components. The similarity of components and queries can then be defined using a vector space representation [3], [4] of the set of index keywords. In our work, we focus on Java

J. Cao, W. Nejdl, and M. Xu (Eds.): APPT 2005, LNCS 3756, pp. 184 – 193, 2005.

components. The main contribution of our proposal is that it demonstrates the importance of component structure to deriving an accurate ranking of the relevance of components to queries.

The paper is organized as follows: Section 2 reviews related work and introduces the novel features of our approach; Section 3 describes the proposed hierarchy weighting scheme and the vector space based on hierarchy weighting for component ranking; Section 4 presents some experiments and analysis. The paper concludes with a discussion of the implications of our findings.

2 Related Work

Much of the existing literature on component retrieval techniques is based on their application to component repositories that take the form of well organized, documented and maintained libraries of software components. Such repositories are undoubtedly a valuable resource to a software engineer but, by their very nature, they are highly labour intensive. In contrast, our concern in the present paper is with retrieving components from the internet. Typically little or no documentation is available for such components and any indexing must be done automatically because of the huge numbers of components to be considered.

There are two main approaches to ranking in existing component retrieval systems: ranking by usability and ranking by relevance (Table 1). Ranking by usability, which attempts to identify well engineered components that can readily be incorporated as part of a complex piece of software, is outside the scope of the present paper. Here we are concerned only with ranking by relevance which attempts to identify components that meet the users' functional requirements.

Table 1. Categories of component ranking

Ranking by usability	Ranking by relevance
SourceForge [12]	Domain, facet classification [15], [11]
Based on formal specification [10], [7]	TF-IDF [14]
Based on usage relationship (Spars [8])	Natural language processing [13]
	Concept lattice [9]
	Ontology processing [5], [6]

The various approaches to ranking components by relevance differ chiefly in the complexity of the representations used to describe both components and queries. Semantic based component retrieval [13] uses a natural language representation. Components are described by specific domain models in natural language. Relevant components are retrieved using a closeness measure based on semantic and syntactic analysis. The use of natural language provides a natural and flexible way to express users' queries and describe the context of a component. Unfortunately, this approach presupposes that natural language descriptions of components either exist or can be constructed so it is not really a practical method for retrieving components from the internet where documentation may be sparse or non-existent. Furthermore, the natural

language processing involved is likely to make any such system very slow if large numbers of components must be examined.

Some approaches simplify natural language processing by using concept lattices [9] and ontology processing [5], [6]. Ranking by adopting an ontology builds a relationship between domain models and reuse repository. The limitations of this proposal are that it assumes the existence of an ontology and search quality depends on the availability and accuracy of ontology and domain model. Alternatively concept lattices can be used to represent the relationship between keywords and components. As with the natural language approach, these methods presuppose that a great deal of work has been done in constructing an appropriate ontology and categorizing the components appropriately. Hence again, it appears unsuitable for retrieving components from the internet.

Components can be classified into different categories according to domain and facet information [11], [15]. Ranking of components is based on the relationship of domains. However, the granularity of ranking based on domain classification is too big to rank each individual component properly. Some components that can not be classified into any pre-existing categories or the intersections of categories could result in loss of some components. The flexibility of component retrieval can be lost by forced classification of components.

All of the above approaches seem more suited to aid retrieval from well maintained and systematically organized software repositories than for searching the internet. An alternative is to make use of the standard text information retrieval techniques that have proved so effective in searching for ordinary text documents on the internet. Such approaches typically index a document by the keywords it contains and match queries to documents on the basis of the keywords they have in common. The majority of such systems associate a weight with each occurrence of a keyword that provides an estimate of how much evidence the keyword provides about the content of the document.

Washizaki and Fukazawa [14] have ranked components using TF-IDF weighting, in which the weight is proportional to frequency of the keyword in the component and inversely proportional to the number of components in the repository that include that keyword. Although this approach can achieve some success, it is limited because it does not exploit all the information that is readily available from a component. In particular, it only considers keyword frequency and takes no account of the structure of a component; the significance of a keyword may well depend on where it appears within the component. Naming information has proved to be effective in component clustering [1], [2].

Consequently it appears worthwhile to develop a weighting scheme that reflects not only the frequency of a keyword but also its location within the structure of a software component.

3 Hierarchical Weighting and Vector Space Representation

The work reported in this paper is concerned with retrieving Java components. The decision to concentrate on Java was taken for two main reasons. First, very large numbers of Java components, often taking the form of Java Beans, are available in the public domain on the internet. Second, the Java facilities of reflection and introspection permit the exploration of the structure of such components.

Programmers often provide clues to the function of a component in the names they give to the various entities of which it is composed. These include the names given to the classes and methods and the name of the file containing the component. Hence index keywords can be obtained by searching for legitimate words within the collection of names used by the programmer. Some of these keywords will provide more information than others about the function of the component. Thus, in addition to extracting the keywords, it is desirable to associate a weight with a each keyword, in a each component, that reflects how much evidence it provides about component functionality. The main hypothesis of this paper is that the location of a keyword within the structure of a component provides a strong indication of how much functional evidence it provides.

Let $C = \{c_1, c_2, ...c_i, ...c_n\}$ be the set of all the components in the entire collection where n is the total number of components. $K = \{k_1, k_2, ...k_i, ...k_m\}$ is the set of index keywords extracted from the entire collection of components. A weight W_{ij} is assigned to each index keyword k_i in the component c_j in order to measure the importance of k_i in expressing the content of component c_j. If keyword k_i appears in the component c_j, $W_{ij} > 0$; if not, $W_{ij} = 0$.

Java components are normally compressed as jar files that can include several classes, each of which contains several methods. Keywords are extracted from the file names, class names and method names. Set $k^{jar} = (k_1^{jar}, k_2^{jar}, \ldots, k_N^{jar})$ comprises the keywords retrieved from the jar file name, set $k^{c_i} = (k_1^{c_i}, k_2^{c_i}, \ldots k_r^{c_i})$ comprises the keywords retrieved from the name of class c_i, and set $k^{f_i} = (k_1^{f_i}, k_2^{f_i}, \ldots k_s^{f_i})$ contains the keywords retrieved from name of method f_i. The relationship of keywords from different levels is shown in Fig. 1.

Thus, the complete set of keywords for a given component is:

$$K = (k_1^{jar}, k_2^{jar}, ...k_N^{jar}) \cup (\bigcup_{i=1}^{n} (k_1^{c_i}, k_2^{c_i}, ...k_{r_i}^{c_i})) \cup (\bigcup_{j=1}^{m} (k_1^{f_j}, k_2^{f_j}, ...k_{s_j}^{f_j}) \quad (1)$$

Keywords from higher levels of the hierarchy are more likely to convey information about the overall function of the component. Hence it is appropriate to give greater weight to keywords from higher levels in the hierarchy. Thus, $W_{ij}^{jar} > W_{ij}^{class} > W_{ij}^{method}$

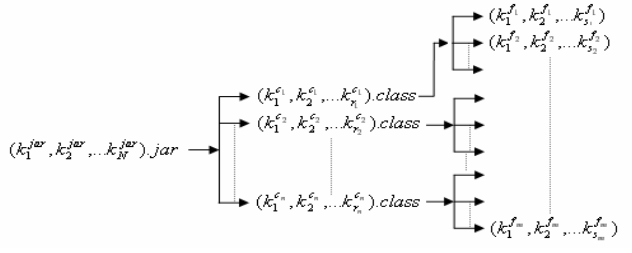

Fig. 1. Keywords hierarchical level breakdown

where W_{ij}^{lev} denotes the weight assigned to the *i*th keyword at level *lev* in the *j*th component. The number of retrieved keywords usually increases as one moves down the hierarchy. Typically there is only one jar file which will contain several classes, each of which contains several methods. Hence a simple way of assigning greater weight to keywords retrieved from higher levels is:

$$W_{ij}^{lev} = \frac{f_{ij}}{\psi^{lev}} \tag{2}$$

where f_{ij} is the frequency of occurrences of keyword k_i and ψ^{lev} is the number of entities at hierarchy level *lev* in the *j*th component. It is clear that, provided a jar file contains more than one class and each class contains more than one method, then if a given keyword occurs with the same frequency at all levels of the hierarchy $W_{ij}^{jar} > W_{ij}^{class} > W_{ij}^{method}$. The total weight assigned to a give keyword i in a particular component j is thus:

$$W_{ij} = W_{ij}^{jar} + W_{ij}^{class} + W_{ij}^{method} = f_{ij}^{jar} + \frac{f_{ij}^{class}}{\psi^{class}} + \frac{f_{ij}^{method}}{\psi^{method}} \tag{3}$$

Hence the frequency of keywords and their location within the structure of a component are combined to estimate their importance in describing the function of that component.

These hierarchical keyword weights can be used in two distinct ways to match queries to components. The first, termed HW, is simply to give each component a similarity score formed as the sum of the hierarchical weights of all words appearing in the query. A more sophisticated alternative, termed VS-HW, is to adopt a vector space representation [3]. Each component is represented as a vector as $\vec{c}_j = (W_{1j}, W_{2j}, ..., W_{mj})$ in the space of all index terms. A query is represented as a vector $\vec{q} = (W_{1q}, W_{2q}, ..., W_{mq})$ where W_{iq} denotes the importance of the *i*th keyword to the query q (always either 0 or 1 in the present study). The similarity of component and the query, $Sim(q, c_j)$, is then defined as:

$$Sim(q, c_j) = \frac{\sum_{i=1}^{t} W_{ij} \cdot W_{iq}}{\sqrt{\sum_{i=1}^{m} (W_{ij})^2} \sqrt{\sum_{i=1}^{t} (W_{iq})^2}} = \frac{\sum_{i=1}^{t} (f_{ij}^{jar} + \frac{f_{ij}^{class}}{\psi^{class}} + \frac{f_{ij}^{method}}{\psi^{method}})}{\sqrt{\sum_{i=1}^{m} (f_{ij}^{jar} + \frac{f_{ij}^{class}}{\psi^{class}} + \frac{f_{ij}^{method}}{\psi^{method}})^2} \sqrt{\sum_{i=1}^{t} (W_{iq})^2}} \tag{4}$$

4 Experimental Results

In order to investigate the efficacy of this hierarchical weighting scheme for reliably ranking components, approximately 10,000 components were retrieved from the internet using a randomly seeded spider program. Keywords were then identified as fragments of filenames, class names and method names using a dictionary based approach. A total of 21,778 such keywords were found. In addition to the two ranking schemes based on hierarchical weightings, comparative experiments were also conducted using the TF-IDF weighting scheme [3] which is widely used in

conventional text retrieval. It differs from our hierarchical weighting scheme chiefly in that in that it assigns the same weight to a keyword wherever it appears within the structure of a component. Both direct (TF-IDF) and vector space (VS-TF-IDF) variants of TF-IDF ranking were implemented.

Searches were carried out using 50 different queries; 20 were single word queries, the remainder comprised either two or three keywords. The total number of components retrieved (i.e. achieving a score greater than zero) for each query varied but fell in the range 140-180 with an average of 158. The components returned were categorized as relevant or irrelevant to the query by inspection; the average number of relevant results was 67.

Three methods were used to compare the performance of the four ranking schemes: average cumulative relevant results distribution, precision distribution and precision histograms [3].

4.1 Average Cumulative Relevant Results Distribution

The cumulative relevant results distribution graph is a plot of the number of relevant results retrieved in the highest ranked x components as x is increased from 1 to T, the total number of components retrieved. An ideal ranking scheme would rank all the relevant components before all those that are irrelevant. Hence the ideal distribution takes the form of a straight line from 0,0 to R,R, where R is the number of relevant components, followed by a second (flat) straight line from R,R to T,R. The quality of a ranking scheme is indicated by how closely it approaches this ideal. Completely random ranking would result in a single straight line from 0,0 to T,R.

Fig. 2 shows the average cumulative relevant results distribution for all four ranking schemes over the 50 queries. It can be seen that all four algorithms generally perform much better than random ranking, although the TF-IDF results are indistinguishable from random ranking in the 20 highest ranked components. None achieves ideal

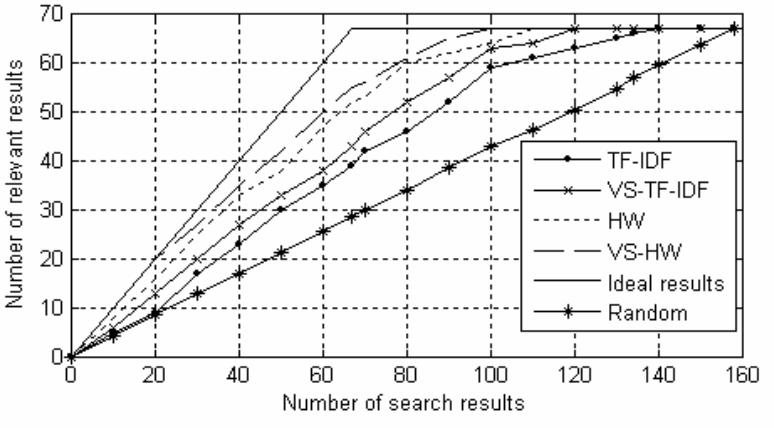

Fig. 2. Average cumulative relevant results distributions

performance, but it is clear that VS-HW performs better than the other three procedures. The performance of HW is only slightly worse. The rankings achieved by the two TF-IDF procedures are markedly worse.

4.2 Distribution of Mean Precision

The precision of a retrieval procedure is simply defined as the proportion of all retrieved items that are relevant. The rth-precision is the precision achieved for the r highest ranked retrieved items. The mean value of rth-precision over all values of r provides another way of comparing the rankings achieved by different ranking procedures, since a higher mean value indicates that more of the highly ranked items were relevant.

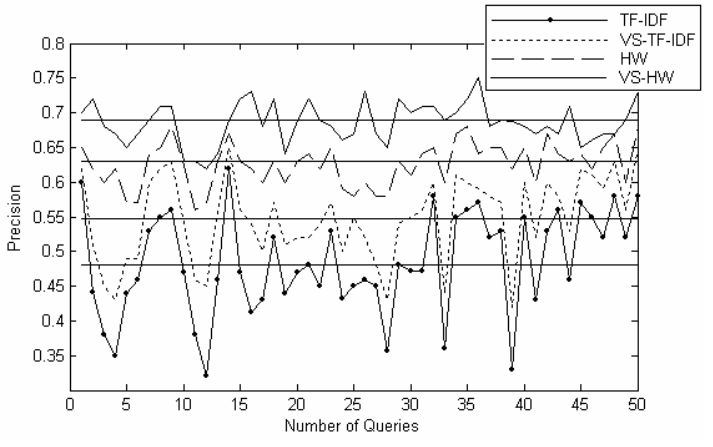

Fig. 3. Mean rth-precision distributions

Fig. 3 shows the mean rth-precision values achieved by the four ranking procedures for each of the 50 queries. To provide a baseline indication of performance across all the queries, the horizontal lines indicate the average mean rth-precision achieved by each procedure over all queries. That is, each line indicates the value of:

$$Mean\overline{RP}(A) = \frac{\sum_{i=0}^{N} \overline{RP}(A \mid i)}{N} \tag{5}$$

where $\overline{RP}(A \mid i)$ is the mean rth-precision achieved by ranking algorithm A on the ith query and N is the total number of queries.

It is clear that, as in the cumulative relevant results distributions, the two algorithms using hierarchical weighting perform considerably better than the two based on TF-IDF. Not only are their overall performances higher, as indicated by the baselines, but they are also more consistent since the deviations from the baseline are much smaller. Once again, the best results are obtained using VS-HW and the worst using TF-IDF.

4.3 Precision Histograms

Precision histograms [3] allow comparisons between two ranking algorithms by considering the differences between their rth-precision values at a specific value of r: the total number of relevant results for the current query. Consider two ranking algorithms, A and B, and let M be the total number of relevant results. Let $MP_A(i)$ and $MP_B(i)$ be the M-th precision values achieved by algorithm A and B for query *i*. Hence the difference between these two value $MP_{A/B}(i)$, is:

$$MP_{A/B}(i) = MP_A(i) - MP_B(i) \tag{6}$$

If $MP_{A/B}(i)$ is zero, the two algorithms have equivalent performance for that query. Positive values of $MP_{A/B}(i)$ imply that algorithm A performs better than algorithm B while a negative values of imply B has better performance.

Figure 4 shows the histograms achieved when each of the hierarchical weighting procedures are compared with each of the TF-IDF procedures on all 50 queries. The results are consistently positive demonstrating that the hierarchical weighting procedures perform better on the entire test set.

A quantitative measure of the difference between two algorithms across a set of queries can be obtained by considering the mean value of $MP_{A/B}(i)$:

$$\overline{MP_{A/B}} = \frac{\sum_{i=1}^{N} (\mathrm{MP_A(i)} - \mathrm{MP_B(i)})}{N} \tag{7}$$

Computing these mean differences for the four histograms demonstrates again that the performance of the two procedures using hierarchical weighting is superior to both of those using TF-IDF weighting (See Table 2).

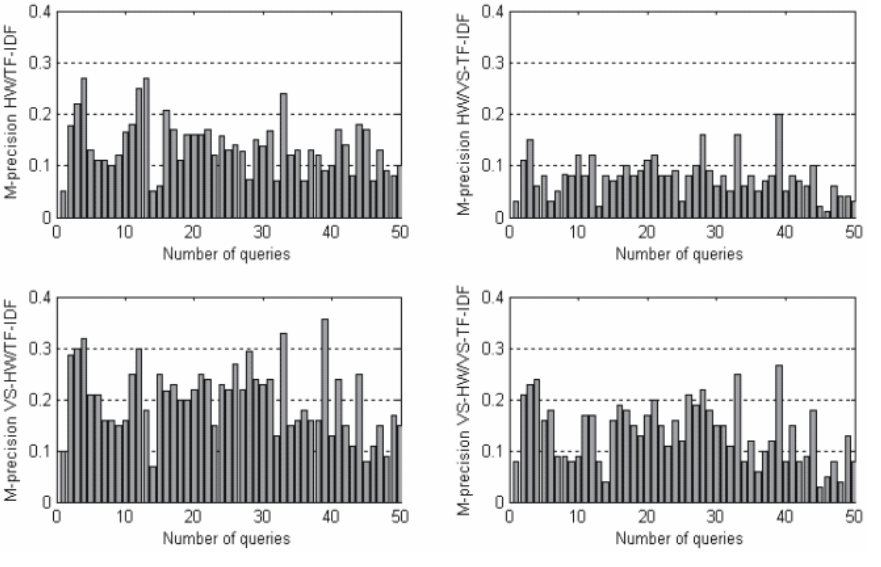

Fig. 4. M-th Precision Histograms

Table 2. Mean Mth-precision differences

Procedures Compared	Mean Mth-precision difference
HW/TF-IDF	0.138
HW/VS-TF-IDF	0.078
VS-HW/TF-IDF	0.20
VS-HW/VS-TF-IDF	0.136

5 Conclusion

In this paper, we have proposed a new approach to software component ranking that exploits the hierarchical structure of components to estimate how much information a keyword, extracted from an entity's name, provides about the overall function of the component. A simple scheme has been proposed, in which the weight associated with the occurrence of a keyword within an entity's name is inversely proportional to the number of entities of the same type in the component. This hierarchical weighting scheme has been incorporated into two ranking algorithms: one which simply uses the summed weight of all keyword and one which uses a vector space representation. The performance of these algorithms has been assessed by comparing them to their counterparts that use TF-IDF weighting and take no account of the component structure. Our results demonstrate the consistent superiority of the hierarchical weighting algorithms. The vector space algorithm performed slightly better than the simple summed weight method.

The work reported here is part of a larger project to develop a complete system for retrieving software components from the internet and it is clearly to this field that it makes the most direct contribution. However, we suggest that our findings may be of wider interest. In particular, the use of hierarchical weighting may be of similar value in information retrieval applications that involve any type of semi-structured text documents.

References

1. Andritsos, P., Tzerpos, V.: Information-Theoretic Software Clustering. IEEE Transactions on Software Engineering. Vol. 31, Issue 2 (2005) 150 - 165
2. Anquetil, N., Lethbridge, T.C.: "Recovering Software Architecture from the Names of Source Files," J. Software Maintenance: Research and Practice, vol. 11, pp. 201-221, May 1999.
3. Baeza, R., Neto, B.: Modern Information Retrieval. ACM Press, Addison Wesley, New York (1999)
4. Belew, R. K.: Finding Out About A Cognitive Perspective on Search Engine Technology and the WWW. Cambridge University Press (2000)
5. Bernstein, A., Klein, M.: Towards High-Precision Service Retrieval. Proc. International Semantic Web Conference (ISWC-02), Sardinia, Italy (2002)
6. Braga R.M.M., Mattoso, M., Werner, C.M.L.: The use of mediation and ontology technologies for software component information retrieval. Symposium on Software Reuse (SSR'01), Toronto, Canada, (2001)

7. Fischer, B.: Specification-Based Browsing of Software Component Libraries. Journal of Automated Software Engineering, Vol. 7, No. 2, (2000) 179-200
8. Inoue, K., Yokomori, R., Fujiwara, H., Yamamoto, T., Matsushita, M., Kusumoto, S.: Component Rank: Relative Significance Rank for Software Component Search. In: Proc. International Conf. on Software Engineering (ICSE2003), Portland, OR, (2003) 14–24
9. Lindig, C.: Concept-based component retrieval. In: Working Notes of the ZJCAI-95 Workshop: Formal Approaches to the Reuse of Plans, Proofs, and Programs. Kohler, J., Giunchiglia, F., Green, C., Walther, C. (eds) (1995) 21-25
10. Penix, J., Alexander, P.: Efficient Specification-Based Component Retrieval. Automated Software Engineering, Vol. 6. Kluwer Academic Publishers (1999) 139-170
11. Seacord, R., Hissan, S., Wallnau, K.: Agora: A Search Engine for Software Components. IEEE Internet Computing, Vol.2, No.6 (1998)
12. Sourceforge.: Sourceforge.net. http://sourceforge.net/. Accessed August 4th (2005)
13. Sugumaran, V., Storey, V.C.: A Semantic-Based Approach to Component Retrieval. The DATA BASE for Advances in Information Systems. Vol. 34, No. 3. ACM SIG Management Information Systems (2003) 8-24
14. Washizaki, H., Fukazawa, Y.: Component-Extraction-based Search System for Object Oriented Programs. Proc. 8th International Conference on Software Reuse, Lecture Notes in Computer Science, Vol. 3107, Springer-Verlag, Berlin Heidelberg New York (2004)
15. Zhang, Z., Svensson, L., Snis, U., Srensen, C., Fgerlind, H., Lindroth, T., Magnusson, M., Stlund, C.: Enhancing Component Reuse Using Search Techniques. Proceedings of IRIS 23. Laboratorium for Interaction Technology, University of Trollhttan Uddevalla (2000)