Improving Source Code Search with Natural Language Phrasal Representations of Method Signatures

Emily Hill
Deptartment of Computer Science
Montclair State University
Montclair, NJ 07043
Email: hillem@mail.montclair.edu

Lori Pollock and K. Vijay-Shanker
Deptartment of Computer and Information Sciences
University of Delaware
Newark, DE 19716
Email: {pollock, vijay}@cis.udel.edu

***Abstract*—As software continues to grow, locating code for maintenance tasks becomes increasingly difficult. Software search tools help developers find source code relevant to their maintenance tasks. One major challenge to successful search tools is locating relevant code when the user's query contains words with multiple meanings or words that occur frequently throughout the program. Traditional search techniques, which treat each word individually, are unable to distinguish relevant and irrelevant methods under these conditions. In this paper, we present a novel search technique that uses information such as the position of the query word and its semantic role to calculate relevance. Our evaluation shows that this approach is more consistently effective than three other state of the art search techniques.**

***Keywords*-code search; concern location; software maintenance**

I. Introduction and Background

Similar to how we use Google to search the web, software search tools match a developer's query with comments and identifiers in the source code to identify relevant program elements [1], [2], [3], [4]. Most software search tools treat a program as a "bag of words" [5] (i.e., words are treated as independent occurrences with no context or relations between them taken into account) and calculate relevance of each document (i.e., a method) by scoring the document based on the query and the word occurrences in the code.

Unfortunately, bag of words approaches can suffer from reduced accuracy by ignoring the relationships between words. For example, consider searching for the query "add item" in a shopping cart application. The presence of "add" and "item" in two separate statements of the same method does not necessarily indicate that the method is performing an "add item" action—the method may be *adding an action* to the system's queue and then *getting the item field* of another object in the system. Ignoring the relationships between words causes irrelevant results to be returned by the search, distracting the user from the relevant results. Thus, knowing how words occur together and where they occur can help distinguish between relevant and irrelevant search results. We use *phrasal concept* to describe a concept expressed as a sequence of words [6].

Some search technique go beyond bag of words to capture phrasal concepts. The Action-Oriented Identifier Graph (AOIG) captures phrasal concepts in the form of <verb, direct object> pairs from method signatures and comments to find actions that cross-cut object-oriented systems [4]. However, the AOIG cannot model all phrasal concepts or linguistic relationships. In this paper, we investigate using more detailed natural language information in phrasal concepts to increase code search effectiveness.

This paper describes how phrasal concepts (PCs) can be leveraged to improve software search, particularly by using the context and semantic role of query words within the method signature. Unlike AOIG and information retrieval-based approaches, our PC-based search technique introduces a relevance scoring mechanism that integrates information about the position of query words in the code as well as the semantic role of query words in phrasal concepts. We investigate the effect of our PC-based score on search by comparing it with 3 state of the art search tools.

II. A PC-based Score for Source Code Search

In this paper, we introduce a PC-based scoring function, pc, to score the relevance of program elements by integrating location, semantic role, head distance, and usage information:

- *Location.* When a method is well-named, its signature summarizes its intent, while the body implements it using a variety of words that may be unrelated. A query word in the signature is a stronger indicator of relevance than the body.
- *Semantic role.* Prior research has shown that using semantic roles such as action and theme can improve search effectiveness [4]. We extend this idea by distinguishing where query words occur in terms of additional semantic roles.
- *Head distance.* The closer a query word occurs to the head, or right-most, position of a phrase, the more strongly the phrase relates to the query word. For example, the phrase "image file" is more relevant to "saving a file" than "file server manager".
- *Usage.* If a query word frequently occurs throughout the rest of the program, it is not as good at discriminating between

relevant and irrelevant results. This idea is commonly used in information retrieval techniques [5].

A. Usage

Usage information captures how frequently a query word appears in the rest of the source code. We capture usage information for a word w using the common information retrieval measure of *inverse document frequency* (idf) [5]:

$$idf(w) = 1 - \frac{df(w)}{N} \tag{1}$$

where df is the total number of methods, or "documents", in which a query word appears, and N is the total number of methods in the system. Traditionally, IDF is calculated by taking the log of $N/df(w)$, with values ranging from $[0, log(N)]$. We modified the traditional IDF scoring function to make it a linear function from $[0, 1)$ to consistently limit the contribution of a single query word regardless of the number of methods in the program.

B. Head Distance

Head distance approximates how strongly a phrase (i.e., phrasal concept) relates to a query word. For example, consider the concept of "adding an auction" in an auction sniping program that allows users to place proxy bids on online auction sites such as eBay. The query to search for this concept would be "add auction". If the head distance of the query word is not taken into account, the following 3 methods are all considered to be equally relevant:

```
AuctionServerMgr.addAuctionServerMenus()
HTMLDump.addAuctionLink()
JBidMouse.addAuction(String auctionSrc)
```

Clearly, the last method, JBidMouse.addAuction, is more relevant than the first two methods. Specifically, addAuctionServerMenus() is not adding an auction but a *server menu*, and addAuctionLink is adding a link. In both cases, the query word is being used to modify the theme's head. Head distance allows us to differentiate between strong occurrences of query words in the head position, and less relevant occurrences to the left of the head.

Given a phrase p and a query word q:

$$head(q,p) = \begin{cases} 0, & \text{if } q \notin p \\ \frac{1}{1+min\_distance(q,p)}, & \text{if } q \in p \end{cases} \tag{2}$$

where $min\_distance$ is the minimum distance of the query word from the head position of the phrase. Equation 2 takes the inverse of the minimum distance $+1$ so that a query word in the head of a phrase has a maximum impact of 1, getting progressively smaller as the query word drifts farther from the head. If the query word does not appear, $head = 0$. We use the minimum distance of the query word from the head position because a single query word may occur multiple times in a single phrasal concept.

C. Semantic Role

Prior research has shown that using semantic roles such as action and theme can improve search effectiveness [4]. We extend this idea by scoring relevance based on 4 semantic roles: action (verb), theme (i.e., direct object), secondary arguments (indirect objects), and any auxiliary arguments. We capture the semantic roles for a given method signature using a novel Software Word Usage Model (SWUM) [7].

The binary relevance score used in prior work only looked for query words in the action or theme semantic roles [4]. We generalize beyond this idea by taking advantage of query words in secondary or auxiliary roles, and associating relevance weights with each semantic role.

Given a query Q and a method x, semantic role information is scored by function $swum$:

$$swum(x,Q) = \sum_{q \in Q} \Big(idf(q) * \max_i (\beta_i * head(q,i)) \Big) \tag{3}$$

where $i \in \{action(x),\ theme(x),\ secondaryArgs(x),\ auxArgs(x)\}$. Each semantic role is assigned a different weight, β_i, and we sum the maximum contribution for each query word. We ensure a query word can only contribute to the $swum$ score once by taking the maximum contribution for any semantic role's head distance score and weight.

As observed in our prior work with verb-direct object search [4], occurrences of query words in a method's action or theme are an important indicator of relevance. Thus, we give the action and theme the highest weights of $\beta_{action} = 1$ and $\beta_{theme} = 1$. Although secondary and auxiliary arguments are less likely to capture the main intent of a method, they can help differentiate between relevant and irrelevant results. For example, consider searching for "sort style" with methods getStyle and sortXMLByStyle. Based only on action and theme, getStyle would be as highly ranked as sortXMLByStyle. By taking secondary arguments into account, sortXMLByStyle would get the higher score. Thus, we define $\beta_{secondaryArgs} = 0.5$ for secondary arguments found in the name, and $\beta_{auxArgs} = 0.25$ for any remaining auxiliary arguments in the signature.

D. Location

Because a method's signature typically summarizes its intent, while the body may implement it using unrelated words, we score query word occurrences in signature and body locations differently. Given a query Q and a method x, we define our scoring function, $pc(x,Q)$ to be:

$$pc(x,Q) = signature(x,Q) + body(x,Q) \tag{4}$$

$$signature(x,Q) = \max(\beta_{swum} * swum(x,Q), \beta_{sig} * lex(x_{sig},Q)) \tag{5}$$

$$body(x,Q) = \beta_{body} * \frac{lex(x_{body},Q)}{|Q|} \tag{6}$$

In Equation 5, we take the maximum of our PC-based score of the signature ($swum$) and a best-effort lexical score (lex). In Equation 6, we use the same lexical score normalized by the number of query words. The body includes all words from the method's comments, identifiers, and literals.

To calculate lex in Equations 5 and 6, we first split and stem all the identifiers in the signature. Then, we lexically search for the query term or the stemmed query term. Similar to Equation 3, we sum the idf score of each lexically matched query word. Using a lexical search allows us to give non-zero scores to method signatures that have insufficient identifier splitting for extracting semantic role information.

Unlike traditional information retrieval techniques, which count the frequency of each query word's contribution, lex uses a binary score for each query word. In general, using frequency-based scores will bias the search in favor of long methods, since longer documents are more likely to contain more occurrences of the query words. If length normalization is used, the search will instead be biased in favor of very short methods. In contrast, our goal is to bias the search to methods containing the query terms, regardless of length. For example, consider a method related to sorting, populateTreeByStyle, which is 1500 lines long and contains just 1 relevant statement that calls the sort function. Later, the programmer decides to refactor the long populateTreeByStyle method into a number of smaller methods. The call to the sort function is now located within a short 5-line method, called getResults. We believe that both the original 1500 line method and the refactored 5 line method are equally relevant for containing a call to the sort function.

We tuned the pc scoring function using a training set of 5 manually-mapped search tasks. Based on our training sample, we define weights for the coefficients as $\beta_{swum} = 1$, $\beta_{sig} = 0.05$, and $\beta_{body} = 0.1$. We multiply lex by a very small β_{sig} coefficient to ensure properly parsed signatures with semantic role information are always ranked more highly than signatures only matched with lex. For example, using lex to search for the word "adds" with stem "add" would return irrelevant matches like "padding" in addition to relevant signatures like the unsplit "additem" or splittable "addsItem". In contrast, using lex ensures that when an identifier is consistently not split correctly, pc will still find relevant results (e.g., matching textfield with query "text field").

As shown in Equation 6, we also analyze body information with lex. We use a small coefficient ($\beta_{body} = 0.1$) to ensure body information is only used to break ties between similarly ranked methods. To keep the contribution of lex bounded to a maximum of β_{body} in Equation 6, we normalize lex by the number of query words. We investigated more sophisticated and expensive ways of using body information, such as calculating a PC-based body score, but results did not improve on our training data.

III. Evaluation

We evaluate our pc scoring function by comparing it with existing state of the art search techniques on 8 search tasks.

A. Design

In this study, we compare 5 search techniques: ELex, GES, FindConcept, and two variants of pc:

- *ELex* is a regular expression search similar to UNIX grep that returns an unranked list of methods, with no threshold.
- *GES* [2] is based on Google Desktop Search, takes a natural language query as input and returns a ranked list of methods. We select the top 10 results for this study.
- *FindConcept* [4] searches in method signatures and comments for verb and direct object pairs using a unique query mechanism that requires the user to enter a verb and a direct object. Results are ordered by structural connectedness to other results, and the top 10 methods returned.
- *PC10* is our pc score using the top 10 ranked results.
- *PCT* uses the same pc scoring function as PC10, but with a more sophisticated threshold that takes the average of the top 20 results to determine relevance.

a) Subjects: We use 8 of 9 search tasks used in a previous concern location study [4]. The tasks are from 4 programs ranging in size from 23 to 75 KLOC. Since the "add auction" task was used in pc's training set, it was excluded.

The queries for ELex, GES, and FindConcept come from a previous concern location study where each task-tool combination was replicated by 6 human subjects [4]. Because we are interested in investigating the differences between the techniques, and not how well the human subjects could use the tools, we selected the most effective queries for each technique, per task: the query that had the top precision, the top recall, and the top F measure. We followed a similar technique for PC10 and PCT, using the queries from a prior study [8]. Thus, the results in this study represent each search technique at its best, given a human user.

b) Measures: We measure effectiveness using the common information retrieval measures of precision, recall, and F measure [5], where *precision* (P) is the percent of search results that are relevant, *recall* (R) is the percent of all relevant results that were returned as search results, and the *F measure* (F) is high only when both precision and recall are similarly high. Because precision and recall are inversely related, a single query typically captures *either* high recall (by returning many results) *or* high precision (by returning few, but very relevant results) [5]. Thus, it is unusual for a single query to yield both high precision and high recall.

c) Threats to Validity: We tried to ensure a fair comparison across search techniques by using only the most effective queries for each task. The results may not generalize beyond Java, although we expect the results to be consistent across other object-oriented languages.

B. Results and Analysis

Figures 1, 2, and 3 show box and whisker plots capturing the overall Precision (P), Recall (R), and F Measure (F) results for the 5 search techniques. Based on the F Measure, ELex appears inferior to the other search techniques. The PC-based techniques, PC10 and PCT, appear to be more

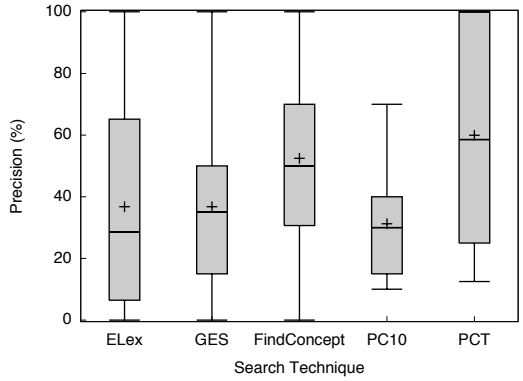

Fig. 1. Precision results for state of the art search techniques.

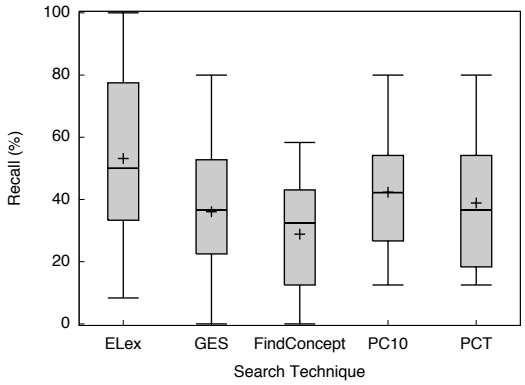

Fig. 2. Recall results for state of the art search techniques.

consistently effective than FindConcept or GES. These results are confirmed by the precision and recall results in Figures 1 and 2. In terms of precision, PCT is a clear front-runner closely followed by FindConcept. For recall, PC10, PCT, and GES appear to have similar results. We also analyzed results by search task. We found that PC outperforms the other search techniques for 3 tasks, GES outperforms the others for 3 tasks, FindConcept is the best for 1 task, and they all tie for 1.

For most queries in this study, ELex typically returns too many results. This ensures ELex finds many relevant results, but too many irrelevant ones. In Figure 2, PC10 and PCT begin to approach ELex's high recall, without sacrificing precision.

Overall, PCT is a very competitive search technique when the query words match relevant signatures. However, when body information is important to locating the relevant code, GES is the best state of the art technique in this study. Although GES outperformed PCT, PC10, and FindConcept for some of the tasks, its performance in general seems to be unpredictable. When GES did not have the best performance, it tended to be little better, and sometimes even worse, than ELex. In contrast, even though PCT did not always have the best results, it was usually competitive.

To investigate this observation, we ranked the approaches from 1–5 based on their maximum F Measure score for each task, giving ties the same rank. Using this measure, PCT is the most highly ranked technique with an average rank of 2.38 and

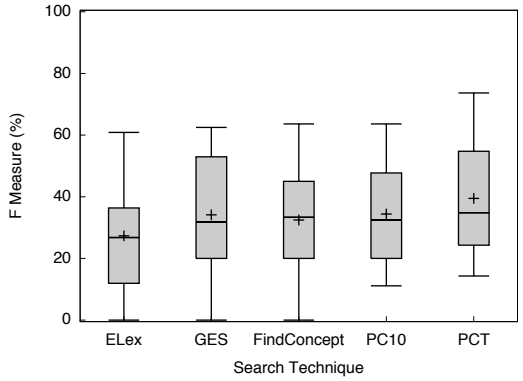

Fig. 3. F Measure results for state of the art search techniques.

a standard deviation (std) of 1.18. GES has an average of 2.75 (std 1.19), PC10 an average of 2.88 (std 1.64), FindConcept an average of 3.00 (std 0.93), and ELex and average of 3.50 (std 1.41). From these results we can see that PCT and GES are the best overall techniques in this study, with PCT consistently ranked more highly overall.

IV. Conclusion

In this paper, we presented a novel scoring function (PCT) for source code that weights query words based on their location, semantic role, head distance, and usage information. We compared our PCT score with three state of the art search techniques. Our results show that PCT and GES are the best overall techniques in the study, with PCT consistently ranked more highly overall. GES typically performed best when body information was important to locating the relevant code, whereas PCT performed best when signature information was more useful. In future, we plan to compare with additional search techniques on more search tasks, as well as explore the role of signature vs. body information in locating source code.

Acknowledgments

This material is based upon work supported by National Science Foundation Grant Nos. CCF-0702401 and CCF-0915803.

References

[1] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *Proc. Work. Conf. Rev. Eng.*, 2004.

[2] D. Poshyvanyk, M. Petrenko, A. Marcus, X. Xie, and D. Liu, "Source code exploration with Google," in *Proc. Int'l Conf. Soft. Maint.*, 2006.

[3] D. Poshyvanyk and A. Marcus, "Combining formal concept analysis with information retrieval for concept location in source code," in *Proc. Int'l Conf. Prog. Comp.*, 2007.

[4] D. Shepherd, Z. P. Fry, E. Hill, L. Pollock, and K. Vijay-Shanker, "Using natural language program analysis to locate and understand action-oriented concerns," in *Proc. Int'l Conf. Aspect-Oriented Soft. Dev.*, 2007.

[5] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY: Cambridge University Press, 2008.

[6] R. Jackendoff, *Semantic Structures*. Cambridge, MA: MIT Press, 1990.

[7] E. Hill, "Integrating natural language and program structure information to improve software search and exploration," Ph.D. dissertation, University of Delaware, Aug. 2010.

[8] E. Hill, L. Pollock, and K. Vijay-Shanker, "Automatically capturing source code context of NL-queries for software maintenance and reuse," in *Proc. Int'l Conf. Soft. Eng.*, 2009.