Available online at: https://ijact.in

Date of Submission	19/01/2019
Date of Acceptance	14/02/2019
Date of Publication	28/02/2019
Page numbers	3047-3052 (6 Pages)

ISSN:2320-0790

COMPUSOFT
An International Journal of Advanced Computer Technology

REFORMULATION OF NATURAL LANGUAGE QUERIES ON SOURCE CODE BASE USING NLP TECHNIQUES

Swathi B.P[1], Anju R[2]

Department of I&CT,
Manipal Institute of Technology,
Manipal Academy of Higher Education,
swathi.bp@manipal.edu, anju.r@manipal.edu

Abstract: Source code retrieval is a branch of text retrieval which helps developer find a piece of code from the code base. The developer can obtain the required code from the code base by issuing a query on the source code base. Generally, a developer who has been working on the code base since a long time will know how to formulate his/her query in order to get a good search result. A developer who is novice to the code base will not know what terms he/she has to include in query to obtain a good search result. In fact, a system should allow developer to issue natural language queries. This arises a need for query reformulation to optimize the developer query when the query does not contain terms from code base. This work has conducted extensive study on areas where natural language queries are applied and the various reformulation techniques. In this work, semantic query reformulation technique is applied on the natural language queries on the source code base. Our discussion and results prove how semantically right word and a word which is in context of the source code can be obtained which acts as a replacement for a query term which is not present in the source code base.

Keywords: Natural Language Processing, Query reformulation, Similarity score, Query Expansion, Natural Language Queries.

I. INTRODUCTION

Information retrieval (IR) is a task of retrieving relevant documents from a large collection of documents. One of the branches of Information Retrieval is Text Retrieval in which information is retrieved in the form of text. Retrieval takes place only when query matches against the documents, and the documents which are returned have very close relevance with the terms in the query. Source code retrieval is an application of text retrieval where information retrieved is source code. Source code retrieval is significantly necessary for developers in software industry to find methods or classes during bug fixing [18][19]. A modification in source code is not only performed by software developers but by any individual who would like to work on open source projects. Generally, the open source project does not consist of software artifacts. As a result, a developer who is a novice to the code base, will have difficult time in finding out the class or method where he has to do modify the code. At present, the integrated development environments (IDE) allow a developer to search for a piece of code/ method/ class using exact term that is present in the source code base. The search may be slightly advanced using a regular expression in the query which should still contain a base term from the corpus. In Information Retrieval (IR), one of the fundamental obstacles faced by users is that communicating their informational requirement in a query which a IR engine comprehends. Apart from the query formulation that system understands, the critical problem lies in deciding the set of words or terms in the query that express users need semantically. Defining the former problem in source code retrieval, a developer who is completely accustomed with the source code base will know identifier names used in the code base. But, a developer who is new to the code base is unaware of class names, method name or any other variable names. Therefore, often as an end user of the IDE, developer would like to have a system which allows for natural language querying on the corpus. The convenience with natural language querying is that the developer need not really remember exact terms such as identifier names from the source code base [1]. This leads to formulation of query which does not have terms from code base. At the same time when the query term does not match the corpus, the search result will be poor. To overcome this poor search result, this work focuses on application of semantic reformulation techniques to optimize the developer query. Studying the various reformulation techniques and concluding that application of semantic reformulation technique on natural language queries on source code base can help developer to obtain a better search result even in the situations when they are not acquainted with source code base.

1.1 Application of Natural Language Queries

It is becoming increasingly important for systems in various fields to inculcate natural language querying which intents at making search easier for end users. The research works on natural language querying in various fields have been summarized as follows: Massai et al. [2] have developed an engine for semantic assistance which suggests local points of interest (POIs) and services by exploiting users' queries written in natural language. It estimates the user information need in terms of geographic references. The system adopts NLP and semantic techniques to provide output recommendations on POIs and services which best match the users' requests. Hu et al [3] have proposed a natural language querying mechanism that processes general aggregate queries over Resource Description Framework (RDF) data. Generally a user's query consists of semantic relations and aggregations in them. Hence, a framework called NLAQ (Natural Language Aggregate Query) has been proposed which employs a novel algorithm to automatically understand intention of a user's query.

Rencis et. Al [4] in their work has outlined a method as to how those clinical information systems could be handled by the domain experts themselves with the help of natural language-based query language. The framework defined works upon data stored in the ontology. Their investigation prove that the suggested approach is certainly easy to be used by end-users such as managers and physicians of hospitals as natural language query language is very intuitive to use. Esther et al. [5] have worked to provide 4 different Natural language interfaces (NLIs) that provides a user friendly means of access to Semantic Web data for casual end-users through their natural language query. Salaiwarakul et al [6] have proposed cultural tourism ontology for Thailand which allows users to access tourism information using natural language queries in the Thai language. This work shows how capacity of natural language querying visibly moved the serious limitations of search based on keyword.

Lin et al. [7] have worked for developing a software called TiQi in which users can issue queries related to software artifacts verbally or written in natural language. It receives Natural Language (NL) queries, converts those queries into SQL, and then executes the queries against database. Jamil et al. [7] in their work, have proposed a knowledge-based middleware which is multilevel in nature. This facilitates intent preserving and semantics mapping of a natural language query. We develop abstractions that are multi-level with a concept reasoner and information retrieval engine to dynamically link arbitrary natural language querying to structured queries. Tamas et al. [16] have discussed how natural language can help user query on e-commerce application or ERP systems. The authors have developed a framework names Artemis which accepts query inputs and converts into filters applicable to underlying data models. Castillo-Ortega et al [17] have worked on natural language querying on multidimensional databases. They have come up with a new tool, Linguistic F-Cube Factory which is based on natural language querying on a multidimensional data model to obtain linguistic results.

1.2 Query reformulation

In information retrieval, Query reformulation is a process of reframing a query to reduce the search result mismatch. Query reformulation is otherwise a query optimization technique that provides the best suitable results [9]. The need for reformulation arises when the terms in the query does not match the document collection. Since, there is more than one syntactic arrangement that communicates the same piece of information; natural language processing plays a significant role in query reformulation because after the application of reformulation techniques such as query expansion / reduction, reengagement of query terms or query modification, the query should still mean the same. Reformulation of query can be done manually based on the search results from the initial query. The knowledge and experience about how search engines work also help in query reformulation [11]. Manual query reformulation is an overhead for the information seeker because he has to go through the search result obtained in the first place.

Query expansion is a reformulation technique in which new terms are added to the existing query in order to get more relevant results [10]. A dictionary or general thesaurus could be used in this process. Lioma et al [13] have worked on reformulation of queries using syntactic based reformulation techniques. They also compared their experimental results with pseudo –relevance feedback through which it proved that syntactic based reformulation performs much better than pseudo-relevance feedback.

Semantic query reformulation is another reformulation technique which becomes highly advantageous when there is more than one way of conveying a matter and also makes the user query semantically enriched. It is a technique in which query terms are replaced with the some other terms which semantically mean the same, using ontological support. Angel et al [12] have worked on query reformulation using similarity thesaurus. The objective of query reformulation is to use the underlying knowledge of the database to reformulate a query into a less expensive yet equivalent query. Amshakala et al [9] have exploited WordNet Ontology for query reformulation and have also, optimized the query by eliminating disjunctive clause. Although, adoption of general thesaurus to formulate the user's need does not yield good results [20].

Bissan et al [14] combined Semantic query reformulation technique with query expansion technique to produce a new technique called Semantic Mixed query Expansion and Reformulation Approach (SMERA) that uses these two types of concepts to improve web queries. The choice of terms to be used as expansion in queries is made very intelligently. The approach provides a statistically significant improvement in precision over a competitive query expansion method. Kun Lu et.al [15] in their study have found out the effects of contextual factors and system features on query reformulation. They have also studied the relationship between types of query reformulation technique and search performance. Both the analysis have been given a solution in multilevel modeling which is a single research model. The results disclosed the facts that query reformulation techniques are influenced by system features and users' educational background Also, types of query reformulation had a significant impact on search performance.

Of all the reformulation techniques that we have gone through from various researchers, natural language querying acted upon by sematic query reformulation is the most helpful technique because , a user need not have knowledge about the terms in the database. But, the main speck that we have observed from the earlier works on semantic query reformulation is replacement of semantically right term in the place of that term which is not found in the database. In our work, two developers may refer to the same piece of code with different terms which semantically mean the same.

Therefore, Objective of this work is to find a semantically right term which is context of the source code base as a replacement for the query term which is not present in the source code base. To achieve this, we have built a thesauri from the source code base on which search is launched and similarity thesaurus. The thesaurus built acts as dictionary and query reformulation is performed using this dictionary. To build the thesaurus from source code base, we have considered benchmark open source project as input to our model. The benchmark open source projects are considered to have documentation (comments) written by the developers of the project. A mapping from words present in the comments of the source code to words in the similarity thesaurus is performed which creates a dictionary of words along with their synonyms. A word can be a synonym to many other words. For example, the word 'show' is a synonym to the word 'display' as well as 'exhibit'. But, in actual the word 'show' has to be replaced by 'display', because the 'display' is the word that is present in the source code base. Now this decision needs the context information, which we have obtained through NLP techniques. A similarity metric from NLP which is one of the pre-retrieval metrics has been employed to obtain the similarity between the term and the source code. The comments obtained from the source code are stored in an unstructured data base for further processing. The following sections furnishes in detail about the methodology followed and the result discussions.

II. METHODOLOGY

The proposed framework is mainly divides into four parts namely, retrieval of comments from the source code base, application of natural language processing on comments and developer query, creation of dictionary to map words from comments to words in similarity thesaurus and query reformulation through similarity score computation. In Fig. 2.1, the input to the model is a source code base containing comments. The comments are retrieved based on the regular expression match performed and all the comments fetched are stored in an unstructured database. During the implementation, we have used MongoDB since the comments are unstructured in nature. This phase of comment retrieval followed by the application of natural language processing on comments as well as developer query. We have applied natural language processing techniques such as stemming, lemmatization, stop words removal. After applying NLP techniques on query we obtained a list of query terms which are maintained in a list called Q_L. T_i represents terms in the query. C_iW_j represents words from each comment. S_i represents synonym for each C_iW_j. Similarly the processed comments are maintained in a list C_L. with the help of similarity thesaurus, a dictionary D has been built to store the synonyms of the words which are present in the comments. The dictionary consists of words from comments as key and the synonyms as values. If a term in the query is already present in the code base, then there is no need of replacing such term. Otherwise, find such term in the synonyms that are nothing but values. Next, find the similarity of those keys for which there is a match in the synonym. Replace the query term with that key which has the highest similarity. The psuedocode for aforementioned is sketched in Fig. 2.3 Fig. 2.2 outlines about finding the similarity score of the entire query with respect to the code base. If the similarity score of the entire query is very low, then it is a clear indication that the query needs more terms which are present in the code base in order to obtain a better search result.

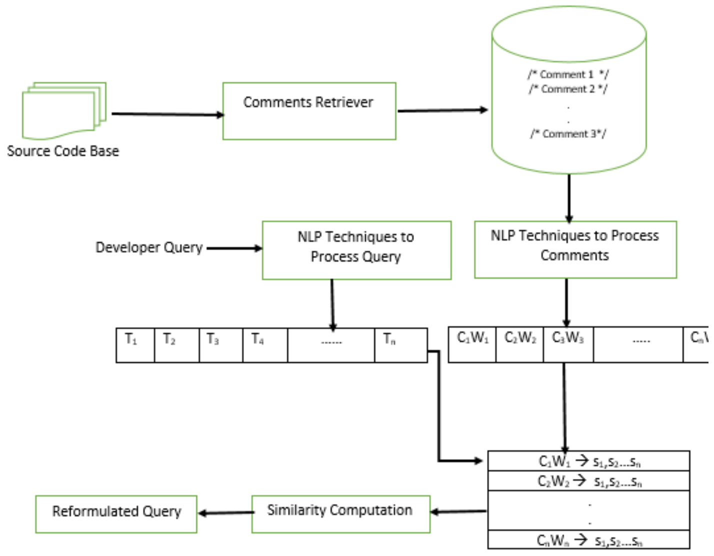

Source Code Base
Comments Retriever
/* Comment 1 */
/* Comment 2 */
/* Comment 3*/
Developer Query
NLP Techniques to Process Query
NLP Techniques to Process Comments
T_1 T_2 T_3 T_4 ...... T_n
C_1W_1 C_2W_2 C_3W_3 ..... C_nW_n
$C_1W_1 \rightarrow S_1,S_2...S_n$
$C_2W_2 \rightarrow S_1,S_2...S_n$
$C_nW_n \rightarrow S_1,S_2...S_n$
Reformulated Query
Similarity Computation

Fig. 2.1 Framework for semantic query reformulation

The similarity score of the term with respect to the code base is obtained by making use of a metric Similarity collection Query (SCQ) from natural language processing [1]. The similarity score of the term with respect to the document is obtained using (1). This focuses on similarity of entire query with source code base.

$$SCQ(t) = (1 + \log(ictf(t,D)).idf(t) \qquad (1)$$

$idf(t) = \log\left(\frac{\|D\|}{\|D_t\|}\right)$	(2)
$\boldsymbol{ictf(t) = \log\left(\frac{\|D\|}{tf(t,D)}\right)}$	(3)
where D: collection of documents, t: key in the key-value pair where the synonym for the term Q is found, D_t :collection of documents containing the term t, *tf(t,D):* frequency of term t in all the documents	

Collection-query similarity (SCQ) over all query terms can be computed using $\sum Q \in$Query SCQ(Q).

III. Results and Discussions

This section briefs about the various ways in which a developer can query on a source code base and how query reformulation is performed by computing similarity score. To demonstrate the cases, we have considered few comments from vlc-media player.

Examples of Comments from vlc-media player:
Unclosed network connection
Return socket handle
Accepts an new connection on a set of listening sockets

Example of Dictionary D created from the model:
Unclosed – open, unlatched, unshut
Network - net, chain, circuitry
Connections – contact, network, relation, association
Return- rebound, entry, arrival, restores
Recovery- rebound, restores, entry, arrival

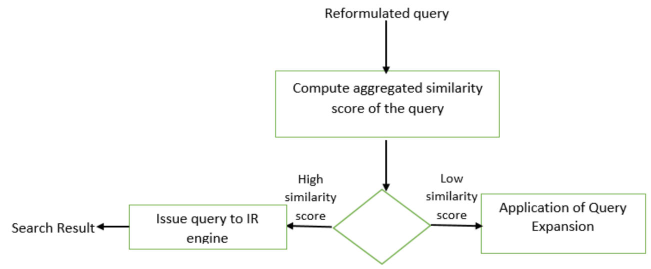
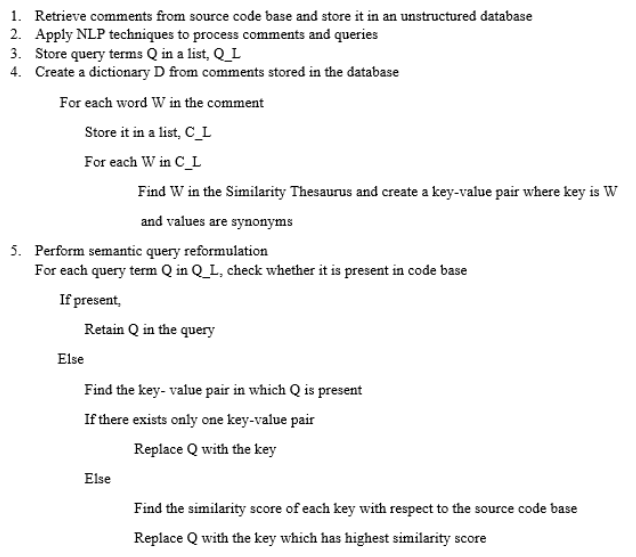


Fig. 2.2. Framework for computation of similarity score of developer query

Case 1: A developer issues a query “unclosed connection”. Since all query terms are present in the code base, reformulation is not required and query is issued to IR engine as it is.

Case 2: A developer issues a query ‘open connection”. Since the word ‘open’ is not present in the code base, it needs to be replaced with the term which is present in the codebase. From the aforementioned thesaurus example, we find that ‘open’ is the synonym for unclosed. There is no ambiguity about the replacement because ‘open’ is the synonym for only one key, i.e ‘unclosed’. Similarity of ‘unclosed’ with source code base is 4.6.

Case 3: A developer issues a query ‘rebound socket handler’. Since ‘rebound’ is a synonym for two keys i.e ‘ return’ and ‘recovery’, we find from similarity score computation that SCQ(return)= 3.2 and SCQ(recovery)=1.2. The low value of SCQ for recovery is because ‘recovery’ is present very few documents. The similarity score of entire query ‘return socket handler’ and ‘recovery socket handler’ are 5.5 and 3.5 respectively.



```
1. Retrieve comments from source code base and store it in an unstructured database
2. Apply NLP techniques to process comments and queries
3. Store query terms Q in a list, Q_L
4. Create a dictionary D from comments stored in the database
     For each word W in the comment
          Store it in a list, C_L
          For each W in C_L
               Find W in the Similarity Thesaurus and create a key-value pair where key is W
               and values are synonyms
5. Perform semantic query reformulation
   For each query term Q in Q_L, check whether it is present in code base
     If present,
          Retain Q in the query
     Else
          Find the key- value pair in which Q is present
          If there exists only one key-value pair
               Replace Q with the key
          Else
               Find the similarity score of each key with respect to the source code base
               Replace Q with the key which has highest similarity score
```

Fig. 2.3 Psuedocode for semantic query reformulation

IV. Conclusion

The results obtained from semantic query reformulation technique applied on natural language queries of the developer legitimates that developer need not remember the exact terms from code base. It also shows that it is inadequate if a query term is directly replaced with a synonym. In addition to finding the similarity, it is necessary to find whether the synonym belongs to the context of the corpus. Our future work focuses on semantic query reformulation along with query expansion. Now, this is necessary if a developer does not provide in his query sufficient terms. Inclusion of extra terms along with semantic reformulation on developer query hopes to improve the search result significantly.

V. References

[1] S. Haiduc, G. Bavota, R. Oliveto, A. D Lucia, and Marcus, Automatic query performance assessment during the retrieval of software artifacts, Proceedings of the 27th IEEE/ACM international conference on Automated Software Engineering, pp. 90-99, 2012.

[2] Massai, Lorenzo and Nesi, Paolo and Pantaleo, Gianni (2019), PAVAL: A location-aware virtual personal assistant for retrieving geolocated points of interest and location-based services, Engineering Applications of Artificial Intelligence, Vol. 77, Elsevier, pp 70-85.

[3] Xin Hu and Yingting Yao and Luting Ye and Depeng Dang (2017), Natural Language Aggregate Query over {RDF}, Information Sciences, abs/1710.07891.

[4] Rencis, Edgars(2018), Towards a Natural Language-based Interface for Querying Hospital Data, Proceedings of 2018 International Conference on Big Data Technologies ICBDT '18, China, pp 25-28

[5] Kaufmann, Esther & Bernstein, Abraham. (2010), Evaluating the Usability of Natural Language Query Languages and Interfaces to Semantic Web Knowledge Bases, SSRN Electronic Journal.

[6] Salaiwarakul, A. (2018), Thai natural language based cultural tourism ontology. ICIC Express Letters. 12. 159-165.

[7] J. Lin et al.(2017), TiQi: A natural language interface for querying software project data, 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), Urbana, IL, pp. 973-977.

[8] Hasan M. Jamil(2017), Knowledge Rich Natural Language Queries over Structured Biological Databases, Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, pp 352-361.

[9] Ka, Amshakala and Nedunchezhian. (2011). WordNet Ontolog Based Query Reformulation and Optimization using Disjunctive Clause Elimination. International Journal of Database Management Systems. 3. 55-63.

[10] Washio, Takashi, and Luo, Jun (2013), Applying NLP Techniques for Query Reformulation to Information Retrieval with Geographical References, Emerging Trends in Knowledge Discovery and Data Mining, Springer Berlin Heidelberg, pp 57-69.

[11] Jeff Huang and Efthimis N. Efthimiadis (2009), Analyzing and evaluating query reformulation strategies in web search logs, Proceedings of the 18th ACM conference on Information and knowledge management (CIKM '09), ACM, 77-86.

[12] Ángel F. Zazo, Carlos G. Figuerola, José L. Alonso Berrocal, and Emilio Rodrıguez (2005), Reformulation of queries using similarity thesauri, Information Processing and Management: an International Journal, Vol. 41, pp. 1163-1173.

[13] C. Lioma and I. Ounis. (2008), A syntactically-based query reformulation technique for information retrieval, Information Processing and Management: an International Journal, Vol. 44, pp. 143-162.

[14] Audeh B., Beaune P., Beigbeder M. (2017) SMERA: Semantic Mixed Approach for Web Query Expansion and Reformulation, Advances in Knowledge Discovery and Management, Studies in Computational Intelligence, Vol 665. Springer, Cham.

[15] Kun Lu, Soohyung Joo, Taehun Lee, and Rong Hu. (2017), Factors that influence query reformulations and search performance in health information retrieval: A multilevel modeling approach, Journal of the Association for Information Science and Technology, Vol. 68, pp 1886-1898.

[16] Tamas, I., & Salomie, I. (2016), Artemis-an extensible natural language framework for data querying and manipulation, Intelligent Computer Communication and Processing (ICCP), 2016 IEEE 12th International Conference, pp. 85-91.

[17] Castillo-Ortega, R & Marín, Nicolás & Sánchez, Daniel & Molina, Carlos (2013), Flexible Querying with Linguistic F-Cube Factory, pp 245-256.

[18] Mills, C., Bavota, G., Haiduc, S., Oliveto, R., Marcus, A. and Lucia, A.D., (2017), Predicting query quality for applications of text retrieval to software engineering tasks, ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 26, pp. 3.

[19] Renuka Sindhgatta (2006), Using an information retrieval system to retrieve source code samples, Proceedings of the 28th international conference on Software engineering (ICSE '06), pp 905-908.

[20] Ellen M. Voorhees and Donna K. Harman (2005), TREC: Experiment and Evaluation in Information Retrieval, Digital Libraries and Electronic Publishing, The MIT Press.