Journal of Computer Science 8 (1): 61-67, 2012
ISSN 1549-3636

First Token Algorithm for Searching Compound Terms Using Thesaurus Database

[1]Yousef Abuzir and [2]Thabit Sabbah
[1]Department of Computer Information Systems,
Al Quds Open University, Salfeet, Palestine,
[2]ICTC Center, QOU, Ramallah, Palestine

Abstract: Problem statement: Searching text materials is the one of the most important operations that carried out by search engines either on web or desktop applications, searching algorithms are required sometimes to find a specific word into a text, others to find a multi word term (pattern matching) into a text. Searching for term into a thesaurus database can be carried out using many searching algorithm such as brute-force algorithm and others. **Approach:** We addressed several issues concerning developing a searching algorithm that search terms into thesaurus database. Two exact algorithms were discussed and compared. The first algorithm, brute-force algorithm and the second one were proposed by this study to enhance brute-force algorithm. **Results:** We proposed an efficient search algorithm and compare it with brute force technique. Computational results showed that our algorithm can provide an efficient search algorithm that reduces the number of queries and the total time required to finish the required task. **Conclusion:** Our study showed an optimum solution for larger size of the studied problem with much less processing time than the brute-force algorithm. The modified algorithm has a higher efficiency to deal with Thesaurus Database searching problems.

Key words: Brute-force, pattern matching, information retrieval, compound terms searching, First Token (FT), thesaurus database, training thesaurus

INTRODUCTION

Searching is the basic process in Information Retrieval (IR) science. Documents, data within documents, relational databases, Schemas and WWW are the main sources where information can be retrieved. Searching information needs a search engines types and different data sources. Text search engines are the most common search engines type, Full-text search is the process of examining all of the words in a computer-stored document(s) or database to match search words supplied by the user. Full-text searching techniques become widely common and supported in either web applications or desktop application programs. Text search is applicable in e-business, human resources departments and others. Also, it is a basic supported feature in any word processing application such as Microsoft Word or database engines like Oracle (Doug *et al*., 2011), MySQL and SqlServer.

A Thesaurus is a list of very important term (single-word or multi-word) in a given domain of knowledge and a set of related terms for each term in the list. It is used for indexing, classifying, searching and text mining. Terms in thesaurus are listed alphabetically and some are hierarchically, this hierarchically indicates the relation between terms, the broader term "BT" represent the super class of the term while the narrower term "NT" represents the subclass (es) of the term. Some thesauri have the USE and Used For (UF) relations to indicate the alternation of terms (Robert, 2006; Abuzir, 2010).

Searching about text into a thesaurus database or any other data sources require the traversing of each term or compound term of the text. Our objective is to introduce an efficient search algorithm within the thesaurus database; this search algorithm can be used in either indexing or information retrieval applications. The next Sections are an overview over the problems we address, Brute-force algorithm and our enhanced algorithm First Token (FT) Algorithm. Finally, discussion, results and conclusion are presented in the last sections.

Background: Searching for text in database or any other data source based on string searching algorithms. These Algorithms check the existence and the location

Corresponding Author: Yousef Abuzir, Department of Computer Information Systems, Al-Quds Open University (QOU), P.O.Box 51800, PostalCode 97917, Jerusalem, Palestine

of a substring (also called pattern) into another string (Lin, 2009; Chen et al., 2011; Sleit et al., 2009).

Many algorithms of string matching were introduced as an enhancement of the simplest string matching algorithm. The Naïve search (brute-force) is the simplest and the less efficient algorithm among string matching algorithms (Lokman and Zain, 2010). Brute-force algorithm is simple to implement, need no preprocessing of text and always find the result if it is exists. It based on making a comparison at each and every possible point while sliding the window of search (Christian and Robert, 2000).

Knuth-Morris-Pratt (KMP) and Boyer-Moore (BM) algorithms (Lin, 2009) are the commonly used algorithms in string matching. Both are similar in idea used, time complexity and both don't perform complicated arithmetic on characters. BM algorithm is more complicated than KMP but it is a little faster in practice. Finite State Machine (FSM) (Cormen, 2001) was introduced as a base for string matching algorithm, this algorithm firstly builds a state table then simulate it on the input text. The bitap algorithm (Shift-or, shift-and or Baeza-Yates-Gonnet algorithm) is a fuzzy string matching algorithm, this algorithm adapts easily to approximate string matching and uses the bitwise techniques, it is efficient if the pattern length is no longer than the memory-word size of the machine (Manber and Wu, 1992).

Benjamin et al., (2006) described XTM system which has the ability to search for text that matches a set of rules or patterns "regular-expression", like social-security numbers, email addresses, phone numbers. This regular-expression matching can be performed concurrently for up to 50 rules. In recent years keyword search over semi structured and structured data has been extensively studied by Fredriksson (2010); Al-mazroi and Rashid, 2011); Alajlan et al. (2009. Other researchers Agrawal et al. (2002); He et al. (2007); Carmel et al. (2003); Vu et al. (2008); treated keyword search in databases as a graph. These approaches are computationally expensive. (Al-mazroi and Rashid, 2011) proposes the combination of two algorithms namely Berry-Ravindran and Skip Search Algorithms to form a hybrid algorithm in order to boost search performance.

MATERIALS AND METHODS

Brute-force algorithm (Lin, 2009) is simple to implement no need of preprocessing of text and always find the result if it is exists. However this technique is proportionally cost growth to the problem size growth, for example consider the problem of finding the number of occurrences of each word within a document that are exists in a database field which is one word term, the brute force technique will traverse all tokens (t) and query the database to check the existence, the total number of queries in this case is (t) times. Suppose that the terms in database field are of length (l-1) tokens, that is mean we can form a compound terms of length (l). The total numbers of queries to search for the compound terms can be calculate by Eq. 1:

$$\sum_{n=1}^{t} n - \sum_{n=1}^{t-l} n \quad (1)$$

The following Fig. 1-3 explains the growth rate in the number of queries with respect to text size and the maximum count of tokens in the database field.

To explain the previous formula and graphs, consider the following text. "Information Retrieval (IR) is the science of searching for documents, or information within document as well as that of searching relational databases and the World Wide Web". Also consider the following list of terms:

Id	Term	Tokens count
1	Information retrieval	2
2	IR	1
3	World Wide Web	3
4	Information technology standards	3

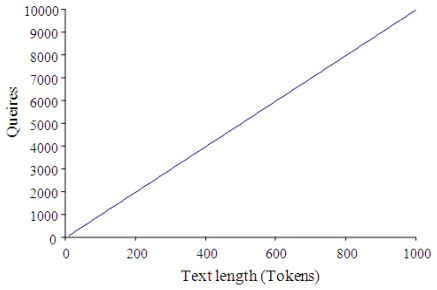

Fig. 1: Growth rate in number of queries related to growth of text size with constant count of tokens in database field (10 tokens)

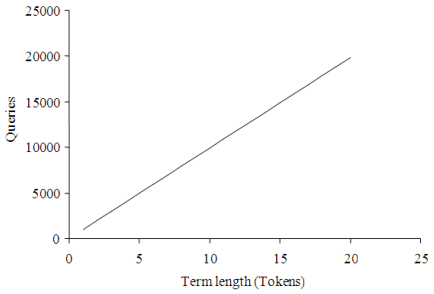

Fig. 2: Growth rate in queries related to growth count of tokens in database field with constant text size (1000 tokens)

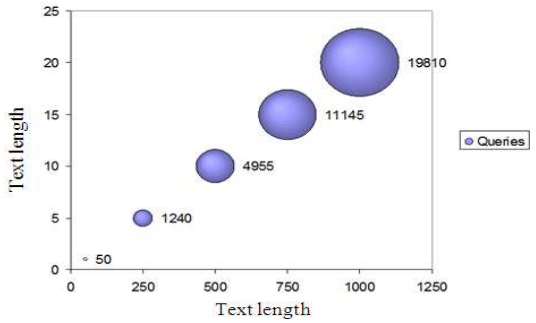

Fig. 3: Growth rate in queries related to growth of both: count of tokens in database field and text size. The volume of bubble in graph represents the number of queries

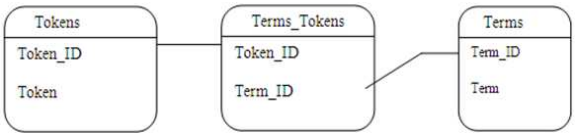

Fig. 4: ER diagram shows the relation between the table

In our first test, we used our sample text about information retrieval. To search the text using brute force algorithm, the text should be traversed 3 times which is the maximum number of tokens in the list. In the first phase the algorithm will search for a single term (token), each word in the text will be used to query the database. In this case, the number of queries equal the number of tokens count in the term. In second phase, a compound term of two words will be considered as one term and this term will be used to query the database. So, the first term in our example will be "Information retrieval "while the second one will be "retrieval IR" and so on, this will yield a (t-1) terms in this round. Third phase will use a term consist of three tokens, starting from the term "Information retrieval IR" and the last term will be "World Wide Web". The number of queries in this round is (t-2). The total number of queries in all the three phases in our example can be calculated using the following Eq. 2:

$$(t + (t-1) + (t-2)) \quad (2)$$

In general, the total number of queries of text consists of (t) tokens and (l) the maximum tokens count in terms in the database is Eq. 3:

$$t + t-1 + t-2 + t-3 + \ldots + (t-l+2) + (t-l+1) + t-l \quad (3)$$

Consider the following series:

$$1 + 2 + 3 + \ldots + (t-l) + (t-l+1) + (t-l+2) + \ldots + t-3 + t-2 + t-1 + t = \sum_{n=1}^{t} n$$

As a result the total number of queries can be expressed by the following formula (4):

$$\sum_{n=1}^{t} n - \sum_{n=1}^{t-l} n \quad (4)$$

From Eq. 4 and based on our text sample we can calculate the total number of queries for Brute-force algorithm. The text contains 14 tokens (t) (tokens are in Bold, the rest are stop words and will be ignored by the system) and the maximum number of Token Count (l) in a term is 3. We can find that the total number of queries is equal to 39 queries:

$$\sum_{n=1}^{14} n - \sum_{n=1}^{14-3} n = \frac{14(14+1)}{2} + \frac{11(11+1)}{2} \quad 105 - 66 = 39$$

The proposed approach-First Token (FT): In this study we proposed an enhanced algorithm to Brute-force algorithm called (FT).

Our study based on the existing approach and the analysis of the effectiveness of different sources on the total number of queries and on the total time. We described the structure of the databases and explained how our approach reduced the number of queries and the total time required to finish the required task.

Database structure: The proposed enhancement depend on creating two other tables related to the main list of terms in the databases, the first one will contain a list of first token of each terms, while the other will contain the Id of terms that begins with specified Token. The following Entity-Relation diagram E-R Diagram (Fig. 4) illustrates the relations between tables.

Table-1 shows an instance of the database from our sample example. The flowchart in Fig. 5 explains how to use the E-R Diagram in Fig. 4 and Table 1.

The proposed algorithm: Fig. 5 shows the main steps in searching for a term in the databases. The process of searching text terms in the database can be performed by traversing the text tokens for one time. In this phase each token of the text will be used to query the table of (Tokens) from the new model.

If the system returns a (TokenID) from Tokens table, this means that two extra queries are needed, the first one is querying the Terms_Tokens table, to get all (TermIDs) that begins with the specified (TokenID). The second one is querying Terms table to get a Temporary List of (Tokens count) for that term Id (TermIDs) and the list of the terms in the thesaurus database (Terms List). The (Tokens count) of terms used to determine the length of the compound term that our system can extract from text collection.

Table 1: The new relationships

Token		Terms-Token		Terms		
Token ID	Token	Token ID	Term ID	Terms ID	Term	Token count
1	Information	1	1	1	Information retrieval	2
2	IR	1	4	2	IR	1
3	World	2	2	3	World Wide Web	3
		3	3	4	Information technology standards	3

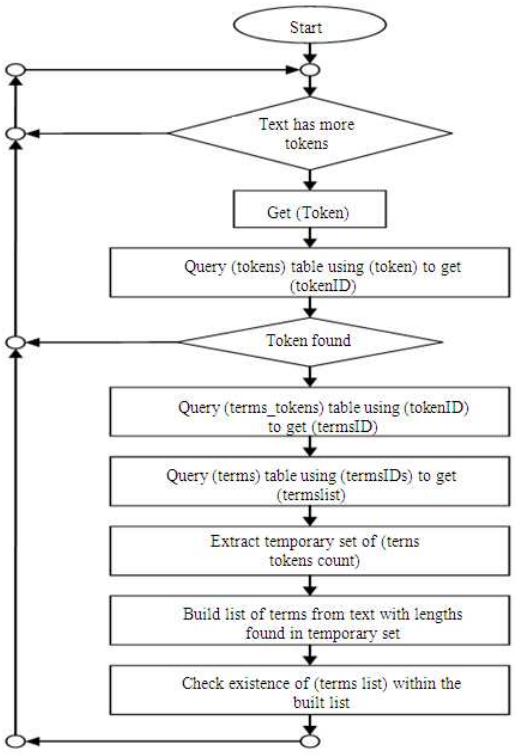

Fig. 5: Searching for a term in the databases model

The system parsed our sample text collections using the tokens counts and constructs a list of compound terms (Build List) start with the term in query. Finally, the system use the list of the terms returned by our query (list of thesaurus terms from the database) to search for the occurrence of these terms in the compound terms extracted and build by our system from the text collection.

This model automated and restricted the construction process of the compound terms from the text collection. It is clear how long is the compound term and the starting term.

Back to our example, starting with token "Information", we query the Tokens Table, this gives us the (TokenID = 1), meaning that we need to perform two extra queries, first we use (TokenID = 1) to query Terms_Tokens Table, resulting the following list of TokenIDs and TermIDs:

TokenID	TermID
1	1
1	4

Now, we query Terms Table for TermIDs 1 and 4. The result of this query will contain the terms and its (TokensCount) as follows:

TermID	Term	Tokens Count
1	Information retrieval	2
4	Information technology standards	3

Based on the previous result, the system build terms from text collection starting with the current token and length of 2 and 3 tokens, the built list will be as follows:

Built terms:
Information retrieval
Information Retrieval (IR)

The final step is to check the existence of terms from query result table within the Built Terms list. Numerically, our example need to make 13 queries to the Tokens table, with 3 extra 2 queries when process the tokens "Information","IR","World". While the brute force technique need to perform 39 queries. The following pseudo code listing of the proposed algorithm illustrates the proposed enhancement approach.

First Token Algorithm: Searching terms into thesaurus database using First Token (proposed name of method) technique:

```
For I = 1 to text.tokensCount
        For j = 1 to tokens.count
If tokens[j] = Text[i]
        L = list of distinct # of tokens for terms starts with Text[i]
        For each length in L
                TempTerm = buildTermfromtextoflength(length)
                For = 1 to termsStartswithToken[j].Count
```

```
                    If
(termsStartswithToken[j]).[k] = tempTerm
        Append thesaurus[i] to result array
        Breack to next length
                                End for k
                        Edn for each
        End for j
End for I
```

RESULTS

An experiment: Our data collections consist of five different thesauri. Table 2 gives a summary of these thesauri. A sample of 15 text collections was used. We test our system with these data collections. We experiment with these collections and databases different length of tokens. The variable length of the tokens ranges were from 50-991 tokens. The system uses stop list to remove noisy terms from the text collections. We ran both algorithms Brute Force and First Token (FT) using our data collections and thesauri. In each experiment we found the average processing time for each algorithm based on the dynamic changing of the length of tokens that range from 50-991. We plotted and compared the result for each experiment.

In the first experiment we used the first thesaurus (Training Thesaurus). The Training Thesaurus constitutes the controlled vocabulary of reference in the field of vocational education and training. We used our Tool ThesCov to built this Thesaurus from Web site related to the domain of Training. The other thesauri were constructed using our Tool ThesCov (Abuzir, 2010).

DISCUSSION

In Table 3 the average time (normalized) for both algorithms was calculated. Comparing our results for brute force algorithm and First Token (FT) algorithm, we can conclude that FT algorithm is more efficient in time on all cases of token length, especially for large number of tokens matching. The graph in Fig. 6 shows the time required for each algorithm using the first thesaurus.

We repeated the test with the other four thesauri and different data collections. A Summary of the average time required for both Brute Force and First Token algorithms to search terms of different length from our text collection using thesauri is shown in Table 3.

Figure 7 and 8 show time required for BF and FT algorithms respectively using the different thesauri. Figure 9 shows time required for BF and FT algorithms using the different thesauri.

The worst case of the proposed enhancement algorithm occurs when each token of the text found in Tokens table that means we need more two extra queries. Here we need the same total number of queries as brute force algorithm.

Table 2: A summary of thesauri

Thesaurus name	# Terms	#Distinct first tokens	Term length average (token)	Max term tokens count	Set of term tokens counts
Thesaurus 1	2522	1749	1.874	12	1, 2, 3,4,5,6,7,8,10,12
Thesaurus 2	3564	2363	1.844	8	1,2,3,4,5,6,8
Thesaurus 3	5800	3475	1.857	7	1,2,3,4,5,6,7
Thesaurus 4	69794	45042	1.903	9	1, 2, 3,4,5,6,7,8,9
Thesaurus 5	19726	10287	2.183	15	1,2,3,4,5,6,7,8,9,10,11,12,14,15

Table 3: Time elapsed to search terms of different length from our text collection in seconds using the different Thesauri

Text length	Thesaurus 1		Thesaurus 2		Thesaurus 3		Thesaurus 4		Thesaurus 5	
(token)	BF average	FT average	BF average	FT average	BF average	FT average	FT average	BF average	FT average	BF average
50	0.002	0.000	0.001	0.000	0.002	0.000	0.000	0.001	0.00	0.002
66	0.002	0.000	0.002	0.000	0.002	0.000	0.000	0.002	0.00	0.002
94	0.003	0.000	0.003	0.000	0.003	0.000	0.000	0.003	0.00	0.003
117	0.004	0.000	0.003	0.000	0.004	0.001	0.000	0.003	0.00	0.004
163	0.006	0.000	0.005	0.001	0.006	0.001	0.000	0.005	0.00	0.006
342	0.009	0.001	0.007	0.001	0.009	0.001	0.001	0.007	0.001	0.009
317	0.012	0.001	0.010	0.001	0.012	0.001	0.001	0.010	0.001	0.012
379	0.014	0.001	0.012	0.002	0.014	0.002	0.001	0.012	0.002	0.014
410	0.015	0.001	0.013	0.002	0.015	0.002	0.015	0.013	0.013	0.015
500	0.019	0.002	0.015	0.002	0.019	0.002	0.002	0.015	0.002	0.019
635	0.024	0.002	0.020	0.002	0.024	0.002	0.002	0.020	0.002	0.024
739	0.028	0.002	0.023	0.003	0.028	0.003	0.002	0.023	0.003	0.028
836	0.032	0.002	0.026	0.003	0.032	0.003	0.002	0.026	0.003	0.032
914	0.035	0.003	0.029	0.003	0.035	0.004	0.003	0.029	0.003	0.035
991	0.38	0.003	0.310	0.004	0.38	0.004	0.003	0.031	0.004	0.038

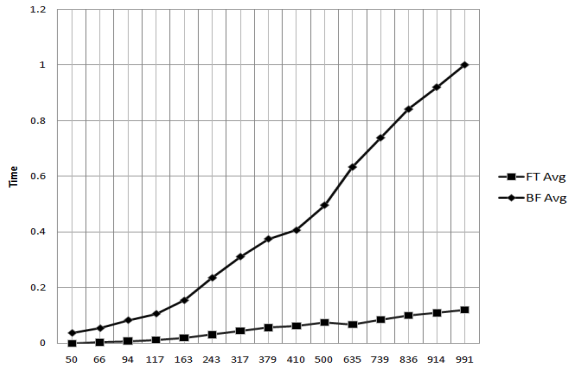

Fig. 6: Time required for BF and FT algorithms

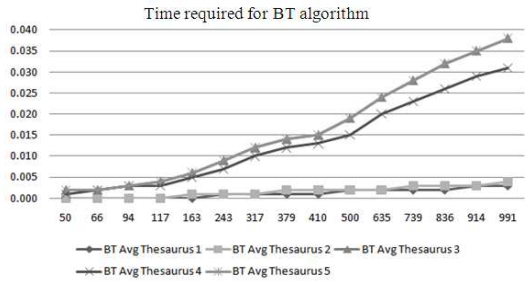

Fig. 7: Time required for BF algorithms using the five different thesauri

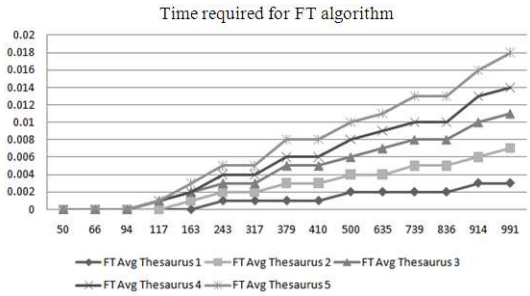

Fig. 8: Time required for FT algorithms using the five different thesauri

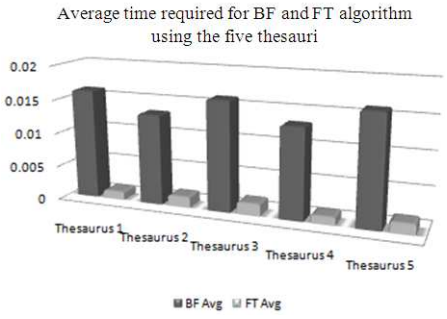

Fig. 9: Time required for BF and FT algorithms using the different thesauri

CONCLUSION

In summary, the proposed approach builds a new database structure (Fig. 4). The system creates these tables only one time. Using these new tables the system decrease the number of queries needed to search in the database. The new structure build new index to be used in searching thesaurus database instead of using the whole database. The system searches for any term in the new indexed tables instead of the original database.

In this study we proposed a string searching algorithm called First Token (FT) as an improvement of the brute force algorithm. Our experiments and data collections showed that the proposed algorithm is efficient. Our algorithm can perform in a faster and more efficient manner than brute force algorithm. Our algorithm decrease the number of queries required to query the databases and search time.

REFERENCES

Doug, G., C. Karen, C. Patrick, G. S. Martin and S.H. Timothy, 2011. Beginning Oracle Application Express 4. 1st Edn., Apress Publication, New York, ISBN: 978-1430231479, pp: 440.

Robert, L.M., 2006. Decisions in thesaurus construction and Use. Inform. Proc. Manage., 43: 958-968. DOI: 10.1016/j.ipm.2006.08.011

Abuzir, Y., 2010. Constructing the medical thesaurus as a tool for indexing, J. AL-Quds Open Univ. Res. Study, Palestine.

Lin, J., 2009. Brute force and indexed approaches to pairwise document similarity comparisons with MapReduce. Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, July 19-23, ACM, New York, USA., pp: 155. DOI: 10.1145/1571941.1571970

Chen, Y., Wang W. and Liu Z., 2011. Keyword-based search and exploration on databases. Proceeding of the IEEE 27th International Conference on Data Engineering, Apr. 11-16, IEEE Computer Society Washington, USA., pp: 1380-1383. DOI: 10.1109/ICDE.2011.5767958

Sleit, A., W. AlMobaideen, M. Qatawneh and H. Saadeh, 2009. Efficient processing for binary submatrix matching. Am. J. Applied Sci., 6: 78-88. DOI: 10.3844/ajassp.2009.78.88

Lokman, A.S. and J.M. Zain, 2010. One-match and all-match categories for keywords matching in

chatbot. Am. J. Applied Sci., 7: 1406-1411. DOI: 10.3844/ajassp.2010.1406.1411.

Christian, L. and H.B. Robert, 2000. Fast exact string pattern-matching algorithms adapted to the characteristics of the medical language. Orig. Investig., 7: 378-391. DOI: 10.1136/jamia.2000.0070378.

Cormen, T.H., 2001. Introduction to Algorithms. 2rd Edn. MIT Press/McGraw-Hill, New York ISBN-13: 978-0262033848, pp: 1180. http://books.google.com.pk/books?id=NLngYyWFl_YC&dq=Introduction+to+Algorithms&lr=&source=gbs_navlinks_s

Manber, U. and S. Wu, 1992. Fast text search allowing errors. Commun. ACM., 35: 83-91. DOI:10.1145/135239.135244

Benjamin, C.B., D.E. Taylor and R.K. Cytron, 2006. A scalable architecture for high-throughput regular-expression pattern matching. Proceeding of the International Symposium on Computer Architecture, (ISCA '06), IEEE Xploor, Boston, pp: 191-202. DOI: 10.1109/ISCA.2006.7

Fredriksson, K., 2010. On building minimal automaton for subset matching queries. J. Inform. Proc. Lett., 110: DOI: 10.1016/j.ipl.2010.09.014

Al-Mazroi, A.A. and N.A. Rashid, 2011. A fast hybrid algorithm for the exact string matching problem. Am. J. Eng. Applied Sci., 4: 102-107. DOI: 10.3844/ajeassp.2011.102.107

Alajlan, N., 2009. Solving square jigsaw puzzles using dynamic programming and the hungarian procedure. Am. J. Applied Sci., 6: 1941-1947. DOI: 10.3844/ajassp.2009.1941.1947.

Agrawal, S., S. Chaudhuri and G. Das, 2002. DBXplorer: A System for Keyword-Based Search over Relational Databases. Proceeding of the 18th International Conference on Data Engineering, Feb. 26-Mar. 1, IEE Xploor, San Jose, pp: 5-16. DOI: 10.1109/ICDE.2002.994693

He, H., H. Wang, J. Yang and P.S. Yu, 2007. BLINKS: Ranked keyword searches on graphs. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 11-14, ACM, New York, NY, USA., pp: 305-316, DOI: 10.1145/1247480.1247516

Carmel, D., Y.S. Maarek, M. Mandelbrod, Y. Mass and A. Soffer, 2003. Searching XML documents via XML fragments. Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 28-Aug. 01, ACM New York, USA., Toronto, Canada, pp: 151-158. DOI: 10.1145/860435.860464

Vu, Q.H., B.C. Ooi, D. Papadias and A.K.H. Tung, 2008. A graph method for keyword-based selection of the top-K databases. Proceedings of the 2008 ACM SIGMOD international conference on Management of Data, June 09-12, ACM, New York, USA., pp: 1378. DOI: 10.1145/1376616.1376707

Al-mazroi, A.A. and A. Rashid, 2011. A fast hybrid algorithm for the exact string matching problem. Am. J. Eng. Applied Sci., 4: 102-107. DOI: 10.3844/ajeassp.2011.102.107