グラフデータベースにおける冗長解抑制を伴った Top-k キーワード検索アルゴリズムに関する研究

王　美蓉† 　張　麗茹† 　金　銀実† 　大森　匡†

†電気通信大学大学院情報システム学研究科〒182–8585 東京都調布市調布ヶ丘 1–5–1

E-mail: †{wang, zhangliru, kim, omori}@hol.is.uec.ac.jp

あらまし　近年のデータベース研究の課題の一つに、大量情報をグラフデータベースとして表してキーワード検索を行う研究がある。著者らのグループは、先行研究である DPBF 算法を改良して、上位 k 個の正確な答えを効率良く求める算法 MDPBF を既に提案している。しかし、MDPBF は、1000 ノード級のグラフでキーワード数 6 による検索でも計算効率が悪くなっていた。この状況を改善するため、本研究では、上位 k 個の解を求めるときの冗長解削除戦略として複数の異なる手法を導入し、それらの下で MDPBF により得られる答えの品質と計算効率を調べ、両者のバランスをとった冗長解削除戦略を提案する。そして、探索空間の削減戦略と併用することにより、10 万ノード級のグラフデータベースにおいても効率的に MDPBF が動作することを目指す。

キーワード　Top-k、キーワード検索、グラフデータベース

Top-k Keyword Search Algorithms with Redundancy Elimination on Large Graph Databases

Meirong WANG† 　Liru ZHANG† 　Yinshi JIN† 　and 　Tadashi OHMORI†

† Graduate School of Information Systems, The University of Electro-Communications

1-5-1, Chofugaoka, Chofu City,Tokyo, 182-8585 Japan

E-mail: †{wang, zhangliru, kim, omori}@hol.is.uec.ac.jp

Abstract　In this paper, by modifying an existing graph-database search algorithm DPBF [1] in a straightforward strategy, we propose such an algorithm that can find exact top-k answers, with eliminating redundant answers, in an exact ranked order in $O((2^{l+1}mk + 3^l nk \cdot \log k)\log(2^l nk))$ run time (l: the number of keywords; k: top-k; n: the number of nodes in the graph database; m: the number of edges in the graph database) with the space complexity of $O(2^{l+1}nk)$.

Keyword　Top-k, Keyword Search, Graph Database

1. Introduction

Recently, keyword search has been a very popular way to explore information. And many studies supporting keyword search on graph databases have been proposed like DPBF [1], BLINKS [2]. In these studies, a database is modeled by a graph, when a set P of keywords is given as a query, how to find an appropriate sub-graph with smaller cost which satisfies P in that graph is the problem in this field [1, 2, 3, 4].

One definition of this problem is to find the top-k answers which contain all the keywords in the ranked order of the increasing costs of connected trees. This definition is adopted in DPBF [1] and is known as finding the top-k min-cost connected trees based on the algorithm of GST-k (Group Steiner Tree).

DPBF [1] is an algorithm which repeats tree grow/merge operations until GST-k are found. DPBF grows and merges all the trees at different root nodes, so it can get the accurate top-1. It just keeps the min-cost one of the trees with the same root node and the same keywords, so the answers from top-2 to top-k cannot be exact. Furthermore DPBF for finding top-k answers (exactly, DPBF-k in [1]) must produce some redundant answers which are not explained in [1].

Consequently, in order to get the top-k answers exactly and eliminate redundant answers, we propose an improved method named MDPBF by modifying DPBF by a straightforward strategy.

2. The limitation of DPBF

DPBF [1] is a method which aimed at finding the top-k min-cost connected trees which contain all the keywords. The connected tree with smaller cost presents more important information. The main idea of DPBF is repeating tree grow/merge operations to produce GST-k.

Because DPBF loses many potential connected trees which may be answers, the top-k answers got by it are approximate. In [1] there is not clear explaining about the redundancy elimination. In this section, we will introduce three Cases of redundant answers as shown in Figure 1.

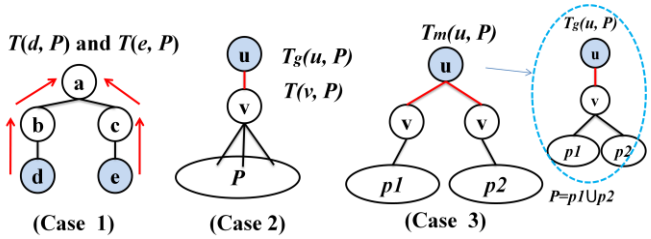

Figure 1: Redundant answers (Cases 1, 2 and 3)

- Case 1: the trees which have the same edges but with different root nodes. We call these trees redundant answer of isomorphs.
- Case 2: the tree which is the super set of the tree which has contained all the keywords. Because this Case of redundant answer is produced after tree-grow operation, we call it redundant answer of growing.
- Case 3: the tree which contains the same edge more than once. This case of redundant answer happens after tree-merge operation, so we call it redundant answer of merging. DPBF just keeps the min-cost one of the trees with the same root node and the same keywords, so this kind of redundant answer can be avoided automatically in DPBF.

3. MDPBF

In order to find the exact top-k answers and eliminate redundant answers, we newly propose an improved method named MDPBF.

3.1 Find exact top-k

From [1], we can see that DPBF just keeps the min-cost one of the trees with the same root node and the same keywords, and that is why DPBF loses answers. In order to improve the quality of answers, instead of keeping the min-cost one, MDPBF keeps top-k of the trees with the same root node and the same keywords. The main idea of MDPBF is the same as DPBF.

The modified equations which are based on equations of [1] are shown as follows.

If node v directly contains a keyword subset $p \subseteq$ P:

$$T_1(v,p) = v \quad (1)$$

If a tree has more than one node:

$$T_n(v,p) = min\_n\{\, TG(v,p) \cup TM(v,p) \,\} \quad (2)$$

Tree Grow: $TG(v,p) = \{\, (v,u) \oplus T_i(u,p) \mid u \in N(v), 1 \le i \le k \,\}$ (3)

Tree Merge: $TM(v,p) = \{\, T_i(v,p_1) \oplus T_j(v,p_2) \mid (p_1 \cap p_2 = \emptyset) \wedge (p_1 \cup p_2 = p, i \times j \le k) \,\}$ (4)

As notations:

- Eq(1): is a leaf node tree without any edge, so its cost is 0.
- $\oplus$ is an operation to merge two trees into a new tree.
- $N(v)$ is a set of neighbor nodes of node v.
- $T_n(v,p)$ means the n-th smallest connected tree with root node v and keyword subset p where $(1 \le n \le k)$ got by function min_n..

When we do the merge operation, there are at least $i \times j$ combinations, for reducing the size of queue, we just do merge where $i \times j \le k$.

MDPBF enumerates candidate trees by using Q_G (global queue) and Q_L (local queue) [5, 6] in the creasing order of their costs until top-k answers are found.

3.2 Avoiding redundant answers

3.2.1 New redundant answer of MDPBF

In fact, except the three cases of redundant answers of DPBF, MDPBF produces a new kind of redundant answer as shown in Figure 2. Note that MDPBF allows keeping top-k of the trees with the same root node and the same keywords. Thus it is possible to keep the top-k trees with the same root node and the same leaf nodes but different paths (Case 4).

See an example shown in Figure 2. Suppose that $T_1(u,\{p1,p2,p3\})$ grows to $T_1'(x,\{p1,p2,p3\})$ $(= T_1 \oplus (u,x))$ and further T_1' grows (along the dotted path from x to v in Figure 2) to $T_2(v,\{p1,p2,p3\})$. If T_2 further grows to $T_3(v,\{p1,p2,p3\})$ $(= T_2 \oplus (v,x))$, T_3 has a cycle and the cycle does not contain any new information about keywords.

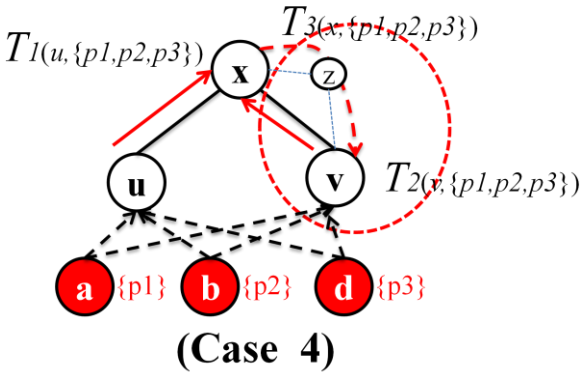

Figure 2: New redundant answer (Case 4)

3.2.2 Redundancy elimination

There are four cases of redundant answers in MDPBF; next we introduce the elimination methods.

Firstly, we eliminate the Case 1 of redundant answers.

- **Method 1 (isomorphs checking)**: Because we aim at finding the top-k connected trees which contain all the keywords in ranked order of increasing costs of connected trees, there are not more than k answers. In order to avoid the Case 1, we just need to check the answers found whether they are the isomorphs.

Second is the Case 2 of redundant answers. We propose two alternative methods to avoid Case 2:

- **Method 2 (redundancy labeling)**: When a tree has contained all the keywords, the new grow/merge trees from it will be redundant answers, so we can give this kind of redundant answers a label to record that they are redundant, and are not allowed to be result..
- **Method 3 (grow stopping)**: Because the merge operation is done when two trees do not have the same keywords, the Case 2 of redundant answers just happens on grow operation. In order to avoid it, when a tree has contained all the keywords, we dequeue it directly without doing the grow operation from it.

Next is the Case 3 of redundant answers. DPBF just keeps the min-cost one of the trees having the same root node and the same keywords, so this case of redundancy can be avoided automatically in DPBF. MDPBF keeps top-k of the trees having the same root node and the same keywords, so this case of redundant answers can be produced by MDPBF.

- **Method 4 (edges checking)**: In Case 3 $T_m(u,P)$ counts the cost of edge (v,u) twice, hence its cost is bigger than $T_g(u,P)(=T(v,p_1\cup p_2)\oplus(v,u))$ which counts the cost of edge (v,u) only once ($T(v,p_1\cup p_2)=T_1(v,p_1)\oplus T_2(v,p_2)$). In order to avoid this kind of redundant answer, when a new edge is added into a tree, we can check whether the new edge has been contained, if so, give up this merge operation.

Final is the Case 4 which contains a cycle.

- **Method 5 (nodes checking)**: In order to avoid the Case 4, one way is to remember all the nodes of a tree directly, then when a new node is added into a tree, we firstly check whether this new node has been contained, if so, we can give up this grow operation.

Except the above the methods, we also proposed a method named *leaf nodes checking* to avoid Cases 3 and 4 at the same time.

- **Method 6 (leaf node checking)**: In Case 3 and Case 4, MDPBF may produce some answers which contain useless edges or nodes, so in order to avoid them, we can check all the trees with the same root node and the same leaf nodes. As a side effect, for each root node v with any keyword subset p, those trees remembered by MDPBF can have the same root node v but they must have different sets of leaf nodes that satisfy p. For example, in Figure 2, we just keep the smaller one between T_1' and T_3 not both.

In summary, we prepare several algorithms with different redundancy elimination methods referred above.

- **DPBF1.1**: with the redundancy elimination methods 1 and 2.
- **DPBF1.2**: with the redundancy elimination methods 1 and 3. The difference with DPBF1.1 is that DPBF1.2 stops the tree grow/merge operations on the trees which have been answers.

Because the DPBF just keeps the min-cost one of the trees with the same root node and the same keywords, DPBF can avoid the Case 3 automatically. Of course, DPBF cannot find exact top-k.

As a practical MDPBF, we propose three versions:

- **MDPBF1.1**: with the redundancy elimination methods 1, 2 and 6. The method 1 is used to avoid the Case 1 and the method 2 is used to avoid the Case 2. The method 6 can avoid Cases 3 and 4.
- **MDPBF1.2**: with the redundancy elimination methods 1, 3, 4 and 5. We used the same method to avoid the Case 1 as MDPBF1.1. We used method 3 to reduce the productions of the redundant candidate trees. And the method 4 is to hold back happens of Case 3 and method 5 aimed at stopping the production of Case 4.
- **MDPBF1.3**: with the redundancy elimination method 1, 3 and 6. In fact, it has the same definition about answer tree as MDPBF1.1. But the difference between them is that MDPBF1.3 stops the tree grow/merge operations on the trees which have contained all the keywords.

3.3 Time and space complexities

Let the size of keyword query be l. When the number of nodes of graph is n, we have to keep $2^l nk$ candidate connected trees. So the maximum size of Q_G is $2^l nk$. The time complexity of MDPBF is $O((2^{l+1}mk+3^l nk\cdot \log k)\log(2^l nk)$.

MDPBF uses Q_G and Q_L (see [5, 6]) to keep all the trees. The maximum number of trees is $2^l nk$ at worst. So the space complexity of MDPBF is $O(2^{l+1}nk)$.

4. Comparison Results

We did experiments to compare these algorithms DPBF1.1, DPBF1.2, MDPBF1.1, MDPBF1.2 and

MDPBF1.3. All the experiments were performed on a 2.66GHz CPU machine with 3GB memory and implemented by JAVA l_5.

4.1 Experiments on small Graph

Firstly, we did experiments to compare answers quality between DPBF and MDPBF. And we found that DPBF produced low quality answers.

Next we performed preliminary experiments to compare the answer quality of MDPBF1.1, MDPBF1.2 and MDPBF1.3. Take Figure 3 as an example, give a set of keywords {p2, a1}, and the nodes 5, 8 and 9 are leaf nodes. MDPBF1.1 and MDPBF1.3 can produce Trees 1, 2 and 3. MDPBF1.2 can produce Trees 1, 2, 3 and 4. In MDPBF1.1 and MDPBF1.3, Tree 4 is considered as redundant because it contains the same leaf nodes and the same root node with Tree 1. Although Tree 4 has different information with Tree 1, its cost is much higher than Tree 1. So we can see that the answer definition of MDPBF1.1 and MDPBF1.3 is different from MDPBF1.2. MDPBF1.1 and MDPBF1.3 try to find the combinations of different leaf nodes at different root nodes. And MDPBF1.2 considers that all the trees with different paths are different and all of them can be answers.

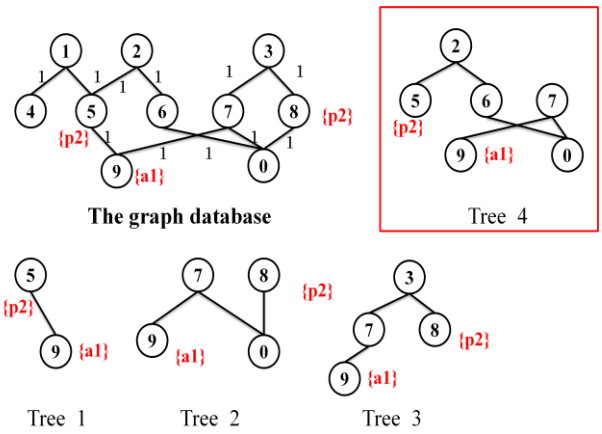

Figure 3: A simple example

4.2 Experiments on 1000-nodes Graph

Next we performed experiments to compare performance of five algorithms referred above. The tests are based on the random graph with 1000 nodes and 1500 edges. Every edge-cost is set to 1. The hit rate for each keyword is 1% (1% of all nodes of graph are leaf nodes for each keyword). k (of top-k) =10. We changed the number of keywords from 1 to 5. The experimental results are shown in Figures 4, 5 and 6. From the experimental results, we can see that MDPBF1.2 and MDPBF1.3 can reduce the size of priority queue and find the top-10 efficiently. Although the quality of answers produced by MDPBF1.1 is high, the performance of MDPBF1.1 is not very well. That is because it produces many candidate trees and the GST-k are found in a wide range. MDPBF1.2 maybe finds the top-k in a narrow range but its efficiency is best. So next we will pay attention on the two algorithms MDPBF1.2 and MDPBF1.3 to make them work well on large graphs.

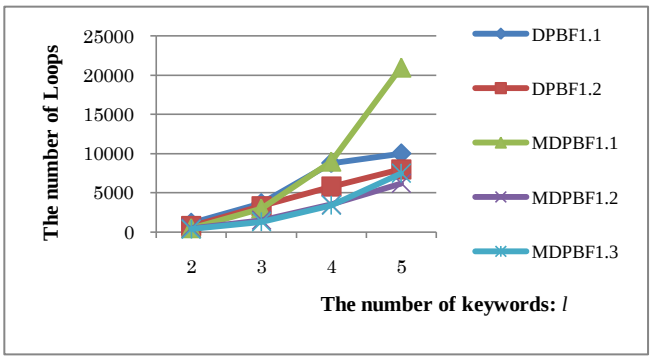

Figure 4: The number of main loop vs. *l* (k=10, n=1000)

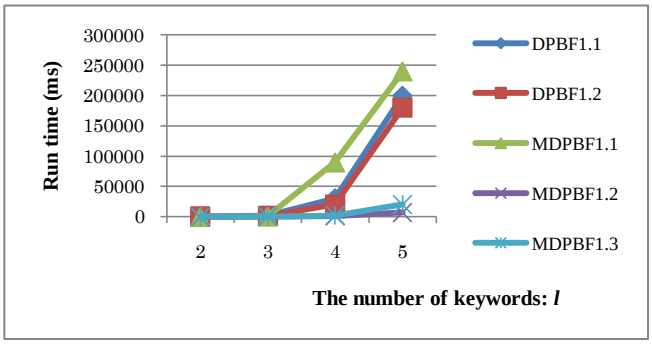

Figure 5: Run time (ms) vs. *l* (k=10, n=1000)

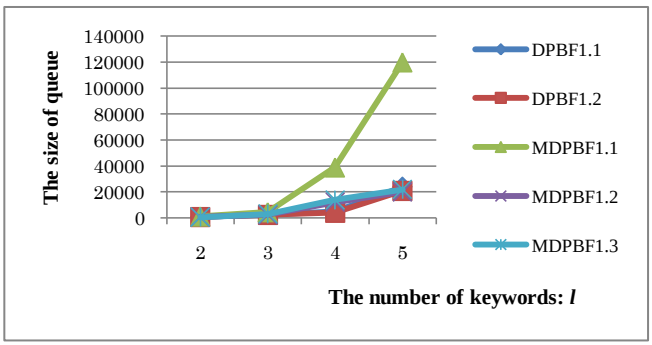

Figure 6: The size of priority queue vs. *l* (k=10, n=1000)

4.3 Experiments on 10000-nodes Graph

Next we uses a random graph contained 10000 nodes and 15000 edges to do experiments. And the hit rate per keyword is 0.1%. The results are shown in Tables 1 and 2.

As notations:

- k: equals to 10
- L: the number of keywords
- Loops: the number of loops when GST-k are found
- QgSize: the maximum size of priority queue (global queue: keep all the candidate connected trees)
- Tmax: the maximum size of edges sets of answer trees
- N1: the number of the trees with the same root node and leaf nodes in candidate trees
- N2 : the number of the trees with the same root node and leaf nodes in answers
- N3: the number of the trees with the same leaf nodes in answers.
- N4: the number of the trees with the same leaf nodes in candidate trees

From Tables 1 and 2, we can see that MDPBF1.2 and MDPBF1.3 just work well when the number of keywords is not bigger than 4. That is because the graph is very sparse, and before we find the top-k, the programs need to repeat tree grow/merge operations many times and the size of priority queue becomes bigger and bigger. However from the two tables, we can also find the size of edges of answer trees (Tmax) is not so big. In order to make MDPBF work well on large graph databases, we need to reduce the search space.

Table 1: Test result of MDPBF1.2 on 10,000 nodes (k=10)

L	Loops	Run time(msec)	QgSize	Tmax	N1	N2
2	2,000	800	6,500	13	130	0
3	10,000	10,000	32,000	16	1,600	0
4	23,000	170,000	100,000	23	7,000	0

Table 2: Test result of MDPBF1.3 on 10,000 nodes (k=10)

L	Loops	Run time (msec)	QgSize	Tmax	N3
2	1,900	1000	7,700	14	2
3	10,900	27,000	45,000	19	4
4	27,000	450,000	135,000	25	1

5. Search space limitation methods

In order to reduce the search space (the size of priority queue), we propose three limitation methods.

Limitation method 1 (L1): limit the max-number (Tmax: an given upper bound) of the edges of connected trees.

Limitation method 2 (L2): limit the max-number of the edges of connected tree based on the number of keywords of this tree (the number of edges ≤ (the number of keywords l_i)* h))($l_i \leq$ L) (h: a given parameter).

Limitation method 3(L3): limit the max-height (H: a given parameter) of the connected tree, in other words L3 limits the max-number of edges between any leaf node and root node.

In fact, when we give a same parameter to L2 and L3 (h=H), the answers got by L2 will contain all the answers got by L3. Namely, L3 has stricter limitation than L2.

Figures 7 and 8 showed the new experimental results on 10,000 nodes graph when Tmax=25, and h=H=5.

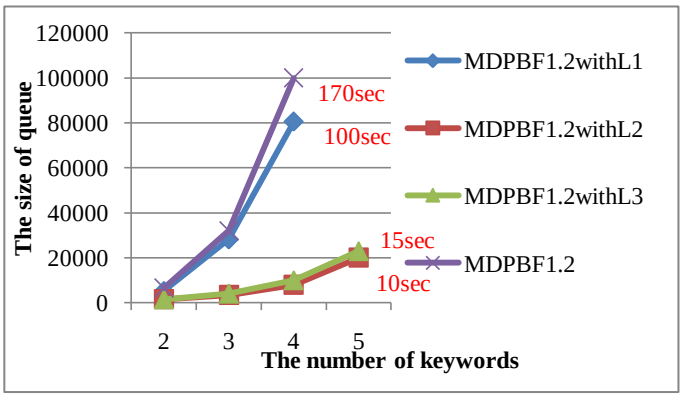

Figure 7: MDPBF1.2 (k=10, n=10000)

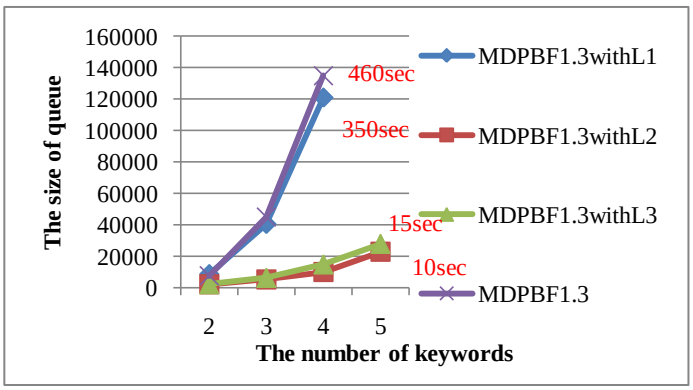

Figure 8: MDPBF1.3 (k=10, n=10000)

Form the Figures 7 and 8, we can say that L2 and L3 can reduce the size of priority queue distinctly and improve search efficiency clearly. When the number of keywords is smaller than 4, L1 cannot reduce search space distinctly.

Finally, we did experiments on a large graph which contains 100,000 nodes and 150,000 edges. The hit rate per keyword is 0.1%.

Tables 3 and 4 are the test results of MDPBF1.2. And Tables 5 and 6 are the results of MDPBF1.3.

Table 3: Test result of MDPBF1.2withL2 on 100,000 nodes (k=30)

L	loops	Run time (msec)	QgSize	The number of trees with the same root node and leaf nodes
2	5,224	3,900	6,890	1
3	11,753	10,637	16,600	0
4	25,600	33,100	38,950	0

Table 4: Test result of MDPBF1.2withL3 on 100,000 nodes (k=30)

L	loops	Run time (msec)	QgSize	The number of trees with the same root node and leaf nodes
2	5,484	4,215	7,256	1
3	21,000	18,759	23,200	0
4	54,200	75,785	57,200	0

Table 5: Test result of MDPBF1.3withL2 on 100,000 nodes (k=30)

L	loops	Run time (msec)	QgSize
2	5,113	4,800	8,785
3	10,700	10,500	20,000
4	25,925	44,600	50,600

Table 6: Test result of MDPBF1.3withL3 on 100,000 nodes (k=30)

L	loops	Run time (msec)	QgSize
2	4,790	4,428	8,400
3	21,050	23,600	30,800
4	52,131	106,700	75,000

Form the four tables, we can find that MDPBF1.2 with Limitation method 2 has the best performance.

6. Conclusions

This paper described redundant answers of DPBF-k and their elimination methods that are not described explicitly in [1]. We next proposed an improved method named MDPBF to find the top-k min-cost connected trees exactly with redundancy elimination.

From the experimental results, we found that MDPBF can produce exact top-k. MDPBF1.1 has working limit on 1000-nodes graph, and MDPBF1.2 and MDPBF1.3 have working limit on 10000-nodes graph. By using the search space limitation methods 2 and 3, MDPBF1.2 and MDPBF1.3 can work well on large graph databases (100000 nodes).

Reference

[1] B.Ding, J.X. Yu, L.Qin, X. Zhang, and X. Lin. Finding Top-k Min-Cost Connected Trees in Databases. In proc. of the 23rd International Conference on Data engineering (ICDE'07), pp.836-845, 2007.

[2] H. He, H. Wang, J. Yang and P. S. Yu. BLINKS ranked Keyword Searches on Graphs. In SIGMOD'07, pp.305-316, 2007.

[3] K. Golenberg, B. Kimelfeld and Y. Sagiv. Keyword Proximity Search in Complex Data Graphs. In ACM SIGMOD'08, pp.927-940, 2008.

[4] B. Kimelfeld and Y. Sagiv. Finding and Approximating Top-k Answer in Keyword Proximity Search. In ACM PODS'06, pp.173-182, 2006.

[5] M.R. Wang, L.J Jiang, L.R. Zhang and T. Ohmori. Exact Top-k Keyword Search on Graph Databases. In SAC'11. 2011. To appear.

[6] M.R. Wang, L.R. Zhang and T. Ohmori. A report on Top-k Keyword Search Algorithms with Redundancy Elimination on Graph Databases. In IPSJ SIG Technical Report (DBS-151-28) Japan. 2010.