PACOKS: Progressive Ant-Colony-Optimization-Based Keyword Search over Relational Databases

Ziyu Lin[1(✉)], Qian Xue[1], and Yongxuan Lai[2]

[1] Department of Computer Science, Xiamen University, Xiamen, China
{ziyulin,xueqian2015}@xmu.edu.cn
[2] School of Software, Xiamen University, Xiamen, China
laiyx@xmu.edu.cn

Abstract. Keyword search over relational databases makes it easier to retrieve information from structural data. One solution is to first represent the relational data as a graph, and then find the minimum Steiner tree containing all the keywords by traversing the graph. However, the existing work involves substantial costs even for those based on heuristic algorithms, as the minimum Steiner tree problem is proved to be an NP-hard problem. In order to reduce the response time for a single search to a low level, a progressive ant-colony-optimization-based algorithm, called PACOKS, is proposed here, which achieves the best answer in a step-by-step manner, through the cooperation of large amounts of searches over time, instead of in an one-step manner by a single search. Through this way, the high costs for finding the best answer, are shared among large amounts of searches, so that low cost and fast response time for a single search is achieved. Extensive experimental results based on our prototype show that our method can achieve better performance than those state-of-the-art methods.

1 Introduction

Keyword search over relational databases and XML documents makes it an easier task to retrieve information from structural and semi-structural data, as users, especially casual users, do not need to learn query languages such as SQL or to know anything about data schemas. Keyword search over these data has been recently thoroughly studied (e.g., [2,3,8,13,16]), and the work can be typically classified into two categories, namely data graph based (e.g., BANKS [2] and EASE [9]) and schema graph based (e.g., DISCOVER [6] and DBXplorer [1]). Schema-graph-based approach enumerates results by directly running SQL statements against DBMS, which means that such approach can only be used in relational data, while the data-graph-based approach is to find out the minimum

Supported by the Natural Science Foundation of China (61303004), the National Key Technology Support Program (2015BAH16F00/F01) and the Key Technology Program of Xiamen City (3502Z20151016).

B. Cui et al. (Eds.): WAIM 2016, Part II, LNCS 9659, pp. 107–119, 2016.
DOI: 10.1007/978-3-319-39958-4_9

Steiner tree by traversing through a data graph, which makes it appropriate for all the data that can be represented as graph.

However, the existing work encountered a challenging issue, i.e., high cost for a single search. As the minimum Steiner tree problem is proved to be an NP-hard problem, the previous data-graph-based solutions, most based on heuristics, usually involve substantial cost for a single search. It is also the case for those schema-graph-based solutions, since there are large amounts of possible expressions to run over the DBMS.

To address the above problem, we first propose an ant-colony-optimization-based algorithm, called ACOKS, to find the minimum Steiner tree from the data graph. Furthermore, we propose a novel progressive ant-colony-optimization-based algorithm, called PACOKS, an abbreviation for Progressive Ant Colony Optimization based Keyword Search, which achieves the best answer in a step-by-step manner through the cooperation of large amounts of searches over time, instead of in an one-step manner by a single search. In other words, the later search result is further optimized based on the earlier one, and the global optimal solution, i.e., the minimum Steiner tree, is gradually achieved through many searches. This way, the high costs for finding the best answer are shared among large amounts of searches, so that low cost and fast response time for a single search is achieved.

To sum up, the main contributions of this paper include:

- An ant-colony-optimization-based algorithm for approximating the minimum or top-k Steiner trees problem.
- A progressive ant-colony-optimization-based algorithm to share the high cost of Steiner tree problem among large amounts of searches so as to achieve very low cost for a single search.
- A prototype to carry out extensive experiments, which confirm the superior performance of our approach over the state-of-the-art methods.

The remainder of this paper is organized as follows. We formalize the minimum Steiner tree problem in Sect. 2. Section 3 discusses the ACOKS algorithm. We discuss in detail the PACOKS algorithm in Sect. 4. Experimental studies are given in Sect. 5, followed by discussions over related work in Sect. 6. Finally, we conclude in Sect. 7.

2 Data-Graph-Based Approach

2.1 Data Graph

The principal of data-graph-based solutions to keyword search over relational databases, is to enumerate the minimum cost Steiner trees by traversing through the data graph constructed from a relational database.

A relational database D can be considered as a directed graph $G(V, E)$, called *data graph*, where V represents the set of nodes, and E the set of edges. A node $u \in V$ represents a tuple in D. A tuple may be inserted into or deleted from a

relation in D, resulting in the change of the data graph. Given two nodes $u \in V$ and $v \in V$, there exists a forward edge $e(u, v)$ in the data graph, from u to v, denoted $u \rightarrow v$, and a backward edge $e(v, u)$, from v to u, denoted $v \rightarrow u$, if there is primary-foreign-key relationship between u and v, where the foreign key is defined on u referencing to the primary key defined on v. Both edges and nodes may have weights, which are identified by the functions of $w(e)$ and $w(u)$ respectively. The weight of a data graph, denoted $w(G)$, is the sum of the weights of all edges, i.e., $w(G) = \sum_{e \in E} w(e)$. Keyword search algorithms score the candidate Steiner trees based on node weights and edge weights, and then rank these trees in the order of decreasing score. Since the focus of this paper is not how to define the weight function, we simply take the weight function proposed in BANKS [2]. In other words, the weight of a node u, denoted $w(u)$, is a function of in-degree. The weight of an edge $e(u, v)$ depends on the its type, i.e., $w(u, v)$=1 for a forward edge representing a primary-foreign-key relationship, and $w(v, u) = w(u, v) * log_2(1 + D_{in}(v))$ for a backward edge, where $D_{in}(v)$ represents the in-degree of a node v.

Example 1. Figure 1 shows a simple example on the publication database DBLP. It consists of four relation schemas (see Fig. 1(a)), i.e., *Authors*, *Papers*, *Cites* and *Writes*. *Authors* has two attributes, *AID* and *Name*, and the primary key is defined on *AID*. *Papers* has two attributes (*PID* and *Title*) with *PID* as the primary key. *Cites* has two foreign keys, *Citing* and *Cited*, both referring to the primary key defined on *Papers*. *Writes* has two foreign keys, *AID* (refer to the primary key defined on *Authors*) and *PID* (refer to the primary key defined on *Papers*). Figure 1(b) and (c) show the database conforming to the relation schemas above and its corresponding data graph respectively. □

2.2 Steiner Tree Problem

In a relational database, the task of keyword search is to find those Steiner trees containing the keywords. These nodes containing the keywords are interconnected by sequences of primary/foreign key relationships among tuples.

Example 2. Figure 1(d) shows two Steiner trees. The left one may be one of the answers for the 4-keyword search $\{database(k_1), XML(k_2), Jim(k_3), Steiner(k_4)\}$, and the right one for the 3-keyword search $\{Jim(k_5), Steiner(k_6), Kate(k_7)\}$, over the publication database in Fig. 1(b). □

The best answer for a keyword search is a minimal Steiner tree.

Definition 1. *[Minimum Steiner tree] Given a data graph $G(V, E)$ and a node set $V' \subseteq V$, a tree T is called a Steiner tree over V', if T contains all the nodes in V'. Let $c(T) = \sum_{e \in E} w(e)$ be the cost of T, where $w(e)$ is the weight of an edge e. We say that T is the minimum Steiner tree, if $c(T)$ is the minimum among all the Steiner trees over V' in G.* □

An extension of minimum Steiner tree is minimum group Steiner tree. Keyword search over a data graph can also be seen as a minimum group Steiner tree problem.

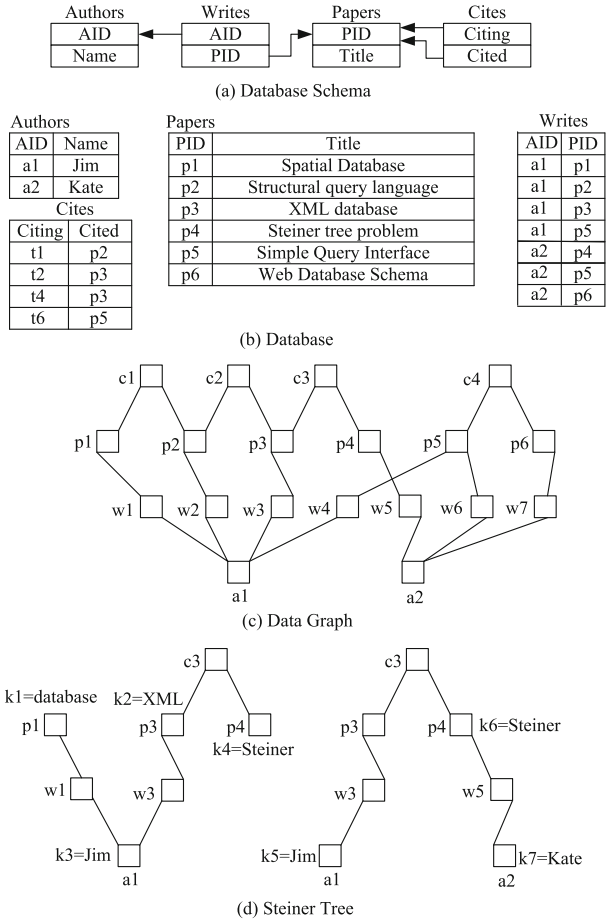


Fig. 1. A simple publication database.

Definition 2. *[Minimum group Steiner tree] Given a data graph $G(V, E)$ and groups $V_1, V_2, ..., V_n \subseteq V$, we say that T is the minimum group Steiner tree over $V_1, V_2, ..., V_n$ in G, if T is a minimum Steiner tree and contains at least one node from each group $V_i (1 \leq i \leq n)$.* □

3 The ACOKS Algorithm

Large amounts of experiments in recent years proved that, ant colony optimization (or ACO in abbreviation) is able to achieve high efficiency in dealing with various kinds of NP-hard problem. Therefore we here propose an ant-colony-optimization-based algorithm, called ACOKS, an abbreviation of ACO-based Keyword Search, to deal with the Steiner tree problem. The basic idea of ACOKS is to find Steiner trees containing the keywords in the data graph through the

cooperation of many ants. When searching for the optimal solution, it aims to find the minimum Steiner tree, while for the top-k answers, it outputs the top-k Steiner trees with minimum costs. As far as the solution for the Steiner tree problem is concerned, it usually uses the method of spanning and cleanup. In this method, it starts from any node of a group V_i and spans to its neighboring nodes, until covering at least one node from each group. Finally, those redundant nodes are deleted from the result tree. The main algorithm of ACOKS is outlined in Algorithm 1. The function of ONE_ANT_MOV(ant) in Algorithm 1 is given in Algorithm 2. The transition rule and pheromone updating for ant colony optimization can be found in many other related research work, so they are not discussed here.

Algorithm 1. ACOKS

```
Input  : keyword search K = {k_1, k_2, ..., k_m};
         data graph G;
Output: top-k Steiner trees;
begin
    foreach k_i ∈ K do
        get the content set C_i of k_i;
    iteration_time←0;
    initialize the pheromone matrix M;
    result_heap ← Φ;
    t ← 0;
    while iteration_time < max_iterlation_time do
        select one node from each C_i and put p ants on it to form ant set S_ants;
        ant_num← m * p;
        foreach ant ∈ S_ants do
            step_num ← 0;
            S_visited(ant) ← Φ;
            S_spanned(ant) ← {k_i};
        while ant_num > 0 do
            foreach ant ∈ S_ants do
                ONE_ANT_MOV(ant)
        iteration_time ← iteration_time + 1;
    output the result trees in result_heap;
```

4 The PACOKS Algorithm

4.1 The Defect of ACOKS

The ACO algorithm has characteristics that do not exist for ants in the natural world. For example, the results can be optimized little by little with the accumulation of pheromone on the paths. By optimization, it means that the

Algorithm 2. ONE_ANT_MOV(ant)

```
begin
    if ant.step_num > max_step_num then
        destroy ant;
    else
        ant.step_num ← ant.step_num + 1;
        t ← t + 1;
        move to next node v satisfying that v ∉ S_visited(ant);
        if there is no other ant arriving at v at time t then
            S_visited(ant) ← S_visited(ant) ∪ v
        else
            S_visited(ant) ← S_visited(ant_other) ∪ S_visited(ant);
            S_spanned(ant) ← S_spanned(ant_other) ∪ S_spanned(ant);
            destroy ant_other;
            if S_spanned(ant) = K then
                output the Steiner tree s_tree composed of nodes in
                S_visited(ant);
                remove redundant nodes from s_tree;
                if score(s_tree) > score(s_tree_min) then
                    remove s_tree_min from result_heap;
                    put s_tree into result_heap;
                else
                    discard s_tree;
                destroy ant;
        update the pheromone matrix M at time t;
```

later solution is closer to the global optimal solution (or *GOS* in abbreviation), compared with the previous one. In other words, for ACOKS, the solution of each iteration is a successive approximation to the *GOS*. Also the ACO algorithm usually can achieve good convergence in most real applications, though the theoretical proof of its convergence is still not available for some cases. Gutjahr et al. [4] discussed the convergence of a graph-based ACO, and proved that the *GOS* can always be found if the amount of ants is large enough. Yang et al. [15] proved that the ACO algorithm for the Steiner tree problem is able to converge to the *GOS*, with the probability of 1-ϵ, when the the number of iterations is large enough, where ϵ is an arbitrarily small value. Therefore, for the above ACOKS algorithm, we can get a conclusion as follows:

– it can not be assured to find the *GOS* if the number of ants is limited; and
– there must be enough ants if the *GOS* is to be achieved.

However, the contradiction is that it will take too long and be unacceptable for user, if it involves a large amount of ants for a single user search, though the *GOS* can be finally achieved through this way. On the other hand, if we want to limit the response time for a single search to a low level, the number of

ants must be limited to a small amount. However, this may influence the result quality and lead to low search effectiveness.

4.2 The PACOKS Algorithm

To resolve the issue above, we here propose the progressive-ACO-based algorithm, or PACOKS in abbreviation, which aims to achieve the *GOS* through a step-by-step approach rather than a one-step approach. In other words, the result of the current search is a further optimization upon that of the previous one, so that the result of every search is a successive approximation of the *GOS*. In this way, the expensive costs of finding the *GOS* are shared among a large amounts of searches, and the cost for a single search is reduced to a very low level, resulting in fast response times for it.

Now we consider how to find the best answer for a search by PACOKS, which can be easily extended to support top-k search.

Algorithm 3. PACOKS

Input : keyword search $K = \{k_1, k_2, ..., k_m\}$
Output: the best answer $result\_tree$

begin
  construct complete graph G_K with $k_i (\in K)$ as vertex;
  foreach $e(k_i, k_j) \in G_k$ **do**
    $s_tree(k_i, k_j) \leftarrow$ACOKS$(k_i, k_j)$;
    $weight(k_i, k_j) \leftarrow$COST$(s_tree(k_i, k_j))$;
  $min_tree \leftarrow$MINTREE(G_K);
  foreach $e(k_i, k_j) \in min\_tree$ **do**
    $min\_tree \leftarrow$ REPLACE$(min_tree, e(k_i, k_j), s_tree(k_i, k_j))$;
  $result\_tree \leftarrow min\_tree$;

Algorithm 3 shows the main steps of PACOKS, in which the functions are as follows:

- COST(s_tree): compute the cost(or weight) of the Steiner tree s_tree;
- ACOKS(k_i, k_j): the ant-colony-optimization based algorithm, as is shown in Algorithm 1, for keyword pair$\{k_i, k_j\}$;
- MINTREE(G_K): compute the minimum spanning tree of G_K;
- REPLACE(min_tree, $e(k_i, k_j)$, $s\_tree(k_i, k_j)$): replace the edge $e(k_i, k_j)$ in min_tree with $s\_tree(k_i, k_j)$.

Since the three algorithms, COST, MINTREE, and REPLACE, are so simple, they are not given in detail here.

Example 3. Figure 2 is an example explaining the process of PACOKS, which includes the following steps:

- Step 1: Given a keyword search K, including four keywords k_1, k_2, k_3 and k_4. Construct a complete graph G_K, with k_1, k_2, k_3 and k_4 as vertexes;

– Step 2: Call the algorithm ACOKS(k_1, k_2) for the edge $e(k_1, k_2)$, so as to get the minimum Steiner tree $s\_tree(k_1, k_2)$ containing the keywords k_1 and k_2; Similarly, we can get $s\_tree(k_1, k_3)$, $s\_tree(k_1, k_4)$, $s\_tree(k_2, k_3)$, $s\_tree(k_2, k_4)$ and $s\_tree(k_3, k_4)$;
– Step 3: Compute the cost of $s\_tree(k_1, k_2)$, denoted g_1, as the weight of the edge $e(k_1, k_2)$ in G_K. Similarly, we can get g_2, g_3, g_4, g_5 and g_6 as the weights of the other edges in G_K ;
– Step 4: Compute the minimum spanning tree min_tree from G_K, which includes three edges, say, $e(k_1, k_2)$, $e(k_1, k_4)$ and $e(k_3, k_4)$;
– Step 5: For each edge $e(k_i, k_j)$ in min_tree, replace $e(k_i, k_j)$ with $s\_tree(k_i, k_j)$. Therefore, $e(k_1, k_2)$, $e(k_1, k_4)$ and $e(k_3, k_4)$ are replaced with $s\_tree(k_1, k_2)$, $s\_tree(k_1, k_4)$ and $s\_tree(k_3, k_4)$ respectively. Figure 2(b) shows $s\_tree(k_1, k_2)$, $s\_tree(k_1, k_4)$ and $s\_tree(k_3, k_4)$, where $u(k_i)$ represents a node containing the keyword k_i. There exist common nodes, i.e., u_1 and $u(k_1)$, between $s\_tree(k_1, k_2)$ and $s\_tree(k_1, k_4)$, and u_3,$u(k_4)$ between $s\_tree(k_1, k_4)$ and $s\_tree(k_3, k_4)$. Therefore the three Steiner trees can be combined into one result tree based on their common nodes. □

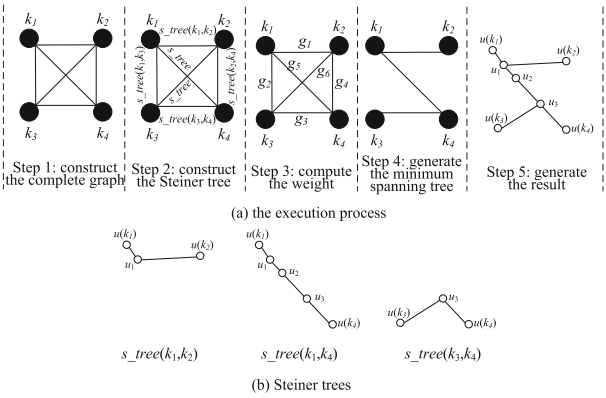


Fig. 2. The execution process of PACOKS.

Note: as Fig. 2 shows, in PACOKS, it only contains two keywords, k_i and k_j, whenever it calls the algorithm ACOKS. If a search K contains more than two keywords, it will call the algorithm ACOKS for each pair of keywords in K. Fortunately, a user will not start a search containing too many keywords.

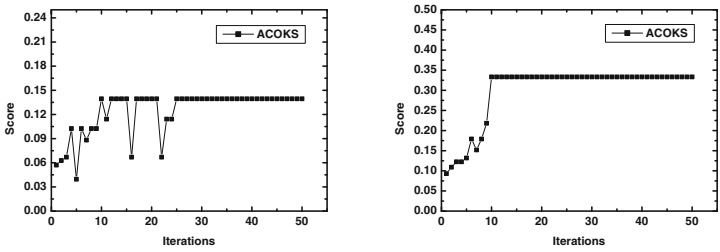
5 Experimental Study

Here we report the performance evaluation of our method. The algorithms are implemented in JAVA. All the experiments were conducted on Intel i7-2600 3.40 GHz CPU, 16.0 GB memory DELL OptiPlex990 PC running Windows Server 2003 and Oracle 11g. As with most other work, we downloaded the

DBLP data (http://dblp.uni-trier.de/xml/) as the testing datasets. The DBLP database involves about 800MB data in the form of XML documents, which are uploaded to the Oracle database through a simple program developed by us. In this way, relational database called DBLP in Oracle database can be populated. Then we write a program to generate data graphs from the relational database DBLP, and put them in the memory. For DBLP, the data graph includes 7,270,404 nodes and 9,047,382 edges. The whole processes of generating the data graphs from DBLP take 89 s. Another program is written to automatically extract terms from DBLP, and a total of 534,124 terms are found. Then an inverted index is built for all the terms, which takes 105 seconds. We compare our methods with BANKS [2] and BLINKS [5].

5.1 Algorithm Training

Algorithm Convergency. We select many keyword pairs to run against ACOKS in the experiments, but only part of the experimental results will be presented here since they all share similar features. The results of two keyword pairs are reported here, i.e., $K_1 = \{Database, Design\}$, $K_2 = \{Zhang, Ullman\}$. Figure 3 shows the relationship between the score of the optimal Steiner tree and the number of iterations of ACOKS, in which the scoring method of BANKS is adopted. We can see from Fig. 3 that the result quality is improved step by step during the algorithm training process, and it can achieve high result quality after the algorithm converges.



(a) K_1 (b) K_2

Fig. 3. The convergence curve of the algorithm training process

Compression Ratio of Pheromone Matrix. During the process of algorithm training, we analyzed the compression ratio of pheromone matrix for many paired keywords, so as to show that our method of constructing pheromone matrix based on paired keywords spanning graph is able to greatly reduce the space costs of the pheromone matrix. Here the compression ratio of the pheromone matrix, denoted f, is defined as N_{pksg}/N_{dg}, where N_{pksg} means the number of elements in the pheromone matrix based on a paired keywords spanning graph, and N_{dg} means the number of elements in the pheromone matrix based on a

data graph. For the pheromone matrix based on a data graph, when it is not optimized, the number of elements in the matrix is the square of the number of nodes in the data graph. In theory, the number of elements in the pheromone matrix based on paired keywords spanning graph is the square of the number of nodes in such graph. However, for a directed graph, an edge may exist from u to v, but there is no edge from v to u, so we will not store any pheromone information for the latter. Furthermore, in our experiments, the maxium step number an ant may move forward, is limited to 50, which is large enough to satisfy any requirements in real applications. Based on the above two aspects, the pheromone matrix in our method can be further simplified. We here select 8 groups of paired keywords, and Table 1 shows the pheromone matrix compression ratios for them. From Table 1, we can see that our method of constructing a pheromone matrix based on paired keywords spanning graph is able to achieve a very high compression ratio. Through our method, all the pheromone matrices for various paired keywords are able to be memory-resident so as to greatly enhance the performance of ACOKS.

Table 1. Pheromone matrix compression ratio

K	N_{ant}	N_{dg}	N_{kp}	$f(\times 10^{-10})$
adriano,gianluca	3029	7270404^2	90735	17.2
alfred,ullman	35176	7270404^2	698974	132
design,database	4095	7270404^2	120231	22.7
information,retrieval	3506	7270404^2	44981	8.51
manfred,joachim	5091	7270404^2	9608	1.82
nikolay,aphrodite	47171	7270404^2	894337	169
pagerank,algorithm	3747	7270404^2	132846	25.1
query,processing	3300	7270404^2	25169	4.76
relational,database	2199	7270404^2	2446	0.463
robert,stephan	7542	7270404^2	213302	40.4
search,keyword	3012	7270404^2	1815	0.343
sunita,soumen	26938	7270404^2	727019	138
vassilis,grigoris	2477	7270404^2	154577	29.2

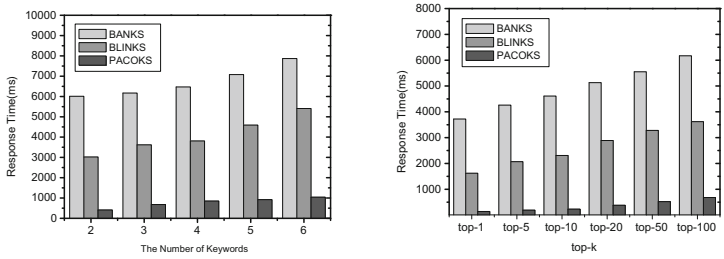
5.2 Search Efficiency

We now evaluate search efficiency of BANKS, BLINKS and PACOKS. We design 50 keyword searches, and the number of keywords, m, alternates between 2 and 6. After the response times for every search are acquired, they are averaged to be the evaluation index. Figure 4(a) shows the performance comparison results for BANKS, BLINKS [5] and PACOKS. We can make the following observations:

– PACOKS can outperform both BANKS and BLINKS for various keyword number. When $m = 2$, it takes more than 6,000 ms for BANKS to return the

answers, and approximately 3000 ms for BLINKS on the DBLP dataset. While PACOKS only runs for 410 ms to get the results. It is because PACOKS is an ant-colony-optimization-based algorithm, which is able to take full advantage of parallel execution among various ants. Furthermore, when PACOKS provides online service, it has gone through the algorithm training period, and the later searches can make full use of the previously accumulated pheromone information, thus being able to achieve high efficiency during its calling ACOKS for each paired keywords $\{k_i, k_j\}$. BANKS and BLINKS, however, do not have such feature as sharing high costs among various searches.

- It performs best for PACOKS when the keyword number is 2, because there is no result subtree merging process. When the keyword number is 3, 4, 5 or 6, the search in PACOKS is split into several keyword pairs and there exists result subtree merging process, which leads to the increase of search response time.
- PACOKS is with good scalability. When m changes from 2 to 6, the response time for PACOKS only increases from 410 ms to 1,040 ms. The increase in time is mainly due to the subtree merging process.



(a) various keyword number (b) top-k search, $m = 3$

Fig. 4. Response time comparison

Furthermore, we compare the three algorithms on the aspect of response time for top-k answers in Fig. 4(b). One can see that our method is able to achieve much better performance than BANKS and BLINKS. For example, PACOKS only runs 190 ms to return the top-5 answers on the DBLP dataset, while BANKS and BLINKS take more than 4,000 ms and 2,000 ms respectively. This is also attributed to the "cost sharing mechanism" adopted by our method.

6 Related Work

Keyword search over structural data (e.g., relational database) and semi-structural data (e.g., XML and HTML documents), has received much attention in the database community in recent years. Existing approaches to support keyword search over relational databases can be typically classified into two categories, namely those based on data graph (e.g., BANKS [2] and EASE [9]) and those based on schema graph (e.g., DISCOVER [6] and DBXplorer [1]).

BANKS [2] is a representative of data-graph-based method, which uses a back expanding search algorithm to traverse the graph. However, back expanding search algorithm deteriorates a lot when it meets a node with large in-degrees. BANKS-II [7] improves the performance of BANKS through bidirectional search. In BLINKS [5], He et al. they proposed a partition-based method and a strategy called cost-balanced expansion, and at the same time, used an additional bi-level index to speed up the traversing process, which achieves better search efficiency than BANKS-II. Li et al. [9] proposed the concept of "r-radius Steiner tree", and keyword search problem is converted into a r-radius Steiner tree problem, which is able to identify some complicated and meaningful structures from database. Simple structures, such as tuple joining trees, are the focus of the literature in the initial research stage. Then research interest is extended to search for more sophisticated structures, such as r-radius Steiner tree [9], community, frequent co-occurring term [14].

DISCOVER [6] and DBXplorer [1] run SQL statements directly against database, while other work, such as [12], take middleware to execute SQL statements. DISCOVER and DBXplorer only focus on the search efficiency instead of result effectiveness, so Liu et al. [10] proposed a new weighted ranking mechanism and carried out extensive experiments to improve effectiveness of keyword search. Luo et al. [11] discussed how to support efficient top-k keyword search over relational database.

7 Conclusion

In this paper, we have addressed the problem of keyword search over relational databases. We proposed ant-colony-optimization-based algorithm, called ACOKS, to deal with minimum group Steiner tree problem. To limit the cost of a single search to a very low level, we further proposed progressive-ant-colony-optimization-based algorithm, called PACOKS, which aims to achieve the final global optimal solution, through the cooperation of searches continuously arriving at the system along the time. Thus the huge costs for finding the global optimal solution, are shared among large amounts of searches, and a single search can achieve very fast response time. The experimental results show that our proposed scheme can achieve both high efficiency and effectiveness.

References

1. Agrawal, S., Chaudhuri, S., Das, G.: DBXplorer: A system for keyword-based search over relational databases. In: Proceedings of ICDE, pp. 5–16 (2002)
2. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using banks. In: Proceedings of ICDE, pp. 431–440 (2002)
3. Djebali, S., Raimbault, T.: SimplePARQL: a new approach using keywords over SPARQL to query the web of data. In: Proceedings of the 11th International Conference on Semantic Systems, SEMANTICS 2015, Vienna, Austria, 15–17 September 2015, pp. 188–191 (2015)

4. Gutjahr, W.J.: A graph-based ant system and its convergence. Future Gener. Comput. Syst. **16**(1), 873–888 (2000)
5. He, H., Wang, H., Yang, J., Yu, P.S.: BLINKS: ranked keyword searches on graphs. In: Proceedings of SIGMOD, pp. 305–316 (2007)
6. Hristidis, V., Papakonstantinou, Y.: DISCOVER: Keyword search in relational-databases. In: Proceedings of VLDB, pp. 670–681 (2002)
7. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: Proceedings of VLDB, pp. 505–516 (2005)
8. Kim, I.-J., Whang, K.-Y., Kwon, H.-Y.: SRT-rank: Ranking keyword query results in relational databases using the strongly related tree. IEICE Trans. Inf. Syst. **97**(D(9)), 2398–2414 (2014)
9. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: Proceedings of SIGMOD, pp. 903–914 (2008)
10. Liu, F., Yu, C., Meng, W., Chowdhury, A.: Effective keyword search in relational databases. In: Proceedings of SIGMOD, pp. 563–574 (2006)
11. Luo, Y., Wang, W., Lin, X., Zhou, X., Wang, J., Li, K.: Spark2: Top-k keyword query in relational databases. TKDE **23**(12), 1763–1780 (2011)
12. Markowetz, A., Yang, Y., Papadias, D.: Keyword search on relational data streams. In: Proceedings of SIGMOD, pp. 605–616 (2007)
13. Park, C.-S., Lim, S.: Effective keyword query processing with an extended answer structure in large graph databases. IJWIS **10**(1), 65–84 (2014)
14. Tao, Y., Yu, J.X.: Finding frequent co-occurring terms in relational keyword search. In: Proceedings of EDBT, pp. 839–850 (2009)
15. Yang, W., Guo, T.: An ant colony optimization algorithm for the minimum steiner tree problem and its convergence proof. Acta Math. Appl. Sinica **29**(2), 352–361 (2006)
16. Zhou, J., Liu, Y., Yu, Z.: Improving the effectiveness of keyword search in databases using query logs. In: Li, J., Sun, Y., Yu, X., Sun, Y., Dong, X.L., Dong, X.L. (eds.) WAIM 2015. LNCS, vol. 9098, pp. 193–206. Springer, Heidelberg (2015). doi:10.1007/978-3-319-21042-1_16