DSI: A Method for Indexing Large Graphs Using Distance Set

Yubo Kou, Yukun Li, and Xiaofeng Meng

Renmin University of China, Beijing, China
{kouyubo,liyukun,xfmeng}@ruc.edu.cn
http://idke.ruc.edu.cn/

Abstract. Recent years we have witnessed a great increase in modeling data as large graphs in multiple domains, such as XML, the semantic web, social network. In these circumstances, researchers are interested in querying the large graph like that: Given a large graph G, and a query Q, we report all the matches of Q in G. Since subgraph isomorphism checking is proved to be NP-Complete[1], it is infeasible to scan the whole large graph for answers, especially when the query's size is also large. Hence, the "filter-verification" approach is widely adopted. In this approach, researchers first index the neighborhood of each vertex in the large graph, then filter vertexes , and finally perform subgraph matching algorithms. Previous techniques mainly focus on efficient matching algorithms, paying little attention to indexing techniques. However, appropriate indexing techniques could help improve the efficiency of query response by generating less candidates. In this paper we investigate indexing techniques on large graphs, and propose an index structure DSI(Distance Set Index) to capture the neighborhood of each vertex. Through our distance set index, more vertexes could be pruned, resulting in a much smaller search space. Then a subgraph matching algorithm is performed in the search space. We have applied our index structure to real datasets and synthetic datasets. Extensive experiments demonstrate the efficiency and effectiveness of our indexing technique.

Key words: Graph Indexing, Distance Set

1 Introduction

In the web age we have witnessed a great increase in modeling data as graphs in many domains, such as XML, the semantic web, biological networks, social networks, dataspace. People often need to query like that: (1) In the chemical compound database, given a specific chemical structure, we need all the compounds that contain the given chemical structure. (2) In the social network, given a specific group description, we need all the corresponding occurrences.

Actually, the database community has long been interested in querying graph databases[2, 5, 6, 7, 8, 9, 10, 11, 13, 19]. The problem previous studies mainly focus on could be defined as: Given a graph database G, and a query Q, we report all the graphs that contain Q as a subgraph. Since subgraph isomorphism checking is NP-complete[1], which means it is infeasible to scan the whole graph database for answers. To efficiently answer queries, the database community usually adopt a "filter-verification" approach, which means graph databases should be indexed at first. However, with the development of bio-informatics and social science applications, many large graphs are emerging. These graphs usually contain tens of thousands of vertexes. On large graphs there are also needs for queries which could be formalized as: Given a large graph G and a query graph Q, we report all the matches of Q in G. This problem is challenging in several aspects:

First, previous techniques about querying graph database couldn't be applied directly to this problem: previous studies aim at finding which graph contains the given query, while this problem is about the exact positions of matches in a graph. Therefore, it is challenging to figure out what to index.

Second, data mining techniques such as frequent subgraph couldn't be directly applied since it is difficult to tell what is frequent in a single graph[3,4].

Third, appropriate indexing techniques are needed for answering queries efficiently and effectively. There have already been some works[14, 15, 16, 17, 18] on this topic. Most of these works propose to first index neighborhood of each vertex in the graph, prune false vertexes through their index, and finally perform subgraph matching algorithm to find each distinct matches. In these works, most attention are paid to efficient matching, while little paid to indexing techniques. However, appropriate indexing techniques could help improve the efficiency of matching algorithm. For a query of size n(the size of a graph refers to the number of nodes), the size of candidate sets for the query nodes are k_1, k_2, .. k_n, respectively. The time cost of subgraph matching algorithm is O($k_1 k_2..k_n$). Thus shrinking each candidate set could help subgraph matching algorithm achieve high performance, especially when n is large.

In this paper, we propose a novel indexing technique named as distance set index(DSI). We have applied DSI to DBLP dataset and synthetic dataset. Our experiments demonstrate that our indexing technique is effective and efficient. Utilizing our indexing technique could generate much less candidates, thus accelerate query response.

To summarize, our contributions are listed as follows:

1. We introduce a new feature (distance set)to represent each vertex's neighborhood. Experiments demonstrate this feature's expressivity.

2. We propose a novel indexing technique, distance set index(DSI). This index utilize distance set for subgraph matching in a large graph, which could help accelerate the query processing.

3. We perform extensive experiments to demonstrate the effectiveness and efficiency of DSI. The experimental results also show it is quite profitable to investigate how to index vertex's neighborhood.

The remainder of this paper is organized as follows: Section 2 presents related work. Section 3 defines the preliminary concepts and problem definition. Section 4 introduces our database preprocessing step. Section 5 describes query processing. Experimental results are delivered in Section 6. Section 7 concludes our work and discusses about future work.

2 Related Work

The database community has a long-term interest in querying the graph database. Most of them investigate feature selection strategies to achieve better pruning power. [2] proposes Graphgrep to enumerate all the pathes within a given length as features. However, pathes is too simple to capture the characteristics of graphs. [13] presents node neighbor tree(NNT) to capture local structure of each vertex, and transform NNT to node projected vector(NPV) for efficient comparing. In these feature selection strategies, a frequent subgraph based approach is very popular. [5] uses gIndex to index all the frequent subgraphs as features. Later, [10] presents FG-index, [11] proves that indexing frequent trees and some special graphs are more beneficial than indexing graphs. There are also some works considering a general view. [6] considers features' relations. [7] and [8] index structures of graphs. Both works assume the graph size be small. [9] proposes to transform graphs into sequences.

There exist some works investigating querying large graphs. [18] proposes to partition the graph first, then process the partitions. [14] studies how to efficiently answer queries without database preprocessing. [15] proposes to index subgraphs composed of each vertex and its adjacent vetexes, and investigates matching algorithms. They further introduce some optimization ideas. [16] also proposes to index neighborhood. What differs is its aim at approximate subgraph matching. [17] proposes neighboring discriminating substructure(NDS) distance as a distance measure. It captures NDS for each pair of vertexes within a given distance to facilitate its matching algorithm. This approach is interesting, however, merely depending on local structures might introduce some false positives. There are also some works focusing on figuring out what is frequent in a large graph[3, 4].

3 Preliminaries

In this section, we introduce the preliminary definitions and concepts, as well as problem definition. We restrict our discussion on undirected acylic labeled graphs. However, our discussion could be easily extended to directed graphs.

Definition 1 (Labeled Graph). *A labeled graph is denoted as* $G=\langle V(G),E(G),\sum, L\rangle$, *where* $V(G)$ *is the set of vertexes,* $E(G)\subseteq V(G)\times V(G)$ *is the set of edges,* $\sum$ *is the set of labels and* L *is the injective function* L*:* $V(G)\rightarrow\sum$*, which assigns each vertex a label from* $\sum$*. Additionally, each vertex in the graph has a unique id.*

Definition 2 (Graph Isomorphism). *Graph isomorphism is a bijection f:* $V(G)\leftrightarrow V(G')$*. Given two labeled graph G and G', if they are isomorphic, they* *satisfy such conditions: (1) $\forall$ u $\in$ V(G), L(u)=L'(f(u)); (2) $\forall$ u, v $\in$ E(G) $\Leftrightarrow$ (f(u),f(v)) $\in$ E(G').*

Graph S is **subgraph isomorphic** to G, if S is isomorphic to at least one subgraph G' of of G. G' is a match of S in G.

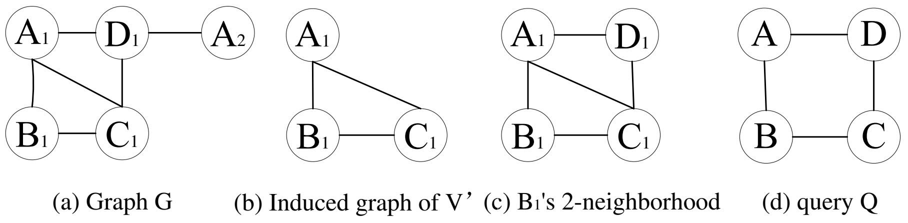

Fig. 1. Examples for preliminaries

Definition 3 (Shortest Distance). *Given vertexes u and v in a graph G, the shortest distance between them is the number of edges in the shortest path between u and v.*

For example, in figure 1(a), the shortest distance between B_1 and D_1 is 2, since the shortest path between them is $B_1A_1D_1$, or $B_1C_1D_1$, both composed of 2 edges.

Definition 4 (Induced Graph). *Given a graph G, and a set of vertexes, denoted as V', V' belongs to V(G), an induced graph of V' is the subgraph composed of V' and any edge whose ends belong to V'.*

Take figure 1(b) as an instance, given V'=$\{A_1, B_1, C_1\}$, the induced graph of V' is shown in figure 1(b).

Definition 5 (K-Neighborhood). *Given a vertex v in a graph G and an integer k, the k-neighborhood of v, denoted as k-N(v), is the induced graph of a vertex set which contains v and the vertexes whose shortest distance from v is no more than k.*

For example, given k=2, B_1's 2-neighborhood is shown in figure 1(c). The graph in figure 1(c) shows the local structure of vertex B_1, which plays an important role in identifying matching vertexes.

Problem Definition(Subgraph Matching): Given a large graph G and a query graph Q, we report all the matches of Q in G.

For example, in figure 1(d), there is one match of Q, which is $\{A_1, B_1, C_1, D_1\}$.

The processing of graph queries contains two steps:

1, Database Preprocessing. This step preprocesses the database graphs by building an index for each vertex. It is important to figure out which feature should be selected, since right feature selection strategy could achieve high pruning power, as well as low cost.

2, query processing. This step contains two substeps. First, **Node Matching**. We retrieve candidate vertexes in G for each node in Q, respectively. These candidates compose a search space for subgraph matching. Second, **Subgraph Matching**. we perform subgraph matching algorithm in the search space to get final matches for Q.

In this work, we mainly study graph indexing techniques to help accelerate query processing.

4 Database Preprocessing

To respond to on-line queries efficiently and effectively, it is popular to build indexes off-line, trading index space for query response time. In this section, we describe our strategy at the preprocessing step. First we introduce a new feature: distance set. Second, we extend the distance set with discriminative substructures. Finally, we describe the index structure, and how to construct the index structure.

4.1 Indexing Unit

Given a node u in Q, a node v in G, and their corresponding k-neighborhoods, the most accurate way to identify whether v is a match of u, is to perform subgraph isomorphism checking between their corresponding neighborhoods. This way takes too much computational cost, since subgraph isomorphism checking is NP-Complete[1]. Therefore, it is necessary to select appropriate features to represent k-neighborhood and prune false vertexes by these features. These features should satisfy two requirements: (1) These features should introduce as less false positives as possible, in other words, these features should have high expressivity, which implies high pruning power; (2) the cost of computing these features should be feasible. To capture the neighborhood information around vertex v, we present a new feature: distance set. First we will give the definition of distance set.

Definition 6 (Distance Set). *Given a vertex v, and a vertex u in its k-neighborhood k-N(v), the distance set between v and u record all the distinct lengths of pathes between them, denoted as DS(v, u). Here pathes do not contain repeated vertexes.*

For example in figure 2(a), given two vertexes A_1 and B_1, DS(A_1, B_1)={1, 2, 3}. Since there are three distinct pathes from A_1 to B_1: A_1B_1, $A_1C_1B_1$ and $A_1D_1C_1B_1$. To profile the neighborhood of a given vertex v, we record each neighbor vertex u's label, and DS(v, u). Suppose there is more than one path of a particular length, say 2, distance set would contain only one 2. Therefore, distance set does not contain duplicates.

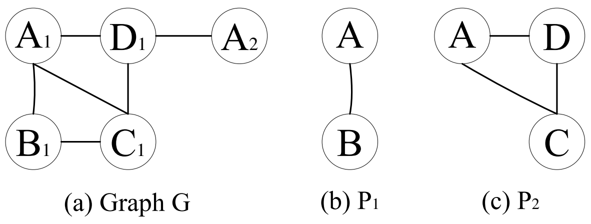

Fig. 2. Examples for distance set

Formally, the conditions for matching a query node u to a vertex v in the database graph is:

for each node u' in k-N(u), there is a vertex v' in k-N(v) such that:

(1) L(u')=L(v')

(2) DS(u,u') $\subseteq$ DS(v,v').

For efficiency's sake, we represent DS in bit vector form, namely, distance set vector(DSV). Each position of a DSV denotes whether the DS contains the specific number. In the foregoing example, DSV(A_1, B_1)=(0, 1, 1, 1). The first position always records whether '0' exists. There is no '0' in DS(A_1, B_1), so the value of the first dimension is 0. There is one '1' in the DS, so the value of the second dimension is 1. The other dimensions' values could be computed in the same way. Thus, we change the condition (2) to that DSV(u,u') dominates DSV(v,v'). Here "dominates" means the value of each dimension in DSV(u,u') is no more than DSV(v,v').

Distance set is vertex-specific in the sense that it captures the structure between v and one vertex in its neighborhood. While vertex-specific distance set provides high expressivity, it could be further improved by utilizing a couple of neighbor vertexes which are inter-connected, in other words, a substructure. To capture more information, such as the structural information between v and one of its discriminative substructures. We extend distance set(DS) to discriminative substructure distance set(DDS). The definition of discriminative substructure could be found in [5]. With a set of k-neighborhoods, we could use frequent subgraph mining techniques to find frequent substructures and select the discriminative ones.

Definition 7 (Discriminative Substructure Distance Set). *Given a vertex v, its k-neighborhood k-N(v), and a discriminative substructure P , P belongs to k-N(v), the discriminative substructure distance set, denoted as DDS(v, P), is a set of distance sets, where each distance set contains all the distinct pathes' distance from a node in P to v. To be specific, none of these pathes contains an edge which belongs to P.*

In this definition, we emphasize the path selecting rule that given a discriminative substructure P, every path from one node of P to the root should not contain any edge of P. This property is important, since it makes discriminative substructure distance set more discriminative. For example in figure 2, let P_1 and P_2 be discriminative substructures. DDS(B_1, P_1)={A.DS'(A_1, B_1), B.DS'(B_1, B_1) }={ A.(2, 3), B.(0)}. DDS(B_1, P_2)={ A.DS'(A_1, B_1), C.DS'(C_1, B_1), D.DS'(D_1, B_1)}=(A.(1), C.(1), D.$\emptyset$).

To represent the distance set of discriminative substructure in one vector, it is essential to order vertexes of discriminative substructures in a unique way. There have been plenty of works discussing canonical form of a graph[5,6,9]. In this paper, since the discriminative substructures we deal with are generally of small size, we adopt a simple lexicographical order.

For example, in figure 2, since P_1 contains two nodes, labeled by A and B, we set A.DS'(A_1, B_1) ahead of B.DS'(B_1, B_1), since A is prior to B lexicographically. Thus we get DDSV(B_1, P_1)=(0, 0, 1, 1, 1), DDSV(B_1, P_2)=(0, 1, 0, 1). In DDSV(B_1, P_1), the first four dimensions correspond to A.DS'(A_1, B_1), while the last three dimensions correspond to B.DS'(B_1, B_1).

Given the definition of discriminative substructure distance set, the pruning condition is transformed to that: for each discriminative substructure P in k-N(u), there is a corresponding discriminative substructure P in k-N(v) such that:

(1) P' is isomorphism to P

(2) DDSV(u,P) is dominated by DDSV(v,P).

There are plenty of works discussing discriminative substructures[5, 11, 17]. In this paper, we adopt the way proposed by [17]. That is, we randomly select 100 k-neighborhoods and conduct frequent subgraph mining algorithm in these k-neighborhoods. After that, we select the most discriminative substructures of size 3 or 4.

To facilitate the process of index construction and query processing, meanwhile keep index size small, we store each DDSV in a bit vector. In the next subsection, we will introduce our index structure.

4.2 Index Structure

In this subsection we consider the index structure to implement the distance set index. We propose a bi-level index structure for distance set index. The first level of the index structure is a list of discriminative substructures. This list contains each distinct discriminative substructure as a unit. To fully cover the large graph, this list also contains each substructure of size 1, namely, each label. The second level of the index structure is bitmaps which contain bit vectors of vertexes who's k-neighborhood contains the discriminative substructure as a subgraph.

For example, given a graph G in figure 2(a), and two discriminative substructures P_1 and P_2 in figure 2(b) and 2(c), our index structure is shown in figure 3. Consider P_1's bitmap, since DDSV(B_1, P_1)=(0, 0, 1, 1, 1), this DDSV is recorded in the second row of the bitmap. Note that the former DDSV only has five dimensions, while in the bitmap, it has eight. We make this extension to achieve uniformity for all the bitmaps.

To index the distance set feature of each vertex, for each discriminative substructure P, we first find vertexes whose k-neighbor contains P. Then we compute

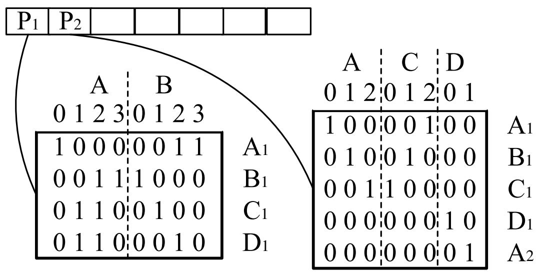

Fig. 3. Distance Set Index

the discriminative substructure distance set for each vertex, and store that vertex and its DDSV in P's bitmap.

5 Query Processing

In this section, we discuss query processing. This step contains two substeps: First, we perform node matching algorithm, which finds candidate vertexes for each node in Q. These candidate vertexes form a search space for later subgraph matching. Second, after we've prepared a set of candidate vertexes for each node, we perform subgraph matching algorithm.in the search space.

Algorithm 1 Node Matching

Input: A large graph G and its DSI, a query Q, a range value k
Output: A search space

```
 1: procedure
 2:     for each node $u_i$ in query do
 3:         C($u_i$) ← {v| v ∈ V(G), L(v)=L($u_i$)};
 4:         for each discriminative substructure P in k-N($u_i$) do
 5:             M($u_i$,P)={v| v ∈ V(G), DDSV(v, P) dominates DDSV($u_i$, P)};
 6:             C($u_i$)=C($u_i$) ∩ M($u_i$,P);
 7:         end for
 8:     end for
 9:     Output C($u_1$)×C($u_2$)×..×C($u_2$);
10: end procedure
```

At the first substep, we first generate DSVs for each feature of node v in Q. After that, for each node v, we perform vertex matching algorithm, shown below. In Algorithm 1, C(u_i) is a set of vertexes who satisfy the matching condition of u_i. M(u_i,P) is a vertex set, where each vertex v's DDSV(v, P) dominates DDSV(u_i,P).

Algorithm 1 first initialize each node's candidate set at step 3. Then for each discriminative substructure P in k-N(u_i), we search our DSI for matching entries, and intersect with C(u). Finally, candidate set is decided when all the features are checked.

At the second substep, we adopt the subgraph matching algorithm proposed by [15]. Generally speaking, this algorithm checks the search space in a depth-first manner for matchings between the graph pattern and the graph. As stated before, our main concern is how indexing techniques could contribute to generating less candidates for each node. In a much smaller search space, the process of subgraph matching is naturally more efficient.

6 Experiments

Our method is written in Java JDK 1.6. All experiments are carried out on a machine with Intel core 2 duo CPU 2.0GHz, 2GB memory and Microsoft Windows XP.

In our experiments, we evaluate our distance set index on real large graph and synthetic graphs. We extract a real large graph from DBLP dataset[20]. In this graph, each vertex represents an author, each edge denotes the coauthor relationship between the two corresponding authors. For each author, the label is set as the conference where he has the most publications. The DBLP graph contains 776068 vertexes, 2498532 edges, and 3512 distinct labels.We also evaluate our methods on synthetic graphs. The synthetic graphs are generated with a heavy-tailed distribution by [21]. In synthetic graphs, we distribute the labels according to Zipf's law, that is, probability of the x^{th} label is proportional to x^{-1}. The default parameter settings are: the number of vertexes is 10000, the exponent is 2.5, the minimum degree and maximum degree is 1 and 1000, the average degree is 5, and the number of labels is 100. In the experiments, we randomly select subgraphs as queries.

Table 1. Parameter Setting

k \ maxL	1	2	3
1	170.2	170.2	170.2
2		123.8	123.8
3			123.3

Next, we conduct experiments to fix the value of k. Selecting a radius of k-neighborhood involves pruning power and index construction cost. To select the most appropriate value of k, we evaluate our methods with different values on DBLP graph and synthetic graph. In our method, there is a hidden parameter, which is the max length of the distance, denoted as maxL. maxL is important in large graph circumstances, due to the special property of large graph. Large graphs are generally sparse, and locally dense. In our experiments, we randomly pick 20 vertexes and their k-neighborhood from the synthetic graph, and retrieve the candidates for each vertex. Table 1 shows the average size of the 20 candidate sets under varying k value and maxL value. According to our experimental results, the candidate set's size does not increase much with k, so we fix both k and maxL at 2.

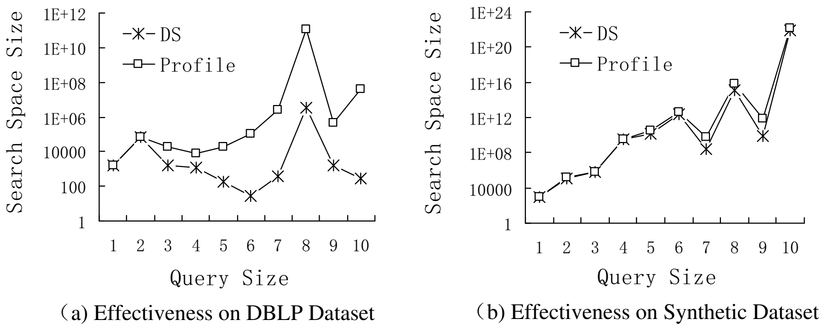

Fig. 4. Effectiveness

We compare DSI's effectiveness with another feature strategy proposed by [15], which is named profile. To get the profile of a vertex, They first transform each vertex's 1-neighborhood into a sequence in lexicographic order. Thus the pruning condition is whether one vertex's profile is a subsequence of the other one's. To compare these two methods, we use search space product measure, which is defined as:

$P=| C(u_1)| \times| C(u_2)| \times..\times| C(u_n)|$

In this experiment, we randomly generate 10 queries for each size in (1, 10). After retrieving the candidate set for each node in query, we compute search space product for each query, then compute the average search space product of all the 10 queries for each size. Figure 4 shows the search space product measure of different methods on two datasets. Our method greatly outperforms profile on DBLP dataset. On the synthetic dataset, our method performs slightly better than profile. This is due to the synthetic graph's particular characteristics that most vertexes whose degree is low, are usually connected to vertexes whose degree is high.

We also measure the index size and index construction time of DSI on these two datasets. The size of DSI on DBLP graph is about 1GB, and it takes us about 100 minutes to build. In synthetic graph, we spend 113 seconds building DSI, whose size is 72.3MB.

Table 2. Effectiveness with Different Parameters on Synthetic dataset

Graph Size	Query Size	Exponent	Min	Max	Avg. Degree	Label Number	DS	Profile
10000	5	1.5	1	1000	5	100	3.81E+5	1.04E+6
10000	5	2	1	1000	5	100	9.74E+11	1.08E+12
10000	5	2.5	1	1000	5	100	1.26E+10	3.76E+10
10000	5	4	1	1000	5	100	4.75E+6	1.69E+7
10000	5	2.5	1	100	5	100	1.06E+9	8.07E+9
10000	5	2.5	1	500	5	100	4.10E+7	1.99E+8
10000	5	2.5	1	1000	3	100	5.83E+4	4.92E+5
10000	5	2.5	1	1000	5	200	1.58E+4	3.92E+4

Next, we vary each parameter of the synthetic dataset at a time to conduct our method against profile. The results is shown in table 2. To specify, we don't test different graph sizes or query sizes, since the scalability has been shown on DBLP dataset. In addition, it is natural to assume that there exist leafs, therefore we don't vary the minimum degree. The last two columns show experimental results of two methods. In this experiment, we find that both methods work well while the graph is more sparse, and the label number is larger, since these characteristics would make each node more selective. Generally speaking, our method performs better.

7 Conclusion and Future Work

In the web age, large graphs are evolving in every aspect of our society, meanwhile the problem of querying large graphs has been aroused. In this paper, we investigate indexing techniques applied to large graphs. We propose distance set index to capture each vertex's local structure. We have applied our method to DBLP dataset,and a synthetic dataset. Extensive experiments show that our distance set index is efficient and effective. Although utilized in a two-way matching algorithm, our method could also be applied to dynamic matching algorithms, such as proposed by [17]. In future work, we plan to investigate utilizing discriminative substructures as basic units in the search space instead of vertexes, and compressing index size of DSI.

8 Acknowledgements

This research was partially supported by the grants from the Natural Science Foundation of China (No.60833005); the National High-Tech Research and Development Plan of China (No.2007AA01Z155); and the Doctoral Fund of Ministry of Education of China (No. 200800020002).

References

1. S. A. Cook. The complexity of theorem-proving procedures. STOC 1971: 151-158.
2. D. Shasha, J. T. L. Wang and R. Giugno. Algorithmics and Applications of Tree and Graph Searching. PODS 2002: 39-52.
3. M. Kuramochi and G. Karypis. Finding Frequent Patterns in a Large Sparse Graph. DMKD 2004: 243-271.
4. M. Fiedler and C. Borgelt. Subgraph Support in a Single Large Graph. ICDM Workshops 2007: 399-404.
5. X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. SIGMOD 2004: 335-346.
6. X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. SIGMOD 2005: 766-777.
7. H. He and A. K. Singh. Closure-tree: An index structure for graph queries. ICDE 2006: 38.
8. D. W. Williams, J. Huan and W. Wang. Graph Database Indexing Using Structured Graph Decomposition. ICDE 2007: 976-985.
9. H. Jiang, H. Wang, P. S. Yu and S. Zhou. GString: A Novel Approach for Efficient Search in Graph Databases. ICDE 2007: 566-575.
10. J. Cheng, Y. Ke, W. Ng. FG-Index: Towards Verification-Free Query Processing on Graph Databases. SIGMOD 2007: 857-872.
11. P. Zhao, J. X. Yu and P. S. Yu. Graph Indexing: Tree + Delta $\geq$ Graph. VLDB 2007: 938-949.
12. C. Chen, X. Yan, P. S. Yu, J. Han, D.-Q. Zhang and X. Gu. Towards graph containment search and indexing. In VLDB, 2007: 926-937.
13. C. Wang and L. Chen. Continuous Subgraph Pattern Search over Graph Streams. ICDE 2009: 393-404.
14. H. Tong, B. Gallagher, C. Faloutsos and T. E-Rad. Fast Best-Effort Pattern Matching in Large Attributed Graphs. In SIGKDD, 2007: 737-746.
15. H. He and A. K. Singh. Graphs-at-a-time Query Language and Access Methods for Graph Databases. SIGMOD 2008: 405-418.
16. Y. Tian and J. M. Patel. Tale A tool for approximate large graph matching. ICDE 2008: 963-972.
17. S. Zhang, S. Li and J. Yang. GADDI Distance Index based Subgraph Matching in Biological Networks. EDBT 2008: 192- 203.
18. L. Zou, L. Chen and Y. Lu. Top-K Subgraph Matching Query in A Large Graph. First PH. D. Workshop CIKM 2007: 139-146.
19. L. Zou, L. Chen, J. X. Yu and Y. Lu. A Novel Spectral Coding in a Large Graph Database. EDBT 2008: 181-192.
20. DBLP Dataset, `http://dblp.uni-trier.de/xml/`
21. F.Viger and M. Latapy. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. COCOON 2005: 440-449.