



PARALLELIZING TREE PROJECTION FOR CERTAIN AND UNCERTAIN GRAPH STREAMS

M Rajasekhara Babu¹, Rashmi rawat², Divya Vani .Y³, Komathi .S⁴

School of computer science & Engineering

VIT University

Vellore-632014, Tamil Nadu

India

Abstract: In today's era, concurrency is the best policy for designing a power efficient program. Concurrent programming is the form of programming in which we calculate a no. of computations separately. In this process we divide a large program into a no of small programs and execute them separately. Concurrency or parallelism helps us to execute a program more efficiently in the terms of power as well as speed. Here we can see that searching over graph databases has come to notice because from the recent analysis, it came into picture that graph databases are used in many fields such as analysis of chemical compounds, functional interpretation, data interpretation, pattern matching over users' visiting logs. Static graph stores all the steady or constant data but in the case of knowledge representation dynamic storage is required. The sub graphs within the graph keep on changing. That's why it becomes difficult to find the exact query stream sub graph. Therefore we propose a concept of Node Neighbor Tree to filter out the false streams of the query graph. Further we generate Node Projection Vectors for these Node Neighbor trees [1]. These NPV helps us to find the desired query streams in both certain and uncertain graphs. Here, we are parallelizing Tree Projection algorithm using Open mp support.

Keywords: Sub graph search, Node-Neighbour tree, Node Projection Vector, Certain graphs, Uncertain Graph

I. Introduction

Now days, graph has been used in many applications like spatial data mining, we represent data sets through graphs. In a chemical reaction we use graphs for finding new components in chemical analysis. In medical data analysis, we analyze images through graphs. In network data traffic, intrusion can be detected through graphs. Because of these applications now many problems arise as well as the solutions, such as in travelling salesman problem we find minimum distance using prims or krushkal algorithm. Same we can see for finding the sub graphs in certain and uncertain graphs. We use here three methods, finding the node neighbor tree, tree projection and dominant set cover algorithm. In which tree projection plays an important part. Here we are trying to enhance the speed of tree projection through parallelization.

II. Related Work

In their paper zhaonian zou,jianzhong Li,Hong Gao and Shuo Zhang proposed the paper named Mining Frequent Sub graph Pattern From Certain And Uncertain Graph Data[1]. They used mining techniques to detect the uncertain graphs [1,2,3]. Due to the graph incompleteness they found some problems of finding mined graph data patterns in an uncertain graph database [1,4,5]. Finally it can be concluded as NP-hard [1,8,10]. They introduced another efficient algorithm called an approximate algorithm to find the patterns of frequent subgraph with error tolerance and confident in that sub graphs [1,11] Finally they showed this algorithm is efficient, scalability and produce an effective result[1,9,12]. Many algorithms are used but three algorithms are more important for finding searching Sub Graph Patterns [1,13,14]. We are using Node neighbor algorithm, Tree projection algorithm, Dominant Set cover algorithm mainly used for searching sub graphs [1,15].

III. Proposed Algorithm

Tree Projection Algorithm: A graph is given and to search the sub graph patterns for given query, Tree projection algorithm is the intermediate algorithm used between Node Neighbor Tree algorithm and Dominated Set Cover algorithm. We are developing this algorithm and implementing the code and parallelizing using openmp input. For each edge we are going to calculate Tree Projection Vector and is also called as dimension, where Tree Projection Vector or dimension is a triple. The triple has the values level of the edge, two node labels of the corresponding edge. We are going to print the Tree Projection Vectors or dimensions of all distinct

edges in the Node Neighbor Tree till the depth of the tree is reached. This Tree Projection algorithm can be used for the Dominant Set Cover algorithm for finding dominant relationships between graphs and also for searching sub graph patterns.

NOTATION: Tree projection Vector is defined as a triple $\langle l, \text{node1}, \text{node2} \rangle$ where l is the level of node1 and node1, node2 are the node labels of the corresponding edge.

TREE PROJECTION

TREE PROJECTION VECTOR:

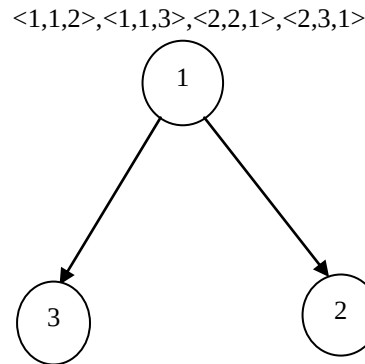


Fig.1.Generation of nodes

Algorithm:

PROCEDURE TREE-PROJECTION VECTOR

```

{
Input: A node neighbor tree having depth d;
Output: Tree projection vectors for tree
(1.)TPV<-0;
(2.)for each level l in the input
(3.)for each edge (vi,vj) at level l
(4.)find TPV[l,label(vi),label(vj)]<-NPT[l,label(vi),label(vj)]+1
(5.)return TPV
}
  
```

IV. Implementation

We give tree as input and get tree projection vectors as output. We are going to implement code for Tree projection algorithm in c language. Here at each level we are projecting the vectors of the tree. The tree projection vector consists of level of the starting node of an edge, node labels of corresponding edge. First we are going to check whether the root is null or not. If the root is null we will not print anything. If the root is not null, then check whether it is having left child or not. If it is having left child then print the level of root, node label, left child label as projection vector. Similarly check whether the root is having right child. If the root is having right child then print the level of root, root node label, right child label as projection vector. The converse of it is also done for printing the tree vectors. Repeat the above process by considering the left child as root and right child as root for printing the tree projection vectors. Repeat this until the tree depth is reached.

We are paralleling the implementation of Tree projection vector algorithm using open mp. Here as the task for printing root left part tree projection vectors is independent of the task for printing root right part tree projection vectors, so the code for implementing them is also independent. As independent tasks or code can be paralleled executed using the parallel sections by threads construct so we are using parallel sections construct to parallelize the code. Also we should use critical construct as to increment the level of the tree synchronously Otherwise the root of the tree will not incremented correctly as different threads are executing the code parallel. As we are parallelizing the code and different threads execute more no of vectors are printed, so to overcome this problem we are using single construct. If the root has left child than the logic or code for printing left sub tree vectors no need to wait for printing vectors of right sub tree, so we are using no wait clause near the single construct. Similarly we are using no wait clause at logic of printing right sub tree vectors.

V. Results

Here, we are comparing our results in terms of execution time, for serial input as well as parallel input.

```

C:\Users\welcome\Documents\Visual Studio 2008\Projects\tree\Debug\tree.exe
creation of tree
enter a value :1
1. INSERTION OF TREE
2. FOR TREE PROJECTION
3.exit
enter ur choice :1
enter a value :2
1. INSERTION OF TREE
2. FOR TREE PROJECTION
3.exit
enter ur choice :1
enter a value :3
1. INSERTION OF TREE
2. FOR TREE PROJECTION
3.exit
enter ur choice :2
< 1, 1, 2>      < 2, 2, 1>      < 2, 2, 3>      < 3, 3, 2>
serial execution time=0.000969
< 1, 1, 2>      < 2, 2, 1>      < 2, 2, 3>      < 3, 3, 2>
parallel execution time=0.002500
1. INSERTION OF TREE
2. FOR TREE PROJECTION
3.exit
enter ur choice :
  
```

Fig.-2 serial time is more than parallel time

From the above result, we can see that for the smaller no. of values serial execution time is less as compared to parallel execution time. So if we use this algorithm for smaller no. of nodes. Serial execution is advisable.

```

C:\Users\welcome\Documents\Visual Studio 2008\Projects\tree\Debug\tree.exe
19> < 19, 19, 18> < 19, 19, 20> < 20, 20, 19> < 20, 20, 21> < 21, 21, 19>
20> < 21, 21, 22> < 22, 22, 21> < 22, 22, 23> < 23, 23, 22> < 23, 23, 24>
24> < 24, 24, 23> < 24, 24, 25> < 25, 25, 24> < 25, 25, 26> < 26, 26, 25>
25> < 26, 26, 27> < 27, 27, 26> < 27, 27, 28> < 28, 28, 27> < 28, 28, 29>
29> < 29, 29, 28> < 29, 29, 30> < 30, 30, 29>
serial execution time=0.028886
< 1, 1, 2> < 2, 2, 1> < 2, 2, 3> < 3, 3, 2> < 3, 3, 4>
4> < 4, 4, 3> < 4, 4, 5> < 5, 5, 4> < 5, 5, 6> < 6, 6, 5>
5> < 6, 6, 7> < 7, 7, 6> < 7, 7, 8> < 8, 8, 7> < 8, 8, 9>
9> < 9, 9, 8> < 9, 9, 10> < 10, 10, 9> < 10, 10, 11> < 11, 11, 10>
10> < 11, 11, 12> < 12, 12, 11> < 12, 12, 13> < 13, 13, 12> < 13, 13, 14>
14> < 14, 14, 13> < 14, 14, 15> < 15, 15, 14> < 15, 15, 16> < 16, 16, 15>
15> < 16, 16, 17> < 17, 17, 16> < 17, 17, 18> < 18, 18, 17> < 18, 18, 19>
19> < 19, 19, 18> < 19, 19, 20> < 20, 20, 19> < 20, 20, 21> < 21, 21, 20>
20> < 21, 21, 22> < 22, 22, 21> < 22, 22, 23> < 23, 23, 22> < 23, 23, 24>
24> < 24, 24, 23> < 24, 24, 25> < 25, 25, 24> < 25, 25, 26> < 26, 26, 25>
25> < 26, 26, 27> < 27, 27, 26> < 27, 27, 28> < 28, 28, 27> < 28, 28, 29>
29> < 29, 29, 28> < 29, 29, 30> < 30, 30, 29>
parallel execution time=0.019692
1. INSERTION OF TREE
2. FOR TREE PROJECTION
3.exit
enter ur choice :
  
```

Fig.-3 parallel time is enhanced than serial time

Here, the result shows that if we move to mass no. of nodes parallel execution time gives better results than serial execution time. We have given the inputs and notice that the results start being differentiated from 30 nodes. Now we will see the comparison of these values through graph representation.

S. No.	No. of nodes	Serial time	Parallel time	Speed up
1.	5	0.000969s	0.002500s	0.001531s
2.	25	0.022645s	0.024332s	0.001687s
3.	50	0.028886s	0.019692s	0.004194s
4.	100	0.031180s	0.025792s	0.005388s
5.	200	0.103497s	0.072731s	0.169234s

Table 1:- Performance Analysis (in 2 core)

The above calculation is for the serial and parallel execution time. Here we can see the behavior of speed up factor through table as well as through graph representation.

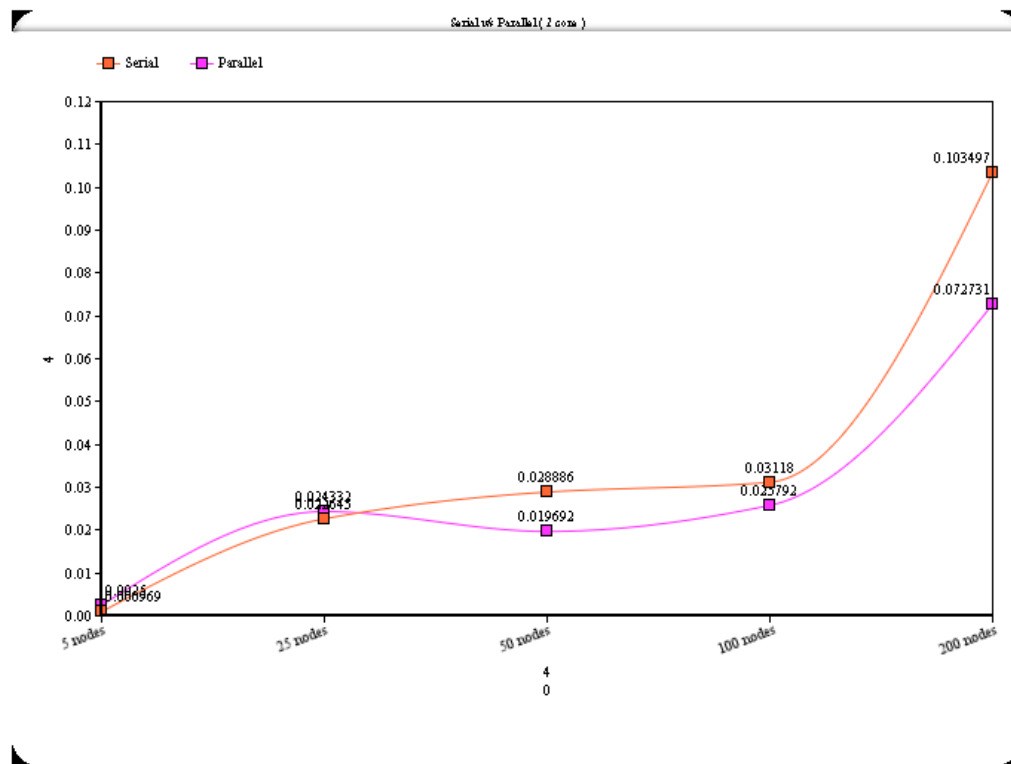


Fig.:-3 Serial vs parallel (in 2 core)

```

C:\I\Tree projection\tree\Debug\tree.exe
19> < 19, 19, 18> < 19, 19, 20> < 20, 20, 19> < 20, 20, 21> < 21, 21, 19>
20> < 21, 21, 22> < 22, 22, 21> < 22, 22, 23> < 23, 23, 22> < 23, 23, 24>
24> < 24, 24, 23> < 24, 24, 25> < 25, 25, 24> < 25, 25, 27> < 26, 27, 25>
25> < 1, 27, 26> < 2, 26, 27> < 26, 27, 28> < 27, 28, 27> < 27, 28, 29>
29> < 28, 29, 28> < 28, 29, 30> < 29, 30, 29>

serial execution time=0.036632
4> < 1, 1, 2> < 2, 2, 1> < 2, 2, 3> < 3, 3, 2> < 3, 3, 4>
5> < 4, 4, 3> < 4, 4, 5> < 5, 5, 4> < 5, 5, 6> < 6, 6, 5>
6> < 6, 6, 7> < 7, 7, 6> < 7, 7, 8> < 8, 8, 7> < 8, 8, 9>
9> < 9, 9, 8> < 9, 9, 10> < 10, 10, 9> < 10, 10, 11> < 11, 11, 10>
10> < 11, 11, 12> < 12, 12, 11> < 12, 12, 13> < 13, 13, 12> < 13, 13, 14>
14> < 14, 14, 13> < 14, 14, 15> < 15, 15, 14> < 15, 15, 16> < 16, 16, 15>
15> < 16, 16, 17> < 17, 17, 16> < 17, 17, 18> < 18, 18, 17> < 18, 18, 19>
19> < 19, 19, 18> < 19, 19, 20> < 20, 20, 19> < 20, 20, 21> < 21, 21, 20>
20> < 21, 21, 22> < 22, 22, 21> < 22, 22, 23> < 23, 23, 22> < 23, 23, 24>
24> < 24, 24, 23> < 24, 24, 25> < 25, 25, 24> < 25, 25, 27> < 26, 27, 25>
25> < 1, 27, 26> < 2, 26, 27> < 26, 27, 28> < 27, 28, 27> < 27, 28, 29>
29> < 28, 29, 28> < 28, 29, 30> < 29, 30, 29>

parallel execution time=0.016658
1. INSERTION OF TREE
2. FOR TREE PROJECTION
3.exit
enter ur choice :
    
```

Fig. 5:- parallel time is less than serial time

Here through this graph we can simply visualize the variation in speed factor. As the no. of nodes increase the parallel time reduces. The larger the no. of nodes will be, the larger the parallel time reduces. Now we will see the performance of this tree projection in 4 core machine.

S. No	No. of nodes	Serial time	Parallel time	Speed up
1.	5	0.0008953s	0.002245s	0.001349s
2.	25	0.0016925s	0.024645s	0.002952s
3.	50	0.0278476s	0.015483s	0.012364s
4.	100	0.0349801s	0.020692s	0.014291s
5.	200	0.0934972s	0.052741s	0.040751s

Table 2:- Performance Analysis (in 4 core)

In Table 2, we have calculated the values in time and the speed up. Now the variations in the execution time can be seen in the graph which is given below:

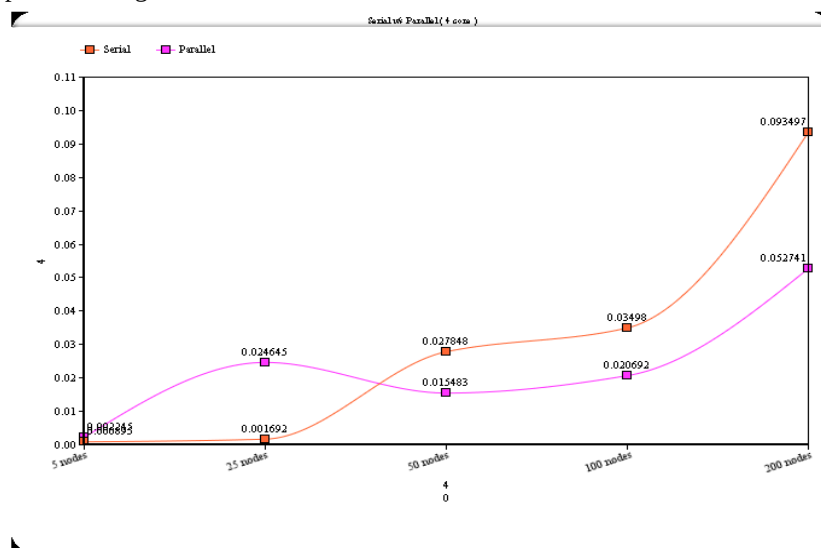


Fig.6:- Serial vs Parallel in 4 Core

From the above graph we can see that the variations which we have seen in the 2 core machine has been increased. We can simplify this as for smaller no. of nodes the serial execution time has been increased and the parallel execution time has also increased. For larger no. of nodes the comparison is also like this only.

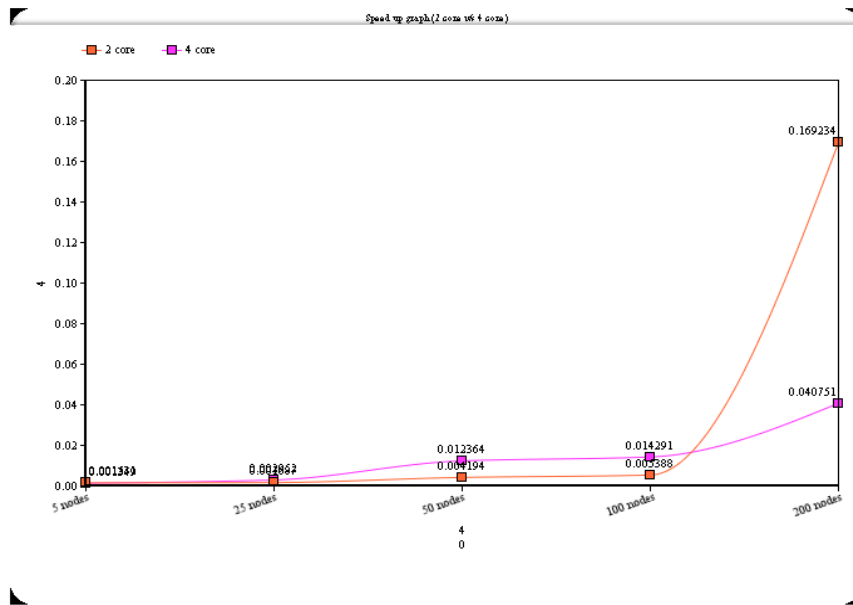


Fig. 7:- Speed up graph (2 core /4core)

Here From the 2 core v/s 4 core speed up graph we can see that as we increase the no. of cores the performance is also increasing form the rapid rate and as we see here we can easily verify that we are increasing the 2 cores and the performance is much higher than the double of the first performance.

After implementing the code for Tree projection algorithm and parallelizing it, we analyze that for small input i.e. if the tree size is small serial code takes less execution time than the parallelized code. And if go on increase the tree size parallel code takes less time for execution than serial code. For two core machine if we give 10 nodes as input the execution time for 10 nodes is 0.008417 for serial code and 0.009437 for parallel code. If we give 22 nodes as input the execution time for tree projection algorithm is 0.022645 for serial code and 0.024332 for parallel code. If we give 50 nodes as input the execution time for tree projection algorithm is 0.025200 for serial code and 0.020574 for parallel code. If we give 62 nodes as input the execution time for tree projection algorithm is 0.031180 for serial code and 0.025792 for parallel code. If we give 150 nodes as input the execution time for tree projection algorithm is 0.077916 for serial code and 0.053999 for parallel code. If we give 200 nodes as input the execution time for tree projection algorithm is 0.103497 for serial code and 0.072731 for parallel code. So we can conclude that for small size tree serial code gives less execution time and if the size of the tree is increasing (number of nodes greater than or equal to 30) then parallelized code takes less execution time. If we talk about cores that are 2 cores and 4 cores than one can easily verify that for 2 cores the serial time and parallel time is increasing slightly whether for 4 core the serial time and the parallel time has increased in a rapid manner. The slight variation at 62 nodes can be seen very easily in the graph plotting.

VI. PROS AND CONS OF THE ALGORITHM WITH THE HELP OF RESULT OBTAINED

The Tree projection method that we have developed is the first idea about parallelizing the tree projection. So there can be many flaws to this model. In our model we are presenting the tree model only for the single root node whether it can be implemented for multiple root nodes also. Here we are generating node neighbor tree for the given graph which is a lightweight process and we are using this process for reducing the computation in finding sub graphs.

By implementing the parallel code for this algorithm we can generate the false alarms in less time. Here the execution time between serial execution and parallel execution can be further enhanced by using some more parameters in the tree projection algorithm.

VII. CONCLUSION & FUTURE WORK

we have done this algorithm parallelization and reached to the conclusion that in the case of large computations parallel program gives better performance. Whether if we will focus on the smaller computations the serial computation will be more better. The process of finding the node neighbor tree and the tree projection can be done separately and the execution time we can also calculate twice for the two processes by using the function `omp_get_time`. Whatever we have discussed previously, from that we can conclude that we are just implementing the tree projection which is the first part in the subgraph pattern search. This algorithm can be implemented for the complete solution of subgraph pattern search in the certain and uncertain graphs by using section clause.

As we know that in continuous subgraph pattern search algorithm, the first phase is to find the node neighbor tree and the second approach is to project a tree, After that we have to do the searching process. There are two searching algorithms that are defined in this paper dominant set cover algorithm, skyline with early stop join method, we can also parallelize them and calculate the difference between the serial and parallel execution and further we can take the whole paper and implement a search algorithm and calculate the differences between the serial and parallel execution time for finding.

VIII. REFERENCES

- [1]. Lei Chan, Changliang Wang, Continuous Subgraph Pattern Search Over Certain and Uncertain Graph, IEEE TRANSACTION ON KNOWLEDGE AND DATA ENGINEERING, VOL. 22, NO. 8, AUGUST 2010
- [2]. Yinglong Xia, Prasanna, V.K., Distributed Evidence Propagation in Junction Trees on Clusters, IEEE TRANSACTION ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 23, Pages 1169-1177, 2012
- [3]. Yinglong Xia and Viktor K., Prasanna, Scalable Node Level Computation Kernels for Parallel Exact Inference, IEEE TRANSACTION ON COMPUTERS, VOL. 58, JAN 2010
- [4]. Kuei-Chiang Lai; Jiun-Jie Jia; Li-Wei-Lin, Hybrid Tree Search Algorithms For Detection in Spatial Multiple Access Systems, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, VOL. 60, Pages 3503-3509, 2011
- [5]. Mario Mendez-Lojo¹ Donald Nguyen² Dimitrios Proutzos² Xin Sui², M. Amber Hassaan³ Milind Kulkarni⁴ Martin Burtscher¹ Keshav Pingali^{1,2} Structure-driven Optimizations for Amorphous Data-parallel Programs, Pages 3-14, 2010
- [6]. Prechelt, L.; Hanssger, S.U, Efficient Parallel Execution of Irregular Recursive Programs, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL 13, Pages 167-178, 2002.
- [7]. C. Chen, X. Yan, P.S. Yu, J. Han, D.-Q. Zhang, and X. Gu, "Towards Graph Containment Search and Indexing," Proc. 33rd Int'l Conf. Very Large Data Bases (VLDB), 2007.
- [8]. J. Cheng, Y. Ke, W. Ng, and A. Lu, "FG-Index: Towards Verification-Free Query Processing on Graph Databases," Proc. ACM SIGMOD, 2007.
- [9]. S. Fortin, "The Graph Isomorphism Problem," technical report, Dept. of Computing Science, Univ. of Alberta, 1996.
- [10]. L. Gao and X.S. Wang, "Continually Evaluating Similarity-Based Pattern Queries on a Streaming Time Series," Proc. ACM SIGMOD, 2002.
- [11]. A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," Proc. ACM SIGMOD, 1984.
- [12]. H. He and A.K. Singh, "Closure-Tree: An Index Structure for Graph Queries," Proc. 22nd Int'l Conf. Data Eng. (ICDE), 2006.
- [13]. H. He and A.K. Singh, "Graphs-at-a-Time: Query Language and Access Methods for Graph Databases," Proc. ACM SIGMOD, 2008.
- [14]. H. He, H. Wang, J. Yang, and P.S. Yu, "Blinks: Ranked Keyword Searches on Graphs," Proc. ACM SIGMOD, 2007.
- [15]. P. Hintsanen and H. Toivonen, "Finding Reliable Subgraphs from Large Probabilistic Graphs," Proc. 12th Int'l Conf. Principles and Practice of Knowledge Discovery in Databases (PKDD), 2008.

- [16]. H. Jiang, H. Wang, P.S. Yu, and S. Zhou, "GString: A Novel Approach for Efficient Search in Graph Databases," Proc. 23rd Int'l
- [17]. Conf. Data Eng. (ICDE), 2007. J.E. Hopcroft and R.M. Karp, "A $n^5=2$ Algorithm for Maximum Matchings in Bipartite," Proc. 12th Ann. Symp. Foundations of Computer Science, 1971.
- [18]. A. Hulgeri and C. Nakhe, "Keyword Searching and Browsing in Databases Using Banks," Proc. 18th Int'l Conf. Data Eng. (ICDE), 2002.
- [19]. R. Jin, S. McCallen, C.-C. Liu, E. Almaas, and X.J. Zhou, "Identify Dynamic Network Modules with Temporal and Spatial Constraints," Proc. Pacific Symp. Biocomputing (PSB), 2009. R. Kali, "The City as a Giant Component: A Random Graph Approach to Zipf's Law," Applied Economics Letters, vol. 10, no. 11, pp. 717-720, 2007.
- [20]. X. Lian and L. Chen, "Monochromatic and Bichromatic Reverse Skyline Search over Uncertain Databases," Proc. ACM SIGMOD, 2008.