Volume 3, Issue 5, May 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Graph Indexing using Distributed and Parallelization Approach

Swati Manekar[*]
Computer Science &Engineering Dept.
G.H.Raisoni College of Engineering, Nagpur
India.

Manish Narnaware [**]
Computer Science &Engineering Dept
G.H.Raisoni College of Engineering, Nagpur
India.

Abstract— Graphs are ubiquitous like chemical compounds, protein structure, road networks, web, social network analysis, etc. So graphs are very important means to model schemaless data with such complicated structures .This paper focuses on indexing structures present in the large graph datasets and proposes a time efficient indexing technique for efficient graph mining for molecular database. First step is to enumerating all frequent substructures in the graph database (derived from search tree) using proper pruning strategies at the time of candidate generation so removing the redundant search which will generate compact index for input database .An inverted index between these structures and graphs are then created. So indexing feature used is frequent subgraphs. But in the real world these datasets are very large in numbers so process of creating index may take large time. The time required has been minimized by using parallel and distributed approach.

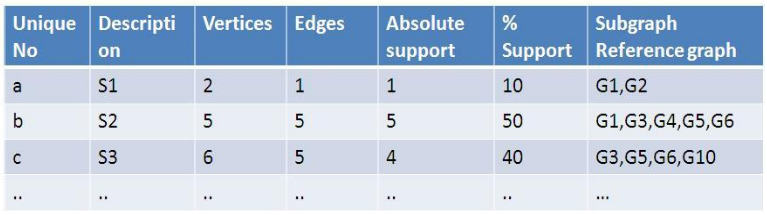
Keywords— Graph indexing, graph mining, frequent structure based approach, parallelization and distributed approach.

I. INTRODUCTION

Graphs are everywhere in chemical compound, social network, road network, biological network, etc .graph mining can be defined as the process of extracting useful information from graph dataset. useful information can be anomaly detection in network of routers, drug discovery in chemical compound, finding strongly connected group on social network, etc. in latest years, a number of data mining and management applications have been designed in the context of graphs and structural data .Many graph mining techniques have been proposed. In graph mining the most interesting issue is to query search in graph database. Sequential scanning of database for query searching would be very hectic, time consuming and complex process. To overcome this index of graph is generally created for filtering noncandidate graphs for given query. Several methods for graph indexing have been proposed. They crate index of graph depending upon some features likes paths, tree, subgraph etc. In our proposed approach we have used frequent subgraphs as an indexing feature .Frequent subgraphs are quality features for indexing as they do not lose the structural information of the graph and generate compact result. Frequent substructures are those which are present in molecules with no less support as a threshold value. As real world data is very large in size so the process of index creation with frequent substructure may take very long time plus memory consumption will be more. To overcome this problem we proposed an incorporation of parallel and distributed approach to achieve memory and time efficient way to index graph. To run the process of mining frequent subgraphs in parallel we used concurrency class in java. For distributed approach we used cajo framework in java. Why distributed approach as while finding frequent subgraphs the search tree size may get very large and may through exception of memory the solution is to use multiple processors running simultaneously with their own memory.

In a graph dataset D, given a query graph q, a subgraph search algorithm retrieves all graphs in D containing q as a subgraph. This process of finding the subgraph isomorphic to query graph in graph database is nontrivial, as subgraph isomorphism problem is known to be NP-complete as shown in [1].

The indexing process in a graph matching methodology creates an index of the reference graph vertices along with their attributes so that the future referencing of the vertices for matching purpose becomes an efficient process. The data structures used for indexing usually determine the flow of the process. Many features, such as frequent and discriminative subgraph (subtree) features, δ−TCFG features and MimR (maximum information and minimum redundancy) features are mined to build the graph index and have certain significant filtering capabilities. Example of such created index can be shown in fig.1



Unique No	Description	Vertices	Edges	Absolute support	% Support	Subgraph Reference graph
a	S1	2	1	1	10	G1,G2
b	S2	5	5	5	50	G1,G3,G4,G5,G6
c	S3	6	5	4	40	G3,G5,G6,G10
..	..	..	..	..	..	...

Fig.1 Example of Index File

It can be seen from above index file which consist of Unique No field which gives the unique no to frequent substructure found, with their description in Description field .Considering any arbitrary data. Edges and nodes give the number of nodes and edges in corresponding substructure. Absolute support and % support gives the number and percentage of molecules which contain the corresponding fond substructure in given database. Subgraph Reference graphs gives the list of graph containing the respective substructure.

II. Graph Indexing Techniques

Searching for graph similarity is interesting issue in graph mining as subgraph isomorphism is NP Complete problem [1].To speed up the process of similarity search in graph database generally an index of graph is generated. Then this index is used for filtering process i.e. to prune out non candidate graph for query graph .As this filtering phase is very important phase in any similarly searching algorithm so many indexing techniques have been developed for efficient searching. These indexing techniques can be broadly classified in three types as path based approach, substructure based approach and graph decomposition based approach.

GraphGrep [2] and Daylight [3] are Path-based graph indexing methods they uses path as an indexing feature. The approach is like enumerating all paths up to length maxL and indexes them. Where a path is a vertex sequence, V1, v2, ……, vk, s.t.,$\forall 1 \leq i \leq k-1, (v_i, v_{i+1})$ is an edge. It uses the index to identify every graph gi that contains all the paths (up to maxL length) in query q. Main plus point of this method is that paths can be manipulated easier than trees. But the draw back with this method is that the path simple structure loosing structural information of a graph and paths in a graph database increases exponentially, making path-based methods impractical for very large graphs.

On the other hand structured based approach identifies subgraphs to be indexed as an indexing feature. As Yan et. al. indicated that false positive ratio of path based methods would be very high instead, structure-based graph indexing approaches gIndex[4] first searches for the frequent subgraphs in the graph, then indexes these frequent structures**.** A case in the above paper discussed is that frequent subgraph discovery increases complexity and exponential number of frequent fragments may exist under low frequency support. Therefore, in their study, they limit the number of nodes and index frequent structures up to 10 nodes.

Shang et al. [14] put forward a novel indexing technique QuickSI to efficiently compute verification phase for testing subgraph isomorphism and a new feature-based indexing method is developed to have room for QuickSI in the filtering phase. Yan et al. [15] put forward a structure filtering algorithm Grafil to filter out graphs without performing pairwise similarity computations. They tip out that the filtering effectiveness and efficiency greatly depend on the excellence of selected features. Cheng et al. [16], [17] propose a nested inverted-index called FG-index to avoid candidate confirmation by exploiting frequent subgraphs and edges as indexing features. However, when encountering intermittent queries, the method performs poorly, as infrequent subgraphs are not included into the FG-Index. On the other hand, Chen et al. [18] inspect an interesting graph containment search problem, and a contrast subgraph- based indexing model cIndex is proposed to solve this problem. Frequent-subgraph-based indexing methods are also called feature-based indexing methods. They have two main drawbacks. One is that the effectiveness and efficiency of such kind of methods depend on the quality of selected features. The other is that it is difficult to construct and maintain the index because the frequent subgraph mining algorithm usually takes a very long time to compute [19]. To compensate for the latter shortcoming, the authors of [21] and [22] propose frequent-subtree-based indexing methods called TreePi and Closure-Tree. The TreePi approach indexes a graph set by using only subtree patterns to accelerate the filtering procedure, while the Closure-Tree approach uses the data structure called ClosureTree to organize a B-tree-like index on the graph set.

An alternative structural indexing approach to search and process queries efficiently even in very large graphs As indexing features, used commonly observed graph structures: star, complete bipartite, triangle and clique. An important feature of these structures is that each one is comprised from the previous one where clique contains complete bipartite structures and complete bipartite contains star structures.

In structural indexing, they have indexed predefined structures that are commonly observed in complex networks. In particular, index star, complete bipartite, triangle and clique structures in a given graph G = (V,E). An important difference of their approach from the previous studies is that they do not limit the size of subgraph considered in indexing. They have indexed all maximal graphs that match the structure formulation. For instance, a maximal clique is a clique that cannot be extended by adding one more vertex from the graph. However, the substructure size in indexing may be limited when needed since maximal clique search is known to be NPcomplete .In order to reduce computational complexities; they have indexed the structures within the original graph in a consecutive manner. They first identified star structures, and then the complete-bipartite, triangle and clique structures from the preceding one. But problem with this method is that finding complete clique is a NP complete problem also finding these types of structures having limited applications [5].

The tree-based indexing involves a tree data structure. Graph partitioning algorithms are used in order to obtain the vertices at various levels of the tree. For example, Top-k based subgraph matching algorithm uses a G-tree for index construction. This algorithm uses a heap structure for the matching process and to store the final result

One of the recent indexing technique is neighbourhood based method, which is employed by TALE [9], GADDI[8] and SAPPER[7]. During indexing, it is ensured that not only the vertex labels are stored, but the neighbourhood of a vertex is also stored thus ensuring that the structural information is taken into consideration. Since the number of neighbours can be large in a very dense graph, normal storage strategies may prove to be inefficient. To tackle the storage issue, a hashing based methodology called bloom filter is used. TALE and SAPPER use bloom filter [6] for storing the index.

GADDI focuses on a slightly different neighbourhood approach called NDS (Neighbour Discriminating Substructure) distance for indexing. The NDS distance of a pattern P is defined as the number of matches of P present in an induced subgraph of neighbouring vertices. An array is used for each of the vertices of the induced subgraph to store the NDS distances.

In indexing for multileveled graph using SAPPER algorithm plus some enhancement to accommodate multiple labels for vertices and edges given in paper [10]. Data structure used here is List. Indexing done in the five part 1)Labels of the vertex 2)Degree of the vertex 3)labels of neighbours 4)Labels of edges to neighbours 5)the labels of second level neighbours stored in bloom filter. Approached used here in this paper can be explained as follows-Firstly the entire reference graph is loaded into primary memory then indexing process is stated. Data Structure used for storage is array, array (1) – Storing vertex information, array (2) – Storing edge information. Two phases for indexing is used first is vertex processing and second is edge processing. In vertex processing firstly a data structure is initialized, then for each vertex, vertex labels are inserted along with initialization of neighbour list structure. Now in edge processing each edge entry is traversed first then updates the neighbour lists corresponding vertices with the label of its neighbours .Update labels of the edge connecting these neighbours and also update vertex identifiers of these neighbours. After this one label for edge and neighbour vertex stored in neighbour list and finally edge processing completed .Labels of second level neighbours are stored in bloom filter for each vertex.

In graph decomposition based indexing technique, graph decomposition is applied directly to replying queries of isomorphism and subgraph isomorphism [11], [12] .Graph decomposition is applied before going for graph isomorphism testing. Two short coming with these method are one is that they have to enumerate all connected subgraphs and hence complexity is exponential to graph size which is to be decomposed and other is frequent information in decomposition results are not improving the efficiency of graph similarity search.

One more method uses both graph decomposition method and frequent subgraph method for indexing graph[13] Here a graph is decomposed into set of *k*-Adjacent trees and decomposed result are indexed by a *K*-AT index. It store more structural information as compared to normal graph decomposition (breakdown) methods.

So in this paper we proposed an indexing method which uses frequent subgraphs an indexing feature for very large database .By using a proper pruning strategy and parallel approach it minimize the time required to construct an index.

III. PROPOSED METHOD

The main aim of Graph Indexing is to reduce the set of graphs without loss of result, means graph indexing must promise pruning non promising graphs. The graph indexing technique must not be very expensive i.e. it should not take much time and memory to create index. Secondly it should have higher pruning capability without loss of result, and finally it should not generate false answer.To deal with the problem of space we used parallel and distributed approach with multiple processors with their individual memory running in parallel to give good speed. For high pruning capability we used close fragment pruning with the loss less result.

User can define the support to index all frequent subgraphs usually low-support large fragments may be indexed well by their smaller subgraphs. Especially, it will be always chosen the (absolute) minSup to be 1 for size-0 fragment to ensure the completeness of the indexing. This method having two rewards first one is that the number of frequent fragments so obtained is much less than that with the lowest uniform minSup, and second one is low-support large fragments may be indexed well by their smaller subgraphs; thereby it will not miss useful fragments for indexing

By using frequent fragments with the size-increasing support constraint, one has a smaller number of fragments to index. However, the number of indexed fragments may still be huge when the support is low. For example, 1,000 graphs may easily produce 100,000 fragments of that kind. It is both time and space consuming to index them. To overcome this parallel and distributed approach has been used in our approach. This can be explained as follows,

- Identify Frequent Structures in the database; the frequent structures are sub graphs that appear quite often in the graph database.
- Prune redundant frequent structures to maintain a small set of discriminative structures.
- Create an inverted index between frequent structures and graphs in the database.
- Enumerate structures in the graph database build an inverted index between structures and graphs.
- A graph or structure is frequent if its support i.e. occurrence frequency no less than the minimum support threshold.

Indexing feature used here is frequent structures as they are the quality candidates not losing the structural information giving lossless results. This whole process of indexing is parallelized to achieve time efficiency.

We are using molecular database as an input. Firs step is to represent molecules as attributed graph. DFS is used to get into the search tree which will ensure that each vertex get covered in the graph. Now the next step is generation of candidate subgraphs i.e. so called pattern growth. This can be done by extending a fragment by adding node or edge. If it is closing ring then adding an edge only otherwise adding a node.

For finding Frequent Subgraphs it is important to review how a graph database is searched for frequent subgraphs: Starting from the seed node or user defined node for which all possibilities must be tried a subgraph is extended by adding a node or edge in every step. Condition here is that in this stepwise extension process one requires that at least one node which is a part of an added edge must already be there in subgraph. These means that the search is restricted to connected subgraphs which are necessary for most applications. In its most basic form the search considers all possible extensions of the current subgraph by an edge and, if necessary, a node. The set of extensions can be reduced by exploiting a canonical description used in gSpan. Note that, as a consequence of the above, the search produces a numbering of the nodes in each subgraph: the steps in which these nodes are added corresponds to node indexes and similar in case of edges also edges are also associated with the indexes in the order in which they are added. This search generates a spanning tree of the subgraph, which is improved by additional edges. Depth-first and breadth-first search are straightforward systematic methods for constructing a spanning tree of a graph. Other alternatives include a spanning tree construction that first visits all neighbours of a node (like breadth-first search), but then chooses the next node to extend in a depth-first search manner.

As in extending search tree, may generate duplicate subgraphs, so to avoid redundant search Canonical form pruning, closed subgraph pruning is used.

For calculating the support embeddings are stored .The support of the fragment (Sub graph) is determined by simply counting the number of different molecules the embedding refers to. As for large data input frequent fragment can be very large in size so only discriminative fragments are derived which give compact index. Discriminative fragments are those fragments which are high in active molecules, low in inactive molecules. After identifying all frequent discriminative subgraph an inverted index has been created between these subgraphs and the graphs in database .This whole process can be summarized in the form of flow char as shown below

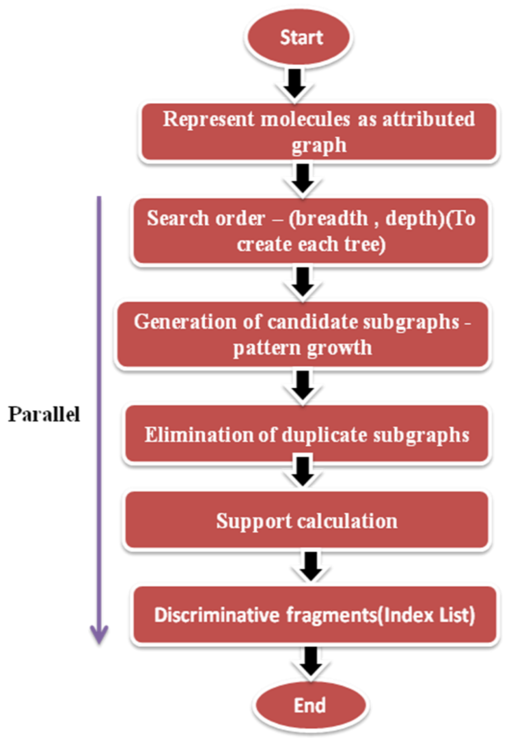

Start
Represent molecules as attributed graph
Search order – (breadth , depth)(To create each tree)
Generation of candidate subgraphs - pattern growth
Elimination of duplicate subgraphs
Support calculation
Discriminative fragments(Index List)
End
Parallel

Fig. 2 Graph Indexing Flow Chart

The simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads give higher speed of execution. So this process of finding frequent subgraphs in the database is divided into multiple threads and executed on different processors in parallel fashion .Ideally, parallel processing makes programs run faster because there are more engines (CPUs or Cores) running it. So we have divided a program of graph miner (finding frequent fragments) in such a way that separate CPUs or cores can execute different portions without interfering with each other. Parallel processing provide faster execution time so higher throughput. Overall execution time for creating the index of graph got reduced. The parallel processing architecture for creating graph index can be shown in fig. 3.

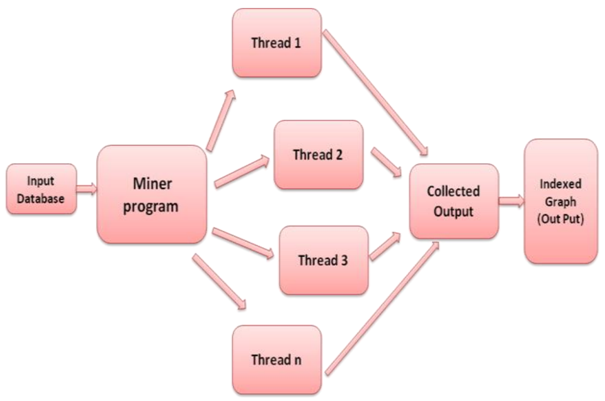


Fig. 3 Parallel Processing Module

Here the miner program is collection of the different methods for finding frequent discriminative fragments in the graph along with the pruning strategies for avoiding irrelevant candidate answers.

In distributed approach there will be main agent and server agents who will serve the request as shown in Fig. 4. We are using cajo frame work in java to run program in distributed environment. As in distributed approach there are multiple processors along with their own memory so memory out of heap space issue is solved. In distributed approach main agent will send request to sever through interface where signatures are created for each task to be done on server agent side. Server agents will run in parallel, all the results are collected in gather phase and displayed.

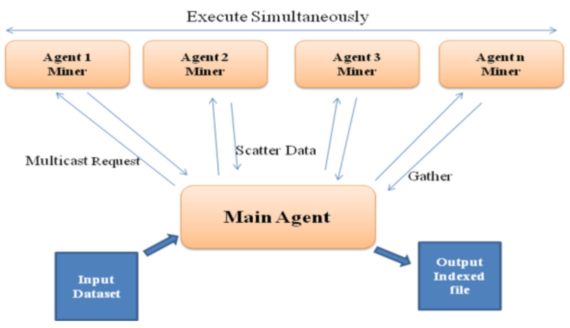


Fig. 4 Distributed Processing Module

IV. Results Calculated

Experiment is conducted on molecular dataset. These molecular dataset is taken from Pubchem molecular Renderer. Up to 100000 molecules are taken as an input. Here two approaches can be seen in fig.5.that is with parallel approach and without parallel approach.

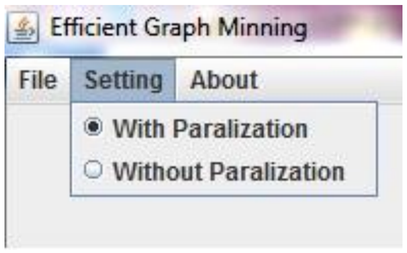


Fig. 5 Setting Tab for parallel and without parallel approach.

V. Comparative Analysis

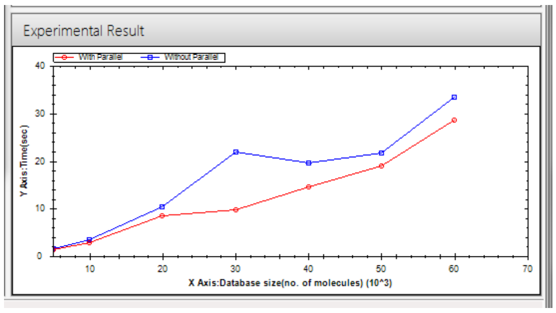


Fig. 6 Comparison between two approaches, with and without parallel approach

Two approaches are compared i.e. with parallel and without parallel approach for graph indexing, x-axis shows Database size i.e. number of molecules in dataset in thousand, the y-axis shows the time in sec to create index. Where blue bar represent the statistical bar for without parallel approach and red line represent with parallel approach.

We got the speed up of 1.3223 and it is calculated as stated in eq. (1).

Speed up factor:

Speedup is defined by the following formula:

$$S_p = \frac{T_1}{T_p} \qquad (1)$$

Where,

- p is the number of processors
- T_1 is the execution time of the sequential algorithm

- T_P is the execution time of the parallel algorithm with *p* processors

Sp=T1/Tp
Sp=16/12.10
Sp=1.3223
So we got the speed up of 1.355
The experiment is performed on the machine with following configuration,
Machine configuration:
Operating System: Microsoft Windows XP
Intel(R) Core(TM) 2Duo CPU
2.20 GHz
1.96 GB of RAM

VI. CONCLUSION

The different approaches for graph indexing having some advantages and drawbacks. In order to speed up graph queries, usually an index of the graph is derived according to some predefined index features. Graph indexing is used for efficient graph mining. As many graph data sets are defined on massive node domains in which the number of nodes in the underlying domain is very large the indexing techniques implement require more time. The performance of the graph indexing has been enhanced and speeded up by using parallelization approach by running the program in parallel on two processors. In distributed approach as all processors are along with their own memory gave memory and time efficient way to generate an index of graph.

REFERENCES

[1] S. A. Cook, "The complexity of theorem-proving procedures," in STOC, 1971, pp. 151–158
[2] D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In Symposium on Principles of Database Systems, pages 39–52, 2002.
[3] C. A. James, D. Weininger, and J. Delany. Daylight theory manual daylight version 4.82. daylight chemical information systems, 2003.
[4] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach, 2004.
[5] Hakan Kardes¸ and Mehmet Hadi G¨unes. Structural Graph Indexing for Mining Complex Networks. 2010 IEEE 30th International Conference on Distributed Computing Systems Workshops.
[6] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," Communications of the ACM, vol. 13, no. 7, pp. 422–426, 1970.
[7] S. Zhang, J. Yang, and W. Jin, "Sapper: subgraph indexing and approximate matching in large graphs," Proceedings of the VLDB Endowment, vol. 3, no. 1-2, pp. 1185–1194, 2010.
[8] S. Zhang, S. Li, and J. Yang, "Gaddi: distance index based subgraph matching in biological networks," in Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. ACM, 2009, pp. 192–203.
[9] Y. Tian and J. Patel, "Tale: A tool for approximate large graph matching," in Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. IEEE, 2008, pp. 963–972.
[10] Varun Krishna , NNR Ranga , Suri G Athithan, MuGRAM: An Approach for Multi-labelled Graph Matching. Centre for Artificial Intelligence and Robotics Bangalore, India. 978-1-4673-0255-5/12/$31.00c 2012 IEEE.
[11] D. Eppstein, "Subgraph Isomorphism in Planar Graphs and Related Problems," J. Graph Algorithms and Applications, vol. 3,no. 3, pp. 1-27, 1999.
[12] J.P. Kukluk, L.B. Holder, and D.J. Cook, "Algorithm and Experiments in Testing Planar Graphs for Isomorphism," J. Graph Algorithms and Applications, vol. 8, no. 3, pp. 313-356, 2004.
[13] Guoren Wang, Bin Wang, Xiaochun Yang," Efficiently Indexing Large Sparse Graphs for Similarity Search" IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 3 MARCH 2012.
[14] H. Shang, Y. Zhang, X. Lin, and J.X. Yu, "Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism," Proc. 34th Int'l Conf. Very Large Data Bases, pp. 364-375, 2008.
[15] X. Yan, P.S. Yu, and J. Han, "Substructure Similarity Search in Graph Databases," Proc. ACM SIGMOD, pp. 766-777, 2005.
[16] X. Yan, P.S. Yu, and J. Han, "Substructure Similarity Search in Graph Databases," Proc. ACM SIGMOD, pp. 766-777, 2005.
[17] J. Cheng, Y. Ke, and W. Ng, "Efficient Query Processing on Graph Databases," ACM Trans. Database Systems, vol. 34, no. 1, pp. 1-44, 2009.
[18] C. Chen, X. Yan, and P.S. Yu, "Towards Graph Containment Search and Indexing," Proc. 33rd Int'l Conf. Very Large Data Bases,pp. 926-937, 2007.
[20] D. Justice and A. Hero, "A Binary Linear Programming Formulation of the Graph Edit Distance," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 28, no. 8, pp. 1200-1214, Aug. 2006.
[21] S. Zhang, M. Hu, and J. Yang, "Treepi: A Novel Graph Indexing Method," Proc. IEEE 23rd Int'l Conf. Data Eng., pp. 966-975, 2007.
[22] H. He and A.K. Singh, "Closure-Tree: An Index Structure for Graph Queries," Proc. 22nd Int'l Conf. Data Eng., p. 38,2006.