Efficiency Improvements for Parallel Subgraph Miners

Abhik Ray, Lawrence B. Holder
School of EECS, Washington State University, Pullman, WA – 99164-2752, USA
{aray, holder}@eecs.wsu.edu

Abstract

Algorithms for finding frequent and/or interesting subgraphs in a single large graph scenario are computationally intensive because of the graph isomorphism and the subgraph isomorphism problem. These problems are compounded by the size of most real-world datasets which have sizes in the order of 10^5 or 10^6. The SUBDUE algorithm developed by Cook and Holder finds the most compressing subgraph in a large graph. In order to perform the same task on real-world data sets efficiently, Cook et al. developed a parallel approach to SUBDUE called the SP-SUBDUE based on the MPI framework. This paper extends the work done by Cook et al. to improve the efficiency of MPI SUBDUE by modifying the evaluation phase. Our experiments show an improvement in speed-up while retaining the quality of the results of serial SUBDUE. The techniques that we have used in this study can also be used in similar algorithms which use static partitioning of the data and re-evaluation of locally interesting patterns over all the nodes of the cluster.

Introduction

Graph data mining is the process of extracting patterns and other useful information from data represented in the form of graphs. One of the interesting patterns in graph data mining is a maximally compressing subgraph (Cook and Holder 2006), i.e., the subgraph of sufficiently large size that best compresses the input graph as per some compression scheme. This pattern allows us to describe the graph data conceptually at multiple levels.

Graphs most intuitively represent real world networks, like social networks, protein interaction networks, and transportation networks etc., which most often range from hundreds of thousands to millions of nodes and edges if not more. The large size of these graphs only increases the problems of subgraph isomorphism and graph isomorphism which are the key computational bottlenecks for graph mining.

In order to make graph mining a part of main-stream data mining, parallel computing techniques have to be used to reduce the execution times. The SUBDUE system (Cook and Holder 1994) was developed to find the most highly compressing subgraphs in a single large graph. Experiments show that it takes SUBDUE a very long time to execute on large real world datasets. Cook et al. developed a parallel version of SUBDUE (Cook et al. 2001) where they explored functional parallelism and data parallelism. They looked into three approaches, namely functional partitioning of the search space, dynamic partitioning of the workload and static partitioning of the dataset across nodes of the system. Overall they found that the static partitioning (SP) approach gave the best results and it required the least overhead of all the other techniques. Our work extends on the work done by Cook et al. in order to further improve the scalability and execution times of SP-SUBDUE.

Background

The compression afforded by a subgraph of a larger graph is determined by the size of the larger graph after replacing each instance of the subgraph with a single vertex. The maximally-compressing subgraph in an input graph is of interest for its ability to illuminate the normative relational patterns in a graph. Algorithms for finding maximally compressing subgraphs are computationally more expensive than ones for the graph transaction setting because of factors like subgraph isomorphism taking place in a larger graph and graph isomorphism occurring between larger subgraphs than are usually found in the graph transaction setting. Also if we are to compute the correct compression value we have to count the set of edge disjoint embeddings of a particular substructure.

SUBDUE

A few algorithms have been developed in order to perform this task. The SUBDUE system developed by Holder et al. (Cook and Holder 1994) is one such system which takes la-

beled or unlabeled graphs as input and performs a variety of supervised and unsupervised learning tasks. Our focus is mainly on making changes to the parallel versions of unsupervised learning though the changes we propose could just as easily be made to improve the efficiency of the parallelized supervised learning algorithm.

SUBDUE's unsupervised learning task is to discover the most highly compressing subgraph based on Rissanen's MDL principle (Rissanen 1989). Discovered substructures are evaluated using the formula given below.

Compression Value = size (Graph)/ (size (Substructure) + size (CompressedGraph))

The *Size* function calculates the size of the graph in terms of description length. SUBDUE's running time is kept within polynomial limits (Rajappa 2003) with the use of computational constraints like using a Beam Search variant to traverse the search space of candidate substructures, by expanding only a user-defined number of substructures, and keeping the size of expanded substructures to a user-defined limit. SUBDUE also uses an inexact graph match, where a substructure is considered to be an instance of another if the edit costs incurred in order to transform it into an isomorphism of the latter does not exceed a user-defined threshold. The SUBDUE system is also capable of using expert guided knowledge in the form of known substructure models that are possibly the most highly compressing ones or by adjusting the cost of each graph match test using graph match rules. It has been applied in areas such as predictive toxicology, network intrusion detection, earthquake analysis, web structure mining, and protein data (Manocha, Cook and Holder 2001) (Noble and Cook 2003) (Su, Cook and Holder 1999).

Parallel SUBDUE

Our experiments have shown that on very large graphs with hundreds of thousands of vertices and edges, SUBDUE exhibits extremely long execution times. As real-world graph datasets get larger, and speed-up techniques for serial processing algorithms are exhausted, parallelization of SUBDUE must be considered. The difficulty with parallelizing SUBDUE as with other knowledge discovery systems is that they require several iterations in order to arrive at results and every iteration is dependent on the output of the previous. Cook et al. looked at several different techniques for parallelizing SUBDUE. They considered the distributed memory architecture over the shared memory architecture as shared memory architectures are often not large enough to store the graph data.

With regards to a functional parallel approach, they developed two schemes, FP-SUBDUE and DP-SUBDUE. In FP-SUBDUE, each processor initially discovers substructures that give a compression value more than 1.0. A master node keeps track of all these substructures in a global search queue. It stores only M (beam-width) unique substructures in this queue in non-decreasing order of compression value. Whenever a child node sends the result of an expansion, the master stores the substructure only if it is better than the M^{th} substructure that it has received so far and non-isomorphic to all M substructures in the queue. The child nodes store only substructures from their search queues that appear in the global search queue. If a child does not have any substructure left, the master gives one to it from the search queue of a child that has more than a threshold number of substructures.

DP-SUBDUE takes the dynamic partitioning approach. Here even though every processor takes as input the entire graph, each core only works on a disjoint set of the input graph and expands the area of the graph it is working on as required. If a processor runs out of work, it requests work from a neighboring processor. In order to avoid duplication of work among processors, a processor cannot expand a substructure to contain a vertex whose label index is less than or equal to the processor ID.

The third approach SP-SUBDUE performs static partitioning of the data among the various processors. Each processor works on its subset of the data and broadcasts its discovered substructure to all other processors to evaluate on their partitions. At the end of all evaluations the master collects the results and reports the best substructure. The quality of results given by SP-SUBDUE depends directly on the quality of the partitions produced by the graph partitioning algorithm being used. Overall SP-SUBDUE was found to give the best results, mostly because it had the least amount of communication overhead. Our work improves upon SP SUBDUE to improve its performance and the quality of its results.

Related Work

While most graph mining algorithms are designed for the single processor environment, there are some that have been extended to be suitable for use in a parallel multiprocessor/ multicore environment. For the single large graph Kuramochi and Karypis developed a frequent graph miner called SiGraM (Kuramochi and Karypis 2004). It contained two algorithms HSiGraM and VSiGraM. HSiGraM or Horizontal SiGraM traversed the search space of frequent subgraphs in a breadth first manner. It essentially relied on a candidate generate and test approach. VSiGraM or Vertical SiGraM traversed the search space in a depth first manner. Reinhardt and Karypis extended this work by parallelizing the VSiGraM algorithm. Reinhardt et al. (Reinhardt and Karypis 2007) parallelized the algorithm in two ways. At a high level they parallelized the two pruning techniques that VSiGraM employs, namely checking whether the size-k subgraph whose expansion is the current size-(k+1) subgraph is actually its generating parent. The second pruning technique is the calculation of frequency for the size-(k+1) candidate subgraph. The parallelization also takes place during the recursive call to the function that extends size-i subgraphs to size-(i+1). At a finer level the authors even parallelize the generation of non-isomorphic size-(i+1) candidate subgraphs from a given size-i frequent subgraph.

SP-SUBDUE2 Master:
Input: M, the number of best substructures to be displayed
Output: B, the set of best substructures
1) begin
2) C = ∅ //*the list of candidate substructures*
3) P = ∅ //*the list of graph partition sizes*
4) N = ∅ //*the list of num instances of substructures*
5) T = ∅ //*the list of #. of recvd. Substructures*
6) I = ∅ //*the list of node IDs that first discover a substructure*
7) S = ∅ // the list of substructure sizes
8) for every child node do
    a) P_i = Size of partition received from child i
9) for every child node do
    a) C_i = Best substructure received from child i
    b) T_i = Number of instances of child i
    c) S_i = Size of substructure received.
    d) I_i = ID of node first discovering this substructure
10) for every substructure C_i do
    a) for every *j in (1 to #nodes)* do
        i) if ($I_i \neq I_j$ and $i \neq j$) send C_i to node j for evaluation
        ii) else $N_i = N_i + T_j$
    b) for every *j in (1 to #nodes)* do
        i) if ($I_i \neq I_j$ and $i \neq j$)
            (1) $N_i = N_i$ + Received number of instances and value for C_i from node j
    c) calculate value for C_i based on N_i, S_i and P_i
11) B = M best substructures from C
12) output B

SP-SUBDUE2 Child:
Input: Graph Partition G_i
Output: Best substructure, Evaluation of substructure
1) begin
2) discover local best substructures and store in L
3) send only best substructure to master
4) receive substructure c_x from master for evaluation on local partition
5) compare c_x to the substructures in L. If any match return the value and number of instances of that substructure
6) else find all instances of c_x in local partition and calculate value
7) return evaluation to master

Figure 1. SP-SUBDUE-2 algorithm

According to the authors the coarser parallelization gave consistently better performance as opposed to the finer parallelization which showed decreased performance as more graphs were added. While our problem is similar to the problem solved by VSiGraM, our approach is slightly different. We take the approach of statically partitioning the input graph among the various processors of the system, whereas in ParVSiGraM, the input graph is not explicitly partitioned. The authors have focused their efforts on functional partitioning. Moreover, SP-SUBDUE makes sure that the child nodes do not have to send data back and forth as they evaluate portions of the graph not present in their local partition.

There has also been some work in extending graph transaction based graph miner algorithms. Buehrer et al. in (Buehrer et al. 2005) designed a parallel approach to gSpan. They evaluated level-wise partitioning and dynamic partitioning methods in order to control the granularity of tasks. They also evaluated three queuing models, namely, global queues, hierarchical queues and distributed queues. They showed that dynamic partitioning and distributed queues give best performance. Meinl et al. (Meinl et al. 2006) discussed a parallel version of MofA and gSpan. Fatta et al. (Di Fatta and Berthold 2006) also discussed a distributed approach to frequent subgraph mining using partitioning of the search space, distributed task queues with dynamic load balancing and a peer-to-peer communication framework.

Approach

The efficiency improvements we make are to the Static Partitioning version of SUBDUE, which we here call SP-SUBDUE. In SP-SUBDUE, the child nodes would work on their local partition. Once they had discovered the substructures, they would report the best one back to the master. As the master received discovered substructures from the children, it would check to see if this substructure was isomorphic to a substructure already in the queue of received substructures. Once the master had received substructures back from all the children, it would sequentially send out each substructure on its queue of discovered substructures to all but the child who had discovered it. The child nodes would then do a subgraph isomorphism test to find all instances of the substructure sent to them for evaluation on their local partitions and report back to the master with the evaluated compression value of the substructure on their local partition. The master would simply add up the compression values for that substructure from all the partitions and store it in a list of evaluated substructures. It would repeat this procedure for all the substructures on its discovered list. At the end it would report the best M substructures on its evaluated list, where M is specified by the user.

We make the following improvements. Initially as the master receives substructures from the children, it checks in its list of previously discovered substructures to see if the current substructure is isomorphic to any in the list. If it is then it stores the ID of the node that first found the substructure along with the substructure (Line 9d). At the end of the discovery phase the discovered list consists of the received substructures along with the ID of the node that first found it. The master then sends a substructure for evaluation only to the children that sent a different best substructure than the one being evaluated currently (Line 10a-i). For the ones that reported the same best substructure the master simply adds up the count of instances discovered by that node for this substructure (Line 10a-ii). The master uses the size of the partitions of every node to compute the final compression value (Line 10c). This value is the actual compression value that would be discovered on the complete input graph taking into account the information lost across the graph's partition boundaries. Whenever a child receives a substructure for evaluation, it performs graph matching to see if it is present in its list of locally discovered substructures (Line 5 – Child). Only if the substructure is not present in this list does the child make a subgraph isomorphism call. The intuition behind this being that if the graph partitioning algorithms do a good job, and the

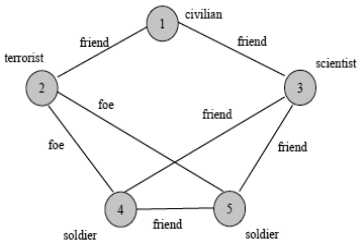

Figure 2. Substructure embedded in synthetic graph

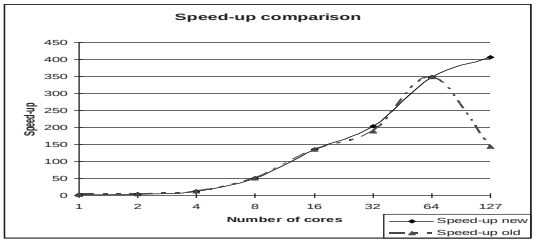

Figure 3. Speed-up of SP-SUBDUE-2 vs SP-SUBDUE

partitions resemble each other to a great degree, the substructures discovered by the child nodes will be similar even though they may vary in their compression value. If so then we would be saving on subgraph isomorphism calls. The SP-SUBDUE-2 algorithm is given in Fig 1.

Experiments

In our experiments we use an IBM Model 1350 HPC made up of 164 Idataplex server nodes with 2 physical processors. Each processor has six cores each making up a total of 12 cores per node operating at 2.66 Ghz. The nodes individually have 24 GB RAM. They use 40 Gb/s Quad Data Rate Mellanox Infiniband in 100% non blocking configuration. We seek to test the scalability of SP-SUBDUE-2 and demonstrate that it scales better than SP-SUBDUE. We also show that SP-SUBDUE-2 gives the same accuracy as serial SUBDUE after evaluating a smaller number of substructures and taking much less time.

Scalability

To perform the scalability testing, we create a synthetic graph with a particular subgraph embedded in it. The algorithm used to create the artificial graph is called "Subgen" and is available from the author. "Subgen" creates graphs of a desired size and with a desired number of embeddings of a provided subgraph, along with additional structure to meet user-specified parameters about graph properties. The instances of that subgraph comprise 70% of the artificial graph. We test scalability by doubling the graph size, keeping the same percentage of subgraph instances and doubling the number of processors. The graphs are partitioned using the METIS algorithm developed by Karypis et al. (Karypis and Kumar 1998). METIS works using a multi-level graph partitioning scheme. The technique used by METIS is to first reduce the size of the graph, partition this smaller graph and then increase the size of the smaller graph to get the actual partitions.

The subgraph that we have chosen is given in Figure 2. Our synthetic graph consists of only this graph as our graph generator program can handle only one embedded substructure at the moment. The distribution of the various node labels and edge labels in the graph are given as follows.

Distribution of vertex labels:

a) soldier - 0.15
b) scientist - 0.05
c) terrorist - 0.1
d) civilian – 0.7

Distribution of edge labels:

a) friend - 0.65
b) foe - 0.35

This particular distribution is chosen to reflect a scenario where there are more 'civilians' than others. The distributions of the edge labels roughly reflect the distribution of the edge labels inside the subgraph. We ran SP-SUBDUE-2 on the graphs, where the graphs as well as the cores in the system double in size, keeping the distributions the same.

TABLE I. SCALABILITY OF SP-SUBDUE-2 ON SYNTHETIC GRAPH

No. cores	**Results**			
	Graph Size	***Time***	***Similarity***	***Compression***
1	750 v, 1550 e	16s	100.00%	2.19973
2	1500 v, 3100 e	14s	100.00%	1.75519
4	3000 v, 6200 e	8s	100.00%	1.59308
8	6000 v, 12400 e	13s	100.00%	1.49339
16	12000 v, 24800 e	8s	100.00%	1.51554
32	24000 v, 49600 e	14s	100.00%	1.47269

Our experimental results as given in Table I show us that in terms of raw execution (wall-clock) time SP-SUBDUE-2 gives similar execution times as the graph size and number of cores is increased, and finds the substructure embedded in the large graph. We were unable to go beyond 32 cores because of the infeasible times taken to generate the synthetic graphs beyond the size of the graph used on 32 cores.

We performed a second scalability test by generating a graph with the same properties as above, only having 24,000 vertices and 49,600 edges. We made partitions of this graph for 1, 2, 4, 8, 16, 32, 64 and 127 child cores. We did not test for 128 child cores as we needed one core for the master and were therefore unable to find a multiple for 129 total nodes. We then ran experiments using these settings for both the new version as well as the old version of SP-SUBDUE. Our results given in Fig 3 and 4 show that while the old version gives speed-up (serial_time/parallel_time) and efficiency (speed-up/#nodes) almost similar to the new, it begins falling off on both

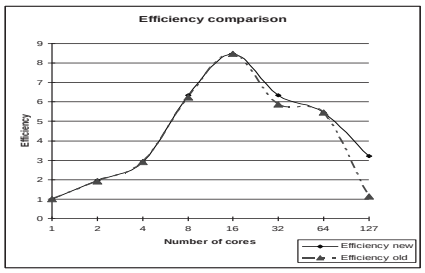

Figure 4. Efficiency of SP-SUBDUE-2 vs SP-SUBDUE

curves earlier than the new version. In our calculations we use the execution time taken from SUBDUE in seconds. We measure the time based on when the master process terminates.

From our experiments we see that while SP-SUBDUE-2 offers only a slight improvement in terms of execution time, it shows improved speed-up and efficiency. This happens because SP-SUBDUE saved on data communication times by simply adding up the values sent after evaluation for a particular substructure to get its final value. SP-SUBDUE-2 incurs these communication costs, due to requesting more data like the size of the local partitions. However the improvements that we have made not only compensate for this overhead, but allow the algorithm to now calculate the true compression value as well as the correct number of instances (the implementation of SP-SUBDUE made an extra addition while computing number of instances). Thus in terms of quality too it gives better results. Figures 3 and 4 show the speed-up and efficiency comparison between SP-SUBDUE (old) and SP-SUBDUE-2 (new). The speed-up of SP-SUBDUE-2 as well as SP-SUBDUE is superlinear because the run-time of SUBDUE is nonlinear in terms of the graph size. As each core is simply running SUBDUE on a smaller graph the time taken by SP-SUBDUE would be less than SUBDUE (Cook et al. 2001).

Quality

The quality of substructures is measured in terms of similarity to the substructures discovered by serial SUBDUE. The quality of the substructures returned by SP-SUBDUE should improve with respect to the compression value up to a certain limit. If the partitions get too small however the quality will decrease since the individual nodes would never see subgraphs larger than the size of the individual partitions even if these subgraphs produced better compression. In order to test quality, with increase in partitions for very large graphs, we use a graph containing nuclear smuggling events and links between them. The original nuclear smuggling dataset contains reports of nuclear smuggling events in Russia (McKay, Woessner and Roule 2001) from which Cook et al. created a graph (Cook et al 2009). Our experimental methodology is as follows. We provide serial SUBDUE with a single large graph created from n-copies of the Nuclear Smuggling graph, where n = 2, 4, 8, 16, 32, 64, 127. To SP-SUBDUE-2 we provide the n-copy of the graph when we use 'n' child nodes and one master node.

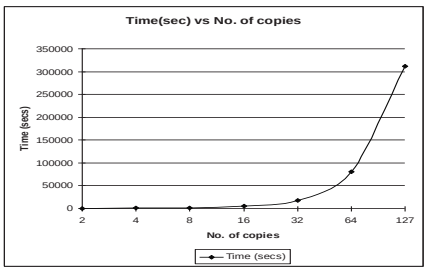

Figure 5. Time taken by serial SUBDUE to discover (6v,5e) substructure setting no. of substructures evaluated to 2036

Every node has its own copy of the Nuclear Smuggling graph. We force the algorithms to consider a larger substructure to illustrate the difference in run-times between serial SUBDUE and SP-SUBDUE-2. We observe that the number of substructures that serial SUBDUE needs to evaluate is larger than the number of substructures that SP-SUBDUE-2 needs to evaluate in order to discover the same best substructure. Moreover the time taken by serial SUBDUE to analyze the increasingly larger graphs follows an exponential growth curve (Fig. 5), whereas the time taken by SP-SUBDUE-2 remains relatively constant (Fig. 6) with a slight increase due to the communication required when switching from using the cores on the same node to cores on separate nodes (this was verified by using 3 cores 11 partitions and 11 cores and 3 partitions respectively). The limit on the number of substructures was chosen using the formula:

$$l = \#init\_subs + (beam * size(desired\_sub))$$

where l = limit of substructures expanded,

#init_subs = number of initial substructures,

beam = beam width, and

size(desired_sub) = size of the desired substructure.

The expression was formulated to force the algorithm to consider substructures of the desired size. The initial substructures considered by the algorithm are all vertices having a count of at least 2. After the algorithm has evaluated these substructures it would then start expanding them by one node. After each pass through the beam, the best substructure would expand by one edge. Hence the number of substructures evaluated by the algorithm before it reaches a substructure of the desired size would be at most: *(beam*size(desired-sub))* .

Results and Discussions

From our experiments we see that SP-SUBDUE-2 gives better speed-up and efficiency than SP-SUBDUE. Table I shows that the time taken by SP-SUBDUE-2 remains the same as the number of cores increases proportionally with the size of the graph. From Fig. 3 and 4 we see that the speed-up and efficiency respectively of SP-SUBDUE-2 is better than SP-SUBDUE as the curve for SP-SUBDUE falls off earlier in both cases than the curves for SP-SUBDUE-2.

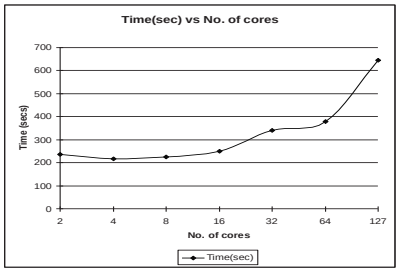

Figure 6. Time taken by SP-SUBDUE-2 to discover (6v,5e) substructure setting no. of substructures evaluated to 945

From Figs. 5 and 6 we can see that serial SUBDUE needs to expand significantly more substructures in order to reach the same conclusion as SP-SUBDUE-2. Consequently its execution time increases to prohibitive limits as the graph size increases.

Conclusions

Our scalability tests show clearly that SP-SUBDUE-2 outperforms SP-SUBDUE. We would have liked to test the quality of results produced by SP-SUBDUE-2 even more thoroughly by actually partitioning a real graph dataset; however we could not do this due to the lack of availability of labeled graphs that could be partitioned using METIS. We will investigate this issue further. Future work also includes extending these improvements or designing new ones for FP-SUBDUE and DP-SUBDUE, and to the supervised learning scenario. The scheme that we have outlined in this paper can be applied to any frequent subgraph miner or maximally-compressing subgraph algorithm that uses static data partitioning. The idea is that if the partitions produced are good, not much information will be lost due to edge cuts. An aggregation of local best substructures would then be a very close approximation to the actual result on the unpartitioned data. An evaluation of the local best substructure of one child by the other children would further increase the accuracy. The evaluation and aggregation can be computed intelligently using some additional memory at the master and child nodes in order to reduce the time taken. While to our knowledge SP-SUBDUE is the only parallel graph miner that uses static partitioning of the input data, ones designed in future can take advantage of our scheme.

Acknowledgments. We would like to thank Randall Svancara of the WSU HPC team for helping us immensely in getting the jobs set up and for his help in figuring out issues that we encountered while testing the system.

References

Cook, D.J., Holder L.B., Galal, G., and Maglothin, R. 2001. Approaches to parallel graph-based knowledge discovery. *Journal of Parallel Distrib. Comput.* 61, 3 (March 2001): 427-446.

Cook, D.J. and Holder, L.B. 1994. Substructure discovery using minimum description length and background knowledge. *J. Artif. Int. Res.* 1, 1 (February 1994): 231-255.

Reinhardt, S. and Karypis, G. 2007. A multi-level parallel implementation of a program for finding frequent patterns in a large sparse graph. In *Proceedings of Twenty First IEEE International Parallel & Distributed Processing Symposium*, 1-8. Long Beach, Calif.: IEEE Press

Buehrer, G., Parthasarathy, S., Nguyen, A., Kim, D., Chen, Y. K., and Dubey, P. 2005. Parallel Graph Mining on Shared Memory Architectures, Technical report, Ohio State University, Columbus, OH.

Cook, D.J. and Holder, L.B. 2006. *Mining Graph Data.* Hoboken, N.J.: John Wiley & Sons.

Karypis, G. and Kumar, V. 1998. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel Distrib. Comput*. 48: 96 -129

Rajappa, S. 2003. Data mining in nonuniform distributed databases. Master's thesis, The University of Texas, 2003.

Cook, D.J., Holder, L.B., Thompson, S., Whitney, P. and Chilton, L. 2009. Graph-Based Analysis of Nuclear Smuggling. *Journal of Applied Security Research* 4: 501-517.

McKay, S.J., Woessner, P.N., and Roule, T.J. 2001. Evidence extraction and link discovery (EELD) seedling projects, database schema description, version 1.0, Technical Report 2862, Veridian Systems Division, August 2001.

Meinl, T., Wörlein, M., Fischer, I., and Philippsen, M. 2006. Mining Molecular Datasets on Symmetric Multiprocessor Systems. In *Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics* 6: 1269 - 1274. Taipei, Taiwan. IEEE Press

Di Fatta, G. and Berthold, M.R. 2006. Dynamic load balancing for the distributed mining of molecular structures. *IEEE Transactions on Parallel and Distributed Systems, Special Issue on High Performance Computational Biology*, 17(8):773-785.

Kuramochi, M. and Karypis, G. 2004. Finding frequent patterns in a large sparse graph. In Proceedings of 2004 SIAM International Conference on Data Mining, 345 – 356. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Pacheco, P.S. 1996. *Parallel Programming with MPI.* San Francisco, Calif.: Morgan Kaufmann Publishers Inc.

Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. and Menon, R. 2001. *Parallel Programming in OpenMP*. San Francisco, Calif.: Morgan Kaufmann Publishers Inc.

Rissanen, J. 1989. *Stochastic Complexity in Statistical Inquiry*. Singapore: World Scientific.

Manocha, N.; Cook, D.J.; and Holder, L.B. 2001. Structural web search using a graph-based discovery system. *Intelligence Magazine* 12(1): 20–29.

Noble, C. and Cook, D. August 2003. Graph-based anomaly detection. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge discovery and Data Mining, Washington, D.C.* New York, NY: ACM Press

Su, S., Cook, D.J. and Holder, L.B. 1999. Knowledge discovery in molecular biology: Identifying structural regularities in proteins. *Intelligent Data Analysis* 3:413–436.