Approaches to Parallel Graph-Based Knowledge Discovery[1]

Diane J. Cook, Lawrence B. Holder, Gehad Galal, and Ron Maglothin
Department of Computer Science Engineering
University of Texas at Arlington
Box 19015, Arlington, TX 76019-0015
Email: {cook, holder, galal, maglothi}@cse.uta.edu
Phone: (817) 272-3606
Fax: (817) 272-3784
URL: http://cygnus.uta.edu/subdue

Contact Author: Diane J. Cook

Abstract

The large amount of data collected today is quickly overwhelming researchers' abilities to interpret the data and discover interesting patterns. Knowledge discovery and data mining systems contain the potential to automate the interpretation process, but these approaches frequently utilize computationally expensive algorithms. In particular, scientific discovery systems focus on the utilization of richer data representation, sometimes without regard for scalability.

This research investigates approaches for scaling a particular knowledge discovery / data mining system, SUBDUE, using parallel and distributed resources. SUBDUE has been used to discover interesting and repetitive concepts in graph-based databases from a variety of domains, but requires a substantial amount of processing time. Experiments that demonstrate scalability of parallel versions of the SUBDUE system are performed using CAD circuit databases, satellite images, and artificially-generated databases, and potential achievements and obstacles are discussed.

1 Introduction

One of the barriers to the integration of scientific discovery methods into practical data mining approaches is their lack of scalability. Many scientific discovery systems are motivated from the desire to evaluate the correctness of a discovery method without regard to the method's scalability.

[1]This work is supported by NSF grant IRI-9502260.

As an example, our SUBDUE system was developed to evaluate the effectiveness of the minimum description length (MDL) principle at discovering regularities in a variety of scientific domains. Description of the serial SUBDUE algorithm and its applications is provided in the literature [6, 8].

Another factor is that some scientific discovery systems deal with richer data representations that only degrade scalability. A number of linear, attribute-value-based approaches have been developed that discover concepts in databases and can address issues of data relevance, missing data, noise, and utilization of domain knowledge [3, 14, 15]. However, much of the data being collected is structural in nature, requiring tools for the analysis and discovery of concepts in structural data [13]. For example, the SUBDUE system uses a graph-based representation of the input data that captures the structural information. Although the subgraph isomorphism procedure needed to deal with this data has been polynomially constrained within SUBDUE, the system still spends a considerable amount of computation performing this task.

The goal of this research is to demonstrate that knowledge discovery in databases (KDD) systems can be made scalable through efficient use of parallel and distributed hardware. To accomplish this goal, we investigate several basic approaches that can be used to parallelize KDD systems and compare the results of these approaches when applied to the SUBDUE discovery system.

Related approaches to scaling data mining and discovery systems have been pursued. Parallel MIMD approaches to concept learning have included partitioning the data set among processors [2, 23, 26] and partitioning the search space among available processors [9, 10, 18, 21]. Data partitioning approaches have also been effective for certain limited approaches to data mining [4, 22] and knowledge discovery [20] on SIMD architectures. Improving the scalability of scientific discovery systems will help break down the barrier that currently excludes these techniques from practical data mining approaches.

Figure 1: Natural rubber atomic structure.

2 Overview of SUBDUE

We have developed a method for discovering substructures in databases using the minimum description length (MDL) principle introduced by Rissanen [24] and embodied in the SUBDUE system. Based on the MDL principle, SUBDUE discovers substructures that compress the original data and represent structural concepts in the data. Once a substructure is discovered, the substructure is used to simplify the data by replacing instances of the substructure with a pointer to the newly discovered substructure. The discovered substructures allow abstraction over detailed structures in the original data. Iteration of the substructure discovery and replacement process constructs a hierarchical description of the structural data in terms of the discovered substructures. This hierarchy provides varying levels of interpretation that can be accessed based on the specific goals of the data analysis.

The substructure discovery system represents structural data as a labeled graph. Objects in the data map to vertices or small subgraphs in the graph, and relationships between objects map to directed or undirected edges in the graph. A *substructure* is a connected subgraph within the graphical representation. This graphical representation serves as input to the substructure discovery system. An *instance* of a substructure in a graph is a set of vertices and edges from the input graph that match, graph theoretically, to the graphical representation of the substructure.

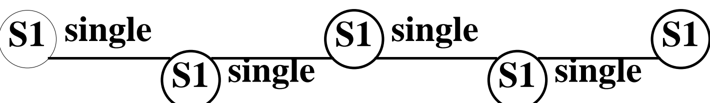


Figure 2: Rubber graph compressed using the discovered substructure.

Figure 1 shows a sample database that is input to SUBDUE, representing the atomic structure of natural rubber. The input graph represents atoms as vertices, and single or double bonds as labeled undirected edges between the vertices. The highlighted substructure, substructure S1, is selected as the best subgraph to describe the input database. The five instances of the substructure are replaced by a single vertex representing the discovered concept. The graph that results from compressing the rubber database is shown in Figure 2. This discovery and compression helps the user understand the graphical database and exploit the knowledge content of the data encoded in the graph vertices and edges.

Figure 3 shows a sample input database containing a portion of a DNA sequence. In this case, atoms and small molecules in the sequence are represented with labeled vertices in the graph, and the single and double bonds between atoms are represented with labeled edges in the graph. SUBDUE discovers substructure S_1 from the input database. After compressing the original database using S_1, SUBDUE discovers substructure S_2, which when used to compress the database further allows SUBDUE to discover substructure S_3. Such repeated application of SUBDUE generates a hierarchical description of the structures in the database.

The substructure discovery algorithm used by SUBDUE is a computationally-constrained beam search. SUBDUE's discovery algorithm is shown in Figure 4. The first step of the algorithm is to initialize ParentList (containing substructures to be expanded), ChildList (containing substructures that have been expanded), and BestList (containing the highest-valued substructures SUBDUE has found so far) to be empty, and to set ProcessedSubs (the number of substructures that have been

Figure 3: Sample results of Subdue on a protein sequence.

expanded so far) to 0. Each of the lists is a linked list of substructures, sorted in nonincreasing order by substructure value. For each unique vertex label, a substructure is assembled whose definition is a vertex with that label, and whose instances are all of the vertices in G with that label. Each of these substructures is inserted in ParentList.

```
SUBDUE( Graph, BeamWidth, MaxBest, MaxSubSize, Limit )
    ParentList = {}
    ChildList = {}
    BestList = {}
    ProcessedSubs = 0
    Create a substructure from each unique vertex label and its single-vertex
        instances; insert the resulting substructures in ParentList
    while ProcessedSubs <= Limit and ParentList is not empty do
        while ParentList is not empty do
            Parent = RemoveHead( ParentList)
            Extend each instance of Parent in all possible ways
            Group the extended instances into Child substructures
            foreach Child do
                if SizeOf( Child ) <= MaxSubSize then
                    Evaluate the Child
                    Insert Child in ChildList in order by value
                    if Length( ChildList ) > BeamWidth then
                        Destroy the substructure at the end of ChildList
            ProcessedSubs = ProcessedSubs + 1
            Insert Parent in BestList in order by value
            if Length( BestList ) > MaxBest then
                Destroy the substructure at the end of BestList
        Switch ParentList and ChildList
    return BestList
```

Figure 4: Subdue's discovery algorithm.

The inner *while* loop is the core of the algorithm. Each substructure is removed from the head of ParentList, and its instances are extended in all possible ways. This is done by adding

a new edge and vertex in G to the instance, or by adding a new edge between two vertices that are already part of the instance. The first instance of each unique expansion becomes a definition for a new child substructure, and all of the child instances that were expanded in the same way become instances of that child substructure. In addition, child instances that were generated by different expansions, and that match the child substructure definition, also become instances of the child substructure. Each child is then evaluated (based on its ability to compress the database following the MDL principle) and inserted in the sorted ChildList. The beam width of the search is enforced by controlling the length of ChildList: after inserting a new child into ChildList, if the length of ChildList exceeds the BeamWidth, the substructure at the end of the list is destroyed. The parent substructure is inserted in BestList; the same pruning mechanism is used to limit the length of BestList to be no greater than MaxBest. When ParentList has been emptied, ParentList and ChildList are switched, so that ParentList now holds the next generation of substructures to be expanded. SUBDUE's running time is constrained to be polynomial by the BeamWidth and Limit (a user-defined limit on the number of substructures to process) parameters, as well as by computational constraints placed on the inexact graph match algorithm.

Because instances of a substructure can appear in different forms throughout the database, an inexact graph match is used to identify substructure instances. Subgraphs are considered to be instances of a substructure definition if the cost of transforming the subgraph into a graph that is isomorphic with the substructure definition does not exceed a user-defined threshold. Transformations between graphs can include addition or deletion of vertices, addition or deletion of edges, vertex label substitutions and edge label substitutions. Performing an inexact match allows the discovered substructures to abstract away minor variations in the substructure instances. The varied instances may be compressed by replacing the instance with a single node representing the substructure (abstracting away instance differences) or with a node and annotations describing the

changes from the substructure definition.

SUBDUE discovers substructures that compress the amount of information necessary to conceptually describe the database. To allow SUBDUE to discover substructures of particular interest to a scientist in a specific domain, the user can direct the search with expert-supplied background knowledge [8]. Background knowledge can take the form of known substructure models that may potentially appear in the database, or graph match rules to adjust the cost of each inexact graph match test. Unlike other existing approaches to graph-based discovery [5, 19, 25, 28, 29], SUBDUE is effective at finding interesting and repetitive substructures in any structural database with or without domain-specific guidance.

The results of the scalability study in section 3 are demonstrated on databases in two different domains. The first type of database is the CAD circuit description of an A-to-D converter provided by National Semiconductor. The graph representation of this database contains 8,441 vertices and 19,206 edges. The second type of database is an artificially-constructed graph in which an arbitrary number of instances of a predefined substructure are embedded in the database surrounded by vertices and edges with random labels and connectivity. The embedded substructure covers almost half of the graph and exhibits significantly less variation in substructure instances than in the CAD database. The tested artificial graph contains 1,000 vertices and 2,500 edges.

To test scalability on even larger databases while maintaining the characteristics of these original two graphs, we generate multiple copies of the CAD and ART graphs and merge the copies together by arbitrarily connecting the individual graphs, yielding a new larger graph. The term "n CAD" thus refers to a graph consisting of n copies of the original CAD database with joining edges added, and "n ART" refers to a graph consisting of n copies of the artificial database with additional joining edges. To ensure that the individual copies are linked tightly together, 20% of the number of edges in the original graph are randomly added to the merged graph.

Usage of domain knowledge	Discovered Substructures	Compression	Nodes Expanded	Human Rating [std dev]	Instances
no domain knowledge	1	2.70	571,370	4.2 [1.2]	9
	2	3.13	46,422	2.7 [1.2]	3
	3	3.57	59,886	2.7 [1.0]	2
graph match rules	1	1.75	145,175	2.7 [1.7]	6
	2	2.04	39,881	2.7 [1.0]	3
	3	2.63	425,772	1.5 [0.8]	2
model knowledge and graph match rules	1	2.13	24,273	4.3 [1.2]	9
	2	2.94	12,960	4.5 [0.8]	4
	3	3.57	7,432	4.5 [0.8]	2

Figure 5: CAD circuit results.

Figure 5 shows the substructures discovered from a CAD circuit with and without background knowledge, and evaluates the results based on compression obtained with the substructure, time required to process the database (measured in terms of number of graph match nodes generated), a human rating of the substructure interestingness, and the number of substructure instances found in the database. The interestingness of SUBDUE's discovered substructures is rated by a group of 8 domain experts on a scale of 1 to 5, where 1 means the discovered substructures do not represent useful information in the domain and 5 means the discovered substructures are very useful.

Figures 3 and 5 demonstrate two applications of SUBDUE. In addition to these examples, SUBDUE has been successfully applied with and without domain knowledge to databases in domains including image analysis, CAD circuit analysis, Chinese character databases, program source code, chemical reaction chains, Brookhaven protein databases, yeast genome databases, and artificially-generated databases. Evaluation of these applications is described elsewhere [7, 11, 12, 27].

3 Scaling KDD Systems

Although reaching the maximum performance of a given knowledge discovery system is specific to the system itself, making use of parallel and distributed resources can significantly affect the scalability of a KDD system. Parallelizing a knowledge discovery system is not easy. The reason is that many KDD systems rely upon heuristics and greedy algorithms to avoid the intractability inherent in an exhaustive approach. Both heuristics and greedy algorithms share the potential of finding a suboptimal solution and, on closer inspection, a sequentially oriented solution. In many cases serial KDD algorithms can perform better if they are provided with enough history of the problem being solved.

In addition, knowledge discovery systems share common parallelization problems. A problem like matrix multiplication (in its standard form) is easily decomposable and thus parallelizable

with minimal synchronization and communication. In contrast, the knowledge discovered in each step by KDD systems depends heavily on what has been discovered in previous steps. Thus, we cannot decompose the work without increasing the synchronization and communication between the parallel processors which usually results in poor performance.

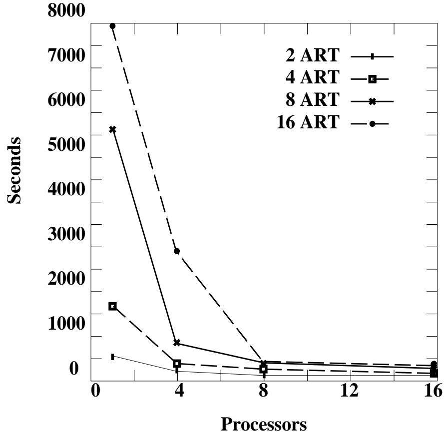
Two main approaches to designing parallel algorithms are the functional parallel approach and the data parallel approach. In the functional parallel approach the algorithm steps are assigned to different processors, while in a data parallel approach each processor applies the same algorithm to different parts of the input data. In our discussion of parallelizing SUBDUE we concentrate on distributed memory architectures because of the improved scalability of such architectures compared to shared memory architectures.

3.1 Functional Parallel Approach—FP-SUBDUE

The main idea behind this algorithm is to divide SUBDUE's search for candidate substructures among processors. The search queue is maintained by one master processor which keeps track of the best discovered substructures. The general description of the algorithm follows.

Each processor starts by discovering initial substructures large enough to obtain a compression greater than 1.0 and informing the master processor of the discovered substructures. The master processor maintains a global search queue containing the substructure definitions. When the master receives the results of an expansion step from a processor, it decides whether to keep the expanded substructures according to the following criteria:

- If that substructure is already discovered, then the new substructure represents a duplication of work and is discarded.
- If the substructure value is not one of the best M so far, then this substructure is discarded.



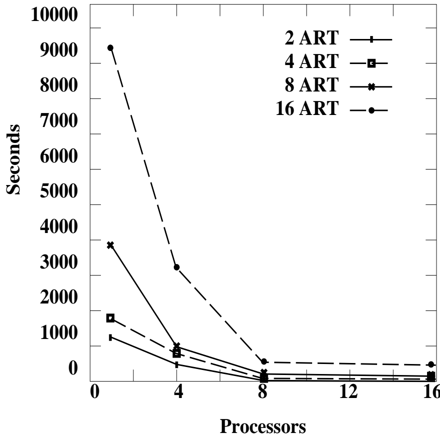


Figure 6: Discovery time of 60 substructures in 2, 4, 8, and 16 ART.

Figure 7: Discovery time of 70 substructures in 2, 4, 8, and 16 ART.

- Otherwise the substructure is kept in the search queue.

Each slave processor keeps or deletes substructures as indicated by the master. If a slave processor does not have any substructures in the global search queue then the master asks another processor which has more than a threshold number of substructures to transfer a substructure to the requesting processor. The algorithm stops when the global search queue is empty or when a certain number of substructures are globally evaluated.

Clearly the type of search employed in this parallel version differs from that of the sequential version. In FP-Subdue any substructure waiting for expansion can be expanded regardless of whether it is the best substructure so far or not. The distribution of effort also allows many substructures to be evaluated that would be discarded in serial Subdue due to the limited beam length.

To demonstrate the speedup of FP-Subdue we test the algorithm on an nCUBE 2 using 1, 4, 8, and 16 processors. Figures 6 through 9 show the resulting decrease in runtime as the number of processors increases for different numbers of generated substructures. A limit on performance

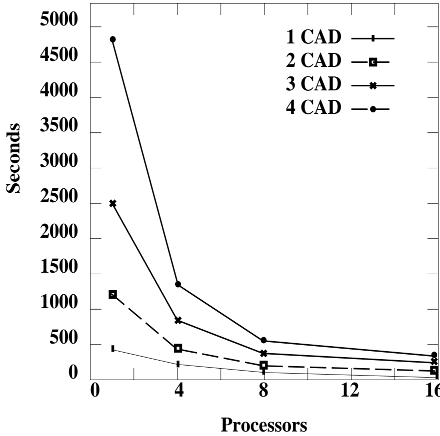


Figure 8: Discovery time of 40 substructures in 1, 2, 3, and 4 CAD.

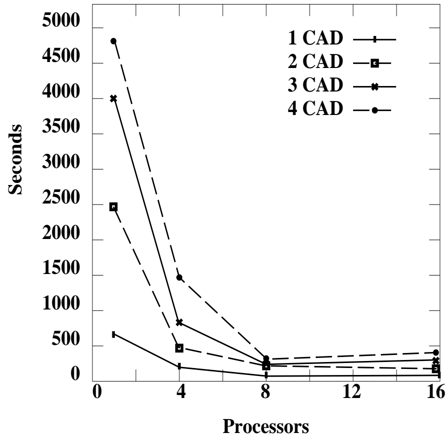


Figure 9: Discovery time of 60 substructures in 1, 2, 3, and 4 CAD.

improvement is reached in each case because of the serial time required to generate and evaluate initial substructures before work is distributed evenly among all processors. The amount of compression achieved may also sometimes increase as the number of processors increases. For example, after 30 substructure evaluations FP-SUBDUE discovered a substructure that is 21% better than any substructure discovered by serial SUBDUE while still yielding a substantial speedup. This is due to the fact that the beam width combined over all processors is greater than a single beam width on the serial machine, and thus a greater number of substructures can be considered.

It may seem that the comparison between this parallel version and the sequential version is not logical as each effectively uses a different type of search, but we should note that if the sequential version were to follow the same kind of search used by the parallel version (for example select the next substructure to expand randomly from the search queue) then the parallel version is guaranteed to give us a near linear speedup, discovering substructures which are as good as those which would be discovered by the sequential version.

3.2 Dynamic Partitioning Approach—DP-Subdue

In the second functional parallel approach, Dynamic-Partitioning Subdue, each processor starts evaluating a disjoint set of the input data. During discovery, each processor enlarges its set as required by the discovered substructures. When DP-Subdue is run on a graph with M vertex labels using P processors, processor i begins processing a candidate substructure corresponding to the ith unique label in the graph ($i < M$). Each processor receives a copy of the entire input graph, and begins processing its assigned candidate substructures. Each processor continues expanding and processing a portion of the possible substructures until the combined number of processed substructures exceeds a given limit or until all processors are idle.

Note that there exists a danger of replicating work across multiple processors. Consider the input graph shown in Figure 10. If label $V1$ is assigned to processor 1 and $V2$ is assigned to processor 2, both processors will expand this single-vertex substructure to the two-vertex substructure highlighted in the figure. If label $V3$ is assigned to processor 3, eventually all three processors will be working on the same candidate substructures. To prevent this duplication of effort, DP-Subdue constrains processors expanding a substructure to only include vertices with a label index greater than the processor ID. In the example scenario, only processor 1 can generate the highlighted substructure.

One hindrance to good performance from a parallel implementation is excessive idling of processors. When a processor runs out of work, it requests work from a neighboring processor. If the neighbor has a sufficient number of substructure candidates left, the neighbor passes the highest-valued substructure and corresponding instances to the requesting processor. If no work can be shared, the requesting processor continues to ask for work until work can be shared or DP-Subdue finishes computation.

When a substructure is transferred to a requesting processor, the requesting processor assumes

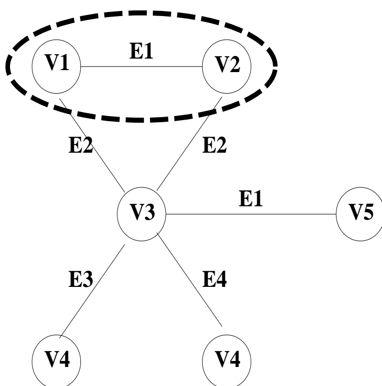


Figure 10: Sample input graph.

the identity of the original processor. This means that the requesting processor can now expand the substructure in all the ways originally permitted to the old processor. To ensure that both processors do not generate the same substructures the original processor keeps a list of all transferred substructures and makes sure that no substructure subsuming a transferred substructure is ever expanded.

Quality control is also imposed on the processors in the DP-Subdue system. One processor is designated as the master processor, and this master regularly receives status information on each processor. This information includes whether the processor is active or idle, the number of processed substructures, and the average value of candidate substructures. Using this information, at regular intervals the master computes the overall average substructure value and sends this information to all processors. Each individual processor then prunes from its list all substructure candidates whose value is less than the global average. In this way no processor is working on a poor substructure candidate that will not likely affect the results of the system. The master processor also collects and sorts the final list of best substructures found from each processor.

The partitions here are logical: the set of all instances of all the candidate substructures discovered by a processor constitutes its partition. Thus, the partitions assigned to different processors

grow dynamically as the discovered substructures become more complex. Clearly, this kind of partitioning will not cause any loss of information because the database itself is not split.

Results from the DP-SUBDUE system indicate that very limited speedup can be achieved by distributing the substructure expansion and evaluation. The work done to limit duplicate work and to load balance the system consumes considerable time in processing and communication. Also as SUBDUE uses a greedy approach many processors follow the same expansion path resulting in a tendency among processors to generate duplicated work. In addition, the memory requirements of this approach are excessive because the entire database is copied on each processor. The speedup achieved is very limited and the results are not included in this paper.

3.3 Static Partitioning Approach—SP-SUBDUE

In this section we illustrate a data parallel approach to substructure discovery by statically partitioning the data among the processors using SP-SUBDUE. This type of parallelism is appealing in terms of memory usage and speedup. A similar form of parallelism can be found in classifier systems that use bagging and boosting methods to create independent classifiers from disjoint training sets and combining a global concept from the individual classifiers [1].

In SP-SUBDUE we partition the input graph into n partitions for n processors. Each processor performs sequential SUBDUE on its local graph partition and broadcasts its best substructures to the other processors. The processors then evaluate the communicated substructures on their local partitions. Once all evaluations are complete, a master processor gathers the results and determines the global best discoveries.

The speedup achieved by SP-SUBDUE as well as the quality of discovered substructures depends on the graph partitioning step. When partitioning the graph, we want to balance the work load equally between processors while retaining as much information as possible (edges along which the

graph is partitioned may represent important information). SP-SUBDUE utilizes the *Metis* graph partitioning package [17]. Metis accepts a graph with weights assigned to edges and vertices, and tries to partition the graph so that the sum of the weights of the cut edges is minimized and the sum of vertex weights in each partition is roughly equal. We assign weights only to edges to ensure that each partition will contain roughly the same number of vertices. The weight of each edge is proportional to the frequency with which it appears in the graph. The motivation for this weight assignment is that the frequently occurring edge labels are more likely to be found in frequently-occurring substructures The run time of Metis to partition the CAD database is very small (ten seconds on average for our databases).

Because the data is partitioned among the processors, SP-SUBDUE can also utilize the increased memory resources of a network of workstations using communication software such as the Parallel Virtual Machine (PVM) system [16]. We implemented SP-SUBDUE on a network of 16 Pentium PCs using PVM.

Figures 11 and 12 graph the run time of SP-SUBDUE on the CAD and artificial databases as the number of processors increases. The speedup achieved with the ART database is frequently superlinear. This is because the run time of sequential SUBDUE is nonlinear with respect to the size of the database. Each processor essentially executes a serial version of SUBDUE on a small portion of the overall database, so the combined run time is less than that of serial SUBDUE. The speedup achieved with the CAD database is usually close to linear and sometimes superlinear. Increasing the number of partitions results in a better speedup until the number of partitions approaches the number of vertices in the graph.

To compare results of these databases with the results of databases not created by merging together smaller components, we also graph the run time of SP-SUBDUE on artificial graphs ranging in size from 1,000 vertices to 4,000 vertices (the number of edges is double the number of vertices)

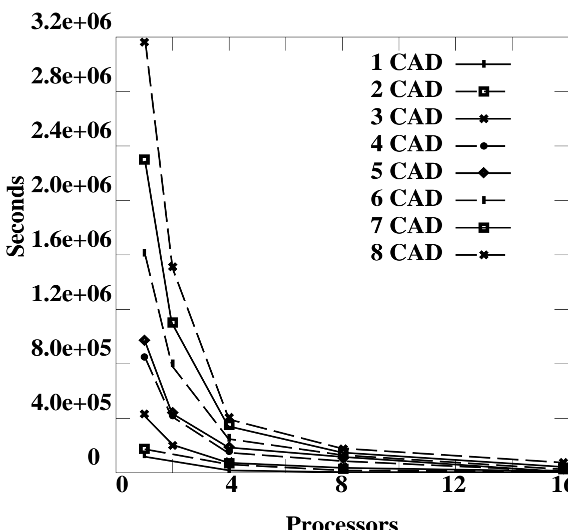


Figure 11: SP-Subdue run time on CAD graphs.

without merging component graphs. The run times of SP-SUBDUE applied to these databases are shown in Figure 13.

Now we turn our attention to the quality of the substructures discovered by SP-SUBDUE. The quality of a substructure is measured in terms of the amount of compression it affords in the original input graph. Let G_s represent the graph G compressed using substructure s. We report $compression = \frac{size(G)}{size(G_s)}$, thus a greater compression value indicates a greater ability to compress the size of the original database using the discovered substructure.

Tables 1 and 2 show the compression achieved for the CAD and ART databases as well as the non-merged artificial graphs when processed by a different number of processors. Regarding the CAD database compression results, we find that a small number of partitions almost always results in a superior compression to that of the sequential version. The reason behind this is the nature of the CAD database. As with many real-world databases, the CAD databases contains many diverse substructures. Treating the entire database as a single partition will result in throwing away good substructures because of the limited search beam. When the database is partitioned

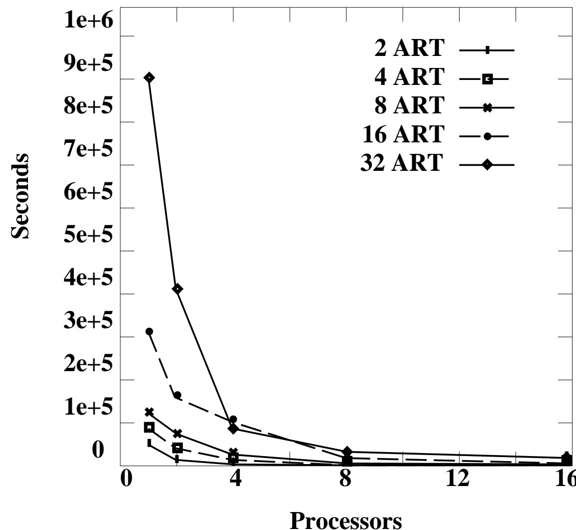


Figure 12: SP-Subdue run time on ART graphs.

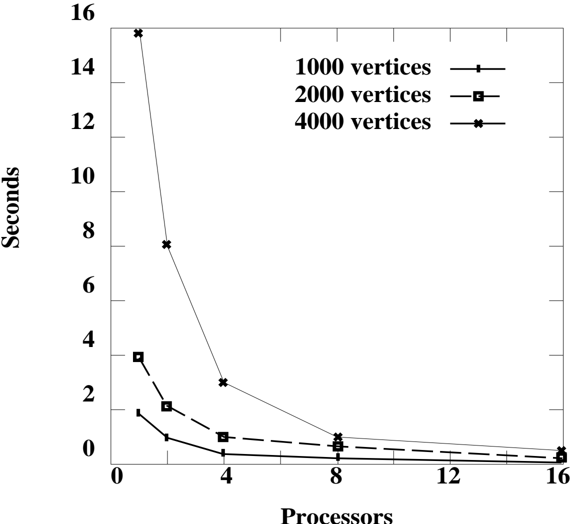


Figure 13: SP-Subdue run time on non-merged artificial graphs.

	Processors				
Database	1	2	4	8	16
1 CAD	1.081	1.280	1.383	1.538	1.436
2 CAD	1.081	1.261	1.294	1.249	1.238
3 CAD	1.081	1.216	1.217	1.261	1.279
4 CAD	1.081	1.189	1.213	1.257	1.297
5 CAD	1.081	1.164	1.188	1.211	1.220
6 CAD	1.081	1.130	1.137	1.149	1.184
7 CAD	1.081	1.123	1.123	1.149	1.155
8 CAD	1.081	1.106	1.113	1.114	1.159

Table 1: CAD database compression results by number of processors.

among several processors, each processor will have a greater chance to deepen the search into the database, because the number of embedded substructures is reduced, resulting in higher-valued global substructures.

The best compression achieved for the merged ART graphs is the compression achieved using the sequential version. This is expected because of the high regularity of this particular set of databases. As mentioned before the merged ART databases have one substructure embedded into it, thus partitioning the database can only cause some instances of this substructure to be lost because of the edge cuts, so we cannot get better compression by partitioning.

Although static data partitioning results in substantial speedup and thus scalability of the discovery system, eventually results will degrade because information is lost along partition boundaries. We investigate two methods of recovering some of this lost information. In the first approach,

	Processors				
Database	1	2	4	8	16
2 ART	2.081	1.979	2.111	2.333	3.080
4 ART	1.969	1.444	1.396	1.333	1.332
8 ART	1.970	1.254	1.147	1.284	1.203
16 ART	1.965	1.094	1.094	1.075	1.073
32 ART	1.866	1.037	1.026	1.025	1.016
1000 Vertices	1.947	2.670	2.920	3.191	3.578
2000 Vertices	2.126	2.709	3.006	3.263	3.348
4000 Vertices	2.131	2.696	2.944	3.140	3.335

Table 2: ART database and non-merged graph compression results by number of processors.

we allow enough overlap between partitions that no edges are lost. In particular, everywhere an edge is cut in the original partition, we add the edge to the partitions containing the edge endpoint vertices, along with the neighboring vertex. In the second approach, we run SP-SUBDUE twice with a different partitioning. No edge or vertex weights are used in the second case, resulting in a different set of partition boundaries. Edges cut in one case are likely to be preserved in the other. When both runs are finished, the substructure yielding the greatest compression over both runs is saved. While this approach increases the run time, speedup is still achieved as a greater number of processors is used.

Figures 14 and 15 show the run times for the original and two modified approaches. The results for the original times, "Overlap" times and "Dual" times are averaged over multiple executions varying the number of processors. As the figures demonstrate, both modified approaches require

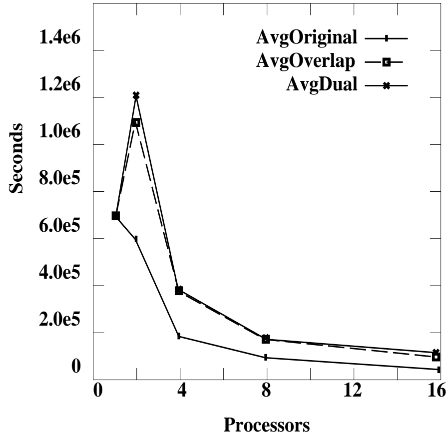


Figure 14: Average run times for CAD databases.

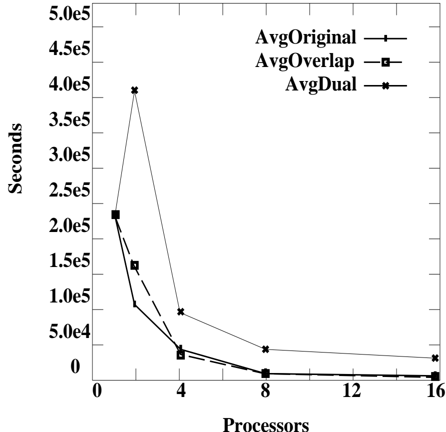


Figure 15: Average run times for ART databases.

substantially longer run time than the original approach. The "Overlap" method requires a longer run time because the number of vertices and edges that must be added to prevent information loss is large, and thus the partitioned graphs remain close to the original graph in size. The "Dual" run times are close to twice the time of the original approach.

Figures 16 and 17 show the compression achieved by each approach, averaged over multiple executions varying the number of processors. These graphs show that the "Overlap" method does not perform as well as the original approach. In these particular graphs, the discovered substructures were fairly small and most instances were found within a partition (not split across partition boundaries). As a result, the inclusion of an overlap area increases the graph size without generally increasing the number of substructure instances, so the resulting compression is less. The best substructures are discovered using the "Dual" approach. Although the original partitions produce the highest-valued substructures in most cases, the second partition does yield better substructures for some of the databases, thus demonstrating that an alternative partition can recapture information lost when the databases are split among processors.

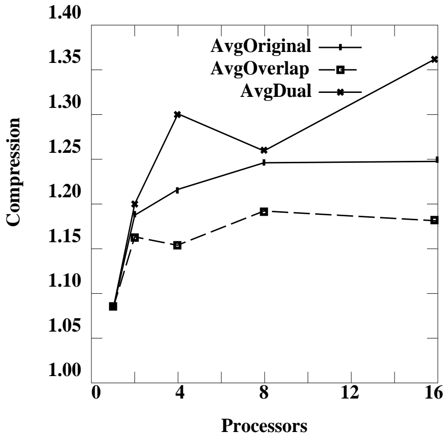


Figure 16: Average compression values for CAD databases.

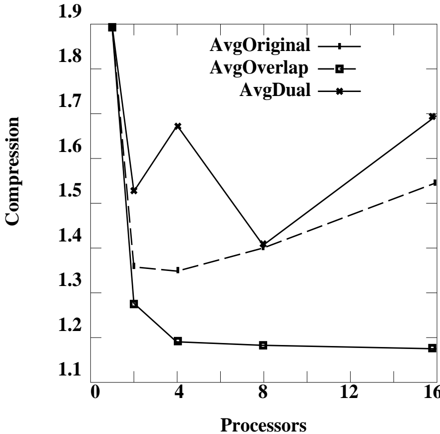


Figure 17: Average compression values for ART databases.

3.4 Comparison

We have described implementations of one functional parallel and two data parallel approaches for improving the scalability of SUBDUE with parallel and distributed resources. When comparing the benefits of the three approaches, DP-SUBDUE is discarded because of poor run time and heavy memory requirements. FP-SUBDUE can prove effective in discovering substructures in very large databases with many embedded substructures due to its unique search algorithm. The reason that the other versions will not discover some of the substructures discovered by FP-SUBDUE is merely because of the limited search queue size (i.e., we can always discover the same substructures by controlling the run time parameters of serial SUBDUE and SP-SUBDUE). The speedup of FP-SUBDUE is plotted along with the speedup of SP-SUBDUE in Figure 18. "ART" speedups are averaged over databases 2ART, 4ART, 8ART, and 16ART, and "CAD" speedups are averaged over databases 1CAD, 2CAD, 3CAD, and 4CAD. The speedups are similar for a smaller number of processors, but show a better runtime performance for SP-SUBDUE as the number of processors increases.

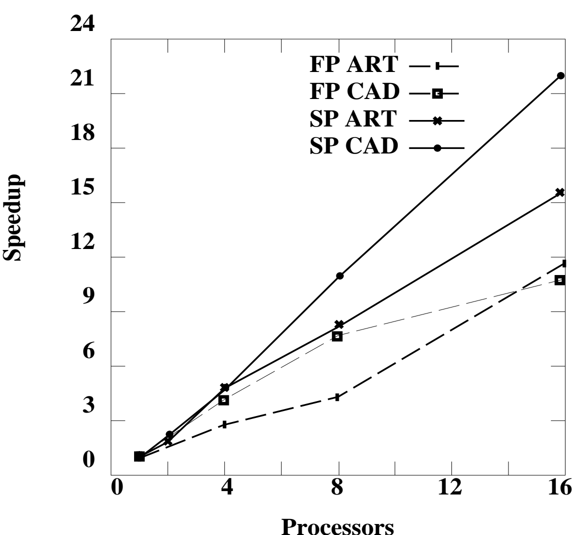


Figure 18: Comparison of FP-Subdue and SP-Subdue speedups.

SP-Subdue is the most interesting approach of all. By partitioning the database effectively, SP-Subdue proves to be a highly scalable system. SP-Subdue can handle huge databases using fewer resources (processing power and memory) than would have been required by the sequential version to handle the same database while discovering substructures of equal or better quality. One of our tested databases representing a satellite image contains 2 million vertices and 5 million edges, yet SP-Subdue is able to process the database in less than three hours. The easy availability of SP-Subdue is greatly improved by using a distributed network of workstations. The minimal amount of communication and synchronization that is required make SP-Subdue ideal for distributed environments. Using the portable message passing interface provided by PVM allows the system to run on heterogeneous networks.

4 Conclusions

The increasing structural component of today's databases requires data mining algorithms capable of handling structural information. The Subdue system is specifically designed to discover

knowledge in structural databases. SUBDUE offers a method for integrating domain independent and domain dependent substructure discovery based on the minimum description length principle. However, the computational expense of a discovery system such as SUBDUE can deter widespread application of the algorithms.

In this paper, we analyze the ability of SUBDUE to scale to large databases. The described parallel and distributed implementations of SUBDUE allow us to investigate methods for improving the scalability of scientific discovery systems. In particular, SP-SUBDUE shows promise as a method of scaling the substructure discovery algorithm to large databases without the need for special-purpose hardware. The distributed implementation is important for our current applications in the biochemical, pharmaceutical, and geological domains where the databases are too large to fit on a single machine. Future work will focus on means of recovering information lost during graph partitioning.

The current version of SUBDUE, available for single processor or multi-processor environments, is available from the SUBDUE home page listed at the beginning of this paper. Improvements are being made to SUBDUE to add concept learning capabilities and apply the system to a greater variety of databases.

References

[1] L. Asker and R. Maclin. Ensembles as a sequence of classifiers. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[2] P. Chan and S. Stolfo. Toward parallel and distributed learning by meta-learning. In *Working notes of the AAAAI-93 workshop on Knowledge Discovery in Databases*, pages 227–240, 1993.

[3] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U. M.

Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 6, pages 153–180. MIT Press, 1996.

[4] S. Clearwater and F. Provost. A tool for knowledge-based induction. In *Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence*, pages 24–30, 1990.

[5] D. Conklin and J. Glasgow. Spatial analogy and subsumption. In *Proceedings of the Machine Learning Conference*, pages 111–116, 1992.

[6] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligent Reseach*, 1:231–255, 1994.

[7] D. J. Cook and L. B. Holder. Graph-based data mining. *to appear in IEEE Intelligent Systems*, 1999.

[8] D. J. Cook, L. B. Holder, and S. Djoko. Scalable discovery of informative structural concepts using domain knowledge. *IEEE Expert*, 10(5):59–68, 1996.

[9] D. J. Cook and R. C. Varnell. Maximizing the benefits of parallel search using machine learning. In *Proceedings of the National Conference on Artificial Intelligence*, 1997.

[10] D. J. Cook and R. C. Varnell. Adaptable incremental deepening search. *Journal of Artificial Intelligence Research*, 1998.

[11] S. Djoko. *The Role of Domain Knowledge in Substructure Discovery.* PhD thesis, Department of Computer Science and Engineering, University of Texas at Arlington, Aug. 1995.

[12] S. Djoko, D. J. Cook, and L. B. Holder. Analyzing the benefits of domain knowledge in substructure discovery. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 75–80, 1995.

[13] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 1, pages 1–34. MIT Press, 1996.

[14] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI Press, Menlo Park, CA, 1996.

[15] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[16] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine, A User's guide and Tutorial for Networked Parallel Computing.* MIT Press, Cambridge, MA, 1994.

[17] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[18] V. Kumar and V. N. Rao. Scalable parallel formulations of depth-first search. In Kumar, Kanal, and Gopalakrishan, editors, *Parallel Algorithms for Machine Intelligence and Vision*, pages 1–41. Springer-Verlag, 1990.

[19] R. Levinson. A self-organizing retrieval system for graphs. In *Proceedings of the National Conference on Artificial Intelligence*, pages 203–206, 1984.

[20] J. T. Potts. Master's thesis:seeking parallelism in discovery programs. Technical report, University of Texas at Arlington, 1996.

[21] C. Powley, C. Ferguson, and R. E. Korf. Parallel heuristic search: Two approaches. In Kumar, Kanal, and Gopalakrishan, editors, *Parallel Algorithms for Machine Intelligence and Vision*, pages 42–65. Springer-Verlag, 1990.

[22] F. J. Provost, B. G. Buchanan, S. H. Clearwater, and Y. Lee. Machine learing in the service of exploratory science and engineering: A case study of the rl induction program. Technical Report ISL-93-6, Intillegent Systems Laboratory, University of Pittsburgh, 1993.

[23] F. J. Provost and D. Hennessy. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 74–79, 1996.

[24] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific Publishing Company, 1989.

[25] J. Segen. Graph clustering and model learning by data compression. In *Proceedings of the Machine Learning Conference*, pages 93–101, 1990.

[26] M. J. Shaw and R. Sikora. A distributed problem solving approach to inductive learning. Technical Report CMU-RI-TR-90-26, Robotics Institute, Carnegie Mellon University, 1990.

[27] S. Su, D. J. Cook, and L. B. Holder. Application of knoweldge discovery to molecular biology: Identifying structural regularities in proteins. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 190–201, 1999.

[28] K. Thompson and P. Langley. Concept formation in structured domains. In D. H. Fisher and M. Pazzani, editors, *Concept Formation: Knowledge and Experience in Unsupervised Learning*, chapter 5. Morgan Kaufmann, 1991.

[29] K. Yoshida, H. Motoda, and N. Indurkhya. Unifying learning methods by colored digraphs. In *Proceedings of the Learning and Knowledge Acquisition Workshop at IJCAI-93*, 1993.