Exploiting Parallelism in Knowledge Discovery Systems to Improve Scalability[*]

Gehad Galal, Diane J. Cook and Lawrence B. Holder
Department of Computer Science and Engineering
University of Texas at Arlington
{galal, cook, holder}@cse.uta.edu

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 approaches hold 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 outlines a general approach for scaling KDD systems using parallel and distributed resources and applies the suggested strategies to the SUBDUE *knowledge discovery system.* 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 that scalability of parallel versions of the* SUBDUE *system are performed using CAD circuit databases 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. As an example, our own SUBDUE system was developed to evaluate the effectiveness of the minimum description length (MDL) principle to discover regularities in a variety of scientific domains [6, 5].

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 [9, 13]. However, much of the data being collected is structural in nature, requiring tools for the analysis and discovery of concepts in structural data [8]. 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 KDD systems can be made scalable by making efficient use of parallel and distributed hardware. To accomplish this goal, we introduce a general approach to parallelizing KDD systems and apply the proposed techniques 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 [1, 14, 17] and partitioning the search space among available processors [11]. Data partitioning approaches have also been effective for certain limited approaches to data mining [3] and knowledge discovery on SIMD architectures. Improving the scalability of scientific discovery systems will help break down the barrier excluding these techniques from practical data mining approaches.

[*]This research is supported by NSF grants IRI-9502260 and IRI-9615272.

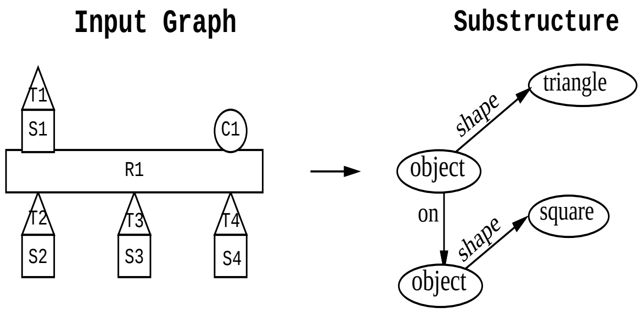


Figure 1: Example substructure in graph form.

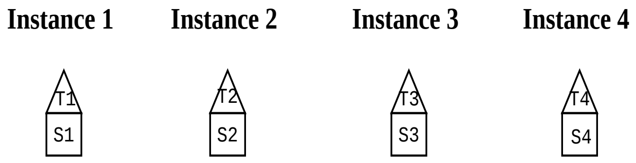


Figure 2: Instances of the substructure.

2 Overview of SUBDUE

We have developed a method for discovering substructures in databases using the minimum description length (MDL) principle introduced by Rissanen [15] and embodied in the SUBDUE system. 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. Figure 1 shows a geometric example of such an input graph. The objects in the figure (e.g., `T1`, `S1`, tt R1) become labeled vertices in the graph, and the relationships (e.g., `on(T1,S1)`, `shape(C1,circle)`) become labeled edges in the graph. The graphical representation of the substructure discovered by SUBDUE from this data is also shown in Figure 1.

An *instance* of a substructure in an input graph is a set of vertices and edges from the input graph that match, graph theoretically, to the graphical representation of the substructure. For example, the instances of the substructure in Figure 1 are shown in Figure 2.

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 finds substructure S_2, which when used to compress the database further allows SUBDUE to find 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. The algorithm begins with the substructure matching a single vertex in the graph. Each iteration, the algorithm selects the best substructure and incrementally expands the instances of the substructure. The new unique substructures become candidates for further expansion. The algorithm searches for the best substructure until all possible substructures have been considered or the total amount of computation exceeds a given limit. Evaluation of each

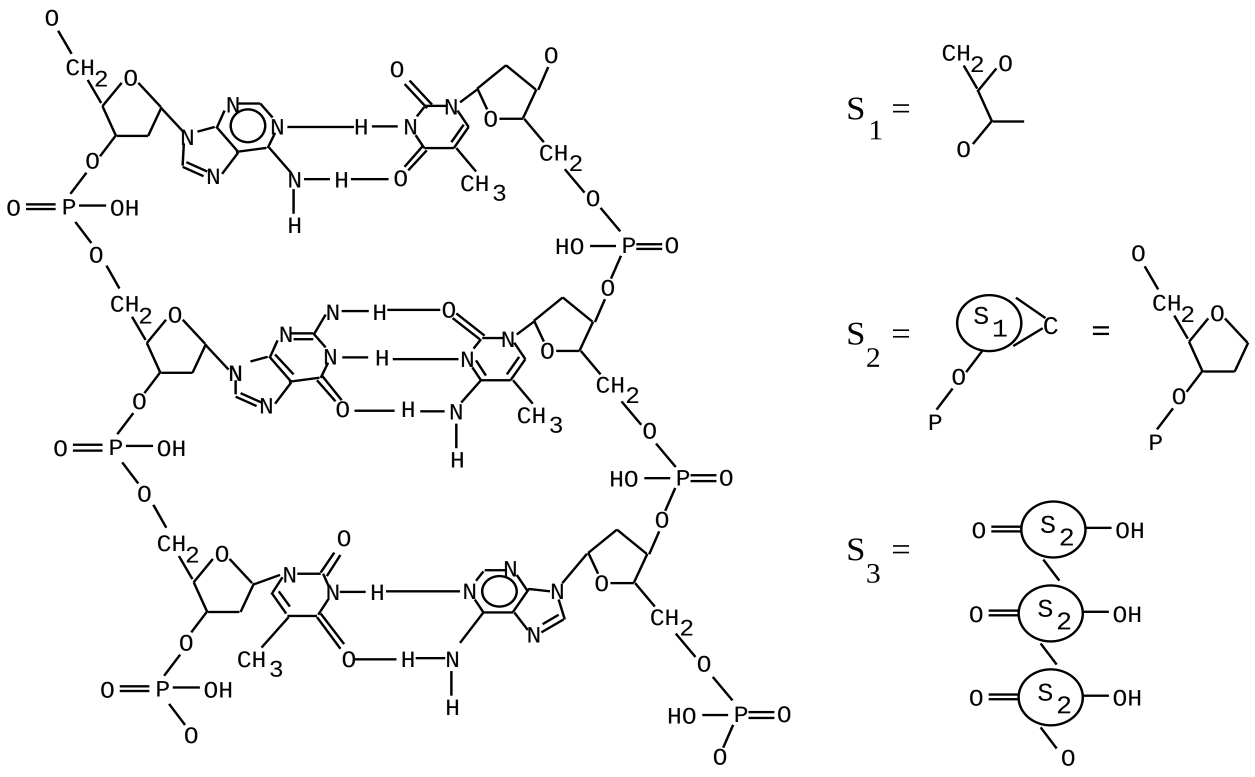


Figure 3: Sample results of Subdue on a protein sequence.

substructure is determined by how well the substructure compresses the description length of the database.

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.

SUBDUE discovers substructures that compress the amount of information necessary to conceptually describe the database. The discovered substructures may not convey important information to a user. To allow SUBDUE to discover substructures of particular interest to a scientist in a given domain, the user can direct the search with expert-supplied background knowledge. 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 [4, 12, 16, 18], SUBDUE is effective at finding interesting and repetitive substructures in any structural database with or without domain-specific guidance.

The results of this scalability study 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 artificial 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 two domains, we generate multiple copies of the CAD and ART graphs and merge the copies together by artificially 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.

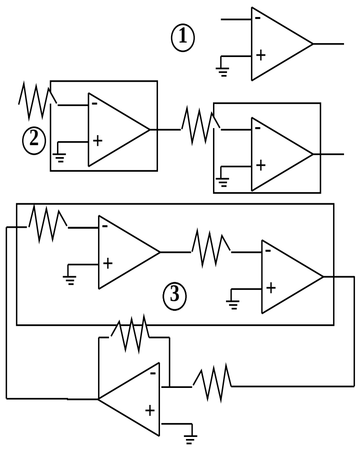
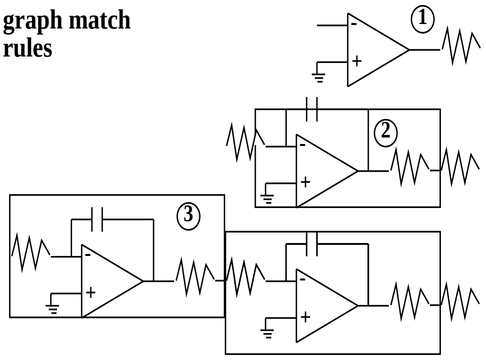
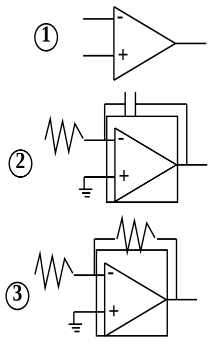
Usage of domain knowledge	Discovered Substructures	Compression	Nodes Expanded	Human Rating [std dev]	Instances
no domain knowledge		0.63	571,370	4.2 [1.2]	9
		0.68	46,422	2.7 [1.2]	3
		0.72	59.886	2.7 [1.0]	2
graph match rules		0.43	145,175	2.7 [1.7]	6
		0.51	39.881	2.7 [1.0]	3
		0.62	425,772	1.5 [0.8]	2
model knowledge and graph match rules		0.53	24,273	4.3 [1.2]	9
		0.66	12,960	4.5 [0.8]	4
		0.72	7,432	4.5 [0.8]	2

Figure 4: CAD circuit – discovered substructures.

Figure 4 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 are rated by a group of 8 domain experts on a scale of 1 to 5, where 1 means not useful in the domain and 5 means very useful.

Figures 3 and 4 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, and artificially-generated databases. Evaluation of these applications is described elsewhere [5, 7].

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, with a closer look, a sequentially oriented solution. In many cases KDD algorithms can perform better if they are provided with enough history of the problem being solved, thus they will perform better in a sequential approach.

In addition, knowledge discovery systems share the 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 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—SQ-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 non-zero compression value 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 kept 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 seen so far, then this substructure is discarded.
- 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 SQ-Subdue any substructure waiting for expansion can be expanded regardless of whether it is the best substructure

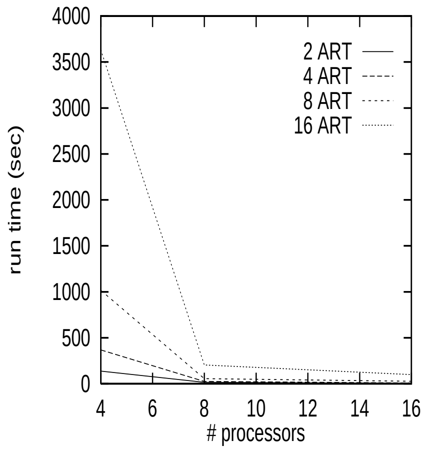


Figure 5: Discovery time of 60 substructures in 2, 4, 8, and 16 ART

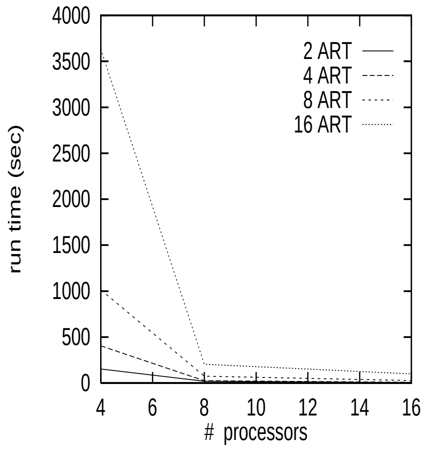


Figure 6: Discovery time of 70 substructures in 2, 4, 8, and 16 ART

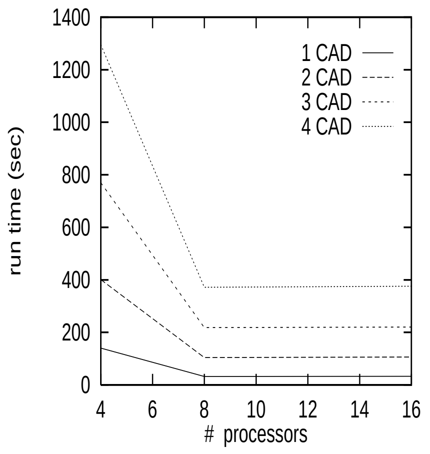


Figure 7: Discovery time of 40 substructures in 1, 2, 3, and 4 CAD

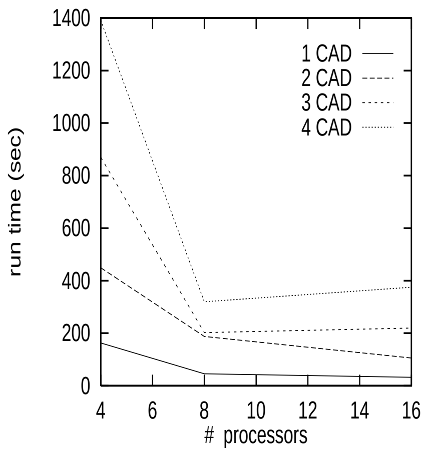


Figure 8: Discovery time of 60 substructures in 1, 2, 3, and 4 CAD

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 SQ-SUBDUE we test the algorithm on an nCUBE 2 with 2, 4, 8, and 16 processors. Figures 5 through 8 show the resulting decrease in runtime as the number of processors increases. A limit on performance 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 SQ-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 that sequential version.

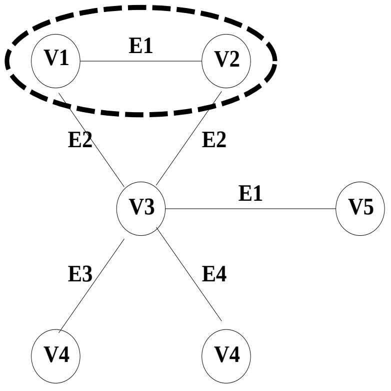


Figure 9: Sample input graph.

3.2 Dynamic Partitioning Approach—DP-Subdue

In the first data partitioning 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 9. 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 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.

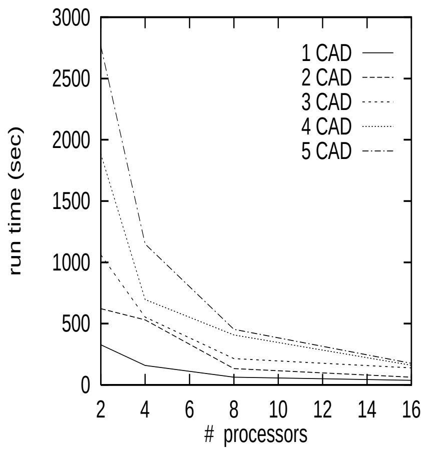


Figure 10: CAD database evaluation time

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 data partitioning 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

Although DP-Subdue approach was not successful, the data partitioning idea itself is very appealing in terms of both memory usage and speedup. Here we illustrate the static partitioning approach to parallelize Subdue.

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

The graph partitioning step is the most important step of the algorithm. The speedup achieved as well as the quality of discovered substructures depends on this step. In 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 [10]. 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. Edge weights are assigned so that the higher the frequency of occurrence of an edge's label, the higher the weight assigned to it. The idea behind this weight assignment is that the frequently occurring edge labels are more likely to represent useful knowledge. The run time of Metis to partition CAD database is very small (ten seconds on average) and is thus not included in the parallel run time.

Figures 10, 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 always 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 always results in a better speedup. On the other hand, information on the borders of partitions is lost which results in poorer compression as the number of partitions increases. This problem can be partially alleviated by allowing the partitions to slightly overlap.

Now we turn our attention to the quality of the substructures discovered by SP-Subdue. Tables 1 and 2 show

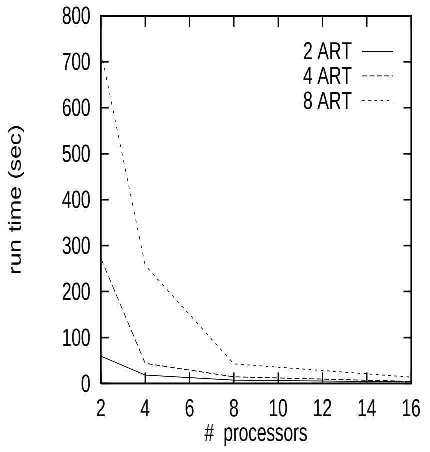


Figure 11: 2, 4, and 8 ART database evaluation time

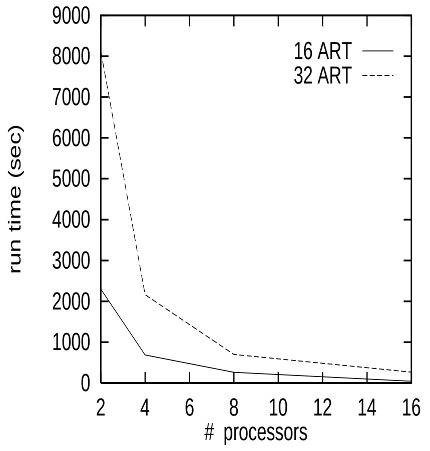


Figure 12: 16 and 32 ART database evaluation time

Table 1: CAD database compression results

Database	1	2	4	8	16
1 CAD	.0746	.1561	.1125	.0287	.0190
2 CAD	.0746	.0823	.1233	.0343	.0171
3 CAD	.0746	.0600	.0501	.0334	.0336
4 CAD	.0746	.1020	.0424	.0490	.0268
5 CAD	.0746	.0792	.0654	.0321	.0179

Table 2: ART database compression results

Database	1	2	4	8	16
2 ART	.5639	.5436	.5453	.5642	.6268
4 ART	.5415	.5200	.3219	.2963	.2252
8 ART	.5418	.5186	.5013	.3101	.2572
16 ART	.5407	.5150	.5050	.4975	.3033
32 ART	.5187	.5072	.4952	.4927	.4903

the compression achieved for the CAD and ART databases when processed by a different number of processors. Regarding the ART database results, it is clear that the best compression achieved is the sequential version compression. This is expected because of the high regularity of the database. As mentioned before the ART database has 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 can not get better compression by partitioning. The compression achieved with a small number of partitions is very close to that of the sequential version.

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 that 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 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.

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. We implemented SP-Subdue on a network of 14 Pentium PCs using PVM. The speedup results are given in Tables 3 and 4.

Table 3: CAD databases distributed version speedups

Database	2	4	8	14
1 CAD	1.48	3.37	7.99	18.73
2 CAD	1.88	3.52	13.64	28.69
3 CAD	1.98	5.54	14.76	33.79
4 CAD	2.58	8.81	15.81	47.18
5 CAD	2.13	6.25	23.27	52.58

Table 4: ART databases distributed version speedups

Database	2	4	8	14
2 ART	4.34	12.76	31.42	63.8
4 ART	2.08	15.69	56.56	139.33
8 ART	2.57	6.09	48.45	162.38
16 ART	2.87	9.96	20.53	171.09
32 ART	3.16	10.31	34.12	82.99

3.4 Comparison

We have described implementations of one functional parallel and two data parallel approaches to 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. SQ-SUBDUE can prove effective in discovering substructure in very 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 SQ-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 SUBDUE$_n$ and SP-SUBDUE).

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 provided with the same amount of resources (processing power and memory) that would have been required by the sequential version to handle the same database while discovering substructures of equal or better quality. One of our databases 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 porting the system to distributed systems. 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 to be 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 wide-spread 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. We are currently exploring methods for improving the scalability of additional discovery methods. In particular, we have developed SIMD and MIMD implementations of the AutoClass discovery system [2], and will research methods of combining the parallel linear-based AutoClass discovery system with the structural approach of parallel SUBDUE. A hybrid linear / structural system will be implemented and used to discover concepts in a variety of earth and space science databases.

References

[1] 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.

[2] 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.

[3] 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.

[4] D. Conklin, S. Fortier, J. Glasgow, and F. Allen. Discovery of spatial concepts in crystallographic databases. In *Proceedings of the ML92 Workshop on Machine Discovery*, pages 111–116, 1992.

[5] 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.

[6] D. J. Cook, L. B. Holder, and S. Djoko. Scalable discovery of informative structural concepts using domain knowledge. *IEEE Expert*, 11(5), 1996.

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

[8] 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.

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

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

[11] 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.

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

[13] Gregory Piatetsky-Shapiro. *Knowledge Discovery in Databases.* AAAI Press, 1991.

[14] 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.

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

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

[17] 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.

[18] 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 Publishers, 1991.