Discovering Concepts in Structural Data

Diane J. Cook, Lawrence B. Holder, and Gehad Galal

Department of Computer Science Engineering

University of Texas at Arlington

{cook, holder, galal}@cse.uta.edu

Abstract

The explosive growth of databases in scientific, industrial, and commercial fields has not been accompanied by a similar growth in our ability to analyze and digest this data. The increasing amount and complexity of data creates an urgent need for automatic database analysis tools. This trend is evident in molecular biology data which continues to grow in both size and complexity.

This research outlines a general approach to automatically discover repetitive and functional concepts in large structural databases. The SUBDUE system discovers substructures that compress the database and represent structural concepts in the data. By replacing previously-discovered substructures in the data, multiple passes of SUBDUE produce a hierarchical description of the structural regularities in the data. To increase the flexibility of the system, we describe methods of incorporating domain-dependent information into the discovery process. Because discovery systems such as SUBDUE are very computationally expensive, we also explore ways of parallelizing the system to improve scalability.

Introduction

The large amount of data collected today is quickly overwhelming researchers' abilities to interpret the data and discover interesting patterns within the data. In response to this problem, a number of researchers have developed techniques for discovering concepts in databases. These techniques work well for data expressed in a non-structural, attribute-value representation, and address issues of data relevance, missing data, noise and uncertainty, and utilization of domain knowledge [Cheesman and Stutz, 1996; Fisher, 1987]. However, recent data acquisition projects are collecting structural data describing the relationships among the data objects. Correspondingly, there exists a need for techniques to analyze and discover concepts in structural databases [Fayyad *et al.*, 1996].

One method for discovering knowledge in structural data is the identification of common substructures within the data. The motivation for this process is to find substructures capable of compressing the data and to identify conceptually interesting substructures that enhance the interpretation of the data. Substructure discovery is the process of identifying concepts describing interesting and repetitive substructures within structural data. Once discovered, the substructure concept can be used to simplify the data by replacing instances of the substructure with a pointer to the newly discovered concept. The discovered substructure concepts allow abstraction over detailed structure in the original data and provide new, relevant attributes for interpreting the 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 goals of the data analysis.

We describe a system called SUBDUE that discovers interesting substructures in structural data based on the minimum description length principle. The SUBDUE system discovers substructures that compress the original data and represent structural concepts in the data. By replacing previously-discovered substructures in the data, multiple passes of SUBDUE produce a hierarchical description of the structural regularities in the data. SUBDUE uses a computationally-bounded inexact graph match that identifies similar, but not identical, instances of a substructure and finds an approximate measure of closeness of two substructures when under computational constraints. In addition to the minimum description length principle, other background knowledge can be used by SUBDUE to guide the search towards more appropriate substructures.

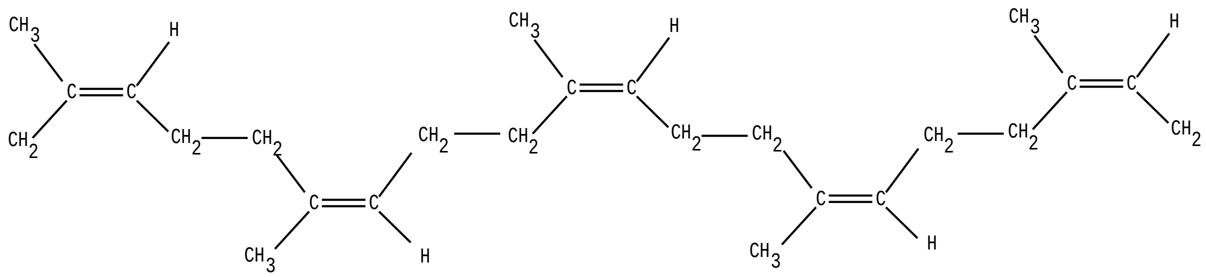

Figure 1: Natural rubber atomic structure

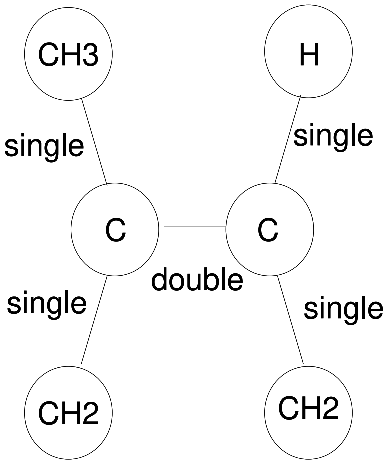

Figure 2: Discovered substructure

Algorithmic Techniques

The substructure discovery system represents structured 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 show a sample graphical representation of an input file to SUBDUE which represents the atomic structure of natural rubber. Given this input database, SUBDUE discovers the subgraph shown in Figure 2 as the best subgraph to describe the input database.

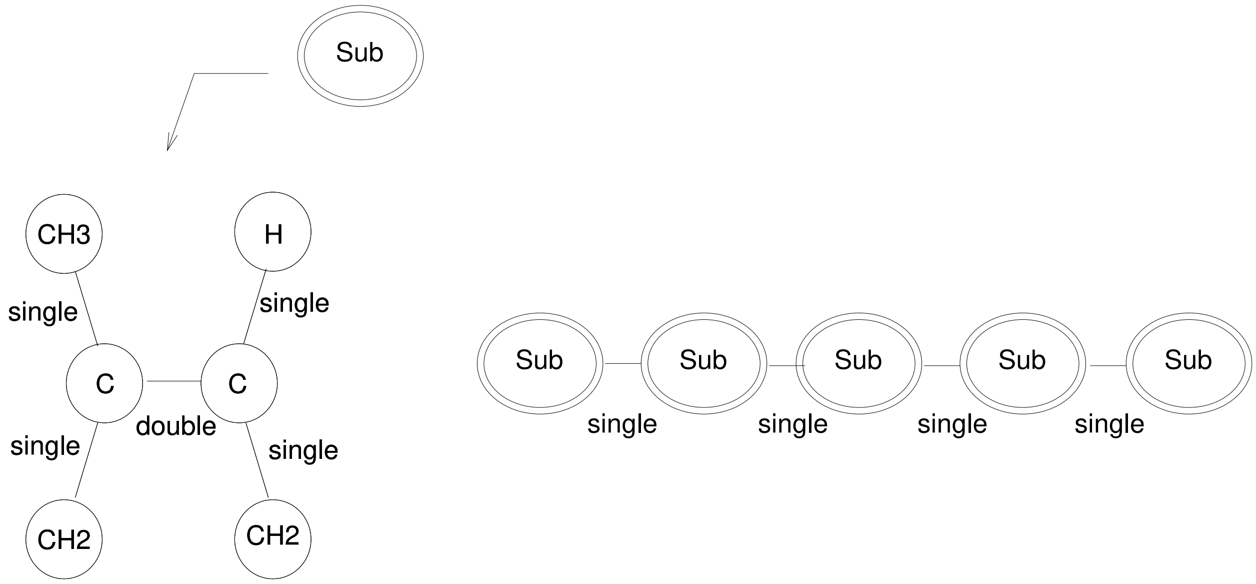

Figure 3: Compressed representation of the graph

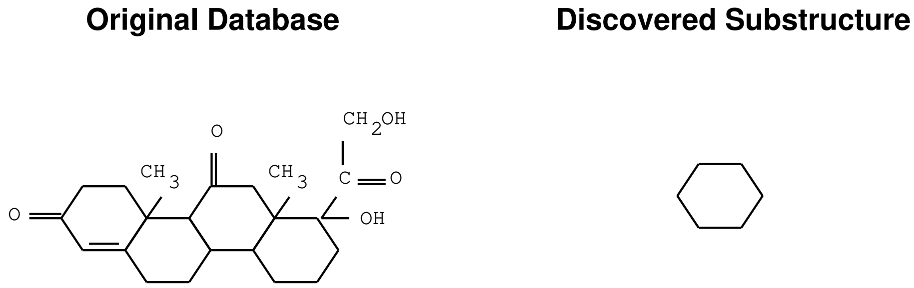

Figure 4: Cortisone atomic structure

An **instance** of a substructure in an input graph is a set of vertices and edges from the input graph that match, isomorphically, to the graphical representation of the substructure. The input graph contains five instances of this concept in the database. As shown in Figure 3, SUBDUE replaces the instances with a pointer to the concept definition and thus redefines the database in terms of this substructure. Figure 4 shows the substructure discovered by SUBDUE in another chemical compound, cortisone.

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

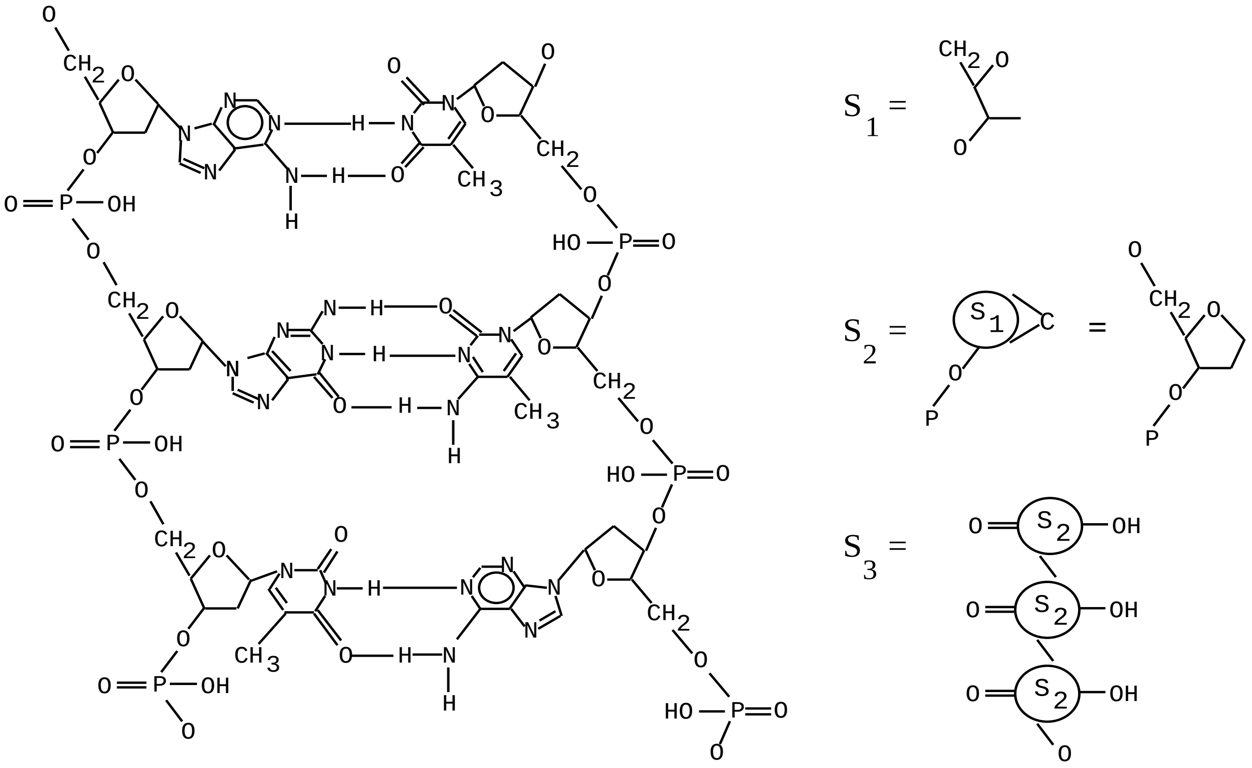

Figure 5: Sample results of Subdue on a dna sequence

iteration through the algorithm selects the best substructure and expands the instances of the substructure by one neighboring edge in all possible ways. The new unique generated substructures (duplicate substructures are removed from the list) 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. The evaluation of each substructure is guided by the MDL principle and other background knowledge provided by the user.

Typically, once the description length afforded by an expanding substructure begins to increase, further expansion of the substructure will not yield a smaller description length. As a result, SUBDUE makes use of an optional pruning mechanism that eliminates substructure expansions from consideration when the description lengths for these expansions increases.

Figure 5 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.

Substructure discovery using the MDL principle

The **minimum description length (MDL)** principle introduced by Rissanen [Rissanen, 1989] states that the best theory to describe a set of data is a theory which minimizes the description length of the entire data set. The MDL principle has been used for decision tree induction [Quinlan and Rivest, 1989], image processing [Pednault, 1989; Pentland, 1989; Leclerc, 1989], concept learning from relational data [Derthick, 1991], and learning models of non-homogeneous engineering domains [Rao and Lu, 1992].

We demonstrate how the minimum description length principle can be used to discover substructures in complex data. In particular, a substructure is evaluated based on how well it can compress the entire data set. We define the minimum description length of a graph to be the minimum number of bits necessary to completely describe the graph. SUBDUE searches for a substructure that minimizes $I(S) + I(G|S)$, where S is the discovered substructure, G is the input graph, $I(S)$ is the number of bits (description length) required to encode the discovered substructure, and $I(G|S)$ is the number of bits required to encode the input graph G with respect to S.

The graph connectivity can be represented by an adjacency matrix. Assume that the decoder has a table of the unique labels in the original graph G, the encoding of the graph consists of the following steps.

1. Determine the number of bits $vbits$ needed to encode the vertex labels of the graph.

2. Determine the number of bits $rbits$ needed to encode the rows of the adjacency matrix A.

3. Determine the number of bits $ebits$ needed to encode the edges represented by the entries $A[i, j] = 1$ of the adjacency matrix A.

The total encoding of the graph takes $(vbits + rbits + ebits)$ bits.

Inexact Graph Match

Although exact structure match can be used to find many interesting substructures, many of the most interesting substructures show up in a slightly different form throughout the data. These differences may be due to noise and distortion, or may just illustrate slight differences between instances of the same general class of structures.

Given an input graph and a set of defined substructures, we want to find those subgraphs of the input graph that most closely resemble the given substructures. Furthermore, we want to associate a distance measure between a pair of graphs consisting of a given substructure and a subgraph of the input graph. We adopt the approach to inexact graph match given by Bunke and Allermann [Bunke and Allermann, 1983].

In this inexact match approach, each distortion of a graph is assigned a cost. A distortion is described in terms of basic transformations such as deletion, insertion, and substitution of vertices and edges. The distortion costs can be determined by the user to bias the match for or against particular types of distortions.

An inexact graph match between two graphs g_1 and g_2 maps g_1 to g_2 such that g_2 is interpreted as a distorted version of g_1. Formally, an inexact graph match is a mapping $f : N_1 \rightarrow N_2 \cup \{\lambda\}$, where N_1 and N_2 are the sets of vertices of g_1 and g_2, respectively. A vertex $v \in N_1$ that is mapped

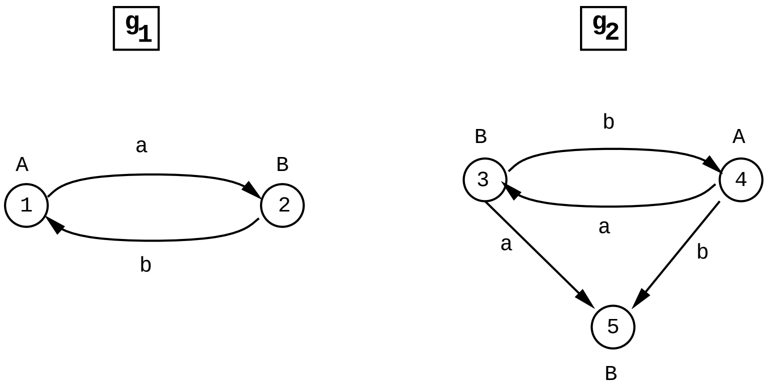

Figure 6: Two similar graphs g_1 and g_2.

to λ (i.e., $f(v) = \lambda$) is deleted. That is, it has no corresponding vertex in g_2.

Given a set of particular distortion costs as discussed above, we define the cost of an inexact graph match $cost(f)$, as the sum of the cost of the individual transformations resulting from f, and we define $matchcost(g_1, g_2)$ as the value of the least-cost function that maps graph g_1 onto graph g_2. If g_2 is isomorphic to g_1, the cost of the match will be 0. The cost of this match increases as the number of dissimilarities between the graphs increases.

Given g_1, g_2, and a set of distortion costs, the actual computation of $matchcost(g_1, g_2)$ can be determined using a tree search procedure. A state in the search tree corresponds to a partial match that maps a subset of the vertices of g_1 to a subset of the vertices in g_2. Initially, we start with an empty mapping at the root of the search tree. Expanding a state corresponds to adding a pair of vertices, one from g_1 and one from g_2, to the partial mapping constructed so far. A final state in the search tree is a match that maps all vertices of g_1 to g_2 or to λ such that for any edge between vertices in g_1, there is an edge between the corresponding vertices in g_2. The complete search tree of the example in Figure 6 is shown in Figure 7. For this example we assign a value of 1 to each distortion cost. The numbers in circles in this figure represent the cost of a state. As we are eventually interested in the mapping with minimum cost, each state in the search tree gets

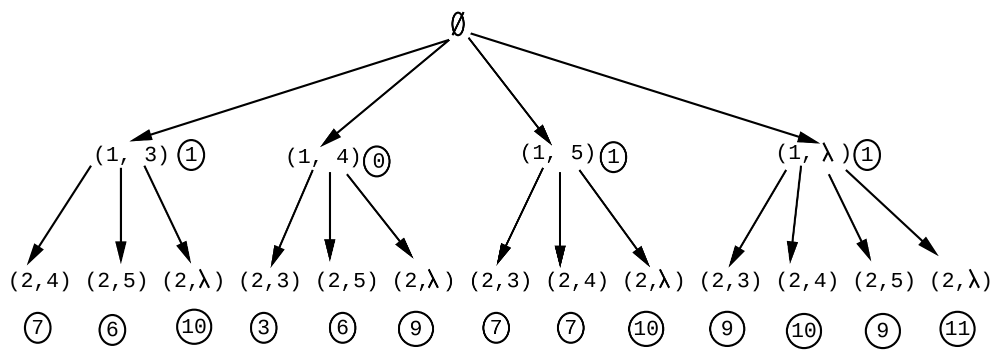

Figure 7: Search tree for computing matchcost(g_1,g_2) from Figure 4.

assigned the cost of the partial mapping that it represents. Thus the goal state to be found by our tree search procedure is the final state with minimum cost among all final states. From Figure 7 we conclude that the minimum cost inexact graph match of g_1 and g_2 is given by the mapping $f(1) = 4$, $f(2) = 3$. The cost of this mapping is 4.

Given graphs g_1 with n vertices and g_2 with m vertices, $m \geq n$, the complexity of the full inexact graph match is $O(n^{m+1})$. Because this routine is used heavily throughout the discovery and evaluation process, the complexity of the algorithm can significantly degrade the performance of the system.

To improve the performance of the inexact graph match algorithm, we extend Bunke's approach by applying a branch-and-bound search to the tree. The cost from the root of the tree to a given node is computed as described above. Nodes are considered for pairings in order from the most heavily connected vertex to the least connected, as this constrains the remaining match. Because branch-and-bound search guarantees an optimal solution, the search ends as soon as the first complete mapping is found.

In addition, the user can place a limit on the number of search nodes considered by the branch-and-bound procedure (defined as a function of the size of the input graphs). Once the number of nodes expanded in the search tree reaches the defined limit, the search resorts to hill climbing using the cost of the mapping so far as the measure for choosing the best node at a given level. By defining such a limit, significant speedup can be realized at the expense of accuracy for the computed match cost.

Adding domain knowledge to the SUBDUE system

The SUBDUE discovery system was initially developed using only domain independent heuristics to evaluate potential substructures. As a result, some of the discovered substructures may not be useful and relevant to specific domains of interest. For instance, in a programming domain, the BEGIN and END statements may appear repetitively within a program; however, they do not perform any meaningful function on their own; hence they exhibit limited usefulness. Similarly, in a biochemical domain, some compounds or substructures may appear repetitively within the data; however, they may not represent meaningful or functional units for this domain. To make SUBDUE's discovered substructures more interesting and useful across a wide variety of domains, domain knowledge is added to guide the discovery process. Furthermore, compressing the graph using the domain knowledge can increase the chance of realizing greater compression than without using the domain knowledge.

In this section we present two types of domain knowledge that are used in the discovery process and explain how they bias discovery toward certain types of substructures.

Model knowledge

Model knowledge provides the discovery system with specific types of structures that are likely to exist in a database and that are of particular interest to a scientist using the system. The model knowledge is organized in a hierarchy of primitive and combinations of primitive well-recognized structures. This hierarchy is supplied by a domain expert.

Although the minimum description length principle still drives the discovery process, domain knowledge is used to input a bias toward certain types of substructures. First, the modified version of SUBDUE can be biased to look specifically for structures of the type specified in the model hierarchy. The discovery process begins with matching a single vertex in the input graph to primitive nodes of the model knowledge hierarchy. If the primitive nodes do not match the input vertices, the higher level nodes of the hierarchy are pursued. The models in the hierarchy pointed to by the matched model nodes in the input graph are selected as candidate models to be matched with the input substructure. Each iteration through the process, SUBDUE selects a substructure from the input graph which has the best match to one of the selected models and can be used to compress the input graph.

The match can either be a subgraph match or a whole graph match. If the match is a subgraph match, SUBDUE expands the instances of the best substructure by one neighboring edge in all possible ways. The newly generated substructure becomes a candidate for the next iteration. However, if the match is a whole graph match, the process has found the desired substructure, and the chosen substructure is used to compress the entire input graph. The process continues until either a substructure has been found or all possible substructures have been considered.

To represent an input graph using a discovered substructure from the model hierarchy, the representation involves additional overhead to replace the substructure's instances with a pointer to the model hierarchy. In some cases, a model definition includes parameters which must also be represented. After a substructure is discovered, each instance of the discovered substructure in the input graph is replaced by a pointer to a concept in the model hierarchy representing the substructure.

Graph match rules

At the heart of the SUBDUE system lies an inexact graph match algorithm that finds instances of a substructure definition. Since many of substructure instances can show up in a slightly different form throughout the data, and each of these differences is described in terms of basic transformations performed by the graph match, we can use graph match rules to assign each transformation a cost based on the domain of usage. This type of domain-specific information is represented using if-then rules such as the following:

IF (domain = BIOCHEMICAL) and ((Vertex1 = HO and Vertex2 = OH) or
(Vertex1 = OH and Vertex2 = HO))
THEN (graph match cost = z)

The graph match rules allow a specification of the amount of acceptable generality between a substructure definition and its instances, or between a model definition and its instances in the input graph. Given $g1$, $g2$, and a set of distortion costs, the actual calculation of similarity can be performed using the search procedure described earlier. As long as the similarity is within the user-defined threshold, the two graphs $g1$ and $g2$ are considered to be isomorphic.

We have tested the performance of SUBDUE with model knowledge, with graph match rules, and with no background knowledge. Performance is measured in terms of the amount of compression afforded by the highest-valued discovered substructure and by amount of substructure functionality. Functionality measures were provided by a team of eight scientists familiar with the application domain – functionality is measured on a scale of 1 to 5. When both types of background knowledge are used, the highest functionality ratings and compression result. The amount of compression resulting from the use of graph match rules alone is lower than using no background knowledge, and both of these cases result in lower human ratings than when both types of background knowledge are used. A more detailed description of these experiments can be found in the literature [Djoko *et al.*, 1996].

Scalability

A goal of knowledge discovery in database (KDD) systems is to discover knowledge in large databases that cannot be effectively processed by humans. For this reason KDD systems are required to handle very large databases. Unfortunately, most KDD systems are computationally expensive, and in most cases the resource requirements of a KDD system grow as the database becomes larger. For these reasons, researchers' attentions are shifting from developing new KDD systems to improving the scalability of existing KDD systems.

A goal of our research is to demonstrate that KDD systems in general, and SUBDUE in particular, can be made scalable by making efficient use of parallel and distributed hardware. One way to improve the scalability of a system is to make use of system resources to allow larger inputs and/or reduce run time. Clearly increasing the resources of a single machine will not benefit us in the limit as the input is usually large enough to exhaust the resources of a single machine. Thus, we have to make use of parallel algorithms. Porting a complicated system to a parallel environment involves consideration of the architecture as well as the nature of the problem itself.

A variety of approaches can be taken to dividing the computational effort among individual processors. However, in many of these approaches the entire database must be stored on each contributing machine. Because we want SUBDUE to provide scalability in storage as well as computation, we introduce a data partitioning approach to SUBDUE, called Static-Partitioning Subdue (SP-SUBDUE).

Using SP-SUBDUE we partition the input graph into n partitions for n processors. Each processor executs the 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 [Karypis and Kumar, 1995]. 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 our test databases is very small (ten seconds on average) and is thus not included in the parallel run time.

Figures 8, 9, and 10 graph the run time of SP-SUBDUE as the number of processors increases on a sample graphs of two types. The first type of database is a CAD circuit description of an A-to-D converter provided by National Semiconductor. The initial graph contains 8,441 vertices and 19,206 edges. In addition, we also test an artificial graph in which an arbitrary number of instances of a predefined substructure are imbedded in the database surrounded by vertices and edges with random labels and connectivity. The tested artificial graph contains 1,000 vertices and 2,500 edges. To create larger databases with the same characteristics as the sample databases, we

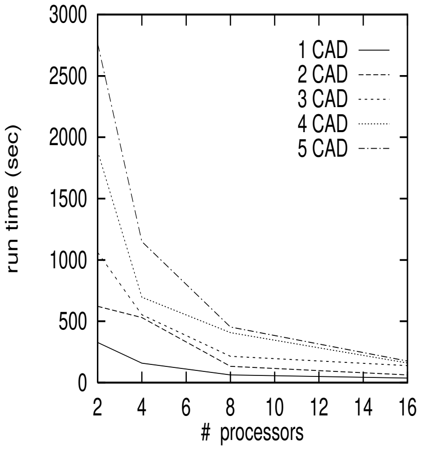

Figure 8: CAD database evaluation time

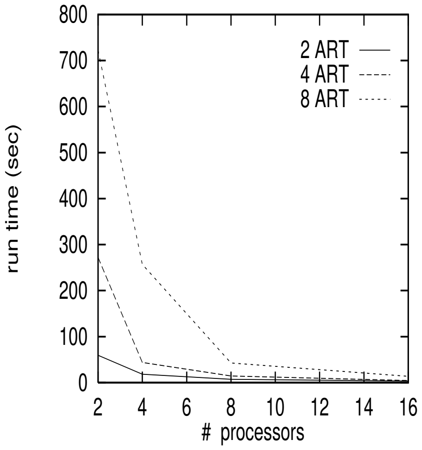

Figure 9: ART database evaluation time

generate multiple copies of these 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 database consisting of n copies of the original CAD database with joining edges added, and "n ART" refers to a database consisting of n copies of the artificial database with additional joining edges.

The speedup achieved with the ART database is always superlinear. This is because the run time of sequential SUBDUE is greater than linear 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.

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

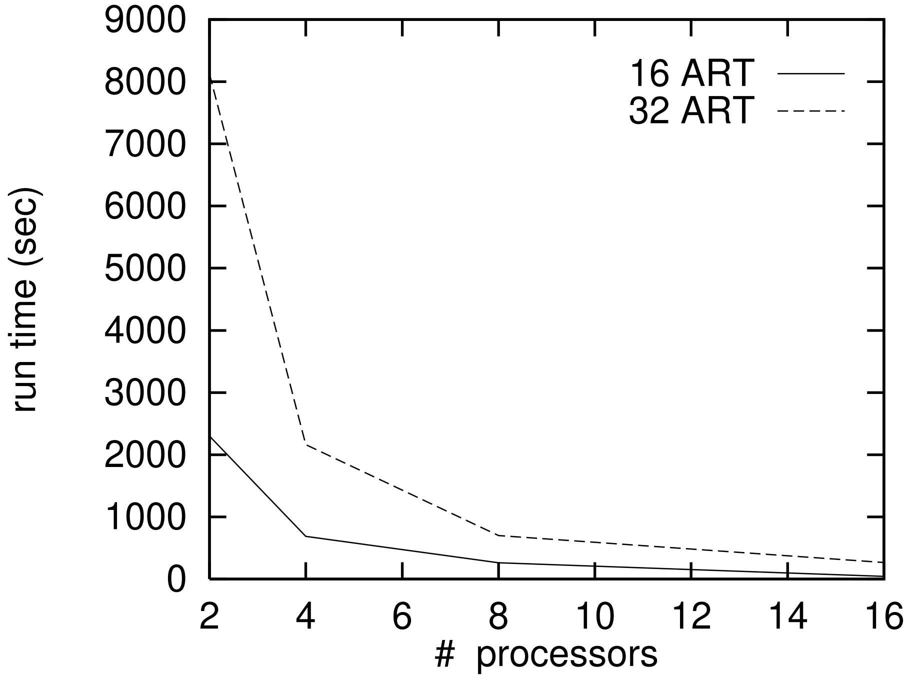

Figure 10: 16 and 32 ART database evaluation time

the edge cuts. As a result, we will generally not get better compression by partitioning, though some compression results improve due to the increased beam width provided by multiple processors. 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 can be found in the nature of the CAD database. As with many real-world databases, the CAD database 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.

Database	1	2	4	8	16
1 CAD	.0746	0.156095	0.112458	0.028686	0.019010
2 CAD	.0746	0.082348	0.123302	0.034331	0.017073
3 CAD	.0746	0.060023	0.050129	0.033402	0.033586
4 CAD	.0746	0.101977	0.042417	0.048962	0.026821
5 CAD	.0746	0.079247	0.065364	0.032131	0.017944

Table 1: CAD database compression results by number of processors

Database	1	2	4	8	16
2 ART	.5639	0.543554	0.545256	0.564162	0.626758
4 ART	.5415	0.519963	0.321946	0.296317	0.225197
8 ART	.5418	0.518611	0.501251	0.310067	0.257190
16 ART	.5407	0.514947	0.504967	0.497538	0.303339
32 ART	.5187	0.507182	0.495180	0.492651	0.490307

Table 2: ART database compression results by number of processors

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 3: CAD 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

Table 4: ART databases distributed version speedups

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 [Geist *et al.*, 1994]. We implemented SP-SUBDUE on a network of 14 Pentium PCs using PVM. The speedup results are given in Tables 3 and 4.

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 makes SP-SUBDUE ideal for distributed environments. Using the portable message passing interface provided by PVM allows the system to run on heterogeneous networks.

Future Work and Generalizations

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

We describe a method for integrating domain independent and domain dependent substructure discovery based on the minimum description length principle. The method is generally applicable to many structural databases, such as chemical compound data, computer-aided design data, computer programs, and protein data. This integration improves SUBDUE's ability to both compress an input graph and discover substructures relevant to the domain of study. In addition, we show that making efficient use of parallel and distributed resources can significantly improve the run-time performance of data-intensive and compute-intensive discovery program such as SUBDUE. We are currently evaluating linear discovery algorithms for inclusion in our system and are adapting our distributed algorithms for application to additional discovery systems.

Defining Terms

graph isomorphism. Graph G_1, defined by vertices V_1 and edges E_1, is isomorphic to graph G_2, defined by vertices V_2 and edges E_2, if there is a one-to-one function mapping vertices of G_1 onto vertices of G_2, $f : V_1 \to V_2$ satisfying the property that $\{u, v\} \in E_1$ if and only if $\{f(u), f(v)\} \in E_2$.

minimum description length principle. The minimum description length principle states that the best theory to describe a set of data is a theory which minimizes the description length of the entire data set.

substructure. A connected graph embedded within another graph.

substructure instance. A subgraph that is isomorphic to the graphical representation of a substructure definition.

References

[Bunke and Allermann, 1983] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.

[Cheesman and Stutz, 1996] P. Cheesman 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.

[Derthick, 1991] M. Derthick. A minimal encoding approach to feature discovery. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 565–571, 1991.

[Djoko *et al.*, 1996] S. Djoko, D. J. Cook, and L. B. Holder. Discovering informative structural concepts using domain knowledge. *IEEE Expert*, 10:59–68, 1996.

[Fayyad *et al.*, 1996] 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.

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

[Geist *et al.*, 1994] 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.

[Karypis and Kumar, 1995] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report TR-95-XXX, Department of Computer Sience, University of Minnesota, 1995.

[Leclerc, 1989] Y. G. Leclerc. Constructing simple stable descriptions for image partitioning. *International Journal of Computer Vision*, 3(1):73–102, 1989.

[Pednault, 1989] E. P. D. Pednault. Some experiments in applying inductive inference principles to surface reconstruction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1603–1609, 1989.

[Pentland, 1989] A. Pentland. Part segmentation for object recognition. *Neural Computation*, 1:82–91, 1989.

[Quinlan and Rivest, 1989] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[Rao and Lu, 1992] R. B. Rao and S. C. Lu. Learning engineering models with the minimum description length principle. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 717–722, 1992.

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

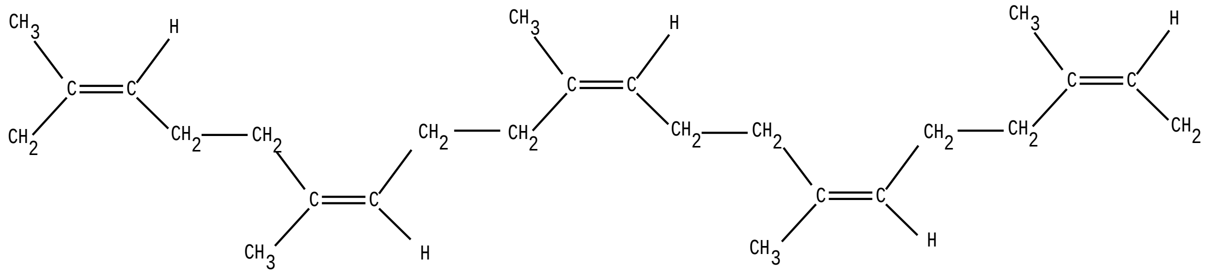

Figure 1: Natural rubber atomic structure

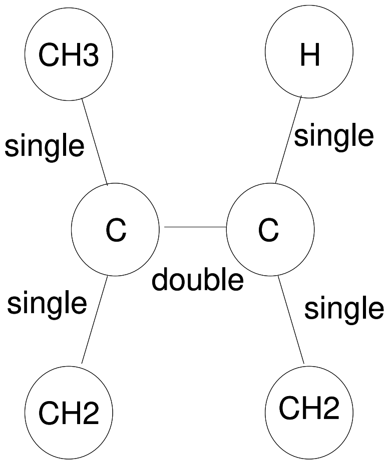

Figure 2: Discovered substructure

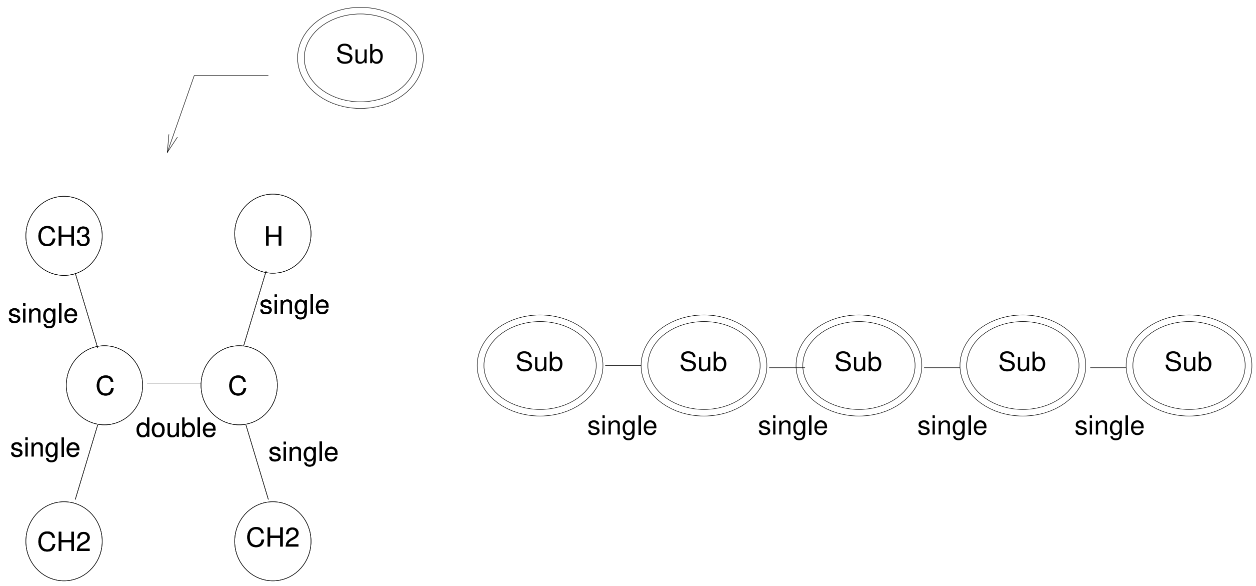

Figure 3: Compressed representation of the graph

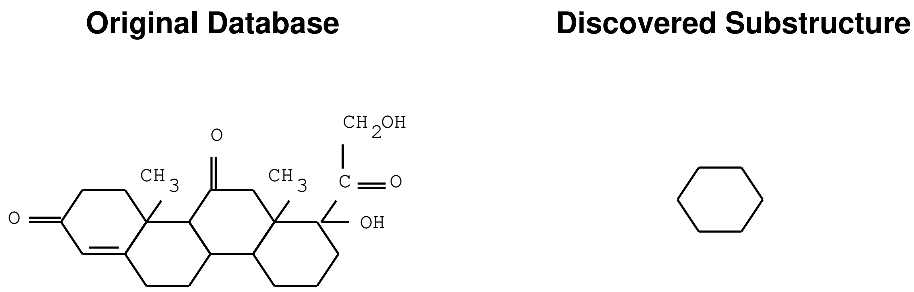

Figure 4: Cortisone atomic structure

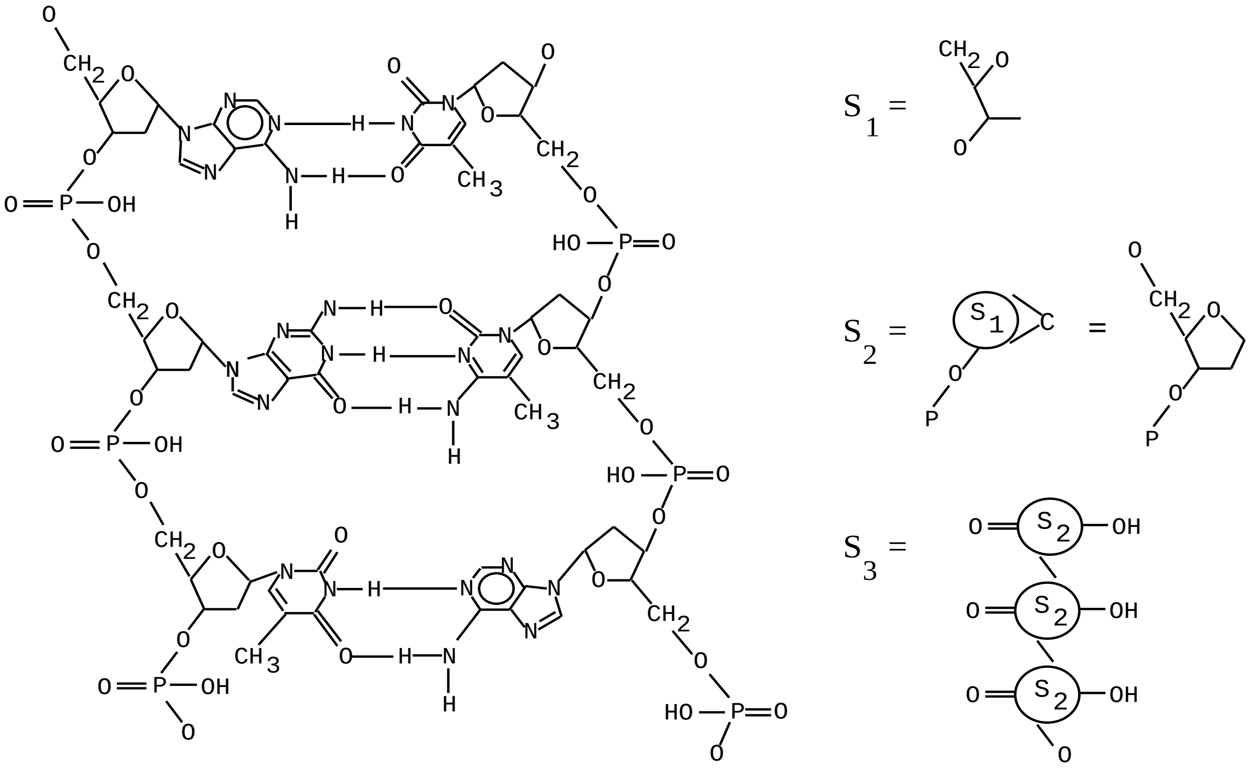

Figure 5: Sample results of Subdue on a dna sequence

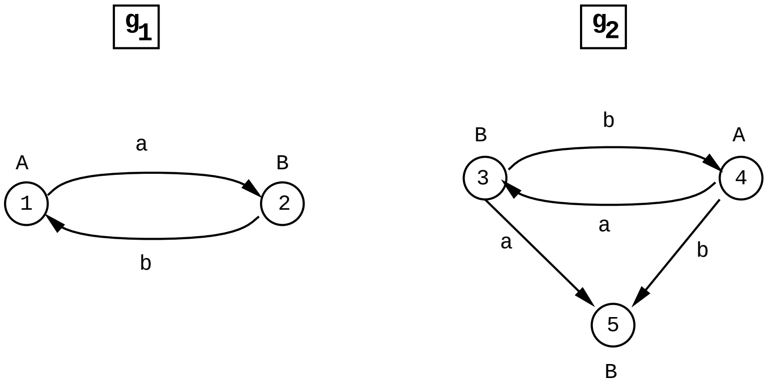

Figure 6: Two similar graphs g_1 and g_2.

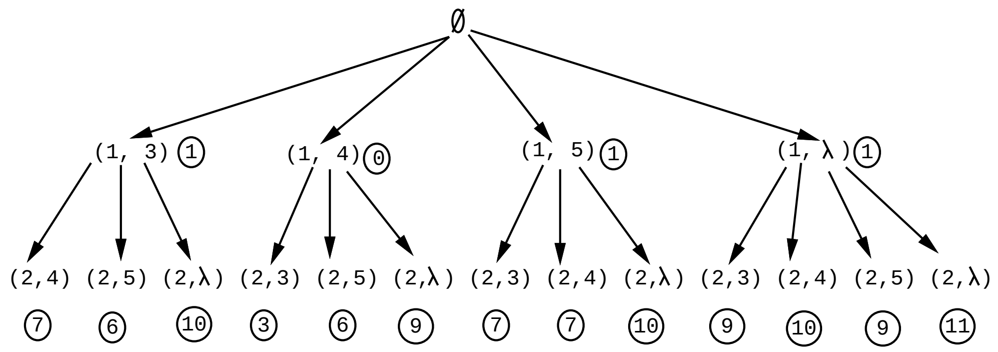

Figure 7: Search tree for computing matchcost(g_1,g_2) from Figure 6.

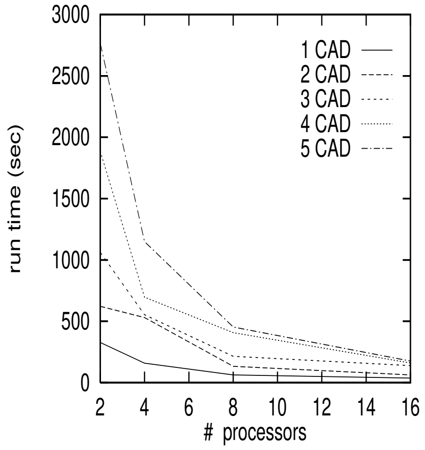

Figure 8: CAD database evaluation time

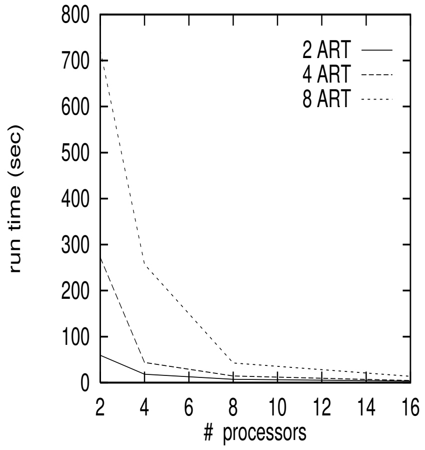

Figure 9: ART database evaluation time

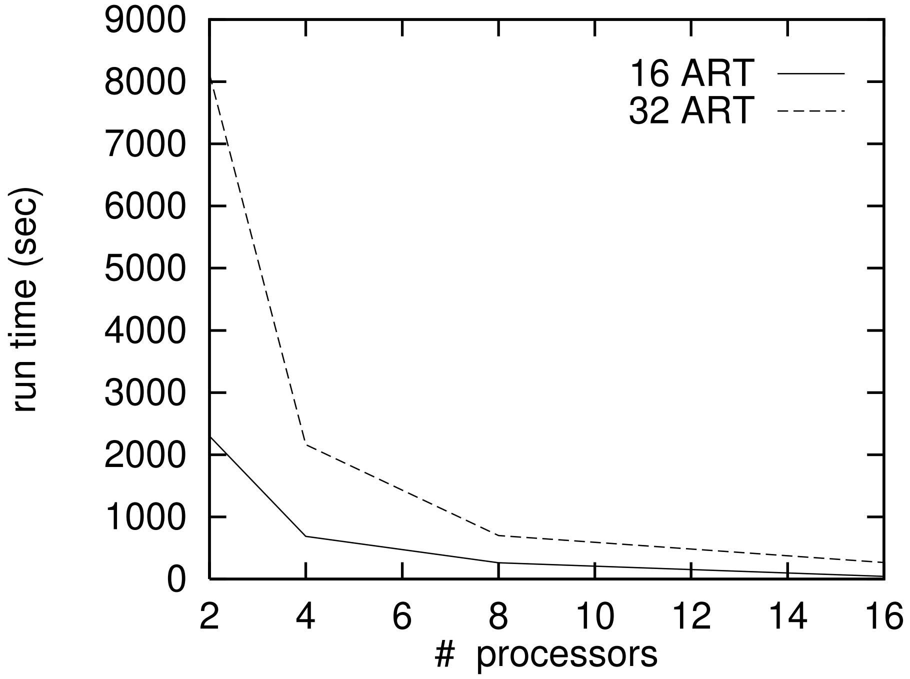

Figure 10: 16 and 32 ART database evaluation time

Database	1	2	4	8	16
1 CAD	.0746	0.156095	0.112458	0.028686	0.019010
2 CAD	.0746	0.082348	0.123302	0.034331	0.017073
3 CAD	.0746	0.060023	0.050129	0.033402	0.033586
4 CAD	.0746	0.101977	0.042417	0.048962	0.026821
5 CAD	.0746	0.079247	0.065364	0.032131	0.017944

Table 1: CAD database compression results by number of processors

Database	1	2	4	8	16
2 ART	.5639	0.543554	0.545256	0.564162	0.626758
4 ART	.5415	0.519963	0.321946	0.296317	0.225197
8 ART	.5418	0.518611	0.501251	0.310067	0.257190
16 ART	.5407	0.514947	0.504967	0.497538	0.303339
32 ART	.5187	0.507182	0.495180	0.492651	0.490307

Table 2: ART database compression results by number of processors

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 3: CAD 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

Table 4: ART databases distributed version speedups