Journal of Machine Learning Research 2 (2001) 19-43 Submitted 7/01; Published 10/01

Graph-Based Hierarchical Conceptual Clustering

Istvan Jonyer JONYER@CSE.UTA.EDU
Diane J. Cook COOK@CSE.UTA.EDU
Lawrence B. Holder HOLDER@CSE.UTA.EDU
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX 76019, USA

Abstract

Hierarchical conceptual clustering has proven to be a useful, although under-explored, data mining technique. A graph-based representation of structural information combined with a substructure discovery technique has been shown to be successful in knowledge discovery. The SUBDUE substructure discovery system provides one such combination of approaches. This work presents SUBDUE and the development of its clustering functionalities. Several examples are used to illustrate the validity of the approach both in structured and unstructured domains, as well as to compare SUBDUE to the Cobweb clustering algorithm. We also develop a new metric for comparing structurally-defined clusterings. Results show that SUBDUE successfully discovers hierarchical clusterings in both structured and unstructured data.

Keywords: Clustering, Cluster Analysis, Concept Formation, Structural Data, Graph Match

1. Introduction

Data mining has become a prominent research area in recent years. One of the major reasons is the ever-increasing amount of data collected in diverse areas of the industrial and scientific world. Much of this data contains valuable knowledge that is not easily retrievable. The increasing speed and capacity of computer technology has made feasible the utilization of various data mining techniques to automatically extract knowledge from this information. Such knowledge may take the form of predictive rules, clusters or hierarchies.

Beyond simple attributes of objects, many databases store structural information about relationships between objects. These structural databases provide a significant source of information for data mining. A well-publicized example is genome data, which is inherently structural (e.g., DNA atoms bonded to other atoms) and therefore benefits from a structured representation. Web data is also commonly represented using structural (hyperlink) as well as textual information. One of the more common ways of representing structural data in a computer is using graphs. Substructure discovery is a data mining technique that—unlike many other algorithms—can process structural data that contains not only descriptions of individual instances in a database, but also relationships among these instances. The graph-based substructure discovery approach implemented in the SUBDUE system has been the subject of research for a number of years and has been shown to be effective for a wide range of applications (Holder and Cook, 1993). Recent examples include the application of SUBDUE to earthquake activity, chemical toxicity domains and DNA sequences (Cook et al., 2000; Holder and Cook, 1993; Chittimoori et al., 1999; Maglothin, 1999). In this project, SUBDUE is applied to hierarchical clustering.

Cluster analysis—or simply clustering—is a data mining technique often used to identify various groupings or taxonomies in real-world databases. Most existing methods for clustering

©2001 Istvan Jonyer, Diane J. Cook and Lawrence B. Holder

apply only to unstructured data. This research focuses on hierarchical conceptual clustering in structured, discrete-valued databases. By structured data, we refer to information consisting of data points and relationships between the data points. This differs from a definition of unstructured data as containing free text and structured data containing feature vectors. Our definition of structured data focuses on the inclusion of data and relationships between the data points.

Section 2 of this paper discusses conceptual clustering in greater depth. Section 3 describes our approach to structural knowledge discovery and an implementation in the SUBDUE knowledge discovery system. Section 4 presents the design and implementation of hierarchical conceptual clustering in SUBDUE and introduces a new measure for evaluating structural hierarchical clusters. Section 5 summarizes the results of applying SUBDUE to examples from various domains and evaluates SUBDUE's success as a clustering tool. Conclusions and future work are discussed in Section 6.

2. Conceptual Clustering

Conceptual clustering has been used in a wide variety of tasks. Among these are model fitting, hypothesis generation, hypothesis testing, data exploration, prediction based on groups, data reduction and finding true topologies (Ball, 1971). Clustering techniques have been applied in as diverse fields as analytical chemistry, image analysis, geology, biology, zoology and archeology. Many names have been given to this technique, including cluster analysis, Q-analysis, typology, grouping, clumping, numerical taxonomy, mode separation and unsupervised pattern recognition, which further signifies the importance of clustering techniques (Everitt, 1980).

The purpose of applying clustering to a database is to gain a better understanding of the data, in many cases by highlighting hierarchical topologies. Conceptual clustering not only partitions the data, but generates resulting clusters that can be summarized by a conceptual description. An example of a hierarchical clustering is the classification of vehicles into groups such as cars, trucks, motorcycles, tricycles, and so on, which are then further subdivided into smaller groups based on observed traits.

Michalski defines conceptual clustering as a machine learning task (Michalski, 1980). A clustering system takes a set of object descriptions as input and creates a classification scheme (Fisher, 1987). This classification scheme can consist of a set of disjoint clusters, or a set of clusters organized into a hierarchy. Each cluster is associated with a generalized conceptual description of the objects within the cluster. Hierarchical clusterings are often described as classification trees.

Numerous clustering techniques have been devised, among which are statistical, syntactic, neural and hierarchical approaches. Clustering is considered an unsupervised learning problem because it consists of identifying valuable groupings of concepts, or facts, that hopefully reveal previously unknown information. Most techniques have some intrinsic disadvantages, however. Statistical and syntactic approaches have trouble expressing structural information, and neural approaches are greatly limited in representing semantic information (Schalkoff, 1992).

Nevertheless, many relatively successful clustering systems have been constructed. An example of an incremental approach is Cobweb, which successively considers a set of object descriptions while constructing a classification tree (Fisher, 1987). This system was created with real-time data collection in mind, where a useful clustering might be needed at any moment. Cobweb's search algorithm is driven by the category utility heuristic, which calculates intra-class similarity and inter-class dissimilarity using conditional probabilities. Instances are introduced into the classification tree at the top, and are moved down either by creating a new class or by merging the instance with an existing class. Other existing classes might also be merged or split to accommodate better definitions of classes.

Labyrinth, an extension to Cobweb, can represent structured objects using a probabilistic model (Thompson and Langley, 1991). Cobweb creates a knowledge structure based on some initial set of instances. Labyrinth is applied one step before Cobweb, resulting in a structure whose formal definition is exactly the same as that produced by Cobweb. When Cobweb is applied after Labyrinth, the resulting algorithm employs both structures to refine the domain knowledge. Both Labyrinth and SUBDUE represent structural cluster definitions. In contrast to the clustering generated by SUBDUE, however, Labyrinth's hierarchy relates parent and child based on attribute-value information, not based on structural information. In addition, only a partial graph match is performed by Labyrinth to determine if an instance is a member of a cluster.

AutoClass is an example of a Bayesian clustering system, which uses a probabilistic class assignment scheme to generate clusters (Cheeseman et al., 1988). AutoClass can process real, discrete or missing values. Another algorithm, called Snob, uses the Minimum Message Length (MML) principle to perform mixture modeling—a synonym for clustering (Wallace and Boulton, 1968).

Hierarchical approaches also exist that target databases containing data in Euclidean space. Among these are agglomerative approaches that merge clusters until an optimal separation of clusters is achieved based on intra- and inter-cluster distances. Divisive approaches are also used that split existing clusters until an optimal clustering is found. These approaches usually have the disadvantage of being applicable only to metric data, which excludes discrete-valued and structured databases. Examples of these are Chameleon (Karypis et al. 1999) and Cure (Guha et al., 1998).

Examining the major differences among the above-mentioned systems, we can see that dichotomies exist between continuous and discrete databases and between structured and unstructured databases. Cobweb can handle discrete, unstructured databases. Labyrinth can process discrete, structural databases. AutoClass can handle discrete or continuous unstructured databases. Lastly, Chameleon and Cure work with continuous-valued, unstructured data.

Few existing systems address the problem of clustering in discrete-valued, structural databases. Labyrinth is one of these systems. SUBDUE's hierarchical clustering algorithm represents another approach, centering on discrete-valued, structural databases that are represented as graphs.

3. Graph-Based Structural Knowledge Discovery

We have developed a method for discovering substructures in databases using the minimum description length principle introduced by Rissanen (1989) 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 substructure definition. The discovered substructures allow abstraction over detailed structures in the original data. Iteration of the substructure discovery 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 data analysis goals. The SUBDUE code and sample databases are available http://cygnus.uta.edu/subdue.

3.1. Graph Representation

SUBDUE accepts as input a database of structured data. This type of data is naturally represented using a graph. The graph representation includes labeled vertices with vertex id numbers and labeled directed or undirected edges, where objects and attribute values usually map

to vertices, and attributes and relationships between objects map to edges (see Figure 1 for an example). A substructure in SUBDUE consists of a subgraph definition and all of the instances of the subgraph (substructure) that occur in the graph.

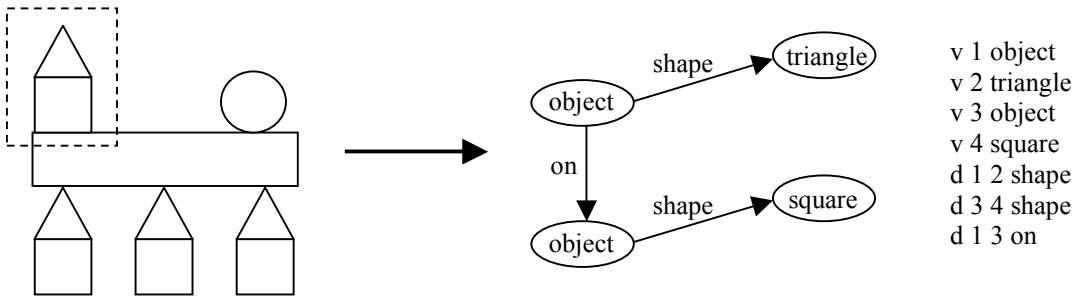

Figure 1: Example substructure in graph form with textual description. The input file syntax is *v id label* for vertices, *d id1 id2 label* for directed edges, and *u id1 id2 label* for undirected edges.

Figure 1 shows a geometric example of a structural database. The graph representation of a substructure discovered in this database is also shown, and one of the four instances of this substructure is highlighted in the picture.

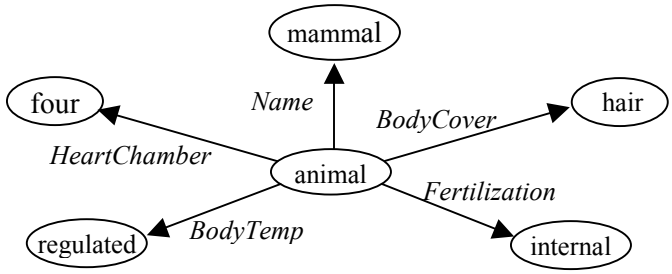

Figure 2: Graph representation of an animal description.

The input graph need not be connected, as is the case when representing unstructured databases. For data represented as feature vectors, instances are often represented as a collection of small, star-like, connected graphs. An example of the representation of an instance from the animal domain is shown in Figure 2. Intuitively, one might map the identifier or target attribute—*Name* in this case—to the center node and all other attributes would be connected to this central vertex with a single edge. This would follow the semantics of most databases. In our experience, however, SUBDUE yields better results using a more general representation including a placeholder node (*animal* in our example) that serves as the center node in the star, a representative of the example.

3.2. Search Algorithm

SUBDUE uses a variant of beam search (see Figure 3). The goal of the search is to find the substructure that best compresses the input graph. A substructure in SUBDUE consists of a substructure definition and all its occurrences in the graph. The initial state of the search is the set of substructures representing each uniquely labeled vertex and its instances. The only search operator is the *Extend-Substructure* operator. As its name suggests, *Extend-Substructure* extends the instances of a substructure in all possible ways by a single edge and a vertex, or by a single

edge if both vertices are already in the substructure. Using the example in Figure 1, a substructure representing the single vertex labeled "object" could be extended to include the vertex labeled "triangle" and the edge labeled "shape" between these vertices during the second iteration of the algorithm. The Minimum Description Length (MDL) principle is used to evaluate the substructures (see Section 3.3).

The search progresses by applying the *Extend-Substructure* operator to each substructure in the current search frontier, which is an ordered list of previously discovered substructures. The resulting frontier, however, does not contain all the substructures generated by the *Extend-Substructure* operator. The substructures are stored on a queue and are ordered based on their ability to compress the graph. The length of the queue, or beam width (Beam), is specified by the user. The user chooses how many substructures of different value—in terms of compression—are to be kept on the queue. Several substructures, however, might have the same ability to compress the graph; as a result, the actual queue length may vary. The search terminates upon reaching a user specified limit on the number of substructures extended, or upon exhaustion of the search space. SUBDUE's run time is polynomial in length of the queue and the user-specified limit on the number of considered substructures. An in-depth analysis of SUBDUE's run time can be found in the literature (Cook et al., 1996), and empirical data indicating the scalability of the serial and parallel versions of the algorithm are also reported (Cook et al., 2000).

Once the search terminates and returns the list of best substructures, the graph can be

```
Subdue ( graph G, int Beam, int Limit )
   queue Q = { v | v has a unique label in G }
   bestSub = first substructure in Q
   repeat
      newQ = {}
      for each S ∈ Q
         newSubs = S extended by an adjacent edge from G
                        in all possible ways
         newQ = newQ ∪ newSubs
         Limit = Limit - 1
      evaluate substructures in newQ by compression of G
      Q = first Beam substructures in newQ
                        in decreasing order of value
      if best substructure in Q better than bestSub
      then bestSub = first substructure in Q
   until Q is empty or Limit ≤ 0
   return bestSub
```

Figure 3: SUBDUE's discovery algorithm.

compressed using the best substructure. The compression procedure replaces all instances of the substructure in the input graph by a single vertex, which represents the substructure. Incoming and outgoing edges to and from the replaced substructure will point to, or originate from, the new vertex that represents the substructure. In our implementation, we do not maintain information on how vertices in each instance were connected to the rest of the graph. This means that we cannot accurately restore the information after compression (this is lossy, rather than lossless, compression). Since the goal of substructure discovery is interpretation of the database, maintaining information to reverse the compression is unnecessary.

The SUBDUE algorithm can be invoked again on this compressed graph. This procedure can be repeated a user-specified number of times, and is referred to as an iteration. The maximum number of iterations that can be performed on a graph cannot be predetermined; however, a graph that has been compressed into a single vertex cannot be compressed further.

3.3. Minimum Description Length Principle

SUBDUE's search is guided by the Minimum Description Length (MDL) principle, originally developed by Rissanen (1989). According to the MDL heuristic, the best substructure is the one that minimizes the description length of the graph when compressed by the substructure (Cook and Holder, 1994). This compression is calculated as

$$Compression = \frac{DL(S) + DL(G \mid S)}{DL(G)}$$

where *DL(G)* is the description length of the input graph, *DL(S)* is the description length of the substructure, and *DL(G|S)* is the description length of the input graph compressed by the substructure. The search algorithm attempts to maximize the *Value* of the substructure, which is the multiplicative inverse of the *Compression*. The description length of a graph is calculated here as the number of bits needed to encode an adjacency matrix representation of the graph. Additional details of the encoding scheme are reported in the literature (Cook and Holder, 1994).

3.4. Inexact Graph Match

When applying the *Extend-Substructure* operator, SUBDUE finds all instances of the resulting substructure in the input graph. A feature in SUBDUE, called inexact graph match, allows these instances to contain minor differences from the substructure definition. This feature is optional and the user must enable it as well as specify the degree of maximum allowable dissimilarity. The command line argument to be specified is *–threshold Number*, where *Number* is between 0 and 1 inclusive - 0 meaning no dissimilarities are allowed, and 1 meaning all graphs are considered the same. A value t between 0 and 1 means that one graph can differ from another by no more than t times the size (number of vertices plus number of edges) of the larger graph.

The dissimilarity of two graphs is calculated as the number of transformations that are needed to make one graph isomorphic to the other. The transformations include adding or deleting an edge, adding or deleting a vertex, changing a label on either an edge or a vertex and reversing the direction of an edge. All of these transformations are defined to have a cost of 1.

Our inexact graph match is based on work by Bunke and Allerman (1983). The algorithm constructs an optimal mapping between the two graphs by searching the space of all possible vertex mappings employing a branch-and-bound search. Although the space requirement is exponential in the size of the graphs, SUBDUE constrains the run time to be polynomial by resorting to hill-climbing when the number of search nodes reaches a predefined function of the size of the substructures. This is a tradeoff between an acceptable running time and an optimal match cost, but in practice, the mappings found are generally at or near optimal (lowest cost).

3.5. Improving the Search Algorithm

In SUBDUE, a *value-based queue* is used to retain substructures with the top values (the number of distinct values is specified by the user) instead of a fixed number of actual substructures. This approach was adopted in order to prevent arbitrarily pruning substructures with value equal to those surviving the pruning and thus to permit the exploration of a larger search space.

The problem with the value-based queue is that the membership in each class, or number of substructures having one of the greatest substructure values, can increase very quickly. For instance, substructures from one value class on the queue, after being extended by applying the *Extend-Substructure* operator, will result in many new substructures that will be similar, and thus offer the same compression (yielding the same evaluation measure). After several steps, the

search queue can grow to a large size. Ironically, most of these subgraphs extend into the same final substructure.

Fortunately, there is a way to prevent the above phenomenon from happening. An important observation is that the operator *Extend-Substructure* is applied to one substructure at a time, and that substructure is extended by only one edge (or edge and a neighboring vertex) at a time. These substructures can be stored on a local value-based queue. Substructures with the same value on this queue may be parents of the same child, because extensions of these substructures could be isomorphic. In particular, we check if the extension that created one of the substructures can be applied to the other substructure as well. If so, one of the substructures can be removed from consideration, because all of its extensions will also be generated by the other substructure. After all checks and subsequent deletions have been performed to the local queue, the queue is copied over to the global queue. This one-step look-ahead procedure is referred to as *purging*, because it cleans the local queue by removing substructures that would introduce redundancy in the search process.

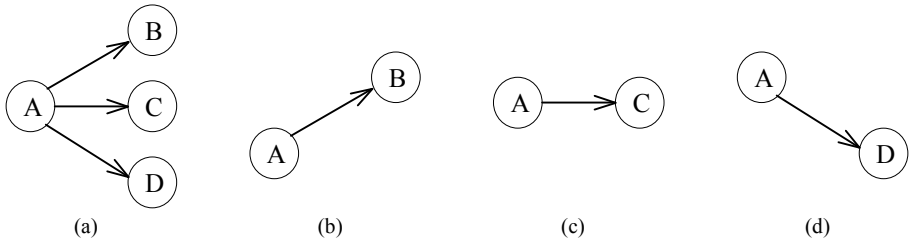

Figure 4: Purging substructures from the queue; (a) best substructure S; (b) substructure of S; (c) substructure of S; (d) substructure of S.

An example of purging is demonstrated in Figure 4. Suppose that substructure S_a shown in Figure 4a occurs in the input graph 20 times. After expanding the substructure representing vertex *A* in all possible ways, the substructures shown in Figures 4b, 4c and 4d emerge. For the sake of argument, suppose that these three substructures occur in the input graph only where substructure S occurs, and therefore they too have 20 instances. Hence, these three substructures would offer the same amount of compression, since they have the same size and same number of instances. The purging algorithm would check if substructure S_b can be extended with vertex *C* of S_c, and if substructure S_c can be extended with vertex *B* of S_b. Since this is the case, substructure S_c would be eliminated from the queue. Next this check is performed on substructure S_b and substructure S_d. The result is similar, and S_d is also eliminated from further expansion. This leaves the queue with one substructure instead of three. Since further extensions of substructure S_b result in substructures that would result from the extensions of substructures S_c and S_d, the same search space is explored using fewer substructure candidates. In the case where S_b, S_c and S_d have different numbers of instances, they will have different values, and therefore will not be compared with each other during purging.

The value-based queue and purging approaches together enable searching of a wider search space while examining potentially fewer substructures in comparison with the fixed length queue. The savings offered by purging has been observed to be substantial since the case described above arises almost every time a substructure is extended. The actual savings depend on particular graphs, the main factor being the connectivity of the graph. The more connected the graph is, the more savings purging offers.

3.6. Additional SUBDUE Features

A number of features are available in SUBDUE that improve the ease of use of the system. Here we describe some of these improvements.

The *–cluster* option initiates cluster analysis using SUBDUE. Cluster analysis is described in detail in Section 4. This option produces a classification lattice in the file "*inputFileName*.dot" that can be viewed with the GRAPHVIZ graph visualization package (Koutsofios and North, 1999). The *–truelabel* option will print the cluster definition into each node of the classification lattice when viewed with Dotty, part of the GRAPHVIZ package. The *–exhaust* option will prevent SUBDUE from terminating after discovering all substructures that can compress the graph, and instead continue until the input graph is compressed into a single vertex. To help evaluate the quality of clusterings the *–savesub* option was introduced. This option saves the definition and all the instances of the best substructure found in all of the iterations. When clustering is enabled, it also saves the classification lattice hierarchy that can be used to reconstruct the discovered substructures. These files may be used with a tool specifically designed for evaluating clusters. An extra output level was also added to display only the essential information concerned with clustering during the discovery process.

The *–prune2 number* option keeps track of local minima with respect to the minimum description length principle (see Section 3.3). The parameter *number* specifies how many more extensions are to be allowed after identifying a local minimum. This option is selected by default for clustering with the argument 2. Its benefits are described in more detail in Section 4, in the context of clustering.

SUBDUE also supports biasing the discovery process. Predefined substructures can be provided to SUBDUE, which will try to find and expand these substructures, this way "jump-starting" the discovery. The inclusion of background knowledge proved to be of great benefit (Djoko et al., 1997). SUBDUE also supports supervised learning, where examples from each class are provided as separate graphs to the system. Substructures are evaluated based on their ability to cover examples in the positive (or target) graph and to not cover examples in the other graph(s). New graphs are classified as positive if they contain the discovered substructure, and negative otherwise (Gonzalez et al., 2001). This method of influencing the discovery process has proven successful in several experiments including the chemical toxicity domain (Cook and Holder, 2000; Gonzalez et al., 2001).

4. Hierarchical Conceptual Clustering of Structural Data

The main goal of this research is to provide a method of performing hierarchical clustering of structural data. This section describes our approach to conceptual clustering of structural data and its implementation using SUBDUE.

Our cluster analysis technique uses the graph-based substructure discovery algorithm to discover substructures that represent clusters (Jonyer et al., 2000). These substructures are then used to build a hierarchy of clusters that describe the input graph. The following subsections describe the background of our approach and its implementation in the SUBDUE system.

4.1. Identifying Clusters

The SUBDUE algorithm requires one iteration to find a substructure that best compresses the input graph. This substructure represents a single cluster in our hierarchy. The members of the cluster consist of all the instances of the substructure in the input graph.

Within a single iteration, SUBDUE has several ways to decide when to stop. SUBDUE always has a single best substructure at the head of the queue, so in effect it could stop at any

point. SUBDUE has a *limit* which specifies the maximum number of substructures to consider in a single iteration. By default, the limit is set to half the size of the input graph (number of vertices plus number of edges). This number has been observed to be sufficiently large to allow the discovery of the best substructure. To minimize wasted effort, SUBDUE would stop the discovery process right after the best substructure is discovered during each iteration.

A new feature, *prune2*, attempts to find the best stopping point. This option keeps track of the compression afforded by each discovered substructure (see Section 3.3). When a minimum value is found, SUBDUE will continue only for a limited number of substructure extensions. If a new minimum is found during this time, the count is reset and SUBDUE continues further. This strategy assures that each iteration of SUBDUE returns the substructure that is responsible for the first local minimum. As discussed later, this is just what the clustering algorithm needs. Since *prune2* will stop the discovery, setting a limit is not necessary when *prune2* is used. This is the default setting for our cluster analysis.

4.2. Creating Hierarchies of Clusters

After each iteration, SUBDUE can be instructed to physically replace each occurrence of the best substructure by a single vertex, this way compressing the graph. The resulting compressed graph can then be used as the new input graph and be input to SUBDUE to discover a substructure that best compresses the new graph.

This iterative approach to clustering imposes more and more hierarchy on the database with each successive iteration. Using the fact that each new substructure discovered in successive iterations may be defined in terms of previously-discovered substructures, a hierarchy of clusters can be constructed. When clustering is enabled, the number of iterations is set to indefinite. As a result, SUBDUE will iterate until the best substructure in the last iteration does not compress the graph. If the *–exhaust* option is enabled, SUBDUE iterates until the input graph is compressed into a single vertex. This default behavior may be overridden by explicitly specifying the number of iterations to be performed, in essence specifying the number of clusters to be discovered.

Hierarchies are typically viewed as tree structures, and are used this way in many previous works on hierarchical clustering. We found, however, that in structured domains a strict tree representation is inadequate. In these cases, a lattice-like structure emerges instead of a tree. Therefore, newly discovered clusters are used to build a *classification lattice*. A classification lattice can be used to perform classification, in a method similar to the classification tree use by Fisher and others.

The classification lattice is a consequence of the fact that any cluster definition—except for the very first one—may contain previously-defined clusters. If a cluster definition does not contain any other clusters, it is inserted as the child of the root. If it is a specialization of another cluster, it is inserted as the child of that cluster, the number of branches indicating the number of times the parent cluster is in the definition of the child cluster. If the cluster definition includes more than one other cluster, then it is inserted as the child for all of those clusters.

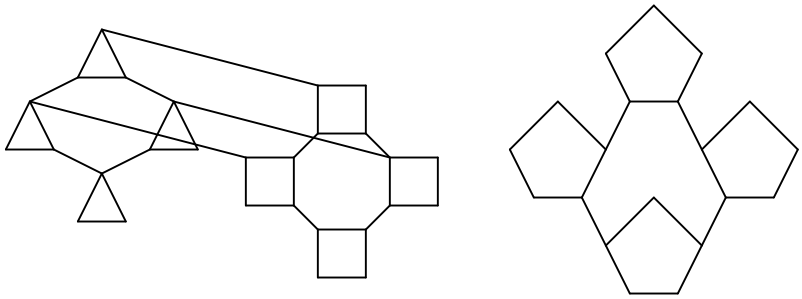

Figure 5: Artificial domain.

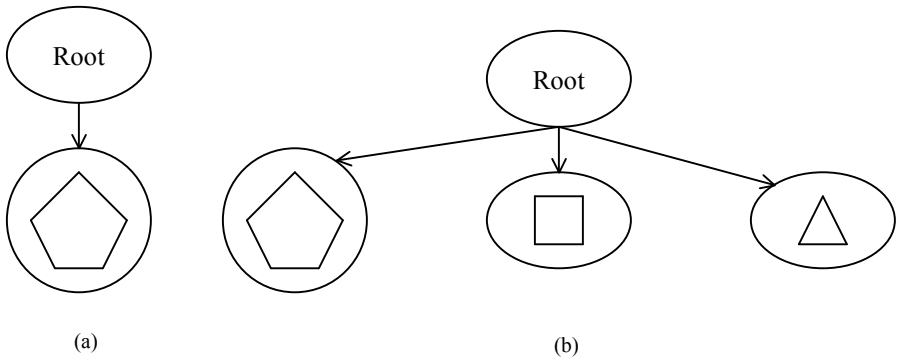

Figure 6: Clustering of the artificial domain after one iteration (a) and after three iterations (b).

To provide an example of the explanation above, the generation of a hierarchical conceptual clustering for the artificial domain shown in Figure 5 is demonstrated here. In the first iteration, SUBDUE discovers the substructure that describes the pentagon pattern in the input graph. This comprises the first cluster C_p. This cluster is inserted as a child of the root node. The resulting classification lattice is shown in Figure 6a. During iterations 2 and 3, the square shape (cluster C_s) and the triangle shape (cluster C_t) are discovered, respectively. These are inserted as children of the root as well, since C_s does not contain C_p in its definition, and C_t does not contain either C_p or C_s. The resulting lattice is shown in Figure 6b.

All of the basic shapes (pentagon, square and triangle) appear four times in the input graph. So why are these substructures discovered in the order described above? Since all of them have the same number of instances in the input graph, the size of the substructure will decide how much they can compress the input graph. The substructure describing the pentagon contains five vertices and five edges, the square contains four vertices and four edges, and the triangle contains three vertices and three edges. Given the same number of instances, the larger substructure will better compress the input graph.

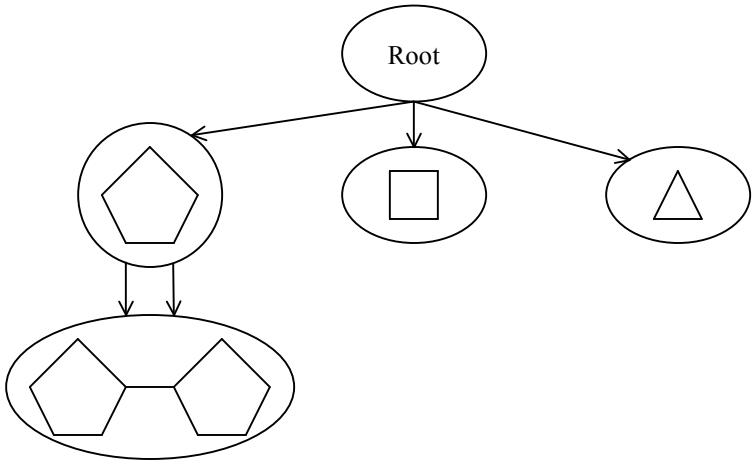

Figure 7: Clustering of the artificial domain after four iterations.

In the fourth iteration, SUBDUE returns the substructure describing two pentagon shapes connected by a single edge. There are only two instances of this formation in the graph, not four, since no overlapping of instances is permitted. This cluster is inserted into the classification lattice as the child of the cluster describing the pentagon, because that cluster appears in its

definition. The resulting classification lattice is shown in Figure 7. There are two links connecting this new cluster to its parent, because the parent cluster definition appears twice.

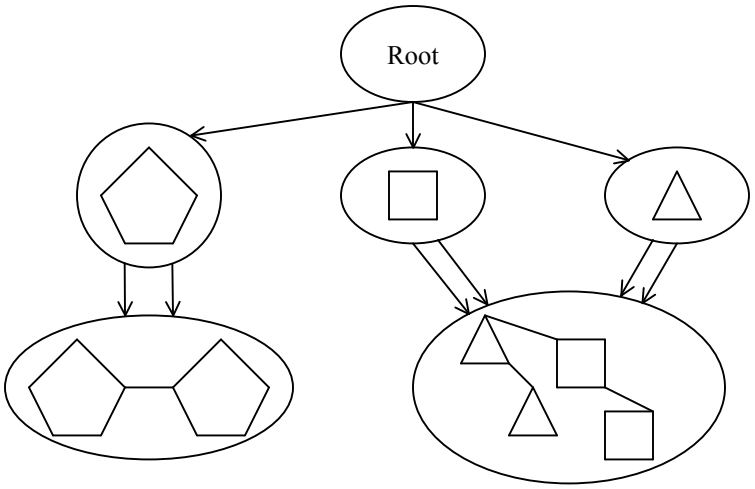

Figure 8: Clustering of the artificial domain after five iterations.

During iteration 5, a substructure is discovered that contains a pair of squares connected by an edge, a pair of triangles connected by an edge, and these two pairs are connected by a single edge. This substructure has two instances in the input graph. This cluster is inserted as a child of two clusters in the first level of the lattice, both of which appear in the definition of this new cluster. The resulting lattice is depicted in Figure 8. Since both parent cluster definitions appear twice in the new cluster, there are two links from each of these parents to the new node.

4.3. First Minimum Heuristic

SUBDUE searches the hypothesis space of classification lattices. During each iteration of the search process (while searching for each cluster), numerous local minima are encountered. The global minimum, however, tends to be one of the first few local minima. For clustering purposes, the first local minimum is used as the best cluster definition. The reason for this is as follows. SUBDUE starts with all the single-vertex instances of all unique substructures, and iteratively expands the best ones by a single edge. The local minimum encountered first is therefore caused by a smaller substructure with more instances than the next local minimum, which must be larger, and have fewer instances. A smaller substructure is more general than a larger one, and should function as a parent node in the classification lattice for any more specific clusters.

Consider the plot of a sample run shown in Figure 9. The horizontal axis of the plot shows the number of the substructure being evaluated (in order of discovery), and the vertical axis indicates the compression offered by the substructures (smaller values are better). Figure 9 shows one global minimum, appearing at substructure number 37. Several local minima occur before this substructure. Those minima, however, are caused by the dissimilarities in compression among the substructures on the queue in each search iteration. For instance, if the maximum queue length is set to be four, then there will be approximately four substructures in the queue after each extension. These four substructures will offer different amounts of compression, the first in the queue offering the most, the last in the queue offering the least. This is reflected in Figure 9.

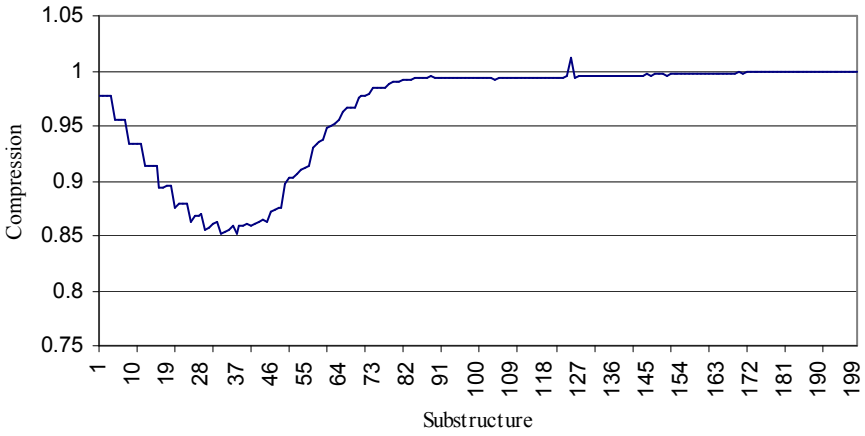

Figure 9: Compression of substructures as considered during one iteration of SUBDUE.

The staircase-like formation, shown from substructures 1 to 20, reflects similar-valued substructures in the queue (substructures 1 through 4, for example, were in the queue at the same time and had similar values). As the discovery process continues we can see that the head of the queue offers more compression than the tail (as seen in substructures 14 through 17), resulting in local minima. The *prune2* feature, however, does not consider fluctuations within each iteration (pass through the queue), but rather between iterations. In other words, minima are determined by looking at the best substructure in the queue between successive iterations. The first local minimum therefore occurs at substructure number 37. This minimum turns out to be the global minimum as well for this iteration.

As a second example, Figure 10 shows the compression of substructures as discovered by SUBDUE in a different database. The search depicted in Figure 10 features numerous local minima, the first one occurring at substructure number 46. This is not the global minimum, but for clustering purposes this one will be used as the best substructure, according to the described selection criteria.

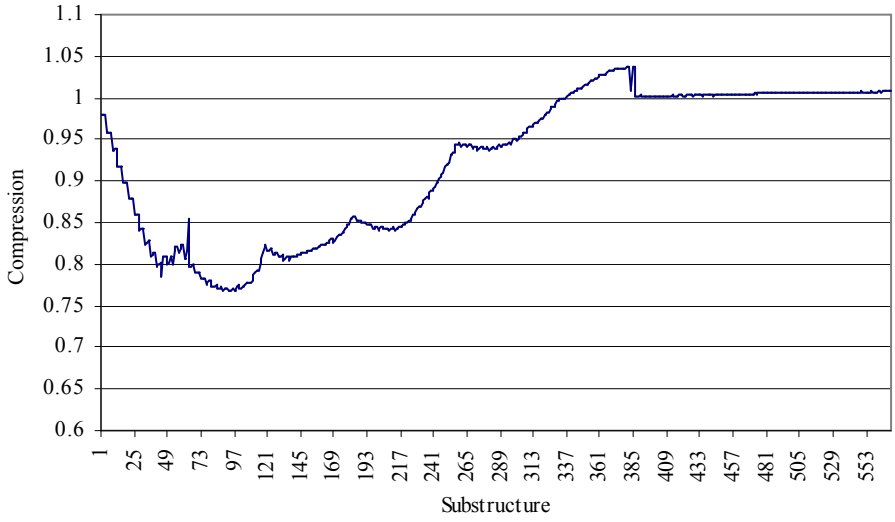

Figure 10: Compression of substructures as considered by SUBDUE during one iteration on an aircraft safety database.

Even though it is possible to use the global minimum as the best substructure, we found that if the global minimum is not the first local minimum (and is therefore not discovered in the current iteration), SUBDUE may generate *overlapping clusters*. Overlapping clusters are those that include the same information. For example, in a particular clustering of the vehicles domain, two clusters may include the information "number of wheels = 4". In the case of SUBDUE, the substructure representing the global minimum may be discovered in a later pass through the database and often will share information with the earlier (local minimum) cluster. This suggests that perhaps a better clustering could be constructed in which this information resided in a cluster at a higher level.

4.4. Implementation

This section discusses the implementation details for cluster analysis in SUBDUE. Most of the clustering functionalities center around building and printing the classification lattice. We will also describe the Dotty visualization package with emphasis on interpreting the classification lattice displayed by Dotty.

A classification lattice describes a hierarchical conceptual clustering of a database. Each node in the lattice represents a cluster. The classification lattice is a tree-like data structure with the special property that one node may have several parents. Information stored in a node includes the substructure definition and instances, pointers to children, number of children, number of parents, the substructure label, a descriptive label and a shape flag.

The substructure label specifies the vertex label (e.g., "Sub1") assigned to the substructure that represents the cluster. This label is automatically assigned to the substructure when replacing each occurrence of the substructure with a single vertex during compression. This information is useful for identifying the parents of a cluster.

The descriptive label contains information about the cluster definition in an easy-to-read format. This has significance when displaying the lattice with Dotty. The label is generated when the *–truelabel* option is set by reporting all pairs of vertices connected by an edge using the format *sourceVertex edge: targetVertex*. For example, if a substructure contains two vertices labeled *car* and *red*, connected by an edge labeled *color*, the descriptive label would read *car color: red*.

The shape flag determines the shape of the cluster when displayed by Dotty. The shape of the cluster is just another visual aid in interpreting the cluster lattice. By default, all clusters are displayed with an oval shape. When the *–exhaust* option is set, however, SUBDUE is instructed to form clusters out of substructures that do not compress the input graph further, and these clusters are given a rectangular shape.

4.5. Visualization

The GRAPHVIZ graph visualization package is used to display the cluster results (Koutsofios and North, 1999). When clustering is enabled, a file with the *.dot* extension is created. This file can be used by the program *dot* to create a PostScript file, or by Dotty to view it interactively. From Dotty one can directly print the lattice to a printer or a file. Dotty also allows the rearrangement of clusters, and the changing of cluster parameters.

Consider the portion of a classification lattice shown in Figure 11. The root node contains the file name of the input graph. Nodes other than the root node contain the sub-label of the substructure that defines the cluster, the number of instances the substructure has in the input graph (shown in brackets) and a series of descriptive labels. Each line, except for the first one, contains a descriptive label. Clusters on the same level are shown using the same color (we

modify the visualization for this paper to use multiple line textures). In some cases the lattice can become highly interconnected, and the colors are useful in identifying levels of the lattice.

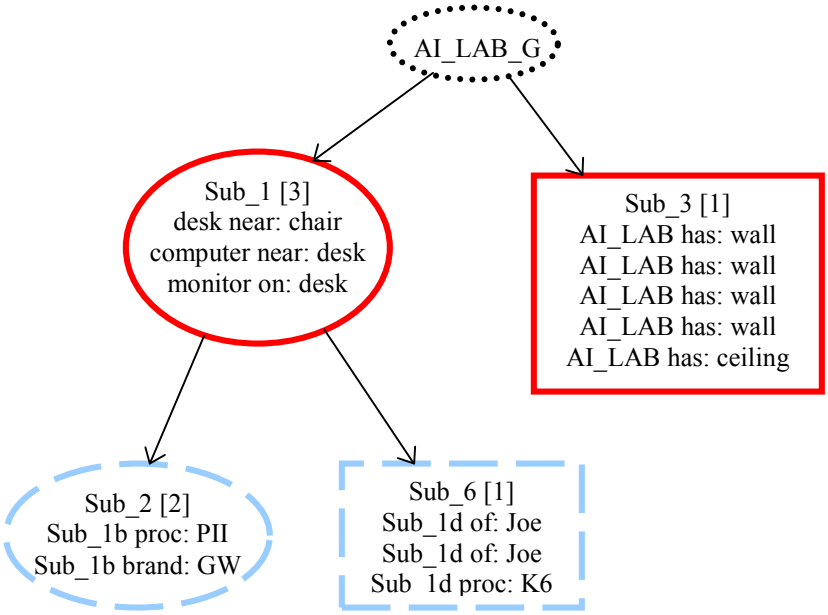

Figure 11: Example of a classification lattice produced by SUBDUE and visualized by Dotty.

4.6. Cluster Evaluation Metrics

Conventional (non-hierarchical) clusters have been evaluated using the observation that instances within a cluster should be similar to each other, and instances in separate clusters should be significantly different. The measure suggested by this observation can be defined as:

$$ClusteringQuality = \frac{InterClusterDistance}{IntraClusterDistance},$$

where *InterClusterDistance* is the average dissimilarity between members of different clusters, and *IntraClusterDistance* is the average dissimilarity between members of the same cluster. If a clustering receives a large clustering quality value, the clusters are distinctly defined, yielding a desirable clustering.

4.6.1. Defining Good Hierarchical Clusterings

When generating hierarchical clusterings, the previous metric cannot be applied. The main reason for this limitation is that clusters are organized into a hierarchy, and two clusters with an ancestral relationship are not completely disjoint. Therefore, it does not make sense to compute the average inter-cluster distance value between all pairs of clusters. Instead, only clusters that have a common parent may be meaningfully compared. Cobweb's category utility metric (Fisher, 1987) and the partition utility function introduced by Markov (2001) cannot be used as a global measure of an entire hierarchical clustering, because these measures only provide local determinations as to whether the addition of a particular cluster will increase the value of the classification tree.

Hierarchical conceptual clustering systems have been shown to be useful in practice. At the same time, there have been extensive discussions on the performance of the algorithms, as well as their advantages and applicability to certain domains. Most points are demonstrated by example, because of the lack of an objective evaluation measure. Here we introduce an evaluation measure for hierarchical clusterings. This clustering evaluation measure should be distinguished from the Minimum Description Length measure, which is used to evaluate individual substructures within the discovery algorithm. The performance measure defined here could be used to select a clustering from among the space of total possible clusterings, but this approach would be very computationally expensive.

To develop a metric for hierarchical conceptual clusterings, first we need to define what characteristics such clusterings should have. One of the properties we would like to demonstrate is the greatest coverage by the smallest possible number of clusters. This would imply that clusters are general enough to describe the all data while still defining individual concepts. A hypothesis that uses a smaller number of clusters is a simpler hypothesis, which is desirable according to the minimum description length principle.

Another desirable property is big cluster descriptions. The more features a cluster includes, the greater its inferential power (Lebowitz, 1987). Hierarchical conceptual clusterings can be used to classify new data points. A good example is the taxonomy of the animal kingdom, which can be used to classify newly discovered species using our current knowledge about animals already seen. The more traits the new species shares with points in the hierarchy, the easier it is to classify. Therefore, we would like to see well-defined concepts in the cluster hierarchy.

A third property we would like a clustering to demonstrate is minimal overlap between its clusters. No overlap indicates disjoint concepts. Clearly defined concepts are of primary importance in conceptual clustering (Michalski and Stepp, 1983).

These three desirable properties sometimes conflict. The larger the cluster description is, the more likely it is that two clusters will share common features and thus overlap. Conversely, if we remove some attributes from the cluster definition to reduce the number of overlaps, we may lose the inferential power of the cluster. In addition, if we enlarge a cluster description by adding attributes, we are likely to generate a greater number of clusters. Similarly, disallowing overlap may result in a large number of clusters. The goal of a clustering system is to balance these properties to obtain the best possible clustering.

The described features are desirable for both hierarchical and non-hierarchical clusterings and can be measured for each set of clusters. In a hierarchical clustering, the measure can be applied recursively to all clusters in the hierarchy. The quality of the root cluster thus represents the quality of the entire hierarchy. The formulation of a metric to measure cluster quality is presented next.

4.6.2. A New Metric for Hierarchical Conceptual Clustering

The previous section outlined what we seek in a good clustering. This section develops the formulation that encompasses those ideas. According to our set of desirable features, the quality of the cluster lattice L in graph G can be computed by the equation

$$Quality(L,G) = \frac{Diversity(root(L))}{Coverage(L,G)}$$

$$Diversity(C) = \frac{\sum_{i=1}^{Degree(C)-1} \sum_{j=i+1}^{Degree(C)} \sum_{k=1}^{|Child_i(C)|} \sum_{l=1}^{|Child_j(C)|} \frac{distance(Child_{i,k}(C), Child_{j,l}(C))}{\max(\|Child_{i,k}(C)\|, \|Child_{j,l}(C)\|)}}{\sum_{i=1}^{Degree(C)-1} \sum_{j=i+1}^{Degree(C)} (|Child_i(C)| * |Child_j(C)|)}$$

$$+ \sum_{i=1}^{Numchildren(C)} Diversity(Child_i(C))$$

$$Coverage(L,G) = \frac{\left\| \mathbf{Y}_{C \in L} \mathbf{Y}_{i=1}^{|C|} C_i \right\|}{\|G\|}$$

where C represents an individual cluster, C_i refers to the i^{th} instance of cluster C, $|C|$ represents the number of instances of cluster C and $||C_i||$ represents the size of the graph (number of edges plus number of vertices). The function *Degree(C)* returns the number of children of cluster C, and the *distance* operation calculates the difference between the two child cluster instances as measured by the number of transformations required to transform the smaller instance graph into the larger one. The $Child_i(C)$ function returns the i^{th} child of cluster C, and $Child_{i,k}(C)$ returns the k^{th} instance of the i^{th} child of C.

The computation of the quality of a hierarchical clustering is recursive, as indicated by the last term of the *Diversity* function. Because of the recursive nature of the calculation, the quality of the root node of the classification lattice represents the quality of the entire clustering. This value is multiplied by the coverage which serves two purposes: it scales the measure so that clusterings with different coverage may be better compared, and it penalizes clusters that increase the coverage but fail to provide other benefits. Coverage is calculated as the number of vertices and edges from the input graph that are covered by at least one of the clusters, divided by the total number of vertices and edges in the input graph.

To compute the quality of a single cluster, all of its child clusters are pairwise compared and normalized. A pairwise comparison between child clusters is performed using the inexact graph match algorithm discussed in Section 3.4. The value returned by the inexact graph match is an integer signifying the number of operations required to transform one graph into an isomorph of the other. This value is normalized to a 0..1 range by dividing it by the size of the larger graph. The dissimilarity between any two graphs is never greater than the size of the larger graph. In addition, each cluster inherits the quality of its children by adding their quality to its own.

As suggested by the pairwise comparison of child clusters, this metric measures the dissimilarity of child clusters. A larger number, or greater dissimilarity, signifies a better quality. This evaluation heuristic rewards the clusters exhibiting properties discussed in Section 4.6.1. More specific clusters are rewarded, because two such disjoint clusters need more transformations to map one cluster onto the other. This dissimilarity is normalized. For example, two clusters that each contain five vertices and five edges and have a single vertex in common are 90% different, while two clusters that each contain two vertices and one edge and have a single

vertex in common are only 66% different. Section 5.1 shows that this metric provides empirically consistent values for clusterings of varying quality.

Disjoint clusters are also rewarded. The less two clusters overlap, the more distant they are according to the inexact match algorithm. A small number of clusters is rewarded by computing the average of the comparisons of all the instances, this way offsetting the summing effect, which would normally reward a large number of clusters. As we can see, this evaluation heuristic measures all of the desirable properties for a hierarchical clustering.

Consider the clustering of the geometric database shown in Figure 8. The value of this clustering is calculated as the diversity of the root of the lattice divided by the coverage of the lattice with respect to the original graph shown in Figure 5. The combined size (or coverage) of the clusters in the lattice is 48 vertices and 56 edges, or 104. The size of the original graph is 48 vertices and 63 edges, or 111. Thus the Coverage term in the equation is 104 / 111 = 0.9369.

The numerator in the Diversity term is calculated as the pairwise distance between each child cluster instance, divided in each case by the size of the larger instance. The root node has three children, each with four instances. The distance between the pentagon and square clusters divided by the size of the larger cluster (the pentagon), summed over all 16 pairs of instances, is 11.2. Similarly, the sum of the normalized instance distances between the pentagon and triangle instances is 14.4, and between the square and triangle instances is 12.0. The sum of these terms is divided by the total number of instance pairs. There are a total of 16 + 16 + 16 = 48 instance pairs, so the first term in the Diversity function is 0.7833. The second term in this function is 0, because each of these clusters has only one child and thus there are no pairs of child instances to compare. The Quality of the lattice with respect to the input graph is thus 0.7833 / 0.9369 = 0.8360.

5. Results

This section presents analyzes clusters generated using SUBDUE. First, the algorithm's proper behavior is established using an artificially-generated database as the test domain. Next, the algorithm is compared to an existing system. Other applications of the algorithm are also discussed.

5.1. Validation in an Artificial Domain

An artificial domain will serve as an example to demonstrate SUBDUE's ability to generate valid clusterings in structural databases. This artificial domain is depicted in its graph form in Figure 5, where only edges are shown. Vertices in the graph represent the meeting points of the edges. Smaller, clearly recognizable shapes—triangles, squares and pentagons—are embedded in the graph. They are organized into rings, and some edges are added between some of the triangles and squares to somewhat disturb the regularity. The vertices in the graph are labeled as *a*, *b*, *c*, and so on, for each primitive shape. Edges connecting the primitive objects are labeled as T_link, S_link, and P_link, for triangle, square, and pentagon, respectively. Edges connecting different shapes are labeled XY, where X and Y represent the distinct shapes (e.g., TS represents triangle-square link).

SUBDUE was invoked using the command

```
Subdue -cluster -truelabel -prune2 1 artif-tsp2.g
```

where *-cluster* enables clustering, *-truelabel* enables the descriptive labels and *-prune2 1* overrides the default option for clustering, *-prune2 2*, which results in increased sensitivity to local minima. We have observed that in general the larger and more complex the database is, the more clearly defined is the local minimum.

The classification lattice generated by SUBDUE is shown in Figure 8. For clarity, the substructures are shown that define the clusters rather than the textual description extracted from the graph representation. The lattice closely resembles a tree, with the exception that the rightmost leaf has two parents. As the figure shows, smaller, more commonly occurring structures are discovered first, and compose the first level of the lattice. These cover most of the graph; therefore, they are the most general clusters. Subsequently identified clusters are based on these more general clusters which are either combined with each other, or with other vertices or edges to form new, more specific clusters. The result of this process can clearly be seen in the second level of the lattice where two pentagons and a connecting edge comprise a new cluster, and a pair of triangles and a pair of squares comprise another cluster along with three additional connecting edges. The second-level nodes in the classification lattice are connected with two branches from their parents. This means that there are two pentagons used in the bottom-left cluster, and two triangles and two squares are used in the bottom-right cluster. Both of the clusters in the second level have two instances.

SUBDUE performs as expected on this artificial domain. It was able to find the most commonly-embedded structures, and construct the expected classification lattice. To further support the algorithm's validity, the following section compares SUBDUE to an existing hierarchical clustering system.

5.2. Comparison to Cobweb

An experiment devised by Fisher (1987) can serve as a basis for comparison of SUBDUE and Cobweb. This example will also demonstrate SUBDUE's performance on unstructured data.

The database used for the experiment is given in Table 1. The animal domain is represented in SUBDUE as a graph, where attribute names (like *Name* and *BodyCover*) are mapped to labeled edges, and attribute values (like *mammal* and *hair*) are mapped to labeled vertices, as shown in Figure 1.

Table 1: Animal Descriptions.

Name	***Body Cover***	***Heart Chamber***	***Body Temp.***	***Fertilization***
mammal	hair	Four	regulated	Internal
bird	feathers	Four	regulated	internal
reptile	cornified-skin	imperfect-four	unregulated	internal
amphibian	moist-skin	three	unregulated	external
fish	scales	Two	unregulated	external

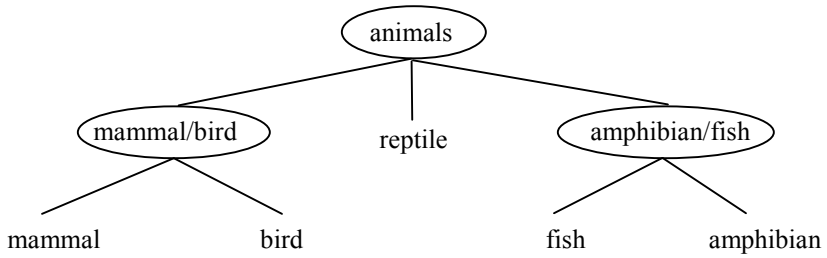

Figure 12: Hierarchical clustering over animal descriptions by Cobweb.

Cobweb produces the classification tree shown in Figure 12, as reported by Fisher (Fisher, 1987). In contrast, SUBDUE generates the hierarchical clustering shown in Figure 13.

SUBDUE's result is similar to that of Cobweb. The "*mammal/bird*" branch is clearly the same. Amphibians and fish are grouped in the same cluster based on their external fertilization, which is grouped the same way by Cobweb. SUBDUE, however, incorporates reptiles with amphibians and fish, based on their commonality in unregulated body temperature. This clustering of the animal domain seems better, because SUBDUE eliminated the overlap between the two clusters (*reptile* and *amphibian/fish*) by creating a common parent for them that describes the common trait. This example also demonstrates that SUBDUE is capable of dealing with unstructured domains successfully.

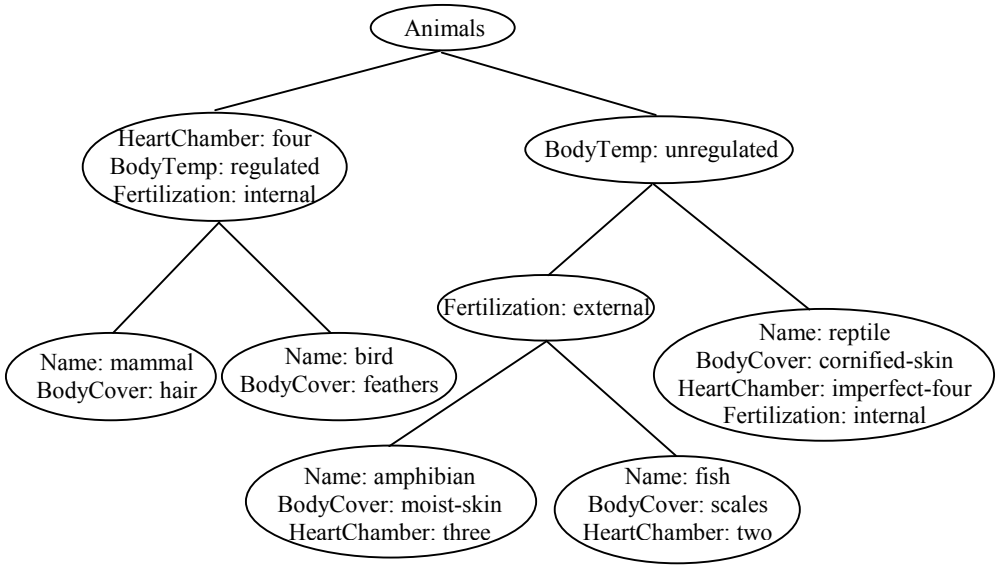

Figure 13: Hierarchical clustering over animal descriptions by SUBDUE.

5.3. Application to the Web Domain

Here we demonstrate the application of SUBDUE to a graph representing a portion of the World Wide Web. Researchers have asserted that a graph forms a natural representation for web data, and hyperlink information is frequently used to enhance web search engines (Chakrabarty et al. 1999, Kleinberg 1998). For this project, we transform web data to a labeled graph for input to SUBDUE. Data collection is performed using a web robot written in Perl. The web robot follows links to pages residing on specified servers, generating a graph file representing the visited pages. The web robot scans each page for URL references contained in that page. A depth-first search through the space of connected web pages is executed to a predefined depth. The labeled graph represents each URL as a vertex labeled "page", with edges labeled "hyperlink" pointing from parent to child URLs. To enhance the graph representation, the web robot extracts words from the "title" field of each HTML page, and adds vertices labeled with each word in the title to the graph. Functions from the WordNet library (Miller et al. 1991) are included to remove non-contributory words and to replace synonyms and abbreviations with a single representative term. Figure 14 shows a portion of the graph generated for the site cygnus.uta.edu.

For this experiment, we generated a graph representing 182 departmental web sites from four universities around the country. Our theory is that departmental web sites have common structural layouts and can thus be clustered on this basis. Over 32,000 web pages were visited, and the resulting graph contains 41,782 vertices and 168,421 edges. We let SUBDUE cluster this graph until no further compression was possible, resulting in 136 substructures. Completing the first iteration of the algorithm took 34 minutes on a 1GHz Pentium PC with 512MB memory.

Because the graph is compressed at the end of each iteration, subsequent iterations are faster. Completing 4 iterations took 48 minutes, and 16 iterations took 68 minutes. The current version of SUBDUE does not write out the file between iterations but keeps all of the information in internal memory. As a result, the memory eventually slowed down the performance of the algorithm, so that completing all 136 iterations took approximately 20 hours. We expect this performance to improve by writing each iteration result to a file rather than adding the new information to internal memory.

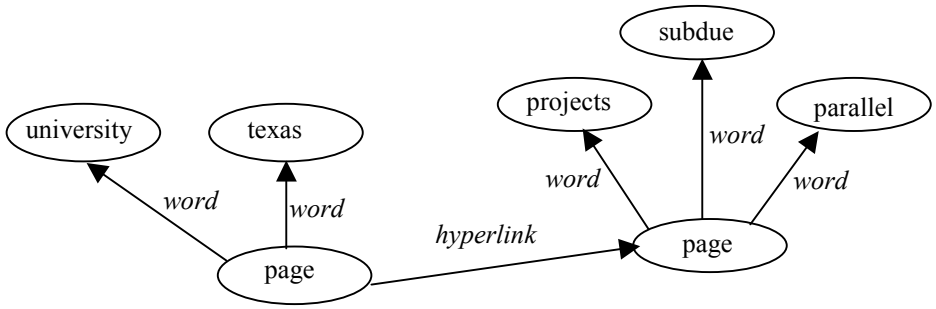

Figure 14: Graph representation of a web site.

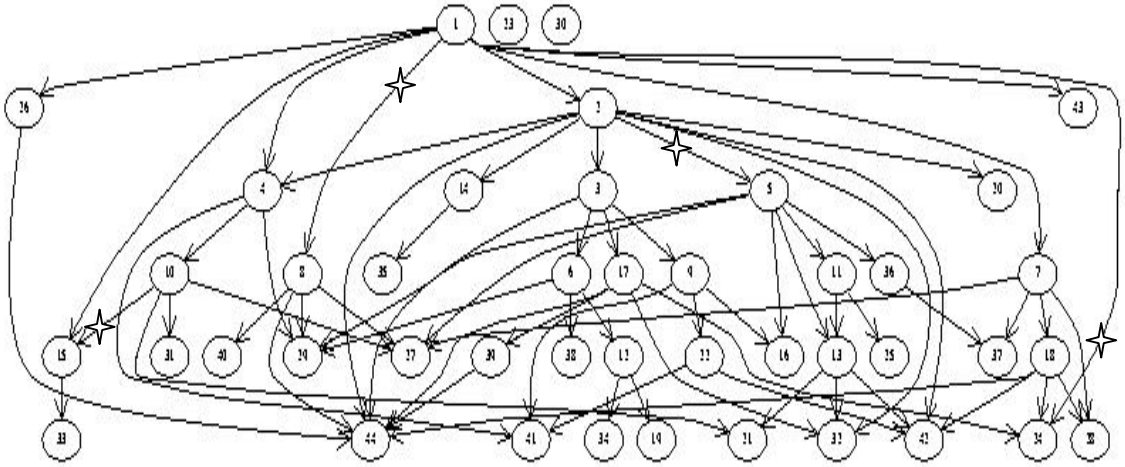

Figure 15. A portion of the web domain lattice generated by SUBDUE. Starred edges represent multiple edges between the pair of nodes.

A portion of the lattice generated by SUBDUE is shown in Figure 15. The first discovered substructure (node 1 in the figure) is defined as three web pages, the first pointing with hyperlinks to the other two pages. The next few discovered substructures expanded this theme, discovering "hub" pages with many links to other pages on the web site. Because the graph contained many more "page" vertices than "word" vertices, clusters of web pages focused on a particular topic did not appear until several levels down in the lattice. Substructure 40, for example, represents a cluster of web pages with pointers to top-level university information pages. Similarly, substructure 43 represents a cluster of departmental web pages with pointers to faculty home pages. The discovered clusters do indeed show common structural regularities within departmental web sites. For the sake of obtaining a timely response, we evaluated the lattice through the first four levels (the lower nodes typically do not add significant values to the overall value). The quality of this lattice using our evaluation measure is 10.08.

We clustered the same database using Cobweb. Because Cobweb cannot represent and process structural information, we represented each page by nine attributes. The first two attributes identify the number of inlinks and outlinks for the page. The remaining seven attributes identify the number of occurrences for the six most common words in the database with a separate attribute for all other words. Cobweb required over 40 hours to complete the clustering. Nodes within this hierarchy primarily contain pages with a similar number of inlinks or outlinks. The quality of this hierarchy evaluated through the first four levels using our measure is 6.23. The main reasons for Cobweb's lower quality measure are lack of diversity between nodes (this is more difficult to achieve without structural information) and the fact that the hierarchy is extremely deep. Lack of structural information makes abstraction of web pages difficult, and thus the hierarchy decomposed nodes to the point where almost every individual data point resides in a leaf node somewhere within the hierarchy.

5.4. Evaluation

The previous sections have shown that SUBDUE's clustering functionality is appealing in many respects. SUBDUE has performed according to expectations in an artificial structured domain, has paralleled an existing system in an unstructured domain, and has discovered clusterings in real-world domains. Here we revisit the artificial domain one more time, in order to provide an objective evaluation of SUBDUE and comparison with the clustering algorithm Cobweb using our evaluation measure.

5.4.1. Self-Diagnostic Evaluation

Due to the relatively large number of parameters in SUBDUE, the system can produce varying results. The evaluation measure can be used to help identify better clusterings generated by specifying different parameters.

An example of this is found in the clustering of the artificial domain. To create a more interesting example in Section 4.2, we deviated from the default option *–prune2 2* to *–prune2 1*. The default parameters produce the clustering shown in Figure 16.

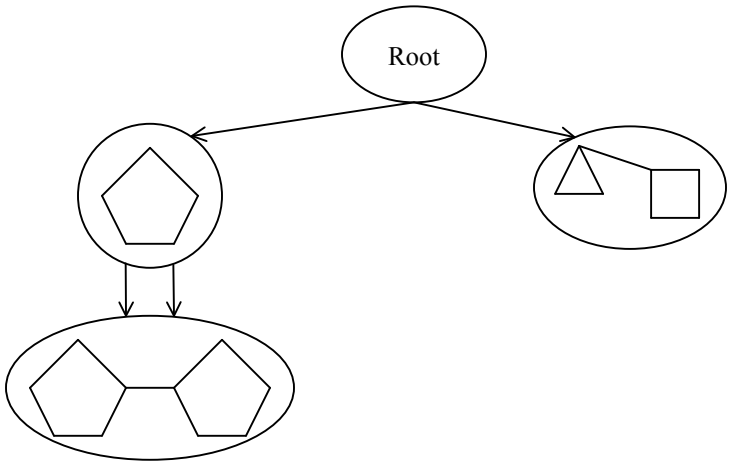

Figure 16: Alternative clustering of the artificial domain.

The clustering depicted in Figure 8 has a clustering quality of 0.836, while the one in Figure 16 has a quality value of 1.105. The difference is that the few number of clusters, larger

cluster definitions and smaller overlap between the clusters in Figure 16 outweigh the visually more pleasing structural representation shown in Figure 8.

5.4.2. Metric-Based Comparison to Cobweb

Earlier we compared SUBDUE to Cobweb on the animal descriptions domain. The evaluation of the system in that section was only anecdotal. In fact, SUBDUE's superiority over Cobweb was based entirely on the observer's opinion. In this section, the performance of both systems will be objectively evaluated.

The clustering generated by SUBDUE (shown in Figure 13) can be directly evaluated by the evaluation tool. This clustering has a quality of 2.32. The classification tree generated by Cobweb, however, needs to be converted into a graph representation that the evaluation tool can analyze. The tree in Figure 12 was converted to a graph using the representation style indicated in Figure 2, but only including attributes that define the cluster. The quality of this clustering is 1.48, according to the evaluation tool.

As a result, we can conclude that SUBDUE generated a clustering that has been shown to be better according to our evaluation metric. The major points of difference between the two clusterings are that Cobweb created a cluster on its own for the instance *reptile*, while SUBDUE incorporated it with amphibians and fish, based on their commonality in unregulated body temperature. This clustering offers a better coverage of instances, at the same time being more general. SUBDUE also eliminated the overlap between the clusters *reptile* and *amphibian/fish*, which is preferable as set forth in our evaluation criteria.

5.4.3. Discussion

We have evaluated the SUBDUE clustering tool in several domains. From both observation and objective analysis, the SUBDUE clustering tool has been shown here to be effective at providing a hierarchical cluster analysis of structured and unstructured data.

As a result of observations and objective evaluations, we can conclude that the best clustering is usually the one that has the minimum number of clusters, with minimum overlap between clusters, such that the entire data set is described. Too many clusters can arise if the clustering algorithm fails to generalize enough in the upper levels of the hierarchy, in which case the classification lattice may become shallow with a high branching factor from the root, and a greater amount of overlap. At the other extreme, if the algorithm fails to account for the most specific cases, the classification lattice may not describe the data entirely. Experimental results indicate that SUBDUE finds clusterings that effectively balance these extremes.

6. Discussion and Conclusions

The purpose of this research is to explore the mostly uncharted territory of hierarchical conceptual clustering in discrete-valued structural databases. There have been numerous attempts at clustering. Most of these, however, are applicable only in unstructured domains that simply enlist object descriptions. SUBDUE overcomes this restriction by representing databases using graphs, which allows for the representation of a large number of relationships between objects.

The technique of cluster analysis is of unquestionable importance. This is demonstrated by the wide variety of fields in which this technique is used, and the different names by which it has been referred. Many databases represent unstructured information, such as a listing of animals and their traits, but many are structured, such as a web data. Cluster analysis is equally applicable to both types of databases. A modern data mining system must be able to handle these different types of data, and operate on them successfully. In fact, many unstructured data sets may be

made structured by a simple preprocessing algorithm. An example of this might be the establishment of relationships among books with the same author in the domain of book listings, or the creation of *near* and *far* relationships, both spatial and temporal, between events in a log of earthquakes. In doing so, a data set can be made more valuable for data mining.

SUBDUE has been demonstrated to be a successful multi-purpose data mining tool in many diverse domains. Since clustering can be applied to any data set that SUBDUE can process, clustering is a very important addition in functionality to SUBDUE as has been demonstrated using various examples.

One of the major contributions of this work is the synthesis of the classification lattice. Previous work in clustering suggested the creation of classification trees, which are inadequate in structured domains. On the other hand, a classification lattice in unstructured domains reduces to a tree, which suggests that classification trees are a proper subset of classification lattices.

Another major contribution is the new evaluation metric we define for hierarchical conceptual clustering. Earlier work in this area has not developed a rigorous evaluation metric. Instead, performance is typically based on the quality of the performance as perceived by an observer, giving only anecdotal justification to their success. Our research provides an objective evaluation metric that reflects the major requirements and tradeoffs of a good quality clustering. We have demonstrated that SUBDUE's performance on unstructured datasets competes with one of the most prominent algorithms so far, perhaps even outperforming it. We also showed SUBDUE's applicability to highly structured domains using an artificial and a web domain.

Future work on SUBDUE includes defining hierarchical clusterings of other real-world domains, and performing comparisons with other clustering systems. Incorporation of the evaluation metric into SUBDUE would also be useful. In this way, SUBDUE could modify its own parameter settings and select the parameter values that yield the best overall results. We would also like to enhance SUBDUE to be able to effectively handle numeric data. Although some work has been done that learns numeric ranges for discovered substructures, more work can be done in this area.

Acknowledgements

This research was supported by National Science Foundation grants IRI-9615272 and IIS-0097517, and the State of Texas Higher Education Coordinating Board Advanced Technology Program grant 003656-45.

References

G. H. Ball. Classification Analysis. *Stanford Research Institute SRI Project 5533*, 1971.

A. Baritchi and D. J. Cook. Discovering structural patterns in telecommunications data. *Proceedings of the Florida Artificial Intelligence Research Symposium*, pp. 82-85, 2000.

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

S. Chakrabarti, B. E. Dom, D. Gibson, J. Kleinberg, R. Kumar, P. Raghavan, S. Rajapolan, and A. Tompkins. Mining the Link Structure of the World Wide Web, *IEEE Computer* 32(8):60-67, 1999.

P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. AutoClass: A Bayesian classification system. *Proceedings of the Fifth International Workshop on Machine Learning*, pp. 54–64, 1988.

D. J. Cook, L. B. Holder, G. Galal, and R. Maglothin. Parallel Approaches to Graph-Based Knowledge Discovery. *Journal of Parallel and Distributed Computing* 61(3):427-466, 2001.

D. J. Cook and L. B. Holder. Graph-Based Data Mining. *IEEE Intelligent Systems* 15(2):32-41, 2000.

D. J. Cook and L. B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research* 1:231-255, 1994.

D. J. Cook, L. B. Holder, and S. Djoko. Scalable Discovery of Informative Structural Concepts Using Domain Knowledge. *IEEE Expert* 10:59-68, 1996.

S. Djoko, D. J. Cook, and L. B. Holder. An Empirical Study of Domain Knowledge and Its Benefits to Substructure Discovery. *IEEE Transactions on Knowledge and Data Engineering*, 9(4):575-586, 1997.

B. S. Everitt, B.S. *Cluster Analysis*. Wiley & Sons, New York, 1980.

D. H. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning* 2:139-172, 1987.

J. A. Gonzalez, L. B. Holder, and D. J. Cook. Structural Knowledge Discovery Used to Analyze Earthquake Activity. *Proceedings of the Florida Artificial Intelligence Research Symposium*, pp. 86-90, 2000.

J. A. Gonzalez, L. B. Holder, and D. J. Cook. Application of Graph-Based Concept Learning to the Predictive Toxicology Domain. *To appear in Proceedings of the Predictive Toxicology Challenge Workshop*, 2001.

S. Guha, R. Rastogi, and K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. *ACM SIGMOD International Conference on Management of Data*, pp. 73-84, 1998.

L. B. Holder and D. J. Cook. Discovery of Inexact Concepts from Structural Data. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):992-994, 1993.

I. Jonyer, L. B. Holder, and D. J. Cook. Graph-Based Hierarchical Conceptual Clustering. *Proceedings of the Florida Artificial Intelligence Research Symposium*, pp. 91-95, 2000.

G. Karypis, E. Han, and V. Kumar. Chameleon: Hierarchical Clustering Using Dynamic Modeling. *IEEE Computer* 32:68-75, 1999.

J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment, *Proceedings of the Ninth ACM-SIAM Symposium on Discrete Algorithms*, pp. 668-677, 1998.

E. Koutsofios and S. C. North. Graphviz - graph drawing software. Available electronically at http://www.research.att.com/sw/tools/graphviz, 1999.

R. Maglothin. Data Mining In DNA: Using the Subdue knowledge discovery system to find potential gene regulatory sequences. Masters Thesis, Department of Computer Science and Engineering, University of Texas at Arlington, 1999.

Z. Markov, A lattice-based approach to hierarchical clustering. *Proceedings of the Florida Artificial Intelligence Research Symposium*, pp. 389-393, 2001.

R. S. Michalski and R. E. Stepp. Learning From Observation: Conceptual Clustering. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Volume 1, Tioga Publishing Company, pp. 331-363, 1983.

R. S. Michalski. Knowledge acquisition through conceptual clustering: A theoretical framework and algorithm for partitioning data into conjunctive concepts. *International Journal of Policy Analysis and Information Systems* 4:219-243, 1980.

G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: An On-line Lexical Database, *International Journal of Lexicography* 3(4):235-244, 1991.

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

J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation* 80:227–248, 1980.

R. Schalkoff. *Pattern Recognition.* Wiley & Sons, New York, 1992.

K. Thompson and P. Langley. Concept formation in structured domains. In D.H. Fisher and M. Pazzani (Eds.), *Concept Formation: Knowledge and Experience in Unsupervised Learning,* Morgan Kaufmann Publishers, Inc., pp. 127-161, 1991.

C. S. Wallace and D. M. Boulton. An Information Measure for Classification. *Computer Journal* 11(2):185-194, 1968.