May 1988 UILU-ENG-88-2223

COORDINATED SCIENCE LABORATORY
College of Engineering

DISCOVERING SUBSTRUCTURE IN EXAMPLES

Lawrence Bruce Holder, Jr.

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-88-2223	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NSF, ONR, and DARPA
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Washington, D.C. 20552, Arlington, VA 22202 Arlington, VA 22209-2308
8a. NAME OF FUNDING / SPONSORING ORGANIZATION NSF/ONR/DARPA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NSF IST-85-11170, N00014-82-K-0186 N00014-87-K-0874
8c. ADDRESS (City, State, and ZIP Code) Washington, D.C. 20552 Arlington, VA. 22202 Arlington, VA. 22209-2308	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION

11. TITLE (Include Security Classification)

Discovering Substructure in Examples

12. PERSONAL AUTHOR(S) Holder Jr., Lawrence Bruce

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	May 1988	107

16. SUPPLEMENTARY NOTATION

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	machine learning; substructure discovery, SUBDUE

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis describes a method for discovering substructure concepts in examples. The method involves a computationally constrained best-first search guided by four heuristics: cognitive savings, compactness, connectivity and coverage. Each heuristic is described in detail along with its role in evaluating an individual substructure concept. The SUBDUE system that implements the method contains a substructure discovery module, a substructure specialization module and an incremental substructure background knowledge module for applying previously discovered substructure concepts. The substructure background knowledge includes both user-defined and SUBDUE-discovered substructures in a hierarchy that is used to determine which substructures are present in the input examples. The system has performed well on a number of examples from different domains and has discovered many interesting substructure concepts such as an aromatic ring and a macro-operator for stacking blocks. The method and implementation of the SUBDUE system are described, and an analysis of experimental results is presented.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

DISCOVERING SUBSTRUCTURE IN EXAMPLES

BY

LAWRENCE BRUCE HOLDER, JR.

B.S., University of Illinois at Urbana-Champaign, 1986

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1988

Urbana, Illinois

DISCOVERING SUBSTRUCTURE IN EXAMPLES

Lawrence Bruce Holder, Jr., M.S.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1988
Robert Earl Stepp III, Advisor

This thesis describes a method for discovering substructure concepts in examples. The method involves a computationally constrained best-first search guided by four heuristics: cognitive savings, compactness, connectivity and coverage. Each heuristic is described in detail along with its role in evaluating an individual substructure concept. The SUBDUE system that implements the method contains a substructure discovery module, a substructure specialization module and an incremental substructure background knowledge module for applying previously discovered substructure concepts. The substructure background knowledge includes both user-defined and SUBDUE-discovered substructures in a hierarchy that is used to determine which substructures are present in the input examples. The system has performed well on a number of examples from different domains and has discovered many interesting substructure concepts such as an aromatic ring and a macro-operator for stacking blocks. The method and implementation of the SUBDUE system are described, and an analysis of experimental results is presented.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Robert Stepp, for his numerous contributions to the ideas within this thesis. His talent for insightful thinking and patient communication has made my work both enjoyable and rewarding.

I would also like to thank Brad Whitehall, Bob Reinke, Bharat Rao, Diane Cook and Bob Moll for their comments and suggestions regarding this work. Thanks also to the rest of the Artificial Intelligence Research Group at the Coordinated Science Laboratory; especially to my office mate, Scott Bennett, for his patient participation in our discussions. Special thanks to our secretary, Francie Bridges, for her ever pleasant assistance.

Finally, I would like to thank my family for their enthusiastic support and encouragement throughout my academic endeavors.

This research was partially supported by the National Science Foundation under grant NSF IST-85-11170, the Office of Naval Research under grant N00014-82-K-0186, by Defense Advanced Research Projects Agency under grant N00014-87-K-0874, and by a gift from Texas Instruments, Inc.

TABLE OF CONTENTS

CHAPTER

CHAPTER

CHAPTER 1

INTRODUCTION

At any given moment the amount of detailed information available from an environment is overwhelming. For example, close observation of a brick wall reveals not only the rows of rectangular bricks, but also the mortar between the bricks, the pitted surface of the bricks and mortar, small cracks in the bricks, etc. Yet, humans have the ability to ignore such detail and extract information from the environment at a level of detail that is appropriate for the purpose of the observation. Even in an unfamiliar environment, humans ignore intricate detail and discern more abstract patterns in the stimuli of the environment [Witkin83]. This thesis describes a computational method for discovering abstract patterns, or substructure, in the descriptions of a structured environment.

When observations at varying levels of detail are necessary, humans are capable of descending into the more minute structure of the environmental stimuli and identifying patterns in terms of these structural primitives. From the observations at different levels of detail, humans may construct a hierarchical description of the environment. For instance, the brick wall can be described as the rectangular bricks in the wall along with the interconnections between the bricks. Furthermore, each brick can be described in terms of the pits, cracks and embedded grains in the brick, and each interconnection can be described by the components of the mortar that combine to form the interconnection. Thus, each level in the description hierarchy represents a different level of detail for representing the environment.

Once such a hierarchy is constructed, similar primitive structures in different environments may suggest the existence of more abstract features higher up in the hierarchy. This abstraction provides a simpler description of the environment and thus allows the human to discover higher

entity representing the substructure, thus simplifying the original input examples. In addition, the newly discovered substructure is specialized by appending additional structure. This specialized substructure describes a larger, more specific portion of the current set of input examples. The substructure background knowledge includes both user-defined and discovered substructure definitions. These are used to identify instances of substructures that exist in a given set of input examples. Chapter 2 concludes by outlining a substructure discovery algorithm incorporating the aforementioned components.

Chapter 3 describes the implementation of the substructure discovery methodology contained in the SUBDUE system. In addition to the substructure discovery module, SUBDUE also includes a substructure specialization module and a substructure background knowledge module. After a substructure is discovered, the substructure is specialized, and both the original and specialized substructures are stored in the background knowledge. Within the background knowledge the substructures are kept in a hierarchy that defines complex substructures in terms of more primitive substructures. Upon receiving a new set of input examples, the background knowledge module determines which of the stored substructures are present in the examples. Thus, as the SUBDUE system runs, a hierarchical representation of selected structures found in the environment is constructed within the background knowledge module.

Chapter 4 presents several experiments with the SUBDUE system. Experiments with the substructure discovery module indicate that the substructure generation and substructure selection processes perform well in guiding the search towards more interesting substructures. Other experiments incorporating both the substructure background knowledge module and specialization module demonstrate the performance improvement obtained by using previously learned substructures in subsequent discovery tasks. The input and output data for each experiment are listed in Appendix A.

Chapter 5 surveys previous work related to substructure discovery. This work dates back to the early 1900's when gestalt psychologists began studying the underlying processes involved in

CHAPTER 2

SUBSTRUCTURE DISCOVERY

The major focus of this work is to investigate methods for discovering substructure concepts in examples. Substructure discovery is the process of identifying concepts describing interesting and repetitive "chunks" of structure within the individual elements of a set of structured examples. Once such a substructure concept is discovered, the descriptions of the examples can be simplified by replacing all occurrences of the substructure with a single form that represents the substructure. The simplified descriptions may then be passed to other learning systems. In addition, the discovery process may be applied repetitively to further simplify the descriptions or to build a hierarchical interpretation of the examples in terms of their subparts.

This chapter presents the components involved in a computational method for substructure discovery. Section 2.1 discusses the motivations for developing such a method and the importance of substructure discovery in the field of artificial intelligence. Section 2.2 defines a *substructure* along with the components comprising a substructure. Methods for generating alternative substructures are presented in Section 2.3, and the techniques used for intelligently selecting among these alternatives are discussed in Section 2.4. Once an appropriate substructure is discovered, the substructure is *instantiated* for each occurrence of the substructure in the input examples. Section 2.5 discusses substructure instantiation. Newly discovered substructures can be specialized to describe larger substructures in the input examples. Section 2.6 discusses substructure specialization. Section 2.7 presents methods for using background knowledge in the substructure discovery process. Finally, Section 2.8 outlines a substructure discovery algorithm incorporating the previously described techniques.

The human ability to perceive structural regularities in the environment provides the final motivation for investigating substructure discovery [Palmer83]. This ability allows humans to extract information from the environment at a level of detail that is appropriate for the purpose of the observation. By perceiving only the appropriate information, humans are better able to learn the concepts implied by the environment.

Thus, the underlying motivations for investigating substructure discovery are the ubiquitous hierarchical structure in the world and the ease with which humans can negotiate the hierarchy. With an appropriate representation for substructures and the substructure hierarchy, a substructure discovery system can perform the tasks just presented.

2.2. Substructure Representation

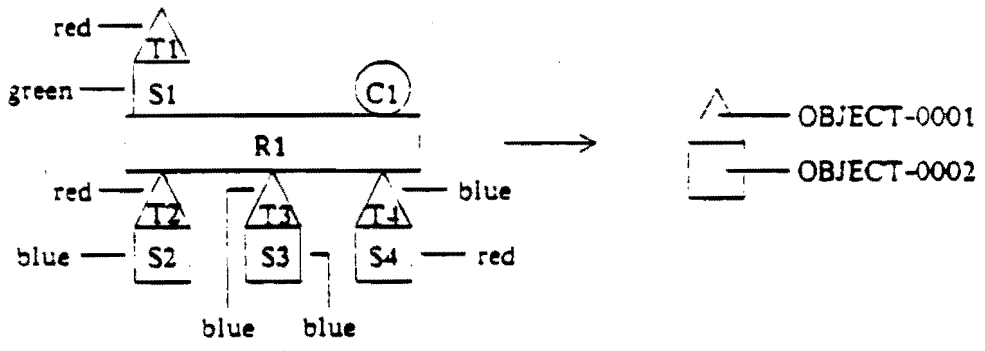
A substructure is both a portion of a collection of structurally-related objects and a description of the concept represented by that portion. For example, the detailed structure composing the concept of a brick is a substructure of a brick wall. The collection of atoms and bonds comprising the concept of an aromatic ring is a substructure of many organic compounds. However, substructure concepts are not always interesting. Upon encountering a brick wall, the concept of a brick may not be as interesting as the concept of a doorway or window. The task of substructure discovery is to find *interesting* substructure concepts in a given specification of structurally-related objects. The representation of structured concepts should be conducive to the task of discovering substructures. This section presents a graphical representation for structured concepts and a language for describing the concepts.

A collection of structured objects can be represented by a graph. Using a graph representation, a substructure is a collection of annotated nodes and edges comprising a connected subgraph of a larger graph. The nodes represent single objects, values, or relations in the substructure, and the edges represent the connections between relations and their arguments. For example, the relation (*on a b*) is represented graphically by three nodes, one annotated with *a*, one annotated with *b*, and one annotated with *on*. The *on* node is connected to both the *a* node and the

element is represented by an asterisk. the substructure <[ON(A,B)=T] [SHAPE(A)=SQUARE] [SHAPE(B)=*]> represents a square on top of some object that has a *shape* attribute, but whose *shape* value is arbitrary. Whereas, the substructure <[ON(A,B)=T] [SHAPE(A)=SQUARE]> represents a square on top of some object that may or may not have a *shape* attribute.

Figure 2.1b illustrates the substructure found in the example shown in Figure 2.1a. Both the input example and the substructure are expressed in the VL_2 language. The expression for the input example in Figure 2.1a is

```
<[SHAPE(T1)=TRIANGLE][SHAPE(T2)=TRIANGLE][SHAPE(T3)=TRIANGLE]
 [SHAPE(T4)=TRIANGLE][SHAPE(S1)=SQUARE][SHAPE(S2)=SQUARE]
 [SHAPE(S3)=SQUARE][SHAPE(S4)=SQUARE][SHAPE(R1)=RECTANGLE]
 [SHAPE(C1)=CIRCLE][COLOR(T1)=RED][COLOR(T2)=RED][COLOR(T3)=BLUE]
 [COLOR(T4)=BLUE][COLOR(S1)=GREEN][COLOR(S2)=BLUE][COLOR(S3)=BLUE]
 [COLOR(S4)=RED][ON(T1,S1)=T][ON(S1,R1)=T][ON(C1,R1)=T]
 [ON(R1,T2)=T][ON(R1,T3)=T][ON(R1,T4)=T][ON(T2,S2)=T]
 [ON(T3,S3)=T][ON(T4,S4)=T]>
```



(a) Input Example (b) Substructure

Figure 2.1. Example Substructure

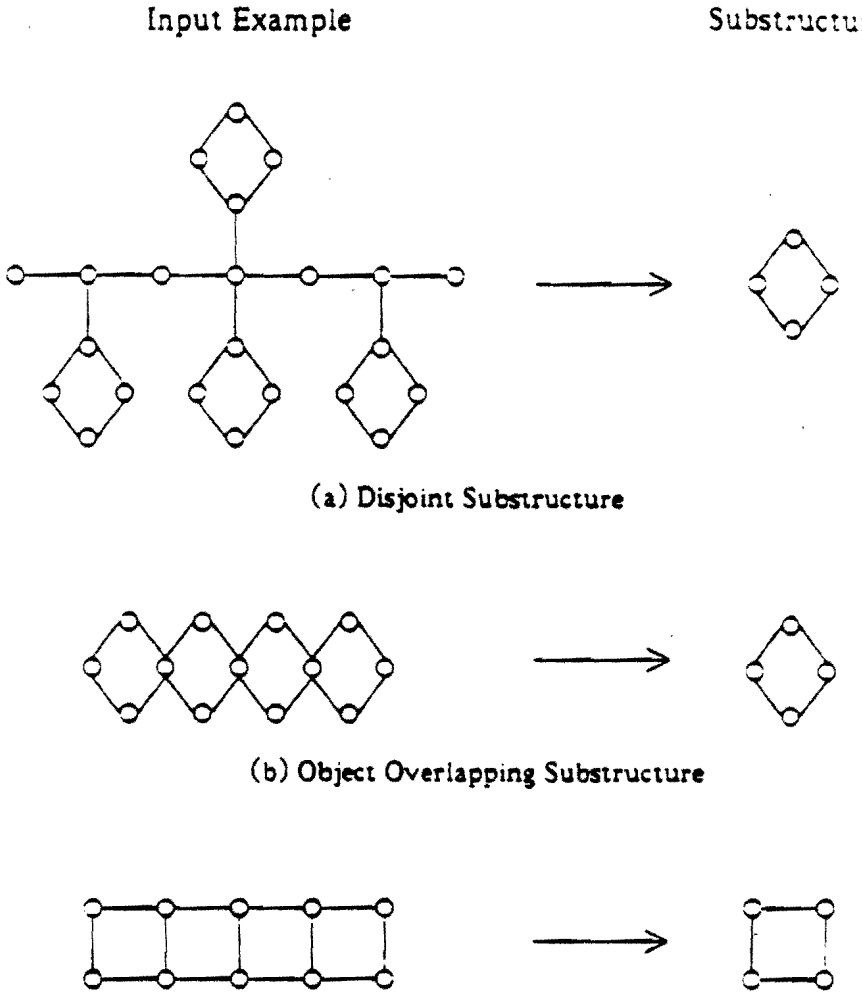


Figure 2.2. Disjoint and Overlapping Substructures

Other substructure evaluation heuristics are adaptations of the cognitive savings to reflect special qualities of a substructure. One such heuristic is *compactness*. Compactness measures the "density" of a substructure. This is not density in the physical sense, but the density based on the number of relations per number of objects in a substructure. The compactness heuristic is a

factors identified in human perception that may apply to substructure evaluation [Kohler47, Wertheimer39]. The SUBDUE system described in Chapter 3 discovers substructures by using the four heuristics presented in this section along with background knowledge to suggest promising substructures and substructure specialization to attach contextual information to newly discovered substructures.

2.5. Substructure Instantiation

Once an interesting substructure is discovered, the input examples can be recast by replacing the objects and relations of each occurrence of the substructure with a single entity representing the abstract substructure. This replacement is termed substructure *instantiation*. After performing one substructure instantiation, a substructure discovery system may continue to discover more abstract substructures in terms of those already instantiated in the input examples. This section considers two methods of substructure instantiation.

One method of substructure instantiation replaces each occurrence by a single object. Difficulties in using this method arise from representation problems. Although all the objects and relations of the occurrence are replaced by a single object, the neighboring relations are not replaced. Therefore, some recollection of the objects involved in the neighboring relations must be maintained. Also, the possibility of overlapping occurrences, as described in Section 2.4, only confounds the instantiation problem. In the case of object overlap, each instantiation of the overlapping occurrences must remember the overlapping components. Similarly, in the case of relation overlap, not only must the overlapping objects be retained, but the overlapping relations as well. Retaining the extra object information is inconsistent with the idea of substructure instantiation using a single new object. Although single object substructure instantiation seems intuitively promising, the accompanying representation problems are difficult to overcome.

An alternative approach to substructure instantiation involves replacing each occurrence of the substructure with a new relation. The arguments to this relation are all the objects in the substructure occurrence. During instantiation, all relations in the occurrence are removed from the

2.3. Substructure Generation

An essential function of any substructure discovery system is the generation of alternative substructures. The substructure generation process constructs new substructures from the objects and relations in the input examples. There are two basic approaches to the generation problem, bottom-up and top-down.

The bottom-up approach to substructure generation begins with the smallest substructures in the input examples and iteratively expands each substructure. The expansion may be accomplished by two different methods. The first method, *minimal expansion*, adds one neighboring relation to the substructure. For example, according to the three neighboring relations (presented in Section 2.2) of the occurrence, <[ON(T1,S1)=T][SHAPE(T1)=TRIANGLE][SHAPE(S1)=SQUARE]>, the substructure in Figure 2.1b would be expanded to generate the following three substructures

<[SHAPE(OBJECT-0001)=TRIANGLE][SHAPE(OBJECT-0002)=SQUARE]
[ON(OBJECT-0001,OBJECT-0002)=T][COLOR(OBJECT-0001)=RED]>
<[SHAPE(OBJECT-0001)=TRIANGLE][SHAPE(OBJECT-0002)=SQUARE]
[ON(OBJECT-0001,OBJECT-0002)=T][COLOR(OBJECT-0002)=GREEN]>
<[SHAPE(OBJECT-0001)=TRIANGLE][SHAPE(OBJECT-0002)=SQUARE]
[ON(OBJECT-0001,OBJECT-0002)=T][ON(OBJECT-0002,OBJECT-0003)=T]>

The second method, *combination expansion*, is a generalization of minimal expansion in which two substructures, having at least one object in common, are combined into one substructure. For example, the substructure in Figure 2.1b could be generated by combining the following two substructures

<[SHAPE(OBJECT-0001)=TRIANGLE][ON(OBJECT-0001,OBJECT-0002)=T]>
<[ON(OBJECT-0001,OBJECT-0002)=T][SHAPE(OBJECT-0002)=SQUARE]>

The top-down approach to substructure generation begins with the largest possible substructures, one for each input example, and iteratively disconnects the substructure into two smaller substructures. As with the expansion approach, substructure disconnection may be accomplished by two different methods. The first method, *minimal disconnection*, removes one relation from a substructure while preserving the resulting substructure's connectivity. For

that have a large number of relations and objects in common. Exhaustive application of minimal disconnection can be avoided by removing only those relations that occur the least in the input examples, yielding substructures with perhaps an increased number of occurrences. Lastly, cut disconnection can be applied more intelligently by removing relations that yield highly connected substructures. The techniques of finding *articulation points* [Reingold77] and *cut points* [Zahn71] are applicable here.

Regardless of how the methods are applied, each method has advantages and disadvantages. Although the top-down approaches allow quicker identification of isolated substructures, they suffer from high computation costs due to frequent comparisons of larger substructures. Both the combination expansion and cut disconnection methods are appropriate for quickly arriving at a larger substructure, but a smaller, more desirable substructure may be overlooked in the process. Also, in the context of building a substructure hierarchy, beginning with smaller substructures is preferred, because the larger substructures can then be expressed in terms of the smaller ones. Minimal expansion begins with smaller substructures, expands substructures along one relation and, thus, is more likely to discover smaller substructures within the computational resource limits of the system.

2.4. Substructure Selection

After using the methods of the previous section to construct a set of alternative substructures, the substructure discovery algorithm must choose which of these substructures to consider the best hypothetical substructure. This is the task of substructure selection. The proposed method of selection employs a heuristic evaluation function to order the set of alternative substructures based on their heuristic quality. This section presents the major heuristics that are applicable to substructure evaluation.

The first heuristic, *cognitive savings*, is the underlying idea behind several utility and data compression heuristics employed in machine learning [Minton87, Whitehall87, Wolff82]. Cognitive savings measures the amount of data compression obtained by applying the substructure to the

occurrence. From the common object symbols within the relations, the overlapping objects and relations of the original occurrences may be reconstructed.

Thus, single relation substructure instantiation remains the more accurate approach for replacing substructure occurrences with a single, more abstract entity representing the original objects and relations in the occurrence. Replacing objects and relations through substructure instantiation reduces the complexity of the input examples and allows subsequent discovery of concepts defined in terms of the instantiated substructures.

2.6. Substructure Specialization

Specializing substructures is an essential capability of a substructure discovery system. For instance, suppose the system finds six occurrences of an aromatic ring substructure within a set of input examples. Three of the occurrences have an attached chlorine atom, and three occurrences have an attached bromine atom. The discovery system may benefit by retaining not only the aromatic ring substructure, but also a more specific aromatic ring substructure with an attached atom whose type is described by the disjunction "chlorine or bromine". Performing this specialization step allows the substructure discovery system to take advantage of additional information in the input examples, avoid learning overly general substructure concepts, and more rapidly discover a specific disjunctive substructure concept. This section presents an approach to substructure specialization.

One technique for specializing a substructure is to perform inductive inference on the extended occurrences of the substructure. An extended occurrence of a substructure is the substructure generated by adding one neighboring relation to the occurrence. The set of extended occurrences consists of the substructures obtained by expanding each occurrence with each neighboring relation of the occurrence. Once the set of extended occurrences is constructed, inductive inference generalization techniques [Michalski83a] can be applied to the newly added neighboring relations and their corresponding values. The resulting generalized neighboring relations are then appended to the original substructure to produce new, specialized substructures.

2.7. Background Knowledge

Although the substructure discovery techniques described in the previous sections work without prior knowledge of the domain, the application of background knowledge can direct the discovery process along more promising paths through the space of alternative substructures. This section considers background knowledge in the form of substructure definitions: that is, candidate substructures that are more likely to exist in the current application domain.

The two major functions of the substructure background knowledge are to maintain both user-defined and discovered substructures and to determine which of these substructures exist in a given set of input examples. In order to take maximum advantage of the hierarchical nature of substructures, the background knowledge is arranged in a hierarchy in which complex substructures are defined in terms of more primitive substructures. This arrangement suggests an architecture similar to that of a truth maintenance system (TMS) [Doyle79]. Primitive substructures serve as the justifications for more complex substructures at a higher level in the hierarchy. When a new substructure is added to the background knowledge, primitive substructures are justified by the objects and relations in the new substructure. These primitive substructures provide partial justification for the new substructure. Furthermore, the TMS architecture provides a simple process for determining which background knowledge substructures exist in a given set of input examples. This determination can be accomplished by first justifying the relations at the leaves of the background knowledge hierarchy with the relations in the input examples. Then, initiate the normal TMS propagation operation to indicate which substructures are ultimately justified by the relations in the input examples. The substructure discovery process may then use these background knowledge substructures as a starting point for generating alternative substructures.

Other forms of background knowledge apply to the task of substructure discovery. Whitehall's PLAND system [Whitehall87] for discovering substructure in sequences of actions uses three levels of background knowledge to guide the discovery process. PLAND's high level

```
L = limit on computation time
S = {base substructures}
D = {}
while (amount_of_computation < L) and (S ≠ {}) do
 BEST-SUB = best substructure selected from S
 S = S - {BEST-SUB}
 D = D ∪ {BEST-SUB}
 E = {alternative substructures generated from BEST-SUB}
 for each e in E do
  if (not (member e D))
    S = S ∪ {e}
return D.
```

Figure 2.4. Substructure Discovery Algorithm

The next step in the algorithm is a loop that continuously generates new substructures from the substructures in S until either the computational limit, L, is exceeded or the set of candidate substructures, S, is exhausted. The loop begins by selecting the best substructure in S. Here, the heuristics of Section 2.4 are employed to choose the best substructure from the alternatives in S. The actual computation performed to compute the heuristic evaluation function depends on the implementation. Section 3.2.2 describes the computation used in SUBDUE; although, other methods may be used. For instance, the heuristics could be weighted, selected by the user, or selected by the background knowledge. However, any substructure selection method should involve the four heuristics described in Section 2.4. Once selected, the best substructure is stored in BEST-SUB and removed from S. Next, if BEST-SUB does not already reside in the set D of discovered substructures, then BEST-SUB is added to D. The substructure generation methods of Section 2.3 are then used to construct a set of new substructures that are stored in E. Those substructures in E that have not already been considered by the algorithm are added to S, and the loop repeats. When the loop terminates, D contains the set of discovered substructures.

CHAPTER 3

THE SUBDUE SYSTEM

An implementation of the substructure discovery methodology described in the previous chapter is contained in the SUBDUE system. Written in Common Lisp on a Texas Instruments Explorer, the SUBDUE program facilitates the use of the discovery algorithm both as a substructure concept discoverer and as a module in a more robust machine learning system. In addition to the heuristic-based substructure discovery module, SUBDUE also includes a substructure specialization module and a substructure background knowledge module for utilizing previously discovered substructures in subsequent discovery tasks. The substructure background knowledge holds both user-defined and discovered substructures in a hierarchy and determines which of these substructures are present in the input examples.

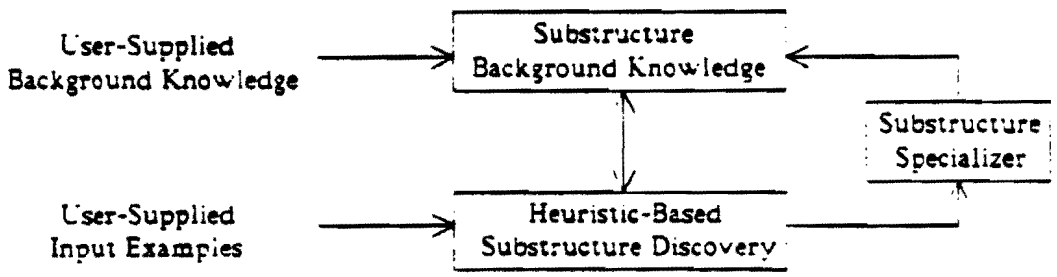
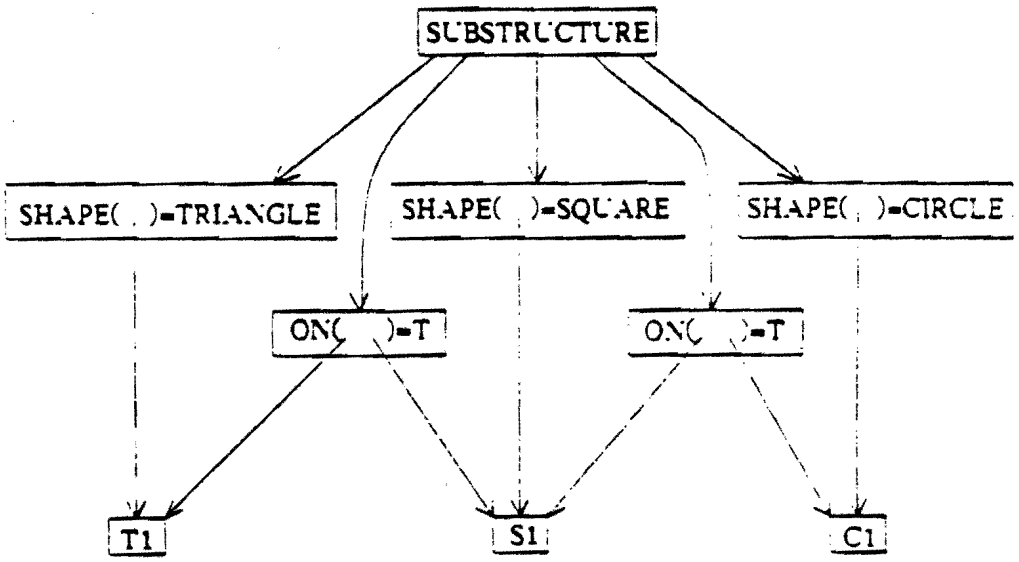


Figure 3.1. The SUBDUE System

(a) Substructure

[SHAPE(T1)=TRIANGLE][SHAPE(S1)=SQUARE][SHAPE(C1)=CIRCLE]
[ON(T1,S1)=T][ON(S1,C1)=T]

(b) External Representation



(c) Internal Representation

Figure 3.2. Substructure Representation

```
SUB = description of substructure to be expanded
NEWSUBS = {}
N = {neighboring relations of the occurrences of SUB}
foreach n in N do
  NSUB = new substructure formed by adding n to SUB
  NEWSUBS = NEWSUBS ∪ {NSUB}
return NEWSUBS.
```

Figure 3.3. Substructure Expansion Algorithm

Figure 3.3 shows the substructure expansion algorithm. The new, expanded substructures are stored in NEWSUBS, initially an empty set. First, the set of neighboring relations of SUB are stored in N. For each neighboring relation, a new substructure is formed by adding the neighboring relation to the original substructure description, SUB. The newly formed substructure is added to the set of expanded substructures, NEWSUBS. After all possible neighboring relations have been considered, the expansion algorithm returns NEWSUBS as the set of all possible substructures expanded from the original substructure.

3.2.2. Selection

The heuristic-based substructure discovery module selects for consideration those substructures that score highest on the four heuristics introduced in Section 2.4: cognitive savings, compactness, connectivity and coverage. These four heuristics are used to order the set of alternative substructures based on their heuristic value in the context of the current set of input examples. With the substructures ordered from best to worst, substructure selection reduces to selecting the first substructure from the ordered list. This section describes the computations involved in the calculation of each heuristic, and how these results are combined to yield the heuristic value of a substructure.

the cognitive savings value depends on the overlap of the occurrences of the substructure in the current set of input examples.

The second heuristic, compactness, measures the density of the substructure. Compactness is defined as the ratio of the number of relations in the substructure to the number of objects in the substructure. Unlike cognitive savings, the compactness of a substructure is independent of the input examples.

$$\text{compactness}(S) = \frac{\#\text{relations}(S)}{\#\text{objects}(S)}$$

For each of the substructures in Figure 2.2, #relations(S) = 4 and #objects(S) = 4; thus, compactness = 4/4 = 1.

The third heuristic, connectivity, measures the amount of external connection in the occurrences of the substructure. Connectivity is defined as the inverse of the average number of external connections found in all occurrences of the substructure in the input examples. Thus, the connectivity of a substructure S with occurrences OCC in the set of input examples E is computed as

$$\text{connectivity}(S,E) = \left(\frac{\sum_{i \in OCC} \left| \text{external\_connections}(i) \right|}{\left| OCC \right|} \right)^{-1}$$

Again, consider Figure 2.2. Each substructure has four occurrences in the input example. In Figure 2.2a, each occurrence has one external connection; thus, connectivity = $(4/4)^{-1} = 1$. In Figure 2.2b and Figure 2.2c, the two innermost occurrences both have 4 external connections and the two outermost occurrences both have 2 external connections, for a total of 12 external connections. Thus, connectivity = $(12/4)^{-1} = 1/3$.

The final heuristic, coverage, measures the amount of structure in the input examples described by the substructure. Coverage is defined as the number of unique objects and relations in

<(SHAPE(T1) = TRIANGLE][SHAPE(T2) = TRIANGLE][SHAPE(T3) = TRIANGLE]
[SHAPE(T4) = TRIANGLE][SHAPE(S1) = SQUARE][SHAPE(S2) = SQUARE]
[SHAPE(S3) = SQUARE][SHAPE(S4) = SQUARE][SHAPE(R1) = RECTANGLE]
[SHAPE(C1) = CIRCLE][ON(T1,S1) = T][ON(T2,S2) = T][ON(T3,S3) = T]
[ON(T4,S4) = T][ON(S1,R1) = T][ON(C1,R1) = T][ON(R1,T2) = T]
[ON(R1,T3) = T][ON(R1,T4) = T]>

First, the algorithm forms the set S of base substructures. Initially, S has only one element, the substructure denoted by <1.1 OBJECT-0001>. This substructure has as many occurrences as there are objects in the input example. The number before a substructure is the *value* of the substructure, as defined in Section 3.2.2. The object names within the substructures are arbitrary symbols generated by the system for each newly constructed substructure.

S = {<1.1 OBJECT-0001>}

Next, we enter the loop, where the best substructure in S is stored in BEST-SUB, removed from S and inserted in the set D of discovered substructures. Next, BEST-SUB is minimally expanded by adding one neighboring relation to BEST-SUB in all possible ways. The newly created substructures are stored in E.

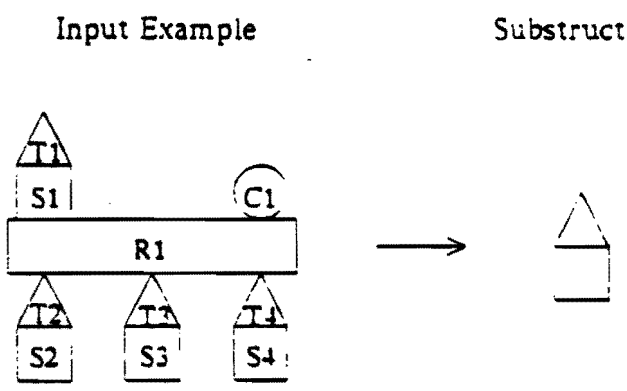


Figure 3.4. Simple Example

In this example the next substructure to be considered, <15.5 [SHAPE OBJECT-0004 = TRIANGLE] [ON(OBJECT-0004,OBJECT-0002) = T] [SHAPE(OBJECT-0002 = SQUARE)>, will emerge as the best substructure. Regardless of the amount of additional computation, this substructure (the substructure in Figure 3.4) will be the best element in the set of discovered substructures returned by the algorithm.

3.3. Substructure Specialization

The substructure specialization module in SUBDUE employs a simple technique for specializing a substructure. This technique is based on the method described in Section 2.6. SUBDUE specializes a substructure by conjoining one attribute relation. The value of the added attribute relation is a disjunction of the values observed in the attribute relations connected to the occurrences of the substructure. To avoid over-specialization the substructure is conjoined with the disjunctive attribute relation representing the minimal amount of specialization among the possible disjunctive attribute relations of the substructure. More specific substructures will eventually be considered after less specific substructures have been stored in the background knowledge, found in subsequent examples, and further specialized. Section 3.3.1 describes the substructure specialization algorithm, and Section 3.3.2 illustrates an example of the specialization process.

3.3.1. Specialization Algorithm

The substructure specialization algorithm used by SUBDUE is shown in Figure 3.5. The algorithm returns all possible specializations for a given substructure in the current set of input examples.

The algorithm proceeds as follows. Given a substructure S with occurrences OCC in the set of input examples E the substructure specialization algorithm in Figure 3.5 returns the set of all possible specializations of S. These specializations will be collected in SPECSUBS, that is initially empty. The algorithm begins by storing in ATTRIBUTES all the attribute relations in the

After considering each attribute relation in ATTRIBUTES, the algorithm terminates and returns the set of specialized substructures stored in SPECSUBS. However, only the minimally specialized substructure is eventually stored in the substructure background knowledge. Therefore, for a substructure, S_{spec}, with newly added attribute relation [REL(OBJ)=UNIQUE_VALUES] and occurrences OCC the following formula is used to measure the amount of specialization in S_{spec}:

$$\text{amount\_of\_specialization}(S_{spec}) = \frac{|\text{UNIQUE\_VALUES}|}{|\text{OCC}|}$$

The substructure with the smallest amount of specialization will then be stored in the substructure background knowledge along with the originally discovered substructure.

3.3.2. Example

As an example of the substructure specialization process, consider Figure 3.6. Figure 3.6a illustrates the same input example of Figure 3.4 with the addition of several *color* attribute relations. After running the heuristic-based substructure discovery algorithm on this example, the same substructure emerges as in Figure 3.4.

S = <[SHAPE(OBJECT-0002)=TRIANGLE][SHAPE(OBJECT-0001)=SQUARE]
[ON(OBJECT-0002,OBJECT-0001)=T]>

Next, the newly discovered substructure is sent to the substructure specialization module. First, all the attribute relations of the occurrences of the substructure are stored in ATTRIBUTES:

ATTRIBUTES = {[COLOR(T1)=RED], [COLOR(T2)=RED], [COLOR(T3)=BLUE],
[COLOR(T4)=BLUE], [COLOR(S1)=GREEN], [COLOR(S2)=BLUE],
[COLOR(S3)=BLUE], [COLOR(S4)=RED]}

Next, the first attribute relation in ATTRIBUTES is given a value 'T' and added to the original substructure. The name of the object argument to the attribute relation is changed from T1 to OBJECT-0002, because OBJECT-0002 is the name of the object in the description of the

Thus, the following specialized substructure is added to SPECSUBS.

S_{spec} = <[COLOR(OBJECT-0002)=BLUE,RED][SHAPE(OBJECT-0002)=TRIANGLE]
[SHAPE(OBJECT-0001)=SQUARE][ON(OBJECT-0002,OBJECT-0001)=T]>

S_{spec} has 4 occurrences and 2 unique values in the newly added attribute relation; thus, amount_of_specialization(S_{spec}) = 2/4 = 1/2. The only other specialized substructure added to SPECSUBS in this example is

S_{spec} = <[COLOR(OBJECT-0001)=BLUE,GREEN,RED][SHAPE(OBJECT-0002)=TRIANGLE]
[SHAPE(OBJECT-0001)=SQUARE][ON(OBJECT-0002,OBJECT-0001)=T]>

This specialized substructure also has 4 occurrences, but 3 unique values; thus, amount_of_specialization(S_{spec}) = 3/4. The first specialized substructure has a smaller amount of specialization. Therefore, only the first substructure, shown in Figure 3.6b, is added to the substructure background knowledge along with the originally discovered substructure.

Specializing the substructures discovered by SUBDUE adds to the substructures information about the context in which the substructures are likely to be found. By minimally specializing the substructure, SUBDUE avoids adding contextual information that is too specific. If the desired substructure concept is more specific than that obtained through minimal specialization, specializing similar substructures in subsequent examples will transform the under-constrained substructure into the desired concept. There is the possibility that even the minimal amount of specialization may over-constrain a substructure. SUBDUE can recover from this problem, because both the original and specialized substructures are retained in the background knowledge. The unspecialized substructure will always be available for application to subsequent discovery tasks.

3.4. Substructure Background Knowledge

The substructure background knowledge module in SUBDUE has two major functions: storing both user-defined and discovered substructures and determining which of these

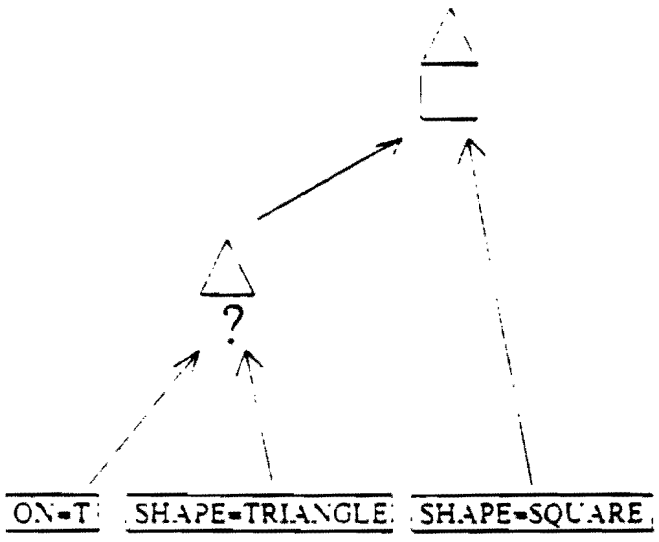


Figure 3.7. Substructure Background Knowledge Example

exactly two justifications. When both justifications are supported, the substructure node is also supported.

As an example, recall the substructure shown in Figure 3.6a. The background knowledge hierarchy for this substructure is shown in Figure 3.7. The question mark appearing in the hierarchy represents an object. Thus, the substructure containing the question mark is [SHAPE(X)=TRIANGLE][ON(X,Y)=T], where the question mark corresponds to the object argument Y. Other objects in the hierarchy are represented by the pictorial equivalent of their *shape* attribute.

Next, suppose either the user or SUBDUE wants to add the substructure from Figure 3.2a to the substructure background knowledge hierarchy in Figure 3.7. The resulting hierarchy is shown in Figure 3.8. This hierarchy is obtained by first treating the new substructure as an example and finding the highest level substructure already in the background knowledge hierarchy that occurs in the new substructure. In this case, the entire hierarchy of Figure 3.7 is justified by the new

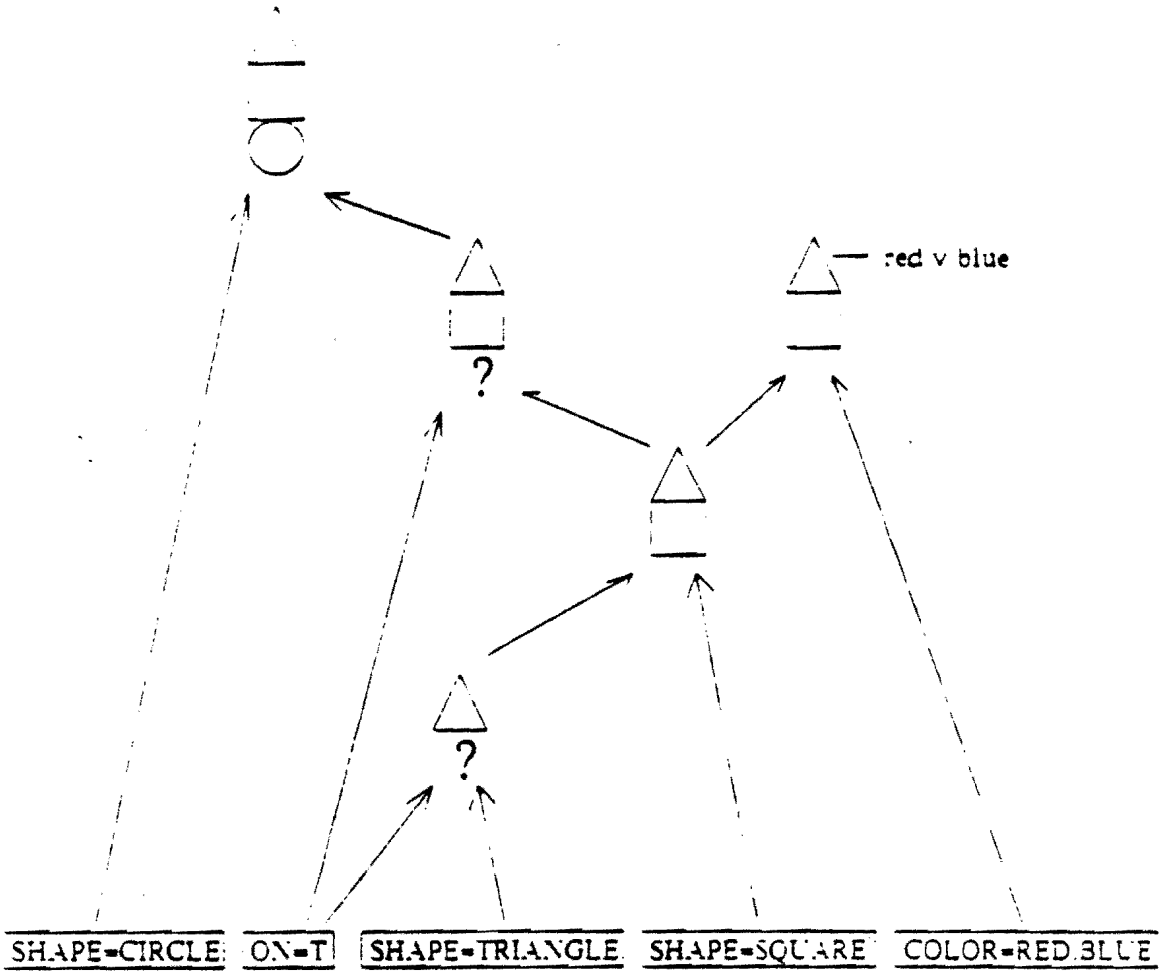


Figure 3.9. Three Background Knowledge Substructures

the user or by SUBDUE, the substructure background knowledge grows incrementally to define the new substructures in terms of the substructures already known.

3.4.2. Identifying Substructures in Examples

As described in Section 2.8, the substructure discovery algorithm includes, in the set of initial substructures, any background knowledge substructures occurring in the set of input examples.

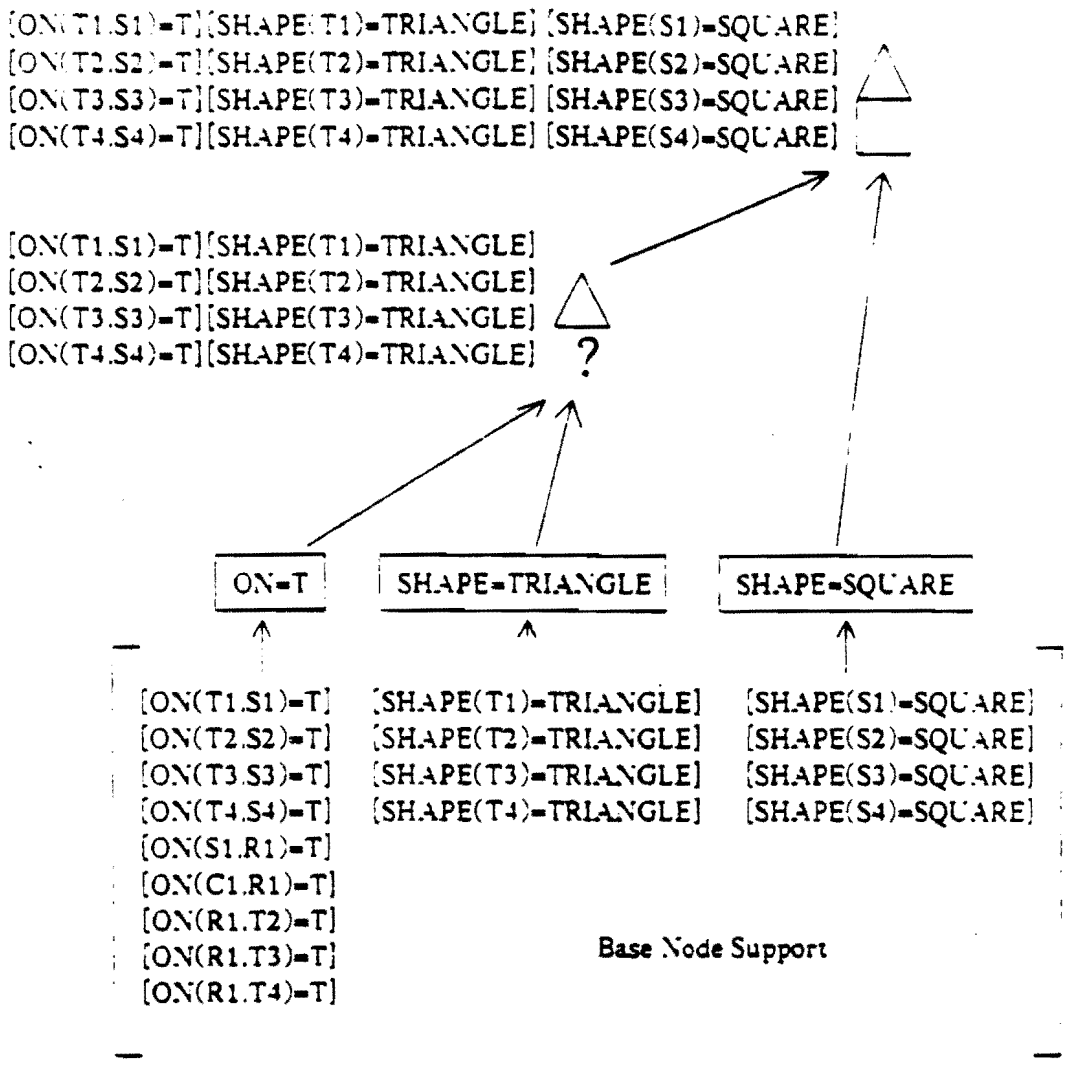
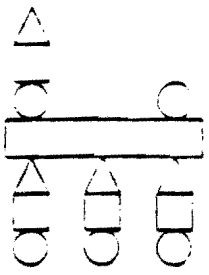


Figure 3.10. Substructure Identification Example

substructure background knowledge, but both specialized and unspecialized substructures discovered by SUBDUE can be retained for use in subsequent substructure discovery tasks.

Input Example:



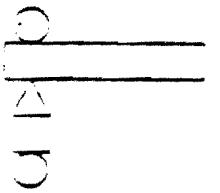
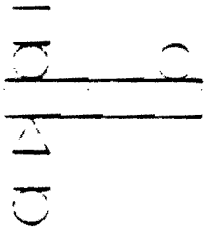
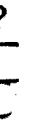
Computation Limit	Three Best Substructures Discovered		
4	Value(S)=8.8	 Value(S)=2.1	 Value(S)=1.7
6	Value(S)=31.2	 Value(S)=9.6	 Value(S)=8.8
10	Value(S)=31.2	 Value(S)=9.6	 Value(S)=9.2
14	Value(S)=31.2	 Value(S)=10.3	 Value(S)=9.6

Figure 4.1. First Example of Experiment 1

relation. Thus, the second substructure in the first row of Figure 4.1 is <(SHAPE X)=SQUARE) (ON(X,Y)=T)>, where the question mark corresponds to the object argument Y. Other objects in the figure are represented by the pictorial equivalent of their *shape* attribute.

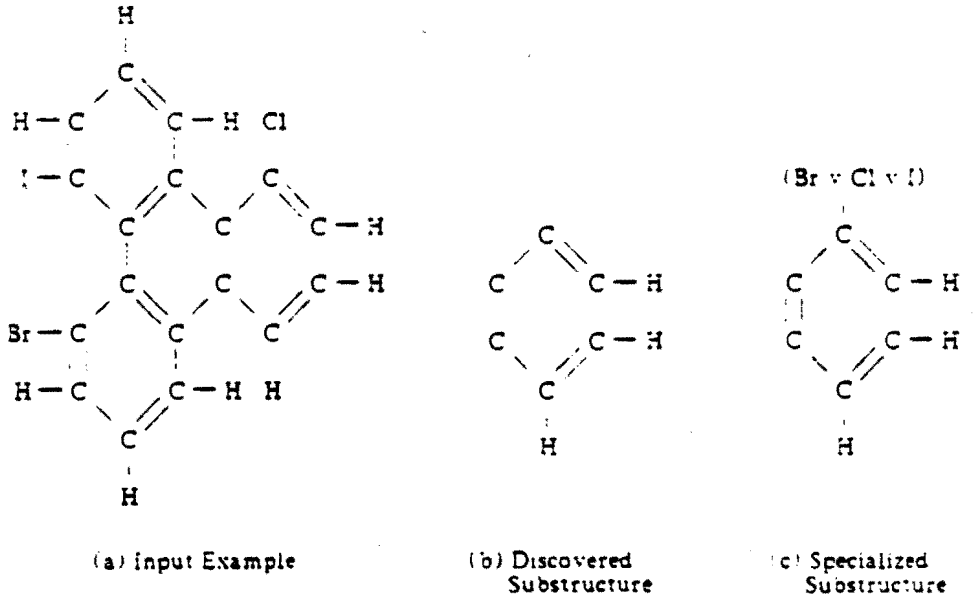
The definition of computational limit given at the beginning of this chapter implies, in the absence of background knowledge, that the best substructure returned by the discovery algorithm cannot contain more relations than the computational limit. If a substructure of a certain size is desired, the computational limit must be set higher than this size. However, the results of Experiment 1 indicate that the limit need not be set much higher than the desired size, if the best overall substructure is indeed of that size. The heuristics appropriately constrain the search to consider substructures along a path of increasing heuristic value towards the best substructure in the input examples.

4.2. Experiment 2: Specialization and Background Knowledge

The ability to retain newly discovered knowledge is beneficial to any learning system. Applying this knowledge to similar tasks can greatly reduce the amount of processing required to perform the task. SUBDUE takes advantage of this idea by specializing discovered substructures and retaining both specialized and unspecialized substructures in the background knowledge. During subsequent discovery tasks, SUBDUE applies the known substructures to the current task. As more examples from similar domains are processed, increasingly complex substructures are discovered in terms of more primitive substructures already known. Eventually, SUBDUE's background knowledge becomes a hierarchical representation of the structure in the domain. Experiment 2 demonstrates SUBDUE's ability to specialize and retain newly discovered substructures and illustrates how these substructures might be applied to a similar discovery task.

The examples for this experiment are drawn from the domain of organic chemistry. Figure 4.3a shows the first example for Experiment 2. The example describes a derivative of the compound Hexabenzobenzene. The best substructure discovered by SUBDUE for this example is shown in Figure 4.3b, and the specialization of this substructure is in Figure 4.3c. Both of these substructures are added to the background knowledge. The resulting background knowledge hierarchy is shown in Figure 4.4. The dashed arrows in Figure 4.4 represent the background knowledge hierarchy defining the discovered substructure of Figure 4.3b.

previously retained substructures in this input example. The previously discovered substructure of Figure 4.3b has six occurrences in the example, and the previously specialized substructure of Figure 4.3c has three occurrences. Each of these substructures is added to the list of base substructures used by the substructure discovery algorithm (see Section 2.8). The previously discovered substructure of Figure 4.3b evaluates to a higher value than the previously specialized substructure; thus, the algorithm begins by considering extensions from the unspecialized substructure. After running the algorithm with a computational limit of 10, SUBDUE produces the substructure in Figure 4.5b as the best discovered substructure. The resulting specialized substructure is shown in Figure 4.5c. Again, both of these substructures are added to the background knowledge. However, SUBDUE takes advantage of the substructures already stored to define the new substructures in terms of the substructures already known. As a result, the background knowledge is extended hierarchically upward to incorporate the new substructures.



(a) Input Example (b) Discovered Substructure (c) Specialized Substructure

Figure 4.5. Second Example for Experiment 2

4.3. Experiment 3: Discovering Classifying Attributes in Multiple Examples

Most machine learning systems assume that the description of the input examples incorporates attributes that are relevant to the learning task. This assumption frequently does not hold, and the best classifying attributes may be those that are synthesized from a combination or a reformulation of the given attributes. A recent approach to conceptual clustering, called goal-oriented conceptual clustering [Stepp86], uses a Goal-Dependency Network (GDN) to suggest relevant attributes on which to focus the attention of the conceptual clustering process.

A GDN directs the conceptual clustering technique implemented in the CLUSTER/CA program [Mogensen87]. In CLUSTER/CA the GDN is provided by the user. However, the user may not always know which attributes or combination of attributes are relevant to a specific problem. In this case, the best substructure discovered by SUBDUE in the given examples can be added to the GDN. The substructure attributes added to the GDN suggest problem-specific features to help focus the conceptual clustering process. Experiment 3 demonstrates how SUBDUE and CLUSTER/CA work together to discover conceptual clusterings based on newly discovered substructure attributes.

Thus far, the operation of SUBDUE has been examined in the context of one input example. SUBDUE operates on multiple input examples in exactly the same manner. SUBDUE always represents the input examples as a graph with single input examples represented as a single connected graph, and multiple input examples represented as a disconnected graph with a connected subgraph component for each example. Because substructures are connected graphs, the substructures discovered in the context of multiple input examples cannot contain structure spanning more than one example.

The examples for this experiment consist of ten trains first introduced by Larson [Larson77] and later used for psychological testing [Medin87]. These same trains are used to demonstrate the operation of CLUSTER/CA [Mogensen87]. The ten trains used in Experiment 3 are shown in Figure 4.7. Cars within a train are connected with an *in-front* relation. Each car is described by the

In CLUSTER/CA the "goodness" of a clustering is measured by a Lexicographical Evaluation Function (LEF) [Michalski80]. The LEF used for this experiment biases CLUSTER/CA toward clusterings with an equal number of examples per cluster, clusterings covering the maximum number of examples and clusterings having the simplest descriptions. Using this LEF and the GDN described in [Mogensen87], the two best clusterings discovered are

Number of cars is "Three" | "Four" | "Five"

Color of engine wheels is "Black" "White"

When the examples are given to SUBDUE, the best substructure found by the heuristic-based substructure discovery module is

<(CAR-LENGTH(OBJECT-0001)=SHORT)(LOAD-NUMBER(OBJECT-0001)=ONE)
(WHEEL-COLOR(OBJECT-0001)=WHITE)>

In other words, the best substructure found is *a short car with white wheels and one load*. By adding this substructure to the original GDN and running CLUSTER/CA again on the same examples, the two best clusterings discovered are

Number of cars is "Three" "Four" "Five"

Number of short cars with white wheels and one load
is "Zero" | "One" | "Two to Four"

The best clustering discovered by CLUSTER/CA is the same as the best clustering discovered without SUBDUE. However, the second best clustering uses the SUBDUE-discovered substructure attribute to cluster the input examples. Thus, according to the LEF, this new clustering is better than the clustering based on the color of the engine wheels. Without the suggestion from SUBDUE, CLUSTER/CA would not have discovered this conceptual clustering.

That CLUSTER/CA was unable to discover the clustering based on the substructure attribute suggested by SUBDUE is mostly due to CLUSTER/CA's bias towards the attributes given in the

In each of these learning paradigms the entire proof tree is considered the macro-operator. Although STRIPS learns subsequences of the plan as macro-operators, the subsequences are chosen arbitrarily. SUBDUE offers a method for discovering "interesting" macro-operators within the structure of the proof tree. Another system that works with the internal structure of a proof tree is the BAGGER system [Shavlik88]. BAGGER *generalizes to N* by finding loops in the proof tree that can be collapsed into one macro-operator representing an iterative instance of the operators within the loop. The PLAND system [Whitehall87] uses a method similar to SUBDUE's to discover macro-operators involving loops and conditionals in observed sequences of plan steps. Section 5.4 discusses similarities and differences between SUBDUE and PLAND.

Experiment 4 shows how SUBDUE can be used to find a macro-operator within the structure of a proof tree. The example for this experiment is drawn from the "blocks world" domain. The operators for this domain are taken from [Nilsson80] and are repeated below:

PICKUP(x)
  Preconditions: ONTABLE(x), CLEAR(x), HANDEMPTY
  Add: HOLDING(x)
  Delete: ONTABLE(x), CLEAR(x), HANDEMPTY

PUTDOWN(x)
  Preconditions: HOLDING(x)
  Add: ONTABLE(x), CLEAR(x), HANDEMPTY
  Delete: HOLDING(x)

STACK(x,y)
  Preconditions: HOLDING(x), CLEAR(y)
  Add: HANDEMPTY, ON(x,y), CLEAR(x)
  Delete: HOLDING(x), CLEAR(y)

UNSTACK(x,y)
  Preconditions: HANDEMPTY, CLEAR(x), ON(x,y)
  Add: HOLDING(x), CLEAR(y)
  Delete: HANDEMPTY, CLEAR(x), ON(x,y)

For this example, suppose the initial world state is as shown in Figure 4.8a, and the desired goal is in Figure 4.8b. The proof tree of operators to achieve the goal is shown in Figure 4.8c. With this proof tree as input, SUBDUE discovers the substructure shown in Figure 4.9. The

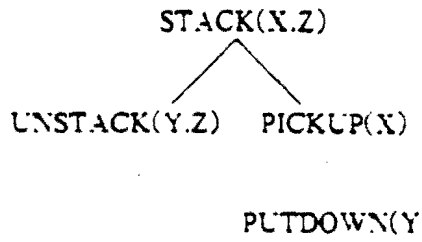


Figure 4.9. Discovered Macro-Operator

system, because the macro-operators may occur in subsequent examples. If the discovered macro-operators are added to SUBDUE's background knowledge, a hierarchy of macro-operators can be constructed. This hierarchy might serve as an initial domain theory for an EBL system.

4.5. Experiment 5: Data Abstraction and Feature Formation

Experiment 5 combines SUBDUE with the INDUCE system [Hoff83] to demonstrate the improvement gained in both processing time and quality of results when the examples contain a large amount of structure. A Common Lisp version of INDUCE was used for this example running on the same Texas Instruments Explorer as the SUBDUE system.

Figure 4.10a shows a pictorial representation of the three positive and three negative examples given to INDUCE. Each of the symbolic benzene rings in the examples of Figure 4.10a corresponds to the detailed description of the atomic structure of the benzene ring, similar to the one shown in the left side of Figure 4.10c. The actual input specification for the six examples contains a total of 178 relations of the form [SINGLE-BOND(C1,C2)=T] or [DOUBLE-BOND(C1,C2)=T]. After 160 seconds of processing time, INDUCE produces the concept shown in Figure 4.10b.

speedup of 3 over INDUCE alone. This experiment demonstrates how the substructures discovered by SUBDUE can improve the results of other learning systems by abstracting over detailed structure in the input and providing new features.

implications of these gestalt theories guided the development of the substructure discovery algorithm.

5.2. Discovering Groups of Objects in Winston's ARCH Program

Winston's ARCH program [Winston75] discovers substructure in order to deepen the hierarchical description of a scene and describe groups of objects as individual concepts. The ARCH program searches for two types of substructure in the blocks world domain. The first type involves a sequence of objects connected by a chain of similar relations. The second type involves a set of objects each having a similar relationship to some "grouping" object. The approach used by the ARCH program begins with a conjecture process that searches for occurrences of the two types of substructure. Next, the revision process excludes from the group those occurrences that fall below a given threshold of the group's average. This section discusses the method by which the ARCH program discovers both types of substructure and how the method compares to that of SUBDUE.

When searching for sequences of objects, the ARCH program considers chains of objects connected by SUPPORTED-BY or IN-FRONT-OF relations. All such chains with three or more objects qualify as a sequence. However, as illustrated in Figure 5.1, not all objects in a sequence

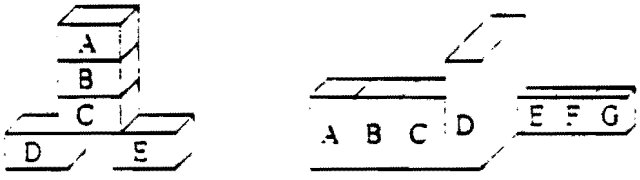


Figure 5.1. Sequence Termination Conditions

A, B, C:
1 SUPPORTED-BY relation to F
2 MARRIES relation to F
3 A-KIND-OF relation to BRICK
4 HAS-PROPERTY-OF relation to MEDIUM

D:
1 SUPPORTED-BY relation to F
2 MARRIES relation to F
3 A-KIND-OF relation to BRICK
4 HAS-PROPERTY-OF relation to SMALL

E:
1 SUPPORTED-BY relation to F
2 MARRIES relation to F
3 A-KIND-OF relation to WEDGE
4 HAS-PROPERTY-OF relation to SMALL

The *common-relationships-list* contains the four relations possessed by more than half the candidates:

Common-Relationships-List:
1 SUPPORTED-BY relation to F
2 MARRIES relation to F
3 A-KIND-OF relation to BRICK
4 HAS-PROPERTY-OF relation to MEDIUM

Next, the procedure measures how typical each candidate is in comparison to the relations in the *common-relationships-list*. The measure is computed as

$$\frac{\text{Number of properties in intersection}}{\text{Number of properties in union}}$$

where the intersection and union are of the candidate's relations list and the *common-relationships-list*. The results of using this measure to compare each candidate are:

A compared to *common-relationships-list*: 4/4 = 1
B compared to *common-relationships-list*: 4/4 = 1
C compared to *common-relationships-list*: 4/4 = 1
D compared to *common-relationships-list*: 3/4 = .75
E compared to *common-relationships-list*: 2/4 = .5

same type. In SUBDUE, the type of an object is an attribute relation, and attribute relations are treated as any other relation during the discovery process. SUBDUE does not constrain the discovered substructures according to the types of the objects in the substructure. Extending the ARCH program to more easily discover substructures with different object types does not seem difficult. Running both the sequence and common relation discovery processes more than once might encourage the ARCH program to build substructure concepts containing objects with different types. However, the new process would still remain dependent upon the blocks world domain.

The final comparison of the two systems involves the representation used to store the discovered substructures. The ARCH program utilizes the semantic network formalism. Here, the substructure node has a TYPICAL-MEMBER link to a general description of the prototype substructure, and each occurrence is linked to the same substructure node with a GROUP-MEMBER. Also, a FORM link notes the substructure type of the node: sequence or common property. In SUBDUE, only the substructure description is retained in the background knowledge. Furthermore, the background knowledge maintains the substructures in a hierarchy that defines complex substructures in terms of previously learned, more primitive substructures. Although the semantic network formalism of the ARCH program can represent hierarchical structures, Winston does not mention this ability explicitly [Winston75]. The substructure representation used by SUBDUE's background knowledge is overly rigid. Eventually, SUBDUE must be able to represent background knowledge other than substructures. For this reason, SUBDUE would benefit from a more general representation such as the semantic network used by the ARCH program.

5.3. Cognitive Optimization in Wolff's SNPR Program

Research toward a comprehensive theory of cognitive development has led Wolff to postulate that optimization is a major underlying goal in building and refining a knowledge structure [Wolff88]. The resulting knowledge structures are optimally efficient for the required tasks. Furthermore, Wolff presents six data compression principles for implementing the optimization

$$CV_e = \frac{f * (S - s)}{S}$$

where f is the frequency of the element in the input text, S is the size of the element and s is the size of the pointer used to replace, or instantiate, the element in the input text. Thus, $f*S-s$ represents the reduction in size of the input text after replacing each occurrence of the element by a pointer to the element. Dividing this term by the cost S of storing the element yields a value that increases as the amount of data compression provided by the element increases. Therefore, SNPR seeks a grammar whose elements maximize CV_e.

SNPR's compression value is similar to SUBDUE's cognitive savings. Recall from Section 3.2.2 that SUBDUE computes the cognitive savings of a substructure as a function of the number of occurrences (frequency) of the substructure and the size of the substructure. If the number of occurrences of the substructure is f, and the size of the substructure is S, then the cognitive savings of the substructure can be expressed as

$$\text{cognitive\_savings} = S * (f - 1) = Sf - S$$

There are two differences between this expression for cognitive savings and the expression for compression value. First, the size of the pointer replacing the substructure occurrences is not considered in the cognitive savings value. Second, the size of the substructure is subtracted from the reduction term in the cognitive savings value; whereas, the size of the element is divided into the reduction term in the compression value. These differences suggest possible future improvements to the cognitive savings measure.

5.4. Substructure Discovery in Whitehall's PLAND System

The PLAND system [Whitehall 87] discovers substructure in an observed sequence of actions. These substructures are termed macro-operators or macrops. PLAND incorporates generalization with different levels of background knowledge to discover three types of macrops: sequences, loops and conditionals. SUBDUE is similar to PLAND in that both systems use the cognitive savings

knowledge rules are typically domain dependent, although PLAND also functions in the absence of this knowledge.

The PLAND system uses the cognitive savings heuristic to evaluate macrop substructures. The cognitive savings used by PLAND is computed as

cognitive_savings = (number of macrop occurrences - 1) * length of macrop

The only difference between this definition and SUBDUE's is the modification in SUBDUE's definition to allow for overlapping substructures. PLAND does not allow macrops in an agenda to overlap. Using the cognitive savings heuristic allows PLAND to select among competing partial macrop agendas and to select complete macrops for instantiation in the input sequence.

Observed Action Sequence:
A B Y X X X Y X X Z Y X X Y X X X X Z

CONTEXT 1

cogsav	macrops
10.0	$M_{1.1} = (X)^*$
11.25	$M_{1.2} = (Y\ M_{1.1}{}^*)^*$
8.5	$M_{1.3} = (M_{1.2}{}^*\ Z)^*$

Instantiated Action Sequence:
A B $M_{1.2}$ Z $M_{1.2}$ Z

CONTEXT 2

cogsav	macrops
2.0	$M_{2.1} = (M_{1.2}{}^*\ Z)^*$

Instantiated Action Sequence:
A B $M_{2.1}$

All interesting macrops discovered.
End.

Figure 5.3. PLAND Example

CHAPTER 6

CONCLUSION

Complex hierarchies of substructure are ubiquitous in the real world and, as the experiments of Chapter 4 demonstrate, in less realistic domains as well. In order for an intelligent entity to learn about such an environment, the entity must abstract over uninteresting detail and discover substructure concepts that allow an efficient and useful representation of the environment. This thesis presents a computational method for discovering substructure in examples from structured domains. Section 6.1 summarizes the substructure discovery theory and methodology discussed in this thesis, and Section 6.2 discusses directions for future work in substructure discovery.

6.1. Summary

The purpose of substructure discovery is to identify interesting and repetitive structural concepts within a structural representation of the environment. Such a discovery system is motivated by the needs to abstract over detail, to maintain a hierarchical description of the environment and to take advantage of substructure within other knowledge-bases to reduce storage requirements and retrieval times. This thesis presents the important processes and methodological alternatives involved in a computational method for discovering substructure.

A substructure discovery system must generate alternative substructures to be considered by the discovery process. Four methods of substructure generation are discussed: minimal expansion, combination expansion, minimal disconnection, and cut disconnection. These methods differ along two dimensions. The expansion methods generate larger substructures by adding structure to smaller ones, while the disconnection methods generate smaller substructures by removing structure from larger ones. The minimal methods add or remove only small amounts of structure

The SUBDUE system is an implementation of the processes involved in substructure discovery. SUBDUE contains three modules: the heuristic-based substructure discovery module, the substructure specialization module, and the substructure background knowledge module. The heuristic-based discovery module utilizes the substructure generation and selection processes and optional background knowledge to identify interesting substructures in the given input examples. The specialization module modifies the substructure to apply in a more constrained environment. The background knowledge module retains both the discovered and specialized substructures in a hierarchy and suggests to the heuristic-based discovery module which of the known substructures apply to the current set of input examples. Experiments with SUBDUE demonstrate the utility of the guidance provided by the heuristics to direct the search towards more interesting substructures and the possible applications of the SUBDUE substructure discovery system in a variety of domains.

The ability to discover substructure in a structured environment is important to the task of learning about the environment. For this reason, substructure discovery represents an important class of problems in the area of machine learning. Operation in real-world domains demands of learning programs the ability to abstract over unnecessary detail by identifying interesting patterns in the environment. Empirical evidence from the SUBDUE system demonstrates the applicability of the concepts presented in this thesis to the task of discovering substructure in examples. Expanding upon these underlying concepts may provide more insight into the development of a substructure discovery system capable of interacting with a real-world environment.

6.2. Future Research

Several extensions and applications of the substructure discovery method and the SUBDUE system require further investigation. Interesting extensions to the substructure discovery method include the incorporation of new heuristics and background knowledge into the discovery process and the ability to discover other types of substructure. Improved methods of substructure instantiation are needed to better capture the abstraction provided by a substructure. The

suggest heuristics to apply during the discovery process according to the current domain and the previous success of similar rule applications.

In addition to increasing the intelligence of the substructure discovery process, further research is needed to increase the scope of the process. Currently, there are several types of substructures that cannot be discovered. For instance, the current discovery algorithm cannot discover recursive substructures, substructures with negation (e.g., <[ON(X,Y)=T] [COLOR(X)≠RED]>), or substructures with constraints on the type of structure to which they can be connected. The ability to discover these types of substructures will require major modifications in both the background knowledge architecture and the operations performed by the discovery algorithm.

Improvements in both the scope and the performance of the substructure discovery process may arise from a better method of substructure instantiation. Current methods do not satisfactorily represent the full amount of abstraction provided by a substructure. Instantiation by a single object retains no information about the way in which the substructure was connected with the rest of the input example. Contrastingly, instantiation by a single relation preserves external connection information for each object in the substructure. An instantiation method that combines both approaches may be more appropriate. Better substructure instantiation methods would allow abstraction to take place automatically within the discovery process. The discovery process could work at several levels of abstraction by instantiating intermediate substructures into the current set of input examples. In this way several hierarchically defined substructures may be discovered with a single application of the discovery algorithm.

Many of the suggested improvements to the substructure discovery process represent extensive modifications to the current methodology. However, most of the improvements involve the use of alternative forms of background knowledge. The proposed extensions to the discovery process may be simplified by appropriate extensions to the substructure background knowledge.

each node of the substructure hierarchy could be used only after identifying the exact substructure to which this knowledge is attached. Developing methods for incorporating "fuzziness" in the match would improve the flexibility of the background knowledge.

One of the major limiting factors in SUBDUE's performance is the sparse amount of knowledge applicable during the discovery process. The proposed extensions to the background knowledge will significantly improve SUBDUE's ability to learn substructure concepts.

6.2.3. Applications

Several applications appear promising for the SUBDUE substructure discovery system. SUBDUE might be used to detect texture primitives and subimages in a visual image, organize and compress large databases, or discover interesting features in the input to other machine learning systems.

One of the goals of visual processing is to construct a high-level interpretation of an image composed only of pixels. In this context, SUBDUE could be used to detect repetitive patterns in the regions of a visual image. Instantiating the patterns to abstract over their detailed representation simplifies the image and allows other vision processing systems to work at a higher level of abstraction.

The need to access information from large databases has become commonplace in most computing environments. Unfortunately, as the amount of knowledge in a database increases, so do the storage requirements and access times. Using the database as input, SUBDUE may discover repetitive substructure in the data. The substructures can be used to compress the database and impose a hierarchical representation. This reorganization reduces the storage requirements of the database and decreases retrieval time for queries referencing data with similar substructure.

An important future application of SUBDUE involves integration with other machine learning systems. Most current systems are unable to process the large number of features available in a real-world environment. SUBDUE could preprocess the input features and abstract over

REFERENCES

[DeJong86] G. F. DeJong and R. J. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning 1*, 2 (April 1986), pp. 145-176. (Also appears as Technical Report UILU-ENG-86-2208, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[de Kleer86] J. de Kleer, "An Assumption-Based Truth Maintenance System," *Artificial Intelligence 28*, (1986), pp. 127-162.

[Doyle79] J. Doyle, "A Truth Maintenance System," *Artificial Intelligence 12*, 3 (1979), pp. 231-272.

[Fikes72] R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence 3*, 4 (1972), pp. 251-288.

[Fu87] K. S. Fu, R. C. Gonzalez and C. S. Lee, *Robotics: Control, Sensing, Vision and Intelligence*, McGraw-Hill, New York, NY, 1987.

[Hoff83] W. A. Hoff, R. S. Michalski and R. E. Stepp, "INDUCE 3: A Program for Learning Structural Descriptions from Examples," Technical Report UIUCDCS-F-83-904, Department of Computer Science, University of Illinois, Urbana, IL, 1983.

[Kohler47] W. Kohler, *Gestalt Psychology*, Liveright Publishing Corporation, New York, NY, 1947.

[Laird86] J. Laird, P. Rosenbloom and A. Newell, "Chunking in Soar: The Anatomy of a General Learning Mechanism," *Machine Learning 1*, 1 (1986), pp. 11-46.

[Larson77] J. Larson, "Inductive Inference in the Variable-Valued Predicate Logic System VL21: Methodology and Computer Implementation," Ph. D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1977.

[Medin87] D. L. Medin, W. D. Wattenmaker and R. S. Michalski, "Constraints and Preferences in Inductive Learning: An Experimental Study of Human and Machine Performance," *Cognitive Science 11*, 3 (1987), pp. 299-239.

[Michalski80] R. S. Michalski, "Pattern Recognition as Rule-Guided Inductive Inference," *IEEE Transactions on Pattern Analysis and Machine Intelligence 2*, 4 (July 1980), pp. 349-361.

[Michalski83a] R. S. Michalski, "A Theory and Methodology of Inductive Learning," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983, pp. 83-134.

[Michalski83b] R. S. Michalski and R. E. Stepp, "Learning from Observation: Conceptual Clustering," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983, pp. 331-363.

[Minton87] S. Minton, J. G. Carbonell, O. Etzioni, C. A. Knoblock and D. R. Kuokka, "Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 122-133.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47-80.

[Wolff88] J. G. Wolff. "Cognitive Development As Optimization." In *Computational Models of Learning*. L. Bolc (ed.). Springer Verlag, Heidelberg, 1988.

[Zahn71] C. T. Zahn. "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters." *IEEE Transactions on Computers* C-20 1 January 1971 pp. 68-86.

:coverage — A boolean value indicating whether or not SUBDUE should apply the coverage heuristic to the cognitive savings, while evaluating a substructure. Default is T.

:use-bk — A boolean value indicating whether or not SUBDUE should consult the background knowledge module for substructures applying to the current set of input examples. Default is NIL.

:discover — A boolean value indicating whether or not SUBDUE should perform the substructure discovery algorithm on the current set of input examples. Default is T.

:specialize — A boolean value indicating whether or not SUBDUE should specialize the best substructure found by the substructure discovery module. Default is NIL.

:inc-bk — A boolean value indicating whether or not SUBDUE should incrementally add the best discovered substructure and the specialized substructure, if requested via the previous keyword, to the background knowledge. Default is NIL.

:trace — A boolean flag for toggling the output of trace information during SUBDUE's execution. Default is T.

The result of a call to the *subdue* function is a list of the substructures discovered in order from best to worst. Each substructure is accompanied by the occurrences of the substructure in the input examples. The substructures in the subsequent output traces appear in the following form:

(Substructure#*n* :value *v* *relations*)
WITH OCCURRENCES:
(*relations*)
(*relations*)
:

The substructure number *n* indicates that this substructure was the *nth* substructure discovered. The value *v* is the result of the value(S) computation from Section 3.2.2 for this substructure. *Relations* is the list of relations defining the substructure or occurrence.

In all the experiments, the default values are used for the *connectivity*, *compactness*, *coverage*, *discover* and *trace* keywords. Also, to conserve space, only the three best substructures discovered by SUBDUE are shown in the output traces, regardless of the computational limit.

```
(on(s4,c4)=t))
(on(c1,c1)=t))
(on(c5,c1)=t))
   .
   .
End trace.
```

A.1.3. Output for First Example of Experiment 1: Limit = 6

```
> (subdue :limit 6)

Begin trace...

Parameters:     limit = 6          connectivity = t     compactness = t     coverage = t
                use-bk = nil       discover = t         specialize = nil    inc-bk = nil

Running substructure discovery...

Discovered the following 6 substructures in 0.23333333 seconds:

Substructure#6 :value 31.219513 ((shape(object-0008)=triangle)(on(object-0008,object-0001)=t)
  (shape(object-0001)=square)(shape(object-0002)=circle)(on(object-0001,object-0002)=t))
WITH OCCURRENCES:
((shape(t1)=triangle)(on(t1,s1)=t)(shape(s1)=square)(shape(c1)=circle)(on(s1,c1)=t))
((shape(t2)=triangle)(on(t2,s2)=t)(shape(s2)=square)(shape(c2)=circle)(on(s2,c2)=t))
((shape(t3)=triangle)(on(t3,s3)=t)(shape(s3)=square)(shape(c3)=circle)(on(s3,c3)=t))
((shape(t4)=triangle)(on(t4,s4)=t)(shape(s4)=square)(shape(c4)=circle)(on(s4,c4)=t))

Substructure#5 :value 9.560976 ((on(object-0008,object-0001)=t)(shape(object-0001)=square)(shape(object-0002)=circle)
  (on(object-0001,object-0002)=t))
WITH OCCURRENCES:
((on(t1,s1)=t)(shape(s1)=square)(shape(c1)=circle)(on(s1,c1)=t))
((on(t2,s2)=t)(shape(s2)=square)(shape(c2)=circle)(on(s2,c2)=t))
((on(t3,s3)=t)(shape(s3)=square)(shape(c3)=circle)(on(s3,c3)=t))
((on(t4,s4)=t)(shape(s4)=square)(shape(c4)=circle)(on(s4,c4)=t))

Substructure#4 :value 8.780488 ((shape(object-0001)=square)(shape(object-0002)=circle)(on(object-0001,object-0002)=t))
WITH OCCURRENCES:
((shape(s1)=square)(shape(c1)=circle)(on(s1,c1)=t))
((shape(s2)=square)(shape(c2)=circle)(on(s2,c2)=t))
((shape(s3)=square)(shape(c3)=circle)(on(s3,c3)=t))
((shape(s4)=square)(shape(c4)=circle)(on(s4,c4)=t))
   .
   .
End trace.
```

A.1.4. Output for First Example of Experiment 1: Limit = 10

```
> (subdue :limit 10)

Begin trace...

Parameters:     limit = 10         connectivity = t     compactness = t     coverage = t
                use-bk = nil       discover = t         specialize = nil    inc-bk = nil

Running substructure discovery...

Discovered the following 10 substructures in 0.6666667 seconds:

Substructure#6 :value 31.219513 ((shape(object-0008)=triangle)(on(object-0008,object-0001)=t)
  (shape(object-0001)=square)(shape(object-0002)=circle)(on(object-0001,object-0002)=t))
WITH OCCURRENCES:
((shape(t1)=triangle)(on(t1,s1)=t)(shape(s1)=square)(shape(c1)=circle)(on(s1,c1)=t))
```

```
Substructure#5 (value 9.5609^6) (on object-0008 object-0001 =t) (shape object-0001 =square) (shape object-0002 =circle)
 (on object-0001 object-0002)=t))
WITH OCCURRENCES:
 ((on t1 s1 =t) (shape s1 =square) (shape c1 =circle) (on s1 c1 =t))
 ((on t2 s2 =t) (shape s2 =square) (shape c2 =circle) (on s2 c2 =t))
 ((on t3 s3 =t) (shape s3 =square) (shape c3 =circle) (on s3 c3 =t))
 ((on t4 s4 =t) (shape s4 =square) (shape c4 =circle) (on s4 c4 =t))
 .
 .

End trace.
```

A.1.6. Input for Second Example of Experiment 1

```
(DefExample
 '(n01 n02 n03 n04 n05 n06 n07 n08 n09 n10)
   (connected nil)
   ((connected (n01 n06) t) (connected (n01 n02) t) (connected (n02 n07) t) (connected (n02 n03) t) (connected (n03 n08) t)
   (connected (n03 n04) t) (connected (n04 n09) t) (connected (n04 n05) t) (connected (n05 n10) t) (connected (n06 n07) t)
   (connected (n07 n08) t) (connected (n08 n09) t) (connected (n09 n10) t))))
```

A.1.7. Output for Second Example of Experiment 1: Limit = 4

```
> (subdue :limit 4)

Begin trace...

Parameters:     limit = 4          connectivity = t       compactness = t       coverage = t
                use-ok = nil       discover = t           specialize = nil      ine-ok = nil

Running substructure discovery...

Discovered the following 4 substructures in 0.98333335 seconds:

(Substructure#2 (value 2.9545455) (connected(object-0001 object-0002)=t))
WITH OCCURRENCES:
 ((connected(n01 n06)=t))
 ((connected(n01 n02)=t))
 ((connected(n02 n07)=t))
 ((connected(n02 n03)=t))
 ((connected(n03 n08)=t))
 ((connected(n03 n04)=t))
 ((connected(n04 n09)=t))
 ((connected(n04 n05)=t))
 ((connected(n05 n10)=t))
 ((connected(n06 n07)=t))
 ((connected(n07 n08 =t))
 ((connected(n08 n09)=t))
 ((connected(n09 n10)=t))

(Substructure#3 (value 2.8085105) (connected(object-0002 object-0003)=t) (connected(object-0001 object-0002 =t)
WITH OCCURRENCES:
 ((connected(n01 n02)=t) (connected(n01 n06)=t))
 ((connected(n06 n07)=t) (connected(n01 n06)=t))
 ((connected(n02 n07)=t) (connected(n01 n02)=t))
 ((connected(n02 n03)=t) (connected(n01 n02)=t))
 ((connected(n02 n03)=t) (connected(n02 n07)=t))
 ((connected(n06 n07)=t) (connected(n02 n07)=t))
 ((connected(n07 n08)=t) (connected(n02 n07)=t))
 ((connected(n03 n08 =t) (connected(n02 n03)=t))
 ((connected(n03 n04 =t) (connected(n02 n03)=t))
 ((connected(n03 n04 =t) (connected(n03 n08 =t))
 ((connected(n07 n08)=t) (connected(n03 n08 =t))
 ((connected(n08 n09)=t) (connected(n03 n08 =t))
```

```
Running substructure discovery...

Discovered the following 6 substructures in 3.2 seconds:

(Substructure#5 :value 5.0 (connected(object-0001,object-0004)=t)(connected(object-0003,object-0004)=t
 (connected(object-0002,object-0003)=t)(connected(object-0001,object-0002)=t))
WITH OCCURRENCES:
((connected(n02,n07)=t)(connected(n06,n07)=t)(connected(n01,n02)=t)(connected(n01,n06)=t))
((connected(n03,n08)=t)(connected(n07,n08)=t)(connected(n02,n03)=t)(connected(n02,n07)=t))
((connected(n04,n09)=t)(connected(n08,n09)=t)(connected(n03,n04)=t)(connected(n03,n08)=t))
((connected(n05,n10)=t)(connected(n09,n10)=t)(connected(n04,n05)=t)(connected(n04,n09)=t))

(Substructure#6 :value 3.25 ((connected(object-0008,object-0003)=t)(connected(object-0001,object-0004)=t
 (connected(object-0003,object-0004)=t)(connected(object-0002,object-0003)=t)(connected(object-0001,object-0002)=t)))
WITH OCCURRENCES:
((connected(n02,n03)=t)(connected(n02,n07)=t)(connected(n06,n07)=t)(connected(n01,n02)=t)(connected(n01,n06)=t))
((connected(n07,n08)=t)(connected(n02,n07)=t)(connected(n06,n07)=t)(connected(n01,n02)=t)(connected(n01,n06)=t))
((connected(n01,n02)=t)(connected(n03,n08)=t)(connected(n07,n08)=t)(connected(n02,n03)=t)(connected(n02,n07)=t))
((connected(n06,n07)=t)(connected(n03,n08)=t)(connected(n07,n08)=t)(connected(n02,n03)=t)(connected(n02,n07)=t))
((connected(n03,n04)=t)(connected(n03,n08)=t)(connected(n07,n08)=t)(connected(n02,n03)=t)(connected(n02,n07)=t))
((connected(n08,n09)=t)(connected(n03,n08)=t)(connected(n07,n08)=t)(connected(n02,n03)=t)(connected(n02,n07)=t))
((connected(n02,n03)=t)(connected(n04,n09)=t)(connected(n08,n09)=t)(connected(n03,n04)=t)(connected(n03,n08)=t))
((connected(n07,n08)=t)(connected(n04,n09)=t)(connected(n08,n09)=t)(connected(n03,n04)=t)(connected(n03,n08)=t))
((connected(n04,n05)=t)(connected(n04,n09)=t)(connected(n08,n09)=t)(connected(n03,n04)=t)(connected(n03,n08)=t))
((connected(n09,n10)=t)(connected(n04,n09)=t)(connected(n08,n09)=t)(connected(n03,n04)=t)(connected(n03,n08)=t))
((connected(n03,n04)=t)(connected(n05,n10)=t)(connected(n09,n10)=t)(connected(n04,n05)=t)(connected(n04,n09)=t))
((connected(n08,n09)=t)(connected(n05,n10)=t)(connected(n09,n10)=t)(connected(n04,n05)=t)(connected(n04,n09)=t))

(Substructure#2 :value 2.9545455 ((connected(object-0001,object-0002)=t)))
WITH OCCURRENCES:
((connected(n01,n06)=t))
((connected(n01,n02)=t))
((connected(n02,n07)=t))
((connected(n02,n03)=t))
((connected(n03,n08)=t))
((connected(n03,n04)=t))
((connected(n04,n09)=t))
((connected(n04,n05)=t))
((connected(n05,n10)=t))
((connected(n06,n07)=t))
((connected(n07,n08)=t))
((connected(n08,n09)=t))
((connected(n09,n10)=t))
  .
  .
  .
End trace.
```

A.1.9. Output for Second Example of Experiment 1: Limit = 10

```
> (subdue :limit 10)

Begin trace...

Parameters:     limit = 10          connectivity = t        compactness = t         coverage = t
                use-bk = nil        discover = t            specialize = nil        inc-bk = nil

Running substructure discovery...

Discovered the following 10 substructures in 30.45 seconds:

(Substructure#5 :value 5.0 (connected(object-0001,object-0004)=t)(connected(object-0003,object-0004)=t
 (connected(object-0002,object-0003)=t)(connected(object-0001,object-0002)=t))
WITH OCCURRENCES:
((connected(n02,n07)=t)(connected(n06,n07)=t)(connected(n01,n02)=t)(connected(n01,n06)=t))
((connected(n03,n08)=t)(connected(n07,n08)=t)(connected(n02,n03)=t)(connected(n02,n07)=t))
```

```
Substructure#6, value 3.25, (connected(object-0008,object-0003)=t, (connected(object-0001,object-0004)=t,
 (connected(object-0003,object-0004)=t) (connected(object-0002,object-0003)=t, (connected(object-0001,object-0002)=t
WITH OCCURRENCES:
 (connected(n02,n03)=t) (connected(n02,n07)=t) (connected(n06,n07)=t) (connected(n01,n02)=t, (connected(n01,n06)=t,
 (connected(n07,n08)=t) (connected(n02,n07)=t) (connected(n06,n07)=t) (connected(n01,n02)=t, (connected(n01,n06)=t,
 (connected(n01,n02)=t) (connected(n03,n08)=t) (connected(n07,n08)=t) (connected(n02,n03)=t) (connected(n02,n07)=t,
 (connected(n06,n07)=t) (connected(n03,n08)=t) (connected(n07,n08)=t) (connected(n02,n03)=t) (connected(n02,n07)=t,
 (connected(n03,n04)=t) (connected(n03,n08)=t) (connected(n07,n08)=t) (connected(n02,n03)=t) (connected(n02,n07)=t,
 (connected(n08,n09)=t) (connected(n03,n08)=t) (connected(n07,n08)=t) (connected(n02,n03)=t) (connected(n02,n07)=t,
 (connected(n02,n03)=t) (connected(n04,n09)=t) (connected(n08,n09)=t) (connected(n03,n04)=t) (connected(n03,n08)=t,
 (connected(n07,n08)=t) (connected(n04,n09)=t) (connected(n08,n09)=t) (connected(n03,n04)=t, (connected(n03,n08)=t,
 (connected(n04,n05)=t) (connected(n04,n09)=t) (connected(n08,n09)=t) (connected(n03,n04)=t, (connected(n03,n08)=t,
 (connected(n09,n10)=t) (connected(n04,n09)=t) (connected(n08,n09)=t) (connected(n03,n04)=t, (connected(n03,n08)=t,
 (connected(n03,n04)=t) (connected(n05,n10)=t) (connected(n09,n10)=t) (connected(n04,n05)=t, (connected(n04,n09)=t,
 (connected(n08,n09)=t) (connected(n05,n10)=t) (connected(n09,n10)=t) (connected(n04,n05)=t) (connected(n04,n09)=t))

End trace.
```

A.2. Experiment 2

Experiment 2 illustrates the use of SUBDUE's substructure specialization module and substructure background knowledge module. Two examples from the chemical domain are presented. The resulting output shows the performance improvement obtained by applying previously discovered substructures to subsequent discovery tasks.

A.2.1. Input for First Example of Experiment 2

```
Def:Example
  n1 n2 h3 n4 h5 n6 cl1 cl2 br1 br2 i1 i2 c01 c02 c03 c04 c05 c06 c07 c08 c09 c10 c11 c12 c13 c14 c15 c16 c17 c18 c19 c20
  c21 c22 c23 c24)
  (single-bond nil) (double-bond nil) (atom-type nil))
   atom-type (n1 nydrogen) (atom-type (h2) hydrogen) (atom-type (h3) hydrogen) (atom-type (h4) hydrogen)
   atom-type (h5) hydrogen) (atom-type (h6) hydrogen) (atom-type (cl1) chlorine) (atom-type (cl2) chlorine)
   atom-type (br1) bromine (atom-type (br2) bromine) (atom-type (i1) iodine) atom-type (i2) iodine
   atom-type (c01) carbon) (atom-type (c02) carbon) (atom-type (c03) carbon) (atom-type (c04) carbon)
   atom-type (c05) carbon (atom-type (c06) carbon) (atom-type (c07) carbon) (atom-type (c08) carbon)
   atom-type (c09) carbon) (atom-type (c10) carbon) (atom-type (c11) carbon) (atom-type (c12) carbon)
   atom-type (c13) carbon) (atom-type (c14) carbon) (atom-type (c15) carbon) (atom-type (c16) carbon)
   atom-type (c17) carbon) (atom-type (c18) carbon) (atom-type (c19) carbon) (atom-type (c20) carbon)
   atom-type (c21) carbon) (atom-type (c22) carbon) (atom-type (c23) carbon) (atom-type (c24) carbon)
   single-bond (c01 i1) t) (single-bond (c02 cl1) t) (single-bond (c05 h1) t) (single-bond (c12 br2) t)
   single-bond (c16 h2) t) (single-bond (c22 h3) t) (single-bond (c24 i2) t) (single-bond (c23 cl2) t)
   single-bond (c20 h4) t) (single-bond (c13 br1) t) (single-bond (c09 h5) t) (single-bond (c03 h6) t)
   single-bond (c01 c04) t) (single-bond (c02 c04) t) (single-bond (c03 c06) t) (single-bond (c05 c08) t)
   single-bond (c06 c09) t) (single-bond (c07 c10) t) (single-bond (c07 c11) t) (single-bond (c08 c12) t)
   single-bond (c10 c14) t) (single-bond (c11 c15) t) (single-bond (c13 c17) t) (single-bond (c14 c18) t)
   single-bond (c15 c19) t) (single-bond (c16 c19) t) (single-bond (c17 c20) t) (single-bond (c19 c22) t)
   single-bond (c21 c23) t) (single-bond (c21 c24) t)) (double-bond (c01 c03) t) (double-bond (c02 c05) t)
   double-bond (c04 c07) t) (double-bond (c06 c10) t) (double-bond (c08 c11) t) (double-bond (c09 c13) t)
   double-bond (c12 c16) t) (double-bond (c14 c17) t) (double-bond (c15 c19) t) (double-bond (c18 c21) t)
   double-bond (c20 c23) t) (double-bond (c22 c24) t)))
```

single-bond(c16,c19)=t] [atom-type(c16)=carbon]
atom-type(c21)=carbon] [double-bond(c18,c21)=t] [atom-type(c18)=carbon] [single-bond(c14,c18)=t
single-bond(c21,c23)=t] [atom-type(h4)=hydrogen] [atom-type(c23)=carbon] [double-bond(c20,c23)=t
atom-type(c14)=carbon] [double-bond(c14,c17)=t] [single-bond(c20,h4)=t] [atom-type(c20)=carbon
single-bond(c17,c20)=t] [atom-type(c17)=carbon]
atom-type(c21)=carbon] [double-bond(c19,c21)=t] [atom-type(c19)=carbon] [single-bond(c15,c19)=t
single-bond(c21,c24)=t] [atom-type(h3)=hydrogen] [atom-type(c24)=carbon] [double-bond(c22,c24)=t
atom-type(c15)=carbon] [double-bond(c15,c19)=t] [single-bond(c22,h3)=t] [atom-type(c22)=carbon]
single-bond(c19,c22)=t] [atom-type(c19)=carbon])

(Substructure#38 :value 22.730703 ([single-bond(object-0058,object-0195)=t] [double-bond(object-0176,object-0171)=t]
atom-type(object-0176)=carbon] [single-bond(object-0013,object-0176)=t] [single-bond(object-0171,object-0058)=t]
atom-type(object-0011)=hydrogen] [atom-type(object-0058)=carbon] [double-bond(object-0058,object-0005)=t]
atom-type(object-0013)=carbon] [double-bond(object-0013,object-0001)=t] [single-bond(object-0005,object-0011)=t]
atom-type(object-0005)=carbon] [single-bond(object-0005,object-0001)=t] [atom-type(object-0001)=carbon]))

WITH OCCURRENCES:

([single-bond(c01,i1)=t] [double-bond(c04,c07)=t] [atom-type(c07)=carbon] [single-bond(c07,c10)=t]
single-bond(c01,c04)=t] [atom-type(h6)=hydrogen] [atom-type(c01)=carbon] [double-bond(c01,c03)=t]
atom-type(c10)=carbon] [double-bond(c06,c10)=t] [single-bond(c03,h6)=t] [atom-type(c06)=carbon]
single-bond(c03,c06)=t] [atom-type(c03)=carbon])

([single-bond(c02,c11)=t] [double-bond(c04,c07)=t] [atom-type(c07)=carbon] [single-bond(c07,c11)=t]
single-bond(c02,c04)=t] [atom-type(h1)=hydrogen] [atom-type(c02)=carbon] [double-bond(c02,c05)=t]
atom-type(c11)=carbon] [double-bond(c08,c11)=t] [single-bond(c05,h1)=t] [atom-type(c08)=carbon]
single-bond(c05,c08)=t] [atom-type(c05)=carbon])

([single-bond(c13,br1)=t] [double-bond(c14,c17)=t] [atom-type(c14)=carbon] [single-bond(c10,c14)=t]
single-bond(c13,c17)=t] [atom-type(h5)=hydrogen] [atom-type(c13)=carbon] [double-bond(c09,c13)=t]
atom-type(c10)=carbon] [double-bond(c06,c10)=t] [single-bond(c09,h5)=t] [atom-type(c09)=carbon]
single-bond(c06,c09)=t] [atom-type(c06)=carbon])

([single-bond(c12,br2)=t] [double-bond(c08,c11)=t] [atom-type(c11)=carbon] [single-bond(c11,c15)=t]
single-bond(c08,c12)=t] [atom-type(h2)=hydrogen] [atom-type(c12)=carbon] [double-bond(c12,c16)=t]
atom-type(c15)=carbon] [double-bond(c15,c19)=t] [single-bond(c16,h2)=t] [atom-type(c19)=carbon]
single-bond(c16,c19)=t] [atom-type(c16)=carbon])

([single-bond(c23,c12)=t] [double-bond(c18,c21)=t] [atom-type(c18)=carbon] [single-bond(c14,c18)=t]
single-bond(c21,c23)=t] [atom-type(h4)=hydrogen] [atom-type(c23)=carbon] [double-bond(c20,c23)=t]
atom-type(c14)=carbon] [double-bond(c14,c17)=t] [single-bond(c20,h4)=t] [atom-type(c20)=carbon]
single-bond(c17,c20)=t] [atom-type(c17)=carbon])

([single-bond(c24,i2)=t] [double-bond(c18,c21)=t] [atom-type(c18)=carbon] [single-bond(c15,c18)=t]
single-bond(c21,c24)=t] [atom-type(h3)=hydrogen] [atom-type(c24)=carbon] [double-bond(c22,c24)=t]
atom-type(c15)=carbon] [double-bond(c15,c19)=t] [single-bond(c22,h3)=t] [atom-type(c22)=carbon]
single-bond(c19,c22)=t] [atom-type(c19)=carbon])

:

Specializing substructure...

Specialized the substructure:

Substructure#37 :value 30.197369 ([single-bond(object-0058,object-0200)=t] [atom-type(object-0171)=carbon]
double-bond(object-0176,object-0171)=t] [atom-type(object-0176)=carbon] [single-bond(object-0013,object-0176)=t]
single-bond(object-0171,object-0058)=t] [atom-type(object-0011)=hydrogen] [atom-type(object-0058)=carbon]
double-bond(object-0058,object-0005)=t] [atom-type(object-0013)=carbon] [double-bond(object-0013,object-0001)=t]
single-bond(object-0005,object-0011)=t] [atom-type(object-0005)=carbon] [single-bond(object-0005,object-0001)=t]
atom-type(object-0001)=carbon])

to the following substructures:

Substructure#0 :value 1/2 ([atom-type(object-0200)=bromine,chlorine,iodine] [single-bond(object-0058,object-0200)=t]
atom-type(object-0171)=carbon] [double-bond(object-0176,object-0171)=t] [atom-type(object-0176)=carbon]
single-bond(object-0013,object-0176)=t] [single-bond(object-0171,object-0058)=t] [atom-type(object-0011)=hydrogen]
atom-type(object-0058)=carbon] [double-bond(object-0058,object-0005)=t] [atom-type(object-0013)=carbon]
double-bond(object-0013,object-0001)=t] [single-bond(object-0005,object-0011)=t] [atom-type(object-0005)=carbon]
single-bond(object-0005,object-0001)=t] [atom-type(object-0001)=carbon])

Adding substructures to 3K...

Added the following substructures to 3K:

```
(infront car1 car2) t) (infront (car2 car3) t) (infront (car3 car4) t)))

(DefExample                    ;Train H
 ((car1 car2 car3)
  ((car-shape nil) (wheel-color nil) (car-length nil) (load-shape nil) (load-number nil) (infront t))
  ((car-shape (car1) engine) (wheel-color (car1) white) (car-shape (car2) closed-rectangle) (car-length (car2) long)
   (load-shape (car2) rectangle) (load-number (car2) one) (wheel-color (car2) white) (car-shape (car3) u-shape)
   (car-length (car3) short) (load-shape (car3) circle) (load-number (car3) one) (wheel-color (car3) white)
   (infront (car1 car2) t) (infront (car2 car3) t))))

(DefExample                    ;Train I
 ((car1 car2 car3 car4 car5)
  ((car-shape nil) (wheel-color nil) (car-length nil) (load-shape nil) (load-number nil) (infront t))
  ((car-shape (car1) engine) (wheel-color (car1) white) (car-shape (car2) open-trapezoid) (car-length (car2) short)
   (load-shape (car2) circle) (load-number (car2) one) (wheel-color (car2) white) (car-shape (car3) jagged)
   (car-length (car3) long) (load-shape (car3) rectangle) (load-number (car3) one) (wheel-color (car3) white)
   (car-shape (car4) open-rectangle) (car-length (car4) short) (load-shape (car4) rectangle) (load-number (car4) one)
   (wheel-color (car4) white) (car-shape (car5) open-trapezoid) (car-length (car5) short) (load-shape (car5) circle)
   (load-number (car5) one) (wheel-color (car5) white) (infront (car1 car2) t) (infront (car2 car3) t)
   (infront (car3 car4) t) (infront (car4 car5) t))))

(DefExample                    ;Train J
 ((car1 car2 car3)
  ((car-shape nil) (wheel-color nil) (car-length nil) (load-shape nil) (load-number nil) (infront t))
  ((car-shape (car1) engine) (wheel-color (car1) white) (car-shape (car2) u-shape) (car-length (car2) short)
   (load-shape (car2) rectangle) (load-number (car2) one) (wheel-color (car2) white) (car-shape (car3) open-rectangle)
   (car-length (car3) long) (load-shape (car3) rectangle) (load-number (car3) two) (wheel-color (car3) white)
   (infront (car1 car2) t) (infront (car2 car3) t)))))
```

A.3.2. Output for Experiment 3

```
> (subdue :limit 30)

Begin trace...

Parameters:     limit = 30      connectivity = t      compactness = t      coverage = t
                use-bk = nil    discover = t          specialize = nil     inc-bk = nil

Running substructure discovery...

Discovered the following 30 substructures in 21.383333 seconds:

(Substructure#4 :value 11.297071 ((car-length(object-0001)=short)(load-number object-0001)=one)
  (wheel-color object-0001)=white))
WITH OCCURRENCES:
((car-length(car2)=short)(load-number(car2)=one)(wheel-color(car2)=white))
((car-length(car3)=short)(load-number(car3)=one)(wheel-color(car3)=white))
((car-length(car2)=short)(load-number(car2)=one)(wheel-color(car2)=white))
((car-length(car3)=short)(load-number(car3)=one)(wheel-color(car3)=white))
((car-length(car2)=short)(load-number(car2)=one)(wheel-color(car2)=white))
((car-length(car3)=short)(load-number(car3)=one)(wheel-color(car3)=white))
((car-length(car4)=short)(load-number(car4)=one)(wheel-color(car4)=white))
((car-length(car5)=short)(load-number(car5)=one)(wheel-color(car5)=white))
((car-length(car3)=short)(load-number(car3)=one)(wheel-color(car3)=white))
((car-length(car3)=short)(load-number(car3)=one)(wheel-color(car3)=white))
((car-length(car3)=short)(load-number(car3)=one)(wheel-color(car3)=white))
((car-length(car2)=short)(load-number(car2)=one)(wheel-color(car2)=white))
((car-length(car4)=short)(load-number(car4)=one)(wheel-color(car4)=white))
((car-length(car5)=short)(load-number(car5)=one)(wheel-color(car5)=white))
((car-length(car2)=short)(load-number(car2)=one)(wheel-color(car2)=white))

(Substructure#6 :value 8.229033 ((wheel-color(object-0008)=white)(infront(object-0008,object-0001)=t)
  (car-length(object-0001)=short)(load-number(object-0001)=one)(wheel-color object-0001)=white))
WITH OCCURRENCES:
```

```
(arg-name arga a (arg-name argb) b (arg-name argc) c (arg-name argd) d (arg-name arge e (arg-name argf) f
arg-name argg) g)
(subop g00 g01) t) (subop g00 g02) t) (before g01 g02) t)
(op-type g01) stack) (stack-arg1 g01 arga) t) (stack-arg2 g01 argc) t) (subop g01 g03) t) (subop g01 g04) t)
(before g03 g04) t)
(op-type g03) unstack) (unstack-arg1 g03 argb) t) (unstack-arg2 g03 argc) t)
(op-type g04) pickup) (pickup-arg g04 arga) t) (subop g04 g05) t)
(op-type g05) putdown) (putdown-arg g05 argb) t)
(op-type g02) stack) (stack-arg1 (g02 argd) t) (stack-arg2 (g02 argg) t) (subop (g02 g06) t) (subop (g02 g07) t)
(before g06 g07) t)
(op-type (g06) unstack) (unstack-arg1 (g06 argf) t) (unstack-arg2 (g06 argg) t) (subop (g06 g08) t) (subop (g06 g09) t)
(before g08 g09) t)
(op-type g08) unstack) (unstack-arg1 (g08 argc) t) (unstack-arg2 (g08 argf) t)
(op-type (g09) putdown) (putdown-arg (g09 argc) t)
(op-type (g07) pickup) (pickup-arg (g07 argd) t) (subop (g07 g10) t)
(op-type (g10) putdown) (putdown-arg (g10 argf) t))))
```

A.4.2. Output for Experiment 4

```
> (subdue)

Begin trace...

Parameters:   limit = 23      connectivity = t   compactness = t   coverage = t
              use-bk = nil    discover = t       specialize = nil  inc-bk = nil

Running substructure discovery...

Discovered the following 23 substructures in 12.566667 seconds:

Substructure#19 :value 4.4621396 ([op-type(object-0006)=pickup][pickup-arg(object-0006,object-0084)=t]
 [subop(object-0001,object-0094)=t][unstack-arg2(object-0094,object-0122)=t][before(object-0094,object-0006)=t]
 [subop(object-0156,object-0001)=t][stack-arg2(object-0001,object-0122)=t][op-type(object-0094)=unstack]
 [unstack-arg1(object-0094,object-0064)=t][stack-arg1(object-0001,object-0084)=t]
 [putdown-arg(object-0021,object-0064)=t][op-type(object-0021)=putdown][op-type(object-0001)=stack]
 [subop(object-0006,object-0021)=t][subop(object-0001,object-0006)=t])
WITH OCCURRENCES:
([op-type(g04)=pickup][pickup-arg(g04,arga)=t][subop(g01,g03)=t][unstack-arg2(g03,argc)=t][before(g03,g04)=t]
 [subop(g00,g01)=t][stack-arg2(g01,argc)=t][op-type(g03)=unstack][unstack-arg1(g03,argb)=t][stack-arg1(g01,arga)=t]
 [putdown-arg(g05,argb)=t][op-type(g05)=putdown][op-type(g01)=stack][subop(g04,g05)=t][subop(g01,g04)=t])
([op-type(g07)=pickup][pickup-arg(g07,argd)=t][subop(g02,g06)=t][unstack-arg2(g06,argg)=t][before(g06,g07)=t]
 [subop(g00,g02)=t][stack-arg2(g02,argg)=t][op-type(g06)=unstack][unstack-arg1(g06,argf)=t][stack-arg1(g02,argd)=t]
 [putdown-arg(g10,argf)=t][op-type(g10)=putdown][op-type(g02)=stack][subop(g07,g10)=t][subop(g02,g07)=t])

Substructure#22 :value 3.2921875 ([op-type(object-0006)=pickup][pickup-arg(object-0006,object-0084)=t]
 [unstack-arg2(object-0094,object-0122)=t][before(object-0094,object-0006)=t][subop(object-0156,object-0001)=t]
 [stack-arg2(object-0001,object-0122)=t][op-type(object-0094)=unstack][unstack-arg1(object-0094,object-0064)=t]
 [stack-arg1(object-0001,object-0084)=t][putdown-arg(object-0021,object-0064)=t][op-type(object-0021)=putdown]
 [op-type(object-0001)=stack][subop(object-0006,object-0021)=t][subop(object-0001,object-0006)=t])
WITH OCCURRENCES:
([op-type(g04)=pickup][pickup-arg(g04,arga)=t][unstack-arg2(g03,argc)=t][before(g03,g04)=t][subop(g00,g01)=t]
 [stack-arg2(g01,argc)=t][op-type(g03)=unstack][unstack-arg1(g03,argb)=t][stack-arg1(g01,arga)=t]
 [putdown-arg(g05,argb)=t][op-type(g05)=putdown][op-type(g01)=stack][subop(g04,g05)=t][subop(g01,g04)=t])
([op-type(g07)=pickup][pickup-arg(g07,argd)=t][unstack-arg2(g06,argg)=t][before(g06,g07)=t][subop(g00,g02)=t]
 [stack-arg2(g02,argg)=t][op-type(g06)=unstack][unstack-arg1(g06,argf)=t][stack-arg1(g02,argd)=t]
 [putdown-arg(g10,argf)=t][op-type(g10)=putdown][op-type(g02)=stack][subop(g07,g10)=t][subop(g02,g07)=t])

Substructure#20 :value 3.2921875 ([op-type(object-0006)=pickup][subop(object-0001,object-0094)=t]
 [unstack-arg2(object-0094,object-0122)=t][before(object-0094,object-0006)=t][subop(object-0156,object-0001)=t]
 [stack-arg2(object-0001,object-0122)=t][op-type(object-0094)=unstack][unstack-arg1(object-0094,object-0064)=t]
 [stack-arg1(object-0001,object-0084)=t][putdown-arg(object-0021,object-0064)=t][op-type(object-0021)=putdown]
 [op-type(object-0001)=stack][subop(object-0006,object-0021)=t][subop(object-0001,object-0006)=t])
WITH OCCURRENCES:
([op-type(g04)=pickup][subop(g01,g03)=t][unstack-arg2(g03,argc)=t][before(g03,g04)=t][subop(g00,g01)=t]
```

(double-bond(c17,c18)=t) (double-bond(c15,c16)=t) (double-bond(c09,c10)=t) (single-bond(c09,c18)=t)
(single-bond(c16,c17)=t) (single-bond(c10,c15)=t) (single-bond(c16,n11)=t) (single-bond(c15,n05)=t)
(atom-type(c18)=carbon) (atom-type(c17)=carbon) (atom-type(c16)=carbon) (atom-type(c15)=carbon)
(atom-type(c10)=carbon) (atom-type(c09)=carbon) (atom-type(h11)=hydrogen))
((double-bond(c17,c18)=t) (double-bond(c15,c16)=t) (double-bond(c09,c10)=t) (single-bond(c09,c18)=t)
(single-bond(c16,c17)=t) (single-bond(c10,c15)=t) (single-bond(c16,n11)=t) (single-bond(c15,n05)=t)
(atom-type(c18)=carbon) (atom-type(c17)=carbon) (atom-type(c16)=carbon) (atom-type(c15)=carbon)
(atom-type(c10)=carbon) (atom-type(c09)=carbon) (atom-type(n05)=hydrogen))
((double-bond(c13,c14)=t) (double-bond(c11,c12)=t) (double-bond(c07,c08)=t) (single-bond(c08,c14)=t)
(single-bond(c12,c13)=t) (single-bond(c07,c11)=t) (single-bond(c14,n10)=t) (single-bond(c13,n09)=t)
(atom-type(c14)=carbon) (atom-type(c13)=carbon) (atom-type(c12)=carbon) (atom-type(c11)=carbon)
(atom-type(c08)=carbon) (atom-type(c07)=carbon) (atom-type(h10)=hydrogen))
((double-bond(c13,c14)=t) (double-bond(c11,c12)=t) (double-bond(c07,c08)=t) (single-bond(c08,c14)=t)
(single-bond(c12,c13)=t) (single-bond(c07,c11)=t) (single-bond(c14,n10)=t) (single-bond(c13,n09)=t)
(atom-type(c14)=carbon) (atom-type(c13)=carbon) (atom-type(c12)=carbon) (atom-type(c11)=carbon)
(atom-type(c08)=carbon) (atom-type(c07)=carbon) (atom-type(h09)=hydrogen))
((double-bond(c13,c14)=t) (double-bond(c11,c12)=t) (double-bond(c07,c08)=t) (single-bond(c08,c14)=t)
(single-bond(c12,c13)=t) (single-bond(c07,c11)=t) (single-bond(c12,n08)=t) (single-bond(c11,br)=t)
(atom-type(c14)=carbon) (atom-type(c13)=carbon) (atom-type(c12)=carbon) (atom-type(c11)=carbon)
(atom-type(c08)=carbon) (atom-type(c07)=carbon) (atom-type(h08)=hydrogen))
((double-bond(c05,c06)=t) (double-bond(c03,c04)=t) (double-bond(c01,c02)=t) (single-bond(c04,c05)=t)
(single-bond(c02,c03)=t) (single-bond(c01,c06)=t) (single-bond(c04,n04)=t) (single-bond(c03,n03)=t)
(atom-type(c06)=carbon) (atom-type(c05)=carbon) (atom-type(c04)=carbon) (atom-type(c03)=carbon)
(atom-type(c02)=carbon) (atom-type(c01)=carbon) (atom-type(h04)=hydrogen))
((double-bond(c05,c06)=t) (double-bond(c03,c04)=t) (double-bond(c01,c02)=t) (single-bond(c04,c05)=t)
(single-bond(c02,c03)=t) (single-bond(c01,c06)=t) (single-bond(c04,n04)=t) (single-bond(c03,n03)=t)
(atom-type(c06)=carbon) (atom-type(c05)=carbon) (atom-type(c04)=carbon) (atom-type(c03)=carbon)
(atom-type(c02)=carbon) (atom-type(c01)=carbon) (atom-type(h03)=hydrogen))
((double-bond(c05,c06)=t) (double-bond(c03,c04)=t) (double-bond(c01,c02)=t) (single-bond(c04,c05)=t)
(single-bond(c02,c03)=t) (single-bond(c01,c06)=t) (single-bond(c02,n02)=t) (single-bond(c01,cl)=t)
(atom-type(c06)=carbon) (atom-type(c05)=carbon) (atom-type(c04)=carbon) (atom-type(c03)=carbon)
(atom-type(c02)=carbon) (atom-type(c01)=carbon) (atom-type(h02)=hydrogen))

Substructure#0 (value 18.550645 ((atom-type(object-0001)=bromine,chlorine,iodine)
(single-bond(object-0004,object-0001)=t) (atom-type(object-0003)=carbon) (double-bond(object-0002,object-0003)=t)
(atom-type(object-0002)=carbon) (single-bond(object-0005,object-0002)=t) (single-bond(object-0003,object-0004)=t)
(atom-type(object-0006)=hydrogen) (atom-type(object-0004)=carbon) (double-bond(object-0004,object-0007)=t)
(atom-type(object-0005)=carbon) (double-bond(object-0005,object-0008)=t) (single-bond(object-0007,object-0006)=t)
(atom-type(object-0007)=carbon) (single-bond(object-0007,object-0008)=t) (atom-type(object-0008)=carbon)))
WITH OCCURRENCES:
((double-bond(c05,c06)=t) (double-bond(c03,c04)=t) (double-bond(c01,c02)=t) (single-bond(c04,c05)=t)
(single-bond(c02,c03)=t) (single-bond(c01,c06)=t) (single-bond(c02,n02)=t) (single-bond(c01,cl)=t)
(atom-type(c06)=carbon) (atom-type(c05)=carbon) (atom-type(c04)=carbon) (atom-type(c03)=carbon)
(atom-type(c02)=carbon) (atom-type(c01)=carbon) (atom-type(h02)=hydrogen) (atom-type(cl)=chlorine))
((double-bond(c13,c14)=t) (double-bond(c11,c12)=t) (double-bond(c07,c08)=t) (single-bond(c08,c14)=t)
(single-bond(c12,c13)=t) (single-bond(c07,c11)=t) (single-bond(c12,n08)=t) (single-bond(c11,br)=t)
(atom-type(c14)=carbon) (atom-type(c13)=carbon) (atom-type(c12)=carbon) (atom-type(c11)=carbon)
(atom-type(c08)=carbon) (atom-type(c07)=carbon) (atom-type(br)=bromine) (atom-type(h08)=hydrogen))
((double-bond(c17,c18)=t) (double-bond(c15,c16)=t) (double-bond(c09,c10)=t) (single-bond(c09,c18)=t)
(single-bond(c16,c17)=t) (single-bond(c10,c15)=t) (single-bond(c17,n12)=t) (single-bond(c18,i)=t)
(atom-type(c18)=carbon) (atom-type(c17)=carbon) (atom-type(c16)=carbon) (atom-type(c15)=carbon)
(atom-type(c10)=carbon) (atom-type(c09)=carbon) (atom-type(h12)=hydrogen) (atom-type(i)=iodine)))

Running substructure discovery...

Discovered the following 10 substructures in 241.31667 seconds:

Substructure#5 (value 34.36344 ((single-bond(object-0013,object-0030)=t) (atom-type(object-0009)=hydrogen)
(atom-type(object-0018)=hydrogen) (single-bond(object-0016,object-0018)=t) (single-bond(object-0012,object-0009)=t)
(atom-type(object-0011)=carbon) (double-bond(object-0010,object-0011)=t) (atom-type(object-0010)=carbon)
(single-bond(object-0013,object-0010)=t) (single-bond(object-0011,object-0012)=t) (atom-type(object-0014)=hydrogen)
(atom-type(object-0012)=carbon) (double-bond(object-0012,object-0015)=t) (atom-type(object-0013)=carbon)
(double-bond(object-0013,object-0016)=t) (single-bond(object-0015,object-0014)=t) (atom-type(object-0015)=carbon)
(single-bond(object-0015,object-0016)=t) (atom-type(object-0016)=carbon)))
WITH OCCURRENCES:
(single-bond(c13,...)=t) (atom-type(n05)=hydrogen) (atom-type(n12)=hydrogen) (single-bond(c17,n12)=t)

```
Specializing substructure...

Specialized the substructure:

Substructure#5 :value 34.36344 (single-bond(object-0013,object-0030)=t)(atom-type(object-0009)=hydrogen)
 (atom-type(object-0018)=hydrogen)(single-bond(object-0016,object-0018)=t)(single-bond(object-0012,object-0009)=t)
 (atom-type(object-0011)=carbon)(double-bond(object-0010,object-0011)=t)(atom-type(object-0010)=carbon)
 (single-bond(object-0013,object-0010)=t)(single-bond(object-0011,object-0012)=t)(atom-type(object-0014)=hydrogen)
 (atom-type(object-0012)=carbon)(double-bond(object-0012,object-0015)=t)(atom-type(object-0013)=carbon)
 (double-bond(object-0013,object-0016)=t)(single-bond(object-0015,object-0014)=t)(atom-type(object-0015)=carbon)
 (single-bond(object-0015,object-0016)=t)(atom-type(object-0016)=carbon))

to the following substructures:

(Substructure#0 :value 1 ((atom-type(object-0030)=bromine,chlorine,iodine)(single-bond(object-0013,object-0030)=t)
 (atom-type(object-0009)=hydrogen)(atom-type(object-0018)=hydrogen)(single-bond(object-0016,object-0018)=t)
 (single-bond(object-0012,object-0009)=t)(atom-type(object-0011)=carbon)(double-bond(object-0010,object-0011)=t)
 (atom-type(object-0010)=carbon)(single-bond(object-0013,object-0010)=t)(single-bond(object-0011,object-0012)=t)
 (atom-type(object-0014)=hydrogen)(atom-type(object-0012)=carbon)(double-bond(object-0012,object-0015)=t)
 (atom-type(object-0013)=carbon)(double-bond(object-0013,object-0016)=t)(single-bond(object-0015,object-0014)=t)
 (atom-type(object-0015)=carbon)(single-bond(object-0015,object-0016)=t)(atom-type(object-0016)=carbon))

Adding substructures to BK...

Added the following substructures to BK:

Discovered: (Substructure#5 :value 34.36344 ((single-bond(object-0013,object-0030)=t)(atom-type(object-0009)=hydrogen)
 (atom-type(object-0018)=hydrogen)(single-bond(object-0016,object-0018)=t)(single-bond(object-0012,object-0009)=t)
 (atom-type(object-0011)=carbon)(double-bond(object-0010,object-0011)=t)(atom-type(object-0010)=carbon)
 (single-bond(object-0013,object-0010)=t)(single-bond(object-0011,object-0012)=t)(atom-type(object-0014)=hydrogen)
 (atom-type(object-0012)=carbon)(double-bond(object-0012,object-0015)=t)(atom-type(object-0013)=carbon)
 (double-bond(object-0013,object-0016)=t)(single-bond(object-0015,object-0014)=t)(atom-type(object-0015)=carbon)
 (single-bond(object-0015,object-0016)=t)(atom-type(object-0016)=carbon))

Specialized: (Substructure#0 :value 1 ((atom-type(object-0030)=bromine,chlorine,iodine)
 (single-bond(object-0013,object-0030)=t)(atom-type(object-0009)=hydrogen)(atom-type(object-0018)=hydrogen)
 (single-bond(object-0016,object-0018)=t)(single-bond(object-0012,object-0009)=t)(atom-type(object-0011)=carbon)
 (double-bond(object-0010,object-0011)=t)(atom-type(object-0010)=carbon)(single-bond(object-0013,object-0010)=t)
 (single-bond(object-0011,object-0012)=t)(atom-type(object-0014)=hydrogen)(atom-type(object-0012)=carbon)
 (double-bond(object-0012,object-0015)=t)(atom-type(object-0013)=carbon)(double-bond(object-0013,object-0016)=t)
 (single-bond(object-0015,object-0014)=t)(atom-type(object-0015)=carbon)(single-bond(object-0015,object-0016)=t)
 (atom-type(object-0016)=carbon))))

End trace.
```

A.3. Experiment 3

Experiment 3 demonstrates SUBDUE's ability to discover substructures that can be used as high-level attributes for classifying multiple examples. The input examples consist of the descriptions of ten trains. The *DefExample* calls for each example are shown, along with the resulting output.

(single c1 c2) t) (double (c1 c3) t) (double (c2 c4) t) (single (c3 c5) t) (single (c4 c6) t) (double (c5 c6) t)
(single (c7 c8) t) (double (c7 c9) t) (double (c8 c10) t) (single (c9 c11) t) (single (c10 c12) t) (double (c11 c12) t)
(single (c13 c14) t) (double (c13 c15) t) (double (c14 c16) t) (single (c15 c17) t) (single (c16 c18) t)
(double (c17 c18) t) (single (c19 c20) t) (single (c12 c21) t) (single (c18 c22) t) (single (c19 c20) t)
(single (c20 c21) t) (single (c21 c22) t) (single (c22 c23) t) (single (c20 c24) t) (single (c21 c25) t)
(single (c22 c26) t)))

(DefExample (compound 6 negative)
'((c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18 c19 c20 c21 c22 c23 c24 c25 c26 c27 c28 c29 c30 c31 c32
c33 c34)
((single nil) (double nil))
((single (c1 c2) t) (double (c1 c3) t) (double (c2 c4) t) (single (c3 c5) t) (single (c4 c6) t) (double (c5 c6) t)
(single (c7 c8) t) (double (c7 c9) t) (double (c8 c10) t) (single (c9 c11) t) (single (c10 c12) t) (double (c11 c12) t)
(single (c13 c14) t) (double (c13 c15) t) (double (c14 c16) t) (single (c15 c17) t) (single (c16 c18) t)
(double (c17 c18) t) (single (c19 c20) t) (double (c19 c21) t) (double (c20 c22) t) (single (c21 c23) t)
(single (c22 c24) t) (double (c23 c24) t) (single (c6 c26) t) (single (c12 c27) t) (single (c18 c25) t)
(single (c24 c29) t) (single (c25 c26) t) (single (c26 c27) t) (single (c27 c28) t) (single (c28 c29) t)
(single (c29 c30) t) (single (c26 c31) t) (single (c27 c32) t) (single (c28 c33) t) (single (c29 c34) t))))

A.5.2. Output for Experiment 5

> (subdue :limit 7)

Begin trace...

Parameters:	limit = 7	connectivity = t	compactness = t	coverage = t
	use-bk = nil	discover = t	specialize = nil	inc-bk = nil

Running substructure discovery...

Discovered the following 7 substructures in 15.183333 seconds:

Substructure#7 :value 178.40707 ([single(object-0011,object-0015)=t][double(object-0015,object-0009)=t]
[double(object-0011,object-0002)=t][single(object-0005,object-0009)=t,[double(object-0005,object-0001)=t]
[single(object-0001,object-0002)=t])
WITH OCCURRENCES:
([single(c4,c6)=t][double(c5,c6)=t][double(c2,c4)=t][single(c3,c5)=t][double(c1,c3)=t][single(c1,c2)=t])
([single(c14,c15)=t][double(c13,c15)=t][double(c12,c14)=t][single(c11,c13)=t,[double(c10,c11)=t][single(c10,c12)=t])
([single(c4,c6)=t][double(c5,c6)=t][double(c2,c4)=t][single(c3,c5)=t][double(c1,c3)=t][single(c1,c2)=t])
([single(c10,c12)=t][double(c11,c12)=t][double(c8,c10)=t,[single(c9,c11)=t][double(c7,c9)=t][single(c7,c8)=t])
([single(c21,c22)=t][double(c20,c22)=t][double(c19,c21)=t][single(c18,c20)=t][double(c17,c18)=t][single(c17,c19)=t])
([single(c27,c28)=t][double(c26,c28)=t][double(c25,c27)=t][single(c24,c26)=t][double(c23,c24)=t][single(c23,c25)=t])
([single(c4,c6)=t][double(c5,c6)=t][double(c2,c4)=t][single(c3,c5)=t][double(c1,c3)=t][single(c1,c2)=t])
([single(c10,c12)=t][double(c11,c12)=t][double(c8,c10)=t][single(c9,c11)=t][double(c7,c9)=t][single(c7,c8)=t])
([single(c16,c18)=t][double(c17,c18)=t][double(c14,c16)=t][single(c15,c17)=t,[double(c13,c15)=t][single(c13,c14)=t])
([single(c28,c29)=t][double(c27,c29)=t][double(c26,c28)=t][single(c25,c27)=t,[double(c24,c25)=t][single(c24,c26)=t])
([single(c34,c35)=t][double(c33,c35)=t][double(c32,c34)=t][single(c31,c33)=t,[double(c30,c31)=t][single(c30,c32)=t])
([single(c40,c41)=t][double(c39,c41)=t][double(c38,c40)=t][single(c37,c39)=t,[double(c36,c37)=t][single(c36,c38)=t])
([single(c4,c6)=t][double(c5,c6)=t][double(c2,c4)=t][single(c3,c5)=t][double(c1,c3)=t][single(c1,c2)=t])
([single(c10,c12)=t][double(c11,c12)=t][double(c8,c10)=t][single(c9,c11)=t][double(c7,c9)=t][single(c7,c8)=t])
([single(c4,c6)=t][double(c5,c6)=t][double(c2,c4)=t][single(c3,c5)=t][double(c1,c3)=t][single(c1,c2)=t])
([single(c10,c12)=t][double(c11,c12)=t][double(c8,c10)=t][single(c9,c11)=t][double(c7,c9)=t][single(c7,c8)=t])
([single(c16,c18)=t][double(c17,c18)=t][double(c14,c16)=t][single(c15,c17)=t,[double(c13,c15)=t][single(c13,c14)=t])
([single(c4,c6)=t][double(c5,c6)=t][double(c2,c4)=t][single(c3,c5)=t][double(c1,c3)=t][single(c1,c2)=t])
([single(c10,c12)=t][double(c11,c12)=t][double(c8,c10)=t][single(c9,c11)=t][double(c7,c9)=t][single(c7,c8)=t])
([single(c16,c18)=t][double(c17,c18)=t][double(c14,c16)=t][single(c15,c17)=t,[double(c13,c15)=t][single(c13,c14)=t])
([single(c22,c24)=t][double(c23,c24)=t][double(c20,c22)=t][single(c21,c23)=t,[double(c19,c21)=t][single(c19,c20)=t])

Substructure#6 :value 74.64602 ([double(object-0015,object-0009)=t][double(object-0011,object-0002)=t]
[single(object-0005,object-0009)=t][double(object-0005,object-0001)=t,[single(object-0001,object-0002)=t])
WITH OCCURRENCES:
([double(c5,c6)=t][double(c2,c4)=t][single(c3,c5)=t][double(c1,c3)=t][single(c1,c2)=t])
([double(c5,c6)=t][double(c1,c3)=t][single(c4,c6)=t][double(c2,c4)=t][single(c1,c2)=t])
([double(c2,c4)=t][double(c1,c3)=t][single(c4,c6)=t][double(c5,c6)=t][single(c3,c5)=t])

[double(c1,c3)=t] [single(c4,c6)=t] [double(c2,c4)=t] [single(c1,c2)=t])
[double(c5,c6)=t] [single(c4,c6)=t] [double(c2,c4)=t] [single(c1,c2)=t])
[double(c1,c3)=t] [single(c4,c6)=t] [double(c5,c6)=t] [single(c3,c5)=t])
[double(c2,c4)=t] [single(c4,c6)=t] [double(c5,c6)=t] [single(c3,c5)=t])
[double(c1,c3)=t] [single(c6,c8)=t] [double(c5,c6)=t] [single(c3,c5)=t])
[double(c13,c15)=t] [single(c11,c13)=t] [double(c10,c11)=t] [single(c8,c10)=t])
[double(c12,c14)=t] [single(c11,c13)=t] [double(c10,c11)=t] [single(c10,c12)=t])
([double(c13,c15)=t] [single(c11,c13)=t] [double(c10,c11)=t] [single(c10,c12)=t])
([double(c10,c11)=t] [single(c14,c15)=t] [double(c12,c14)=t] [single(c10,c12)=t])
([double(c13,c15)=t] [single(c14,c15)=t] [double(c12,c14)=t] [single(c10,c12)=t])
([double(c10,c11)=t] [single(c14,c15)=t] [double(c13,c15)=t] [single(c11,c13)=t])
([double(c12,c14)=t] [single(c14,c15)=t] [double(c13,c15)=t] [single(c11,c13)=t])
([double(c2,c4)=t] [single(c3,c5)=t] [double(c1,c3)=t] [single(c1,c2)=t])
([double(c5,c6)=t] [single(c3,c5)=t] [double(c1,c3)=t] [single(c1,c2)=t])
([double(c1,c3)=t] [single(c4,c6)=t] [double(c2,c4)=t] [single(c1,c2)=t])
([double(c5,c6)=t] [single(c4,c6)=t] [double(c2,c4)=t] [single(c1,c2)=t])
([double(c1,c3)=t] [single(c4,c6)=t] [double(c5,c6)=t] [single(c3,c5)=t])
([double(c2,c4)=t] [single(c4,c6)=t] [double(c5,c6)=t] [single(c3,c5)=t])
([double(c1,c3)=t] [single(c6,c14)=t] [double(c5,c6)=t] [single(c3,c5)=t])
([double(c8,c10)=t] [single(c9,c11)=t] [double(c7,c9)=t] [single(c7,c8)=t])
([double(c11,c12)=t] [single(c9,c11)=t] [double(c7,c9)=t] [single(c7,c8)=t])
([double(c7,c9)=t] [single(c10,c12)=t] [double(c8,c10)=t] [single(c7,c8)=t])
([double(c11,c12)=t] [single(c10,c12)=t] [double(c8,c10)=t] [single(c7,c8)=t])
([double(c7,c9)=t] [single(c10,c12)=t] [double(c11,c12)=t] [single(c9,c11)=t])
([double(c8,c10)=t] [single(c10,c12)=t] [double(c11,c12)=t] [single(c9,c11)=t])
([double(c7,c9)=t] [single(c12,c15)=t] [double(c11,c12)=t] [single(c9,c11)=t])
([double(c20,c22)=t] [single(c18,c20)=t] [double(c17,c18)=t] [single(c14,c17)=t])
([double(c26,c28)=t] [single(c24,c26)=t] [double(c23,c24)=t] [single(c15,c23)=t])
([double(c19,c21)=t] [single(c18,c20)=t] [double(c17,c18)=t] [single(c17,c19)=t])
([double(c20,c22)=t] [single(c18,c20)=t] [double(c17,c18)=t] [single(c17,c19)=t])
([double(c17,c18)=t] [single(c21,c22)=t] [double(c19,c21)=t] [single(c17,c19)=t])
([double(c20,c22)=t] [single(c21,c22)=t] [double(c19,c21)=t] [single(c17,c19)=t])
([double(c17,c18)=t] [single(c21,c22)=t] [double(c20,c22)=t] [single(c18,c20)=t])
([double(c19,c21)=t] [single(c21,c22)=t] [double(c20,c22)=t] [single(c18,c20)=t])
([double(c25,c27)=t] [single(c24,c26)=t] [double(c23,c24)=t] [single(c23,c25)=t])
([double(c26,c28)=t] [single(c24,c26)=t] [double(c23,c24)=t] [single(c23,c25)=t])
([double(c23,c24)=t] [single(c27,c28)=t] [double(c25,c27)=t] [single(c23,c25)=t])
([double(c26,c28)=t] [single(c27,c28)=t] [double(c25,c27)=t] [single(c23,c25)=t])
([double(c23,c24)=t] [single(c27,c28)=t] [double(c26,c28)=t] [single(c24,c26)=t])
([double(c25,c27)=t] [single(c27,c28)=t] [double(c26,c28)=t] [single(c24,c26)=t])
([double(c2,c4)=t] [single(c3,c5)=t] [double(c1,c3)=t] [single(c1,c2)=t])
([double(c5,c6)=t] [single(c3,c5)=t] [double(c1,c3)=t] [single(c1,c2)=t])
([double(c1,c3)=t] [single(c4,c6)=t] [double(c2,c4)=t] [single(c1,c2)=t])
([double(c5,c6)=t] [single(c4,c6)=t] [double(c2,c4)=t] [single(c1,c2)=t])
([double(c1,c3)=t] [single(c4,c6)=t] [double(c5,c6)=t] [single(c3,c5)=t])
([double(c2,c4)=t] [single(c4,c6)=t] [double(c5,c6)=t] [single(c3,c5)=t])
([double(c1,c3)=t] [single(c6,c20)=t] [double(c5,c6)=t] [single(c3,c5)=t])
([double(c9,c10)=t] [single(c9,c11)=t] [double(c7,c9)=t] [single(c7,c8)=t])
([double(c11,c12)=t] [single(c9,c11)=t] [double(c7,c9)=t] [single(c7,c8)=t])
([double(c7,c9)=t] [single(c10,c12)=t] [double(c8,c10)=t] [single(c7,c8)=t])
([double(c11,c12)=t] [single(c10,c12)=t] [double(c8,c10)=t] [single(c7,c8)=t])
([double(c7,c9)=t] [single(c10,c12)=t] [double(c11,c12)=t] [single(c9,c11)=t])
([double(c8,c10)=t] [single(c10,c12)=t] [double(c11,c12)=t] [single(c9,c11)=t])
([double(c7,c9)=t] [single(c12,c21)=t] [double(c11,c12)=t] [single(c9,c11)=t])
([double(c14,c16)=t] [single(c15,c17)=t] [double(c13,c15)=t] [single(c13,c14)=t])
([double(c17,c18)=t] [single(c15,c17)=t] [double(c13,c15)=t] [single(c13,c14)=t])
([double(c13,c15)=t] [single(c16,c18)=t] [double(c14,c16)=t] [single(c13,c14)=t])
([double(c17,c18)=t] [single(c16,c18)=t] [double(c14,c16)=t] [single(c13,c14)=t])
([double(c13,c15)=t] [single(c16,c18)=t] [double(c17,c18)=t] [single(c15,c17)=t])
([double(c14,c16)=t] [single(c16,c18)=t] [double(c17,c18)=t] [single(c15,c17)=t])
([double(c13,c15)=t] [single(c18,c22)=t] [double(c17,c18)=t] [single(c15,c17)=t])
([double(c27,c29)=t] [single(c25,c27)=t] [double(c24,c25)=t] [single(c20,c24)=t])
([double(c33,c35)=t] [single(c31,c33)=t] [double(c30,c31)=t] [single(c21,c30)=t])
([double(c39,c41)=t] [single(c37,c39)=t] [double(c36,c37)=t] [single(c22,c36)=t])
([double(c26,c28)=t] [single(c25,c27)=t] [double(c24,c25)=t] [single(c24,c26)=t])
([double(c27,c29)=t] [single(c25,c27)=t] [double(c24,c25)=t] [single(c24,c26)=t])

```
(double(c17,c18)=t] [single(c15,c17)=t] [double(c13,c15)=t] [sing.e(c13,c14)=t])
 (double(c13,c15)=t] [single(c16,c18)=t] [double(c14,c16)=t] [sing.e(c13,c14)=t])
 (double(c17,c18)=t] [single(c16,c18)=t] [double(c14,c16)=t] [sing.e(c13,c14)=t])
 (double(c13,c15)=t] [single(c16,c18)=t] [double(c17,c18)=t] [sing.e(c15,c17)=t])
 (double(c14,c16)=t] [single(c16,c18)=t] [double(c17,c18)=t] [sing.e(c15,c17)=t])
 (double(c13,c15)=t] [single(c13,c23)=t] [double(c17,c18)=t] [sing.e(c15,c17)=t])
 (double(c20,c22)=t] [single(c21,c23)=t] [double(c19,c21)=t] [single(c19,c20)=t])
(double(c23,c24)=t] [single(c21,c23)=t] [double(c19,c21)=t] [sing.e(c19,c20)=t])
(double(c19,c21)=t] [single(c22,c24)=t] [double(c20,c22)=t] [single(c19,c20)=t])
(double(c23,c24)=t] [single(c22,c24)=t] [double(c20,c22)=t] [single(c19,c20)=t])
(double(c19,c21)=t] [single(c22,c24)=t] [double(c23,c24)=t] [single(c21,c23)=t])
(double(c20,c22)=t] [single(c22,c24)=t] [double(c23,c24)=t] [single(c21,c23)=t])
(double(c19,c21)=t] [single(c24,c29)=t] [double(c23,c24)=t] [single(c21,c23)=t])
   :
   :
End trace.
```