May 1988 UILU-ENG-88-2223 (2)

DTIC FILE COPY

COORDINATED SCIENCE LABORATORY
College of Engineering

AD-A197 778

DISCOVERING SUBSTRUCTURE IN EXAMPLES

Lawrence Bruce Holder, Jr.

88 7 13 005

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

[illegible]

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 NO.

11. TITLE (Include Security Classification)

Discovering Substructure in Examples

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

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

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

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
UNCLASSIFIED

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED
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 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 level regularities in terms of these abstract features. This thesis also proposes a method for maintaining such a hierarchy of substructure and utilizing this knowledge in subsequent substructure identification and discovery tasks.

The existence of these human processes for perceiving substructure suggests that having a computational method for the processes may improve the abilities of current machine learning programs that operate in a detailed structural environment. Such a computational method for substructure identification and discovery could abstract over unnecessary detail in the descriptions of the examples given to a learning program. For instance, suppose the given examples are organic molecules described by the types of atoms in the molecule and the types of bonds between the atoms. For complex molecules the number of features describing the examples is overwhelming and may conceal more abstract substructure concepts such as an aromatic ring. By first discovering the ring of carbon atoms and connected hydrogen atoms as an interesting substructure, a substructure discovery system can replace the group of atoms and bonds by a single entity representing the newly discovered substructure concept. Then, a learning program can discover concepts in terms of this substructure concept instead of just atoms bonded to atoms. Other machine learning programs can benefit from a system that discovers substructure concepts in examples and retains the concepts for use in subsequent learning tasks.

The methodology underlying the proposed computational method for substructure discovery is presented in Chapter 2. The chapter begins by defining a substructure and presenting the language used to describe substructures. Next, the components involved in substructure discovery are discussed. Two essential processes of a substructure discovery system are substructure generation and substructure selection. Substructure generation is the process of constructing new, alternative substructures from an existing substructure. After generating the set of alternative substructures, the discovery system invokes the substructure selection process to choose the best substructures from among the alternatives. Once an interesting substructure is discovered, the occurrences of the substructure within the input examples are replaced, or instantiated, by a single 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 human perception [Wertheimer39]. More recent machine learning research on substructure discovery begins with Winston's work on learning from examples in the blocks world [Winston75]. From this point on, research on the various facets of substructure discovery has proposed several interesting solutions to the problem.

Finally, Chapter 6 concludes with a summary of the research described in this thesis and discusses directions for future research. Since the completion of the research described in this thesis, several extensions, improvements and applications of the substructure discovery method and the SUBDUE system have been revealed. Improvements to the discovery method might be obtained by generating alternative substructures more intelligently or by incorporating specialization into the discovery process rather than as a separate process. Future potential applications of the substructure discovery system include detection of interesting image patterns or textures in a high level vision system and constructive feature formation in other machine learning systems. These and other directions for future research are discussed in Chapter 6.

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.

2.1. Motivations

One motivation for substructure discovery is the overwhelming number of features available in a typical real-world description of an environment. Unless features are preselected by the human experimenter, artificial intelligence programs relying on environmental observations for input would become bogged down in the complexity of processing the large amount of sensory data. Therefore, a program for constructing features from the data and thereby introducing ways to abstract over the unnecessary detail in the observations would improve the performance of other learning programs. Having the new abstracted features can provide learning programs with a basis for formulating concepts based on the more abstract, and possibly more pertinent features [Stepp87].

A second motivation for substructure discovery originates from the hierarchical nature of a structured environment. In other words, given a level of detail with which to describe an environment, there almost always exists a more primitive description in terms of the components of the current features. This fact suggests that an environment can be represented as a hierarchy, where each level in the hierarchy represents a different level of detail with which to represent the environment. By defining newly discovered substructures in terms of more primitive substructures already known, a hierarchical substructure representation is constructed similar to the natural hierarchy of the environment. Other programs may then use this hierarchy to find an appropriate level of abstraction for processing the features of the environment.

The reorganization and compression of knowledge bases provides a third motivation for substructure discovery. The amount of data typically stored in real world knowledge bases is immense. In order to combat the storage and processing limitations on current retrieval systems, the substructure discovery program can be applied to the data of the knowledge base. The underlying substructures found in the data can be used to compress the data and impose a hierarchical representation. The subsequent reorganized data requires less storage space and reduces the retrieval time for queries referencing data containing the same substructures.

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.

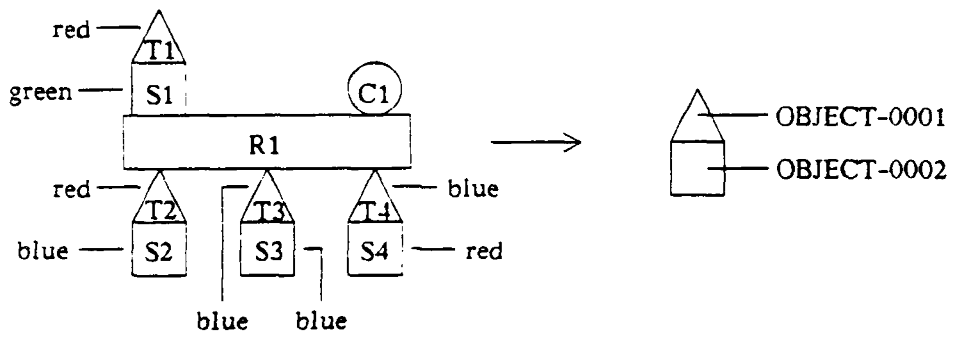
A collection of structured objects can be represented by a graph. Using a graph representation, a substructure is a collection of annotated nodes and edges comprising a connected subgraph of a larger graph. The nodes represent single objects, values, or relations in the substructure, and the edges represent 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 *b* node. As another example, consider the relation *(shape a circle)*. The graph representation consists of three nodes (*shape*, *a* and *circle*) along with an edge from *shape* to the object *a* and an edge from *shape* to the value *circle*. However, substructures represent more than just a collection of nodes and edges. The description of a substructure is a conjunctive concept. Substructure discovery is concerned with identifying substructures that represent interesting concepts, not just interesting graphical structures. Thus, substructures, or equivalently substructure concepts, should be interpreted as a unit of structure described by a conjunctive concept.

An appropriate language for describing substructures in terms of conjunctive concepts is an extension to the first order logic called Variable-valued Logic system 2 (VL_2) [Michalski80] — a subset of the Annotated Predicate Calculus (APC) [Michalski83a]. The VL_2 representation uses a conjunction of relational units called selectors to describe a substructure. A *selector relation* is a two-place relation between a function and the value or values in the range of the function. Functions with only one argument are called *attributes*. For example, the selector relation corresponding to the relation *(shape a circle)* is *(shape(a) circle)*, where *shape(a)* is an attribute and *circle* is a value in the range of the *shape* attribute. In VL_2 this selector relation is written as [SHAPE(A)=CIRCLE]. Likewise, the VL_2 expression for the relation *(on a b)* is [ON(A,B)=T]. VL_2 also allows certain disjunctive concepts to be expressed as a single conjunct. For example, the disjunctive concept [COLOR(C)=RED] $\vee$ [COLOR(C)=BLUE] can be expressed as the single equivalent conjunct [COLOR(C)=RED,BLUE].

Using the VL_2 representation, a *substructure* is defined as a conjunction of connected selector relations. A set of selector relations is *connected* if the equivalent graph representation of the selector relations is connected. *Objects* correspond to single entities within the substructure and act as the arguments to the functions of the selector relations. An object is a primitive element on which selector relations and, ultimately, substructures are defined. Typically, the range of values of the function in a selector relation includes a "don't care" element. This element is used to denote a function that must be present but whose value is immaterial. For example, if the "don't care" 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

After each object of the substructure is assigned an arbitrary symbolic name as shown in the Figure 2.1b, the expression for the substructure is

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

For subsequent discussions, some terminology is needed to describe important aspects of substructures as they relate to a given set of input examples. An *occurrence* of a substructure in a set of input examples is the set of objects and selector relations from the examples which match, graph theoretically, to the graph representation of the substructure. All isomorphisms with the same objects and selector relations are considered the same occurrence. For example, the occurrences of the substructure in the input example of Figure 2.1a are

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

A *neighboring relation* of an occurrence of a substructure is a selector relation in the input example that is not contained in the occurrence, but has at least one object from the occurrence as an argument. For example, the neighboring relations of the first occurrence listed above are [COLOR(T1)=RED], [COLOR(S1)=GREEN] and [ON(S1,R1)=T].

An *attribute relation* of an occurrence of a substructure is a neighboring relation whose only argument is a single object contained in the occurrence. The attribute relations in the first occurrence listed above are [COLOR(T1)=RED] and [COLOR(S1)=GREEN].

An *external connection* of an occurrence of a substructure is a neighboring relation of the occurrence that has as an argument at least one object not contained in the occurrence. In other words, an external connection of an occurrence of a substructure is a selector relation that relates one or more objects in the occurrence to one or more objects not in the occurrence. For the first occurrence listed above, there is only one external connection, [ON(S1,R1)=T].

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 example. the substructure in Figure 2.1b can be generated by removing the [COLOR(OBJECT-0001)=RED] relation from the following substructure

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

The second method. *cut disconnection*, cuts a substructure into two unconnected substructures by removing one relation. Cut disconnection is actually a special case of minimal disconnection; however, a substructure does not always contain a suitable relation on which to perform cut disconnection. Performing cut disconnection on the substructure in Figure 2.1b would generate the following two simple substructures

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

As an example of a substructure having no suitable cut disconnection relation, consider the substructure representing three objects connected in a ring

<[CONNECTED(OBJECT-0001,OBJECT-0002)=T][CONNECTED(OBJECT-0002,OBJECT-0003)=T]
[CONNECTED(OBJECT-0001,OBJECT-0003)=T]>

There does not exist a relation in this substructure that when removed yields two unconnected substructures.

Each method typically has several applications within a given substructure. Therefore, the method must either exhaustively generate all possible resulting substructures or intelligently choose the more promising applications. Intelligent application of the minimal expansion method for generating new substructures considers the occurrences of the substructure being expanded. If each occurrence of the substructure has the same type of neighboring relation, application of the method to this relation would be better than an application to a less repetitive neighboring relation. Combination expansion can be applied more intelligently by combining only those substructures that have a large number of relations and objects in common. Exhaustive appl. ation 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 input examples. The cognitive savings of a substructure represents the net reduction in the complexity of the input examples provided by the substructure. Replacing each occurrence of the substructure by a single conceptual entity reduces the complexity of the input examples. However, there is a gain in complexity associated with the conceptual definition of the new substructure. The net reduction in complexity is determined from the difference between these two measures of complexity. The reduction in complexity of the input examples can be computed as the number of occurrences of the substructure multiplied by the complexity of the substructure. Thus, the cognitive savings of a substructure, S, for a set of input examples, E, is computed as

$$\begin{aligned}\text{cognitive\_savings}(S,E) &= \text{complexity\_reduction}(S,E) - \text{complexity}(S)\\ &= [\text{number\_of\_occurrences}(S,E) * \text{complexity}(S)] - \text{complexity}(S)\\ &= \text{complexity}(S) * [\text{number\_of\_occurrences}(S,E) - 1]\end{aligned}$$

In the above computation of cognitive savings the complexity of the substructure is typically a function of the number of objects, the number of relations, and the arity of the relations in the substructure. The complexity of the substructure represents the cost of retaining the description of the substructure. If portions of the substructure have already been defined, the substructure complexity should reflect the reduced cost of retaining only the previously unknown parts of the substructure.

The number of occurrences of the substructure is more complicated to measure, because occurrences may overlap in the input examples. For instance, Figure 2.2 shows three input examples along with the substructure found by the discovery process; here, the circles represent objects and the lines represent relations. In Figure 2.2a the number of occurrences of the substructure in the example is obviously four. At first glance, the number of occurrences of the substructure in Figure 2.2b may appear to be four; however, the number of non-overlapping occurrences is less than four. Figure 2.2b illustrates the problem of object overlap; likewise, Figure 2.2c illustrates the problem of relation overlap. In view of the overlap problem, computation of the number of occurrences must reflect the number of unique occurrences.

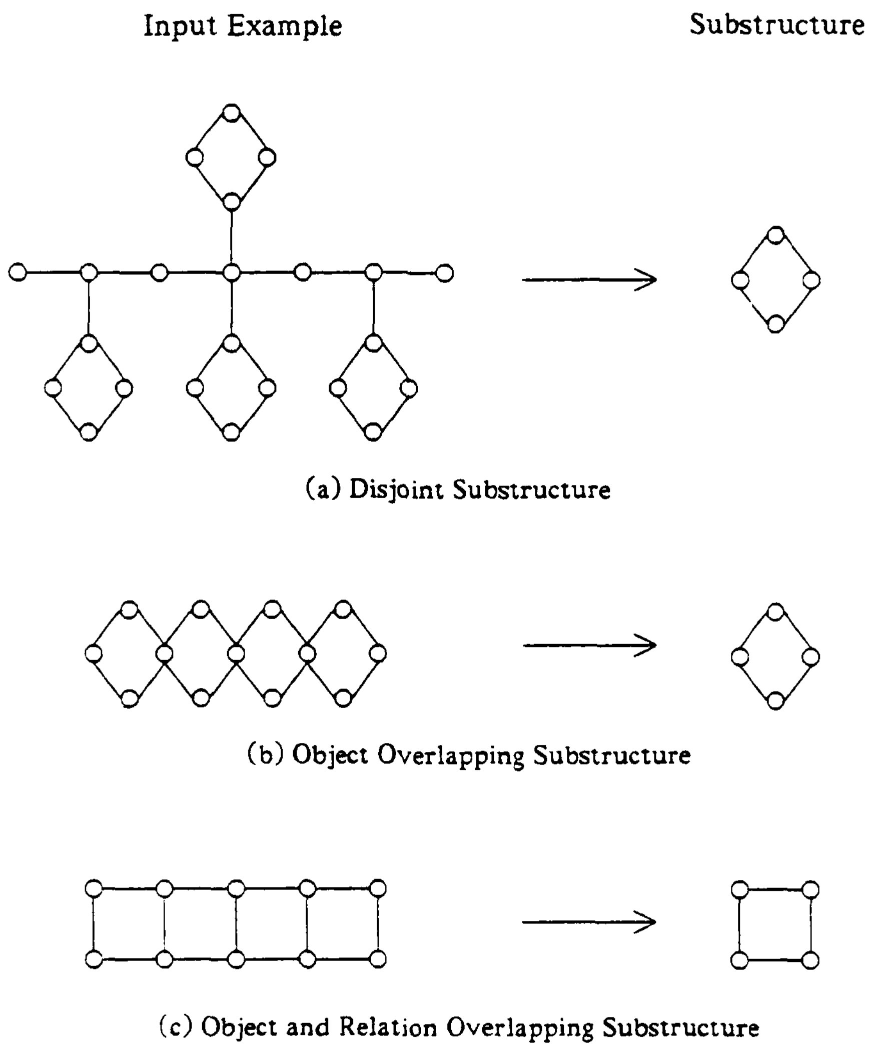

Input Example
Substructure
(a) Disjoint Substructure
(b) Object Overlapping Substructure
(c) Object and Relation Overlapping Substructure

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 generalization of Wertheimer's *Factor of Closure*, which states that human attention is drawn to closed structures [Wertheimer39]. A similar heuristic, also called compactness, is used in the vision literature to describe the density of a region in a visual image [Fu87]. Graphically, a closed substructure has at least as many relations as objects, whereas a non-closed substructure has fewer relations than objects [Prather76]. Thus, closed substructures have a higher compactness value.

Another heuristic that modifies the cognitive savings of a substructure is *connectivity*. This heuristic measures the amount of external connection in the occurrences of the substructure. The connectivity heuristic is a variant of Wertheimer's *Factor of Proximity* [Wertheimer39], and is related to earlier numerical clustering techniques [Zahn71]. These works demonstrate the human preference for "isolated" substructures; that is, substructures that are minimally related to adjoining structure. One method for computing the connectivity measures the "isolation" of a substructure by computing the average number of external connections over all the occurrences of the substructure in the input examples. The number of external connections is to be minimized; therefore, the connectivity value is computed as the inverse of the average to arrive at a value that increases as the number of external connections decreases.

The final heuristic modifying the cognitive savings is *coverage*. The coverage heuristic measures the amount of structure in the input examples described by the substructure. The coverage heuristic is motivated from research in inductive learning and provides that concept descriptions describing more input examples are considered better [Michalski83b]. Multiplying cognitive savings by coverage decreases the original cognitive savings only when the substructure describes a smaller amount of the input examples.

Other heuristics exist for evaluating and selecting substructures. One heuristic involves the use of background knowledge to recognize more promising substructures (see Section 2.7). Also, in the context of specialized substructures, other heuristics could measure the amount of specialization involved and incorporate this measure into the cognitive savings. In addition to the gestalt motivation for the previously discussed heuristics, gestalt theory suggests many additional 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.

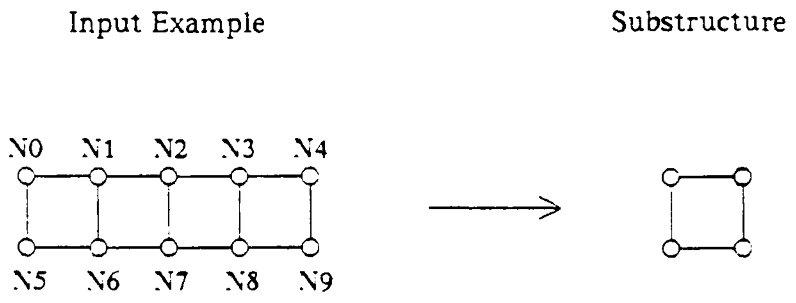
2.5. Substructure Instantiation

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 input example and replaced by the new relation. Other neighboring relations containing arguments that refer to objects in the occurrence are left unchanged. In this way the symbol equality between relation arguments preserves the original input example structure that is not directly described by the substructure. Unfortunately, by retaining all the objects in the occurrence as arguments to the new relation, the relation instantiation method loses the data compression gained by abstracting over the objects in the occurrences of the substructure.

To clarify the single relation substructure instantiation procedure, consider again the input example and substructure of Figure 2.2c redrawn in Figure 2.3a with appropriate object symbol assignments. Figure 2.3b shows the four new relations representing the four occurrences of the substructure in the input example. The relation name is chosen arbitrarily and is the same for each



(a) Example

[SUB-0005(N0,N1,N5,N6)=T][SUB-0005(N1,N2,N6,N7)=T]
[SUB-0005(N2,N3,N7,N8)=T][SUB-0005(N3,N4,N8,N9)=T]

(b) Instantiation

Figure 2.3. Substructure Instantiation Using New Relations

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.

Despite the generalization step in the previous technique, the substructures produced are structurally more specific than the original substructure. Appending a new relation to a substructure (whether or not the relation is generalized) adds a conjunct to the conjunctive description. Thus, the extended conjunctive description describes fewer occurrences over the space of possible input examples.

As an example of the substructure specialization technique, suppose the substructure discovery system discovers the substructure <[ON(A,B)=T]>, and several of the occurrences of this substructure have [SHAPE(A)=SQUARE] and [SHAPE(A)=CIRCLE] as neighboring relations. In this case, the specialization process produces the substructure <[ON(A,B)=T] [SHAPE(A)= CIRCLE,SQUARE]>.

After the originally discovered substructure undergoes the specialization process, both the original and specialized substructures are retained in the background knowledge of the substructure discovery system. The background knowledge may then identify known substructures in subsequent input examples. For instance, suppose the substructure resulting from a previous specialization, <[ON(A,B)=T][SHAPE(A)=SQUARE,CIRCLE]>, is identified by the background knowledge in the given input examples. Furthermore, suppose the neighboring relations of the occurrences of this substructure include the relations [COLOR(A)=RED], [COLOR(A)=BLUE] and [COLOR(A)=RED]. Then one possible specialization of the original substructure is <[ON(A,B]=T][SHAPE(A)=SQUARE,CIRCLE][COLOR(A)=RED,BLUE]>. With the possibility of repeated specialization, the substructure discovery system can learn increasingly specific disjunctive substructure concepts.

By specializing newly discovered substructures, the substructure discovery system learns both general and specific substructure concepts. Retaining these substructures in a background knowledge hierarchy allows the discovery system to identify previously learned substructure concepts in subsequent input examples and provides a more robust set of primitive substructures from which subsequent discovered substructures can be defined.

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 [illegible]ations at the leaves of the background knowledge hierarchy with the relations in the input e[illegible]ples. 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 background knowledge is used to determine the level of abstraction that is appropriate for processing the input sequence. PLAND's medium level background knowledge determines which groups of substructures within a given level of abstraction to process next. The low level background knowledge controls the type of substructures considered. Other details of the PLAND system can be found in Section 5.4.

Currently, only declarative substructure background knowledge is considered for retention in the hierarchy. Yet, there is an equally important and hierarchically organized body of knowledge involved in the retention and determination of substructure function. Chapter 6 discusses future efforts that will attempt to incorporate substructure function into the declarative background knowledge hierarchy.

2.8. Substructure Discovery Algorithm

This section presents a substructure discovery algorithm utilizing the techniques from previous sections. The algorithm is a computationally constrained best-first search guided by the substructure generation and selection techniques presented in sections 2.2 and 2.3. Figure 2.4 outlines the substructure discovery algorithm.

Initially, the algorithm is given one or more input examples and a limit, L, on the amount of computation performed. The algorithm begins by forming the set, S, of base substructures. In the situation where no background knowledge is present, the set of base substructures has only one element, the substructure corresponding to all single objects, with as many occurrences as there are objects in the input examples.[1] When background knowledge is present, this set of base substructures, S, may contain applicable background knowledge substructures in addition to the object substructure. As the algorithm progresses, the discovered substructures will be kept in the set, D, that is initially empty.

[1]In the VL_2 language for expressing these substructures, objects are represented as variables. However, a variable alone does not constitute a well-formed VL_2 expression. Although expressed as an arbitrary object name (i.e. OBJECT-0001), the single object substructure should be interpreted as an implicit relation expressing the existence of the object (i.e. [OBJECT(OBJECT-0001)=T]).

```
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.

Thus, the discovery algorithm is a straightforward implementation of the best-first search paradigm with a computational constraint. The power of the algorithm lies in the substructure generation methods and selection heuristics for choosing among alternative substructures. As subsequent examples will demonstrate, these heuristics perform well in guiding the search toward more promising substructures.

The next chapter describes the SUBDUE system. The SUBDUE system utilizes the substructure discovery algorithm along with substructure specialization and substructure background knowledge to form a robust substructure discovery system.

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.

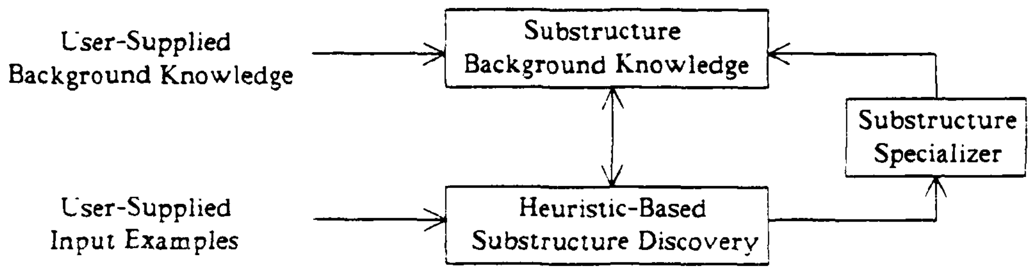

User-Supplied Background Knowledge
Substructure Background Knowledge
Substructure Specializer
User-Supplied Input Examples
Heuristic-Based Substructure Discovery

Figure 3.1. The SUBDUE System

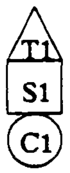
Figure 3.1 illustrates the interaction of the three modules. First, the user provides one or more input examples and optional background knowledge describing substructures that may apply to the examples. Next, the heuristic-based substructure discovery module asks the substructure background knowledge module for any known substructures that apply to the input examples. The discovery module uses the background knowledge substructures, if any, to find interesting substructures in the input examples. Upon exhausting the computational resources, SUBDUE passes the best substructure found to the substructure specialization module. This module appends additional attribute relations to the description of the discovered substructure. Finally, both the original and specialized substructure concepts are added to the substructure background knowledge. The user may then run the system on a new set of input examples, add additional background knowledge, or run the system again on the same examples, perhaps altering the computational limit. SUBDUE has options to deactivate one or more of the modules to investigate the operation of individual modules.

This chapter discusses the substructure representation used in SUBDUE and describes each of the three modules along with examples of their operation.

3.1. Substructure Representation

The SUBDUE system uses the substructure description language of Section 2.2 to communicate with the user. The internal representation of a substructure closely resembles a directed graph. Consider the substructure shown in Figure 3.2a. Figure 3.2b shows the external expression for this substructure, and Figure 3.2c shows the internal representation.

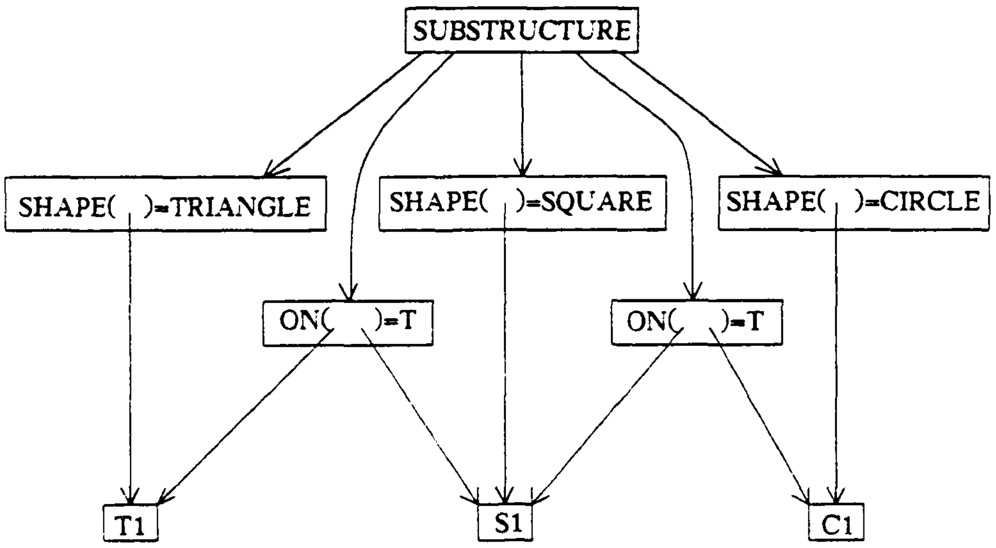
Internally, SUBDUE represents a substructure by the set of relations that comprise the substructure. Each relation contains the name of the relation, the value of the relation, the list of object arguments to the relation and the order-relevancy of the arguments. Objects are represented internally by the literal names used to specify the objects in the external representation. This substructure representation is expressed as a directed graph, as in Figure 3.2c, and facilitates a graph-theoretic comparison between substructures.



(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

Accompanying each internal substructure representation are the heuristic value of the substructure and a list of the occurrences of the substructure in the current set of input examples. Although not essential, keeping the occurrences together with the substructure greatly increases the efficiency of substructure generation, selection and specialization processes.

The dual substructure representations provide a convenient external substructure representation and an efficient internal representation. The external representation allows the user-supplied input examples, the user-supplied substructure background knowledge, and the substructures discovered by SUBDUE to be communicated in the same form. Converting this external representation into an internal, directed graph representation permits increased efficiency in many of the major operations performed by the SUBDUE system.

3.2. Heuristic-Based Substructure Discovery

The heuristic-based substructure discovery module in SUBDUE is an implementation of the substructure discovery algorithm presented in Section 2.8. This implementation uses the exhaustive minimal expansion technique discussed in Section 2.3 for generating alternative substructures. Selection from among the alternative substructures is accomplished by evaluating the substructures using the four heuristics described in Section 2.4: cognitive savings, compactness, connectivity, and coverage. Section 3.2.1 outlines SUBDUE's implementation of the minimal expansion technique for substructure generation, and Section 3.2.2 describes the computations performed during the evaluation of a substructure. A sample execution of the heuristic-based substructure discovery module is presented in Section 3.2.3.

3.2.1. Generation

The heuristic-based substructure discovery module uses the exhaustive minimal expansion technique from Section 2.3 for generating alternative expanded substructures from a single substructure. Recall from Section 2.3 that the minimal expansion technique expands a substructure by adding one neighboring relation.

```
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.

As defined in Section 2.4, the cognitive savings of a substructure, S, for a set of input examples, E, is computed as

$$\begin{aligned}\text{cognitive\_savings}(S,E) &= \text{complexity\_reduction}(S,E) - \text{complexity}(S)\\ &= [\text{number\_of\_occurrences}(S,E) * \text{complexity}(S)] - \text{complexity}(S)\\ &= \text{complexity}(S) * [\text{number\_of\_occurrences}(S,E) - 1]\end{aligned}$$

The *complexity(S)* is defined as the size of the substructure, S, where the size is computed as the sum of the number of objects and relations in the substructure. As discussed in Section 2.4, the *number_of_occurrences(S,E)* is more complicated to compute, because the occurrences may overlap in the input examples. Simply counting all objects and relations in the overlapping occurrences would incorrectly state the true cognitive savings of the substructure. Therefore, the *complexity_reduction(S,E)* is defined to be the number of objects and relations in the occurrences of the substructure, where overlapping objects and relations are counted only once. The number of such objects is referred to as *#unique_objects*, and the number of such relations is referred to as *#unique_relations*. Thus, the cognitive savings of a substructure S with occurrences OCC in the set of input examples E is computed as

$$\begin{aligned}\text{cognitive\_savings}(S,E) &= \text{complexity\_reduction}(S,E) - \text{complexity}(S)\\ &= [\#\text{unique\_objects}(OCC) + \#\text{unique\_relations}(OCC)] - \text{complexity}(S)\\ &= [\#\text{unique\_objects}(OCC) + \#\text{unique\_relations}(OCC)] - \text{size}(S)\\ &= [\#\text{unique\_objects}(OCC) + \#\text{unique\_relations}(OCC)] - [\#\text{objects}(S) + \#\text{relations}(S)]\end{aligned}$$

As an example of the cognitive savings calculation, consider the input examples and corresponding substructures in Figure 2.2. If each circle is considered an object and each line a relation, then for each of the three substructures, #objects(S) = 4, #relations(S) = 4 and there are four occurrences of the substructure in the input example. In Figure 2.2a, #unique_objects(OCC) = 16 and #unique_relations(OCC) = 16; thus, cognitive_savings = [16 + 16] - [4 + 4] = 24. In Figure 2.2b, #unique_objects(OCC) = 13 and #unique_relations(OCC) = 16; thus, cognitive_savings = [13 + 16] - [4 + 4] = 21. In Figure 2.2c, #unique_objects(OCC) = 10 and #unique_relations(OCC) = 13; thus, cognitive_savings = [10 + 13] - [4 + 4] = 15. Therefore, although the substructure is the same, 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 the occurrences of the substructure divided by the total number of objects and relations in the input examples. Thus, the coverage of a substructure S with occurrences OCC in the set of input examples E is computed as

$$\text{coverage}(S,E) = \frac{\#\text{unique\_objects}(OCC) + \#\text{unique\_relations}(OCC)}{\#\text{objects}(E) + \#\text{relations}(E)}$$

In Figure 2.2a there are 32 unique objects and relations in the occurrences of the substructure, and a total of 49 objects and relations in the entire example. Thus, coverage = 32/49. In both Figure 2.2b and Figure 2.2c the occurrences of the substructure describe every object and relation in the input example; thus, coverage = 1.

Ultimately, the *value* of a substructure S for a set of input examples E is computed as the product of the four heuristics.

$$\text{value}(S,E) = \text{cognitive\_savings}(S,E) * \text{compactness}(S) * \text{connectivity}(S,E) * \text{coverage}(S,E)$$

In this way the compactness, connectivity and coverage heuristics adapt the cognitive savings value to reflect specific qualities of the substructure. The *values* of the substructures in Figure 2.2 are

Figure 2.2a: *value* = 24 * 1 * 1 * 32/49 = 15.7
Figure 2.2b: *value* = 21 * 1 * 1/3 * 1 = 7.0
Figure 2.2c: *value* = 15 * 1 * 1/3 * 1 = 5.0

3.2.3. Example

In order to clarify the different operations of the heuristic-based substructure discovery module, this section traces a sample execution of the process. The execution proceeds according to the substructure discovery algorithm (Figure 2.4) utilizing the generation algorithm (Figure 3.3) and the heuristic evaluation function described in the previous section. Figure 3.4 shows the input example and the substructure that will eventually emerge. The input example is given to the system as:

<[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.

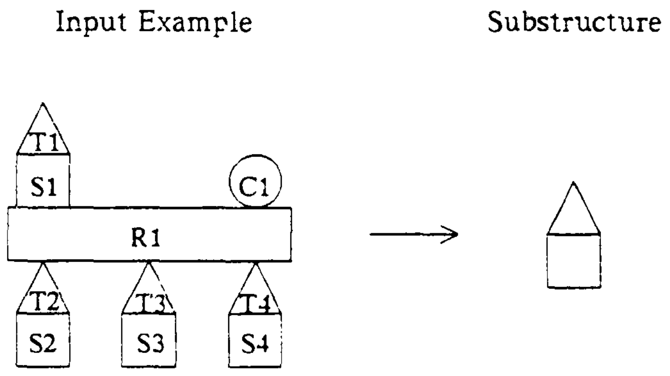


Figure 3.4. Simple Example

E = {<0.9 [SHAPE(OBJECT-0002) = TRIANGLE]> <1.3 [SHAPE(OBJECT-0002) = SQUARE]>
<0.0 [SHAPE(OBJECT-0002) = RECTANGLE]> <1.0 [ON(OBJECT-0003,OBJECT-0002) = T]>
<0.0 [SHAPE(OBJECT-0002) = CIRCLE]>}

Each substructure in E is added to S in order of decreasing value, and the loop repeats.

S = {<1.3 [SHAPE(OBJECT-0002) = SQUARE]> <1.0 [ON(OBJECT-0003,OBJECT-0002) = T]>
<0.9 [SHAPE(OBJECT-0002) = TRIANGLE]> <0.0 [SHAPE(OBJECT-0002) = RECTANGLE]>
<0.0 [SHAPE(OBJECT-0002) = CIRCLE]>}

In the second iteration of the loop, <1.3 [SHAPE(OBJECT-0002) = SQUARE]> is stored in BEST-SUB, removed from S and added to D. E is set to the minimal expansion of BEST-SUB.

E = {<0.0 [ON(OBJECT-0002,OBJECT-0005) = T][SHAPE(OBJECT-0002) = SQUARE]>
<3.3 [ON(OBJECT-0004,OBJECT-0002) = T][SHAPE(OBJECT-0002) = SQUARE]>}

Neither substructure in E has occurred before. Thus, both are added to S, and the loop repeats.

S = {<3.3 [ON(OBJECT-0004,OBJECT-0002) = T][SHAPE(OBJECT-0002) = SQUARE]>
<1.3 [SHAPE(OBJECT-0002) = SQUARE]> <1.0 [ON(OBJECT-0003,OBJECT-0002) = T]>
<0.9 [SHAPE(OBJECT-0002) = TRIANGLE]> <0.0 [ON(OBJECT-0002,OBJECT-0005) = T]
[SHAPE(OBJECT-0002) = SQUARE]> <0.0 [SHAPE(OBJECT-0002) = RECTANGLE]>
<0.0 [SHAPE(OBJECT-0002) = CIRCLE]>}

In the third iteration of the loop, <3.3 [ON(OBJECT-0004,OBJECT-0002) = T] [SHAPE(OBJECT-0002) = SQUARE]> is stored in BEST-SUB, removed from S and added to D. E is set to the minimal expansion of BEST-SUB.

E = {<0.0 [ON(OBJECT-0002,OBJECT-0007) = T][ON(OBJECT-0004,OBJECT-0002) = T]
[SHAPE(OBJECT-0002) = SQUARE]> <15.5 [SHAPE(OBJECT-0004) = TRIANGLE]
[ON(OBJECT-0004,OBJECT-0002) = T][SHAPE(OBJECT-0002) = SQUARE]>
<0.9 [ON(OBJECT-0006,OBJECT-0004) = T][ON(OBJECT-0004,OBJECT-0002) = T]
[SHAPE(OBJECT-0002) = SQUARE]>}

Again, none of the substructures in E have occurred previously; therefore, each substructure is added to S. The loop continues until the computational limit is exceeded or S becomes empty.

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

```
E = current set of input examples
S = description of substructure to be specialized
OCC = {occurrences of S in E}
SPECSUBS = {}
ATTRIBUTES = {attribute relations of OCC}
foreach REL in ATTRIBUTES do
  $S^* = S \wedge$ [REL(OBJ)=*]
  $OCC^*$ = {occurrences of $S^*$ in E}
  UNIQUE_VALUES = {unique values of all REL(OBJ) in $OCC^*$}
  $S_{spec} = S \wedge$ [REL(OBJ)=UNIQUE_VALUES]
  SPECSUBS = SPECSUBS $\cup$ $S_{spec}$
return SPECSUBS.
```

Figure 3.5. Substructure Specialization Algorithm

occurrences of S. Recall from Section 2.2 that an attribute relation of an occurrence is a neighboring relation of the occurrence whose only argument is an object contained in the occurrence. For each attribute relation, REL, in ATTRIBUTES, a new substructure, S^*, is constructed by adding a new attribute relation to the original substructure S. This new relation, [REL(OBJ)=*], is the same as REL except that the value slot of the relation is changed to a "don't care" value that matches any value. For example, if REL is [COLOR(T1)=RED], and T1 corresponds to object OBJECT-0001, then the attribute relation [COLOR(OBJECT-0001)=*] will be added to S and will match any other COLOR attribute, i.e., [COLOR(T2)=RED] or [COLOR(S1)=GREEN], etc.

After S^* is constructed, the occurrences of S^* in E are stored in OCC^*. Next, from the set of occurrences in OCC^*, the actual values of the REL attribute relation are uniquely collected in UNIQUE_VALUES. Then, the specialized substructure, S_{spec}, is constructed by adding to the original substructure S an attribute relation having the same name as REL and UNIQUE_VALUES as the disjunctive list of relation values. Finally, if not already in SPECSUBS, S_{spec} is added to SPECSUBS.

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

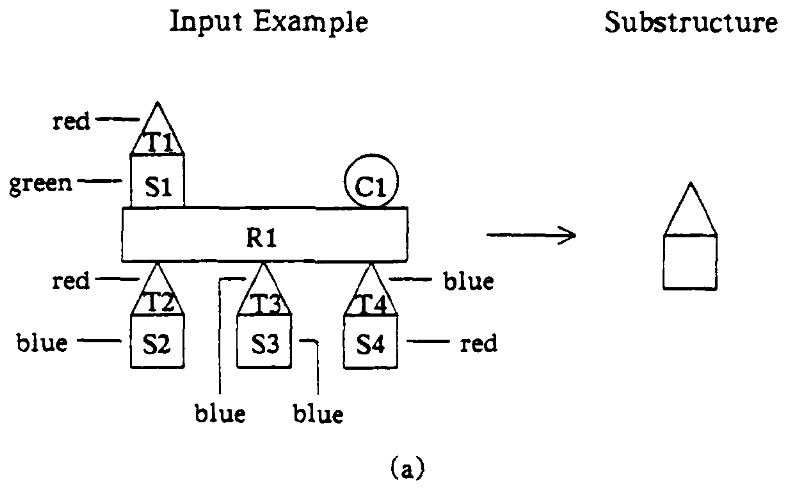
As an example of the substructure specialization process, consider Figure 3.6. Figure 3.6a illustrates the same input example of Figure 3.4 with the addition of several *color* attribute relations. After running the heuristic-based substructure discovery algorithm on this example, the same substructure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



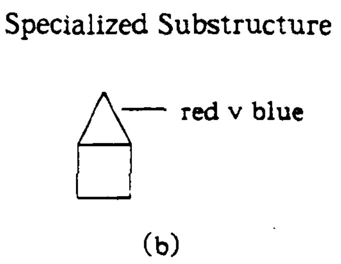


Figure 3.6. Substructure Specialization Example

substructure S that corresponds to the object T1 in the occurrences of the substructure.

S' = <[COLOR(OBJECT-0002)=*][SHAPE(OBJECT-0002)=TRIANGLE]
[SHAPE(OBJECT-0001)=SQUARE][ON(OBJECT-0002,OBJECT-0001)=T]>

The four occurrences of S' are stored in OCC'. and the unique values of their *color* relations are collected in UNIQUE_VALUES.

UNIQUE_VALUES = {RED.BLUE}

Thus, the following specialized substructure is added to SPECSUBS:

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

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

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

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 substructures occur in a given set of input examples. The substructures are stored in a hierarchy, where complex substructures are defined in terms of their more primitive subparts. By storing user-defined substructures, the background knowledge allows the user to contribute prior knowledge of the application domain. Storing discovered substructures makes SUBDUE a closed-loop system in which the knowledge grows incrementally and hierarchically as new substructures are discovered. Each execution of the heuristic-based discovery module queries the background knowledge for substructures occurring in the current set of input examples. Section 3.4.1 describes the architecture of the substructure background knowledge, and Section 3.4.2 outlines the method for identifying background knowledge substructures in a given set of input examples.

3.4.1. Architecture

As suggested in Section 2.7, the architecture of SUBDUE's substructure background knowledge is modeled after a variant of the truth maintenance system called the assumption-based truth maintenance system (ATMS) [de Kleer86]. There are two reasons for choosing the ATMS to model the substructure background knowledge. First, the justification network in an ATMS captures the hierarchical representation inherent in substructures. Second, the maintenance of several environments in which a node is supported allows not only the determination of substructure existence, but provides the occurrences as well. This will become clear in the next section.

In the ATMS used by SUBDUE, there are two types of nodes: base nodes and substructure nodes. The *base nodes* are the leaves of the hierarchical background knowledge tree. A base node represents a single relation with a unique values list. Thus, [ON(X,Y)=T], [COLOR(X)=RED] and [COLOR(X)=RED,BLUE] would be different base nodes in the hierarchy. At the lowest level in the hierarchy, these base nodes serve as justifications for the higher level substructure nodes. Each base node may support any number of substructure nodes at any level in the hierarchy. A *substructure node* represents a substructure formed by adding one base node relation to either a more primitive substructure node or another base node relation. Thus, each substructure node has

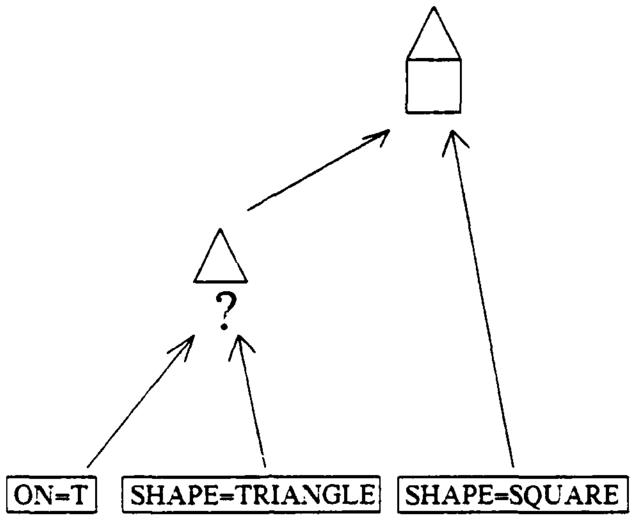


Figure 3.7. Substructure Background Knowledge Example

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

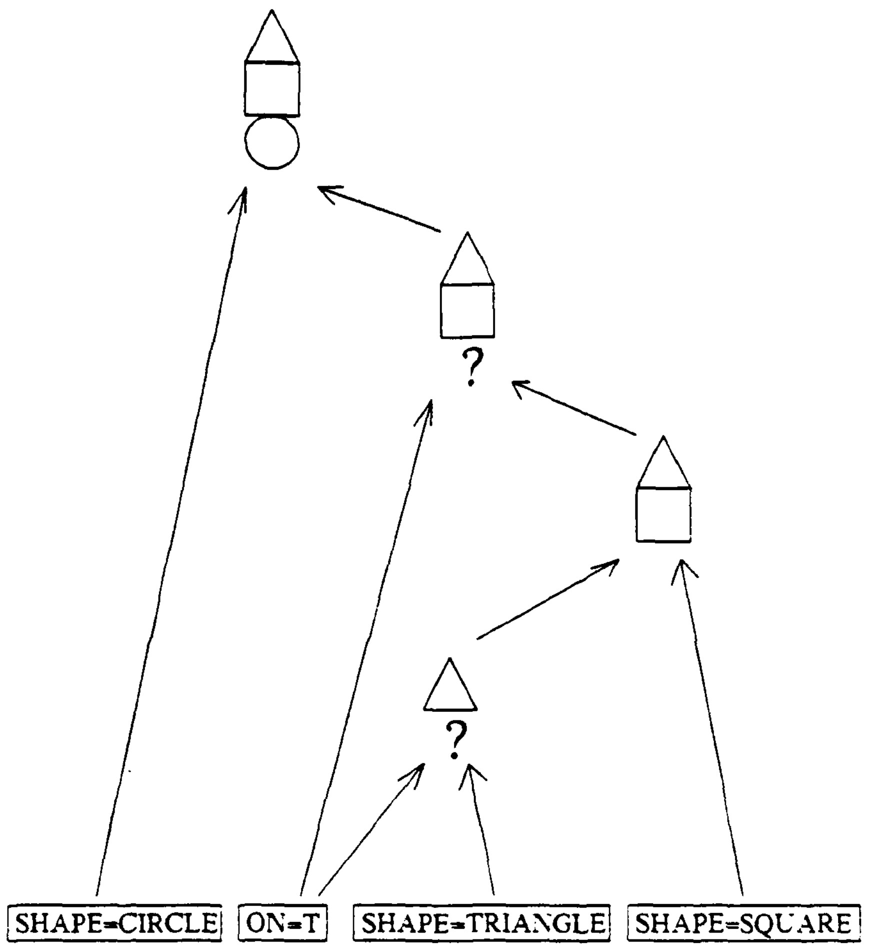


Figure 3.8. Two Background Knowledge Substructures

substructure and is used as support for the new substructure. Again, a question mark is used to represent the object argument to the *on* relation, because the object does not have a *shape* attribute at that level of the hierarchy.

As a final example, Figure 3.9 shows the resulting hierarchy after adding the specialized substructure from Figure 3.6b to the hierarchy of Figure 3.8. As new substructures are added by

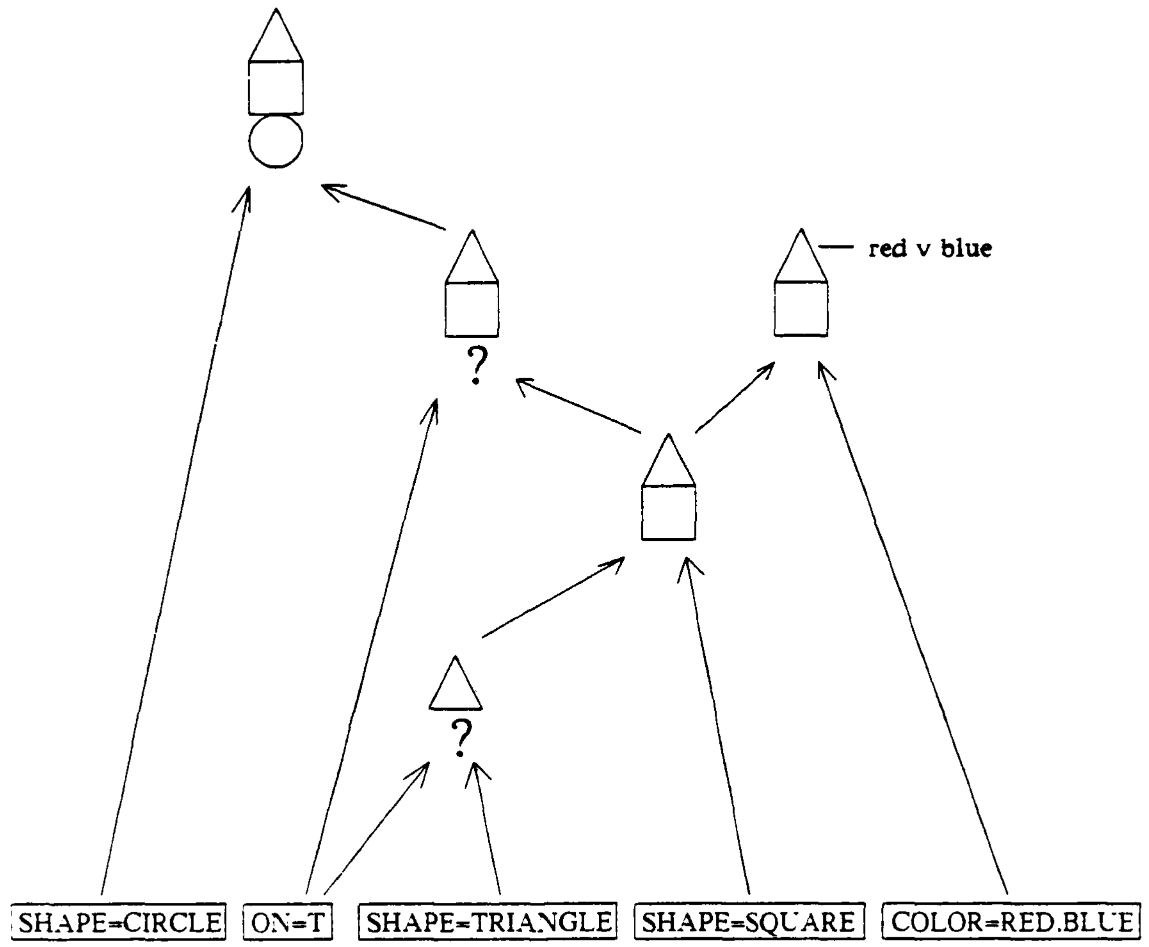


Figure 3.9. Three Background Knowledge Substructures

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.

The identification process described in this section determines which background knowledge substructures occur in the input examples. Furthermore, the identification process also finds each occurrence of the identified substructures. The method of identification used by SUBDUE is illustrated in this section by identifying the substructures in a simple example.

Suppose the current state of the background knowledge is as shown in Figure 3.7, and the background knowledge substructures occurring in the input example of Figure 3.4 are to be identified. First, all the relations in the input example are used as support for the corresponding base node relations in the background knowledge hierarchy. For example, each [SHAPE(X)=TRIANGLE] supports the SHAPE=TRIANGLE base node. Relations having no corresponding base node (e.g., [SHAPE(C1)=CIRCLE]) are ignored. The base node supports resulting from the current example are shown in Figure 3.10.

After the base node supports are constructed, an ATMS-like propagation begins. The only difference between the normal ATMS propagation and that used by SUBDUE's background knowledge is that instead of assigning the entire cross product of the two justifications to the supported substructure node, only the combinations resulting in a valid, connected substructure for that substructure node are retained. For instance, the "triangle on top of something" substructure node in Figure 3.10 is not assigned [ON(C1,R1)=T][SHAPE(T1)=TRIANGLE], because the resulting substructure is not connected. Similarly, the substructure node is not assigned [ON(R1,T2)=T][SHAPE(T2)=TRIANGLE], because the resulting substructure is not a valid instance of the substructure represented by this node. Once the propagation completes, the top level substructure node will be assigned the set of occurrences identified in the input example. The final substructure node assignments are also shown in Figure 3.10.

The ATMS architecture of SUBDUE's substructure background knowledge provides a hierarchical representation and a simple propagation technique for identifying the occurrences of the background knowledge substructures in the current set of input examples. With the substructure background knowledge module, not only can SUBDUE utilize user-supplied

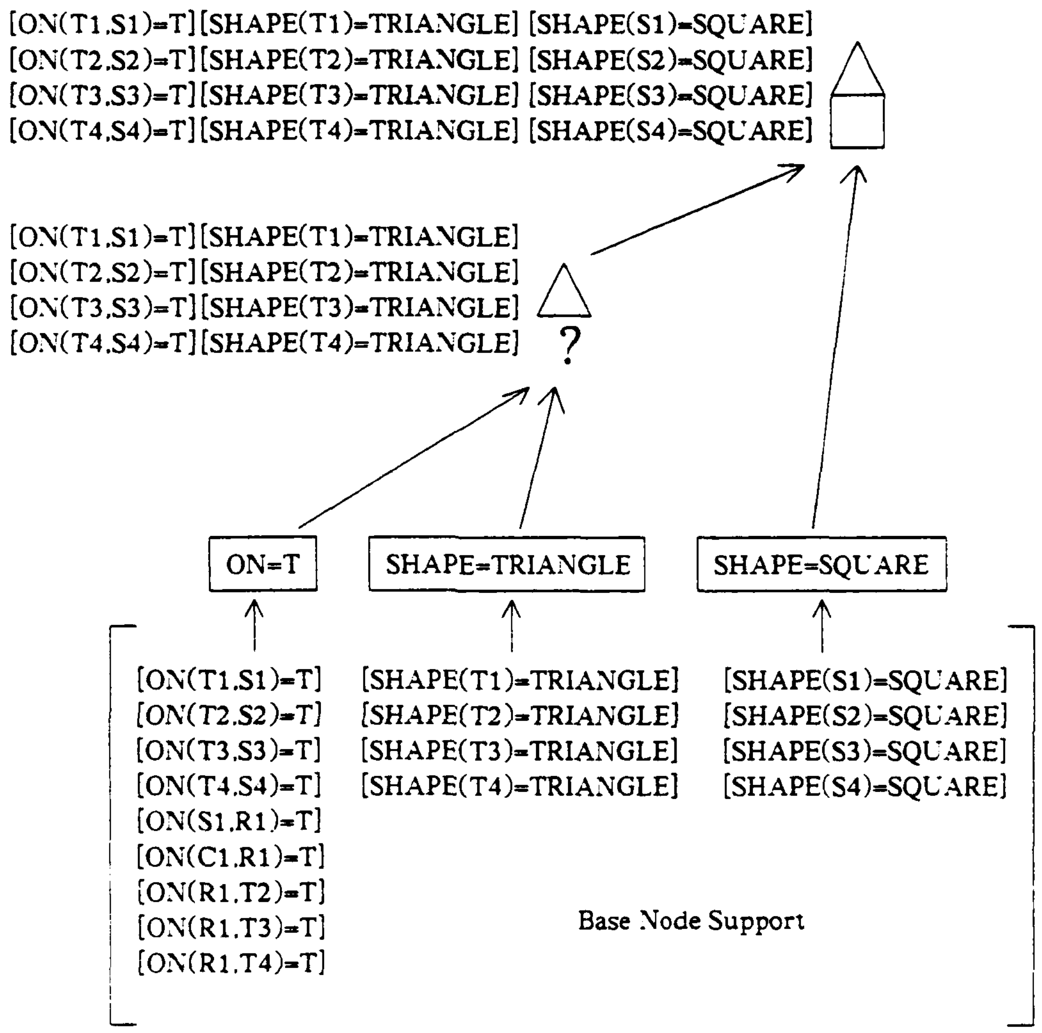

[ON(T1,S1)=T] [SHAPE(T1)=TRIANGLE] [SHAPE(S1)=SQUARE]
[ON(T2,S2)=T] [SHAPE(T2)=TRIANGLE] [SHAPE(S2)=SQUARE]
[ON(T3,S3)=T] [SHAPE(T3)=TRIANGLE] [SHAPE(S3)=SQUARE]
[ON(T4,S4)=T] [SHAPE(T4)=TRIANGLE] [SHAPE(S4)=SQUARE]
[ON(T1,S1)=T] [SHAPE(T1)=TRIANGLE]
[ON(T2,S2)=T] [SHAPE(T2)=TRIANGLE]
[ON(T3,S3)=T] [SHAPE(T3)=TRIANGLE]
[ON(T4,S4)=T] [SHAPE(T4)=TRIANGLE]
?
ON=T
SHAPE=TRIANGLE
SHAPE=SQUARE
[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]
[SHAPE(T1)=TRIANGLE]
[SHAPE(T2)=TRIANGLE]
[SHAPE(T3)=TRIANGLE]
[SHAPE(T4)=TRIANGLE]
[SHAPE(S1)=SQUARE]
[SHAPE(S2)=SQUARE]
[SHAPE(S3)=SQUARE]
[SHAPE(S4)=SQUARE]
Base Node Support

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.

CHAPTER 4

EXPERIMENTS

This chapter presents several experiments that demonstrate SUBDUE's ability to discover substructure in examples, specialize the substructure and utilize the substructure in subsequent discovery tasks. In the experiments, the only user-variable parameter to the system is the computational limit mentioned in Section 2.8. The computational limit represents the maximum number of substructures considered by the heuristic-based discovery module. In other words, the computational limit is the maximum number of iterations allowed for the *while loop* in the substructure discovery algorithm of Figure 2.4. Unless explicitly stated, the computational limit is assumed to be half the number of relations in the current set of input examples.

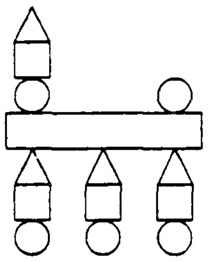
Each experiment is run on a Texas Instruments Explorer. SUBDUE's input and output data for the experiments of this chapter are given in Appendix A.

4.1. Experiment 1: Varying the Computational Limit

For a given set of examples, the number of substructures considered by the substructure discovery algorithm depends on the computational limit imposed on the discovery process. Increasing the computational limit allows the algorithm to consider an increasing number of alternative substructures and improves the chance that the heuristically best substructure is generated. Experiment 1 demonstrates the effects of varying the computational limit. The results obtained by the algorithm on two examples are presented.

Figure 4.1 and Figure 4.2 show the results of the two examples in Experiment 1 run with four different values of the computational limit. As in the illustrations of the background knowledge in Section 3.4.1, the question marks in Figure 4.1 represent object arguments to the *on*

Input Example:



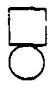
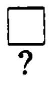
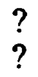
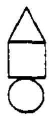
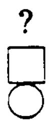
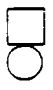
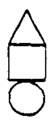
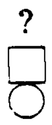
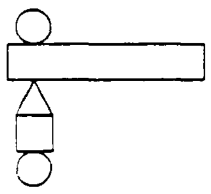
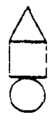
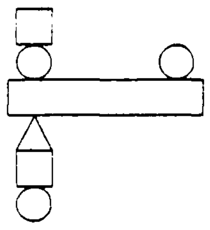
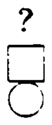
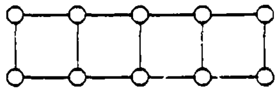
Computation Limit	Three Best Substructures Discovered		
4	 Value(S)=8.8	 Value(S)=2.1	 Value(S)=1.7
6	 Value(S)=31.2	 Value(S)=9.6	 Value(S)=8.8
10	 Value(S)=31.2	 Value(S)=9.6	 Value(S)=9.2
14	 Value(S)=31.2	 Value(S)=10.3	 Value(S)=9.6

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.

Figures 4.1 and 4.2 indicate an important quality of the heuristic evaluation function that applies to most of the examples processed by SUBDUE: the heuristics are well-behaved. The heuristics prevent the best-first search from straying too far from the path towards the substructure with the highest overall heuristic value. The best substructure discovered with a computational limit of six in both Figure 4.1 and Figure 4.2 is the best substructure, according to the heuristics, of all possible substructures in the input example.

Input Example: 

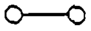
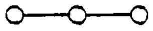
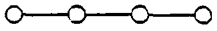
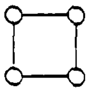
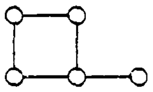
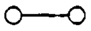
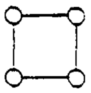
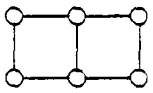
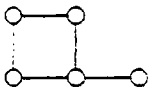
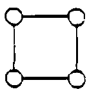
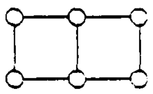
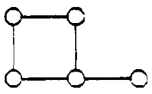
Computation Limit	Three Best Substructures Discovered		
4	 Value(S)=3.0	 Value(S)=2.8	 Value(S)=2.6
6	 Value(S)=5.0	 Value(S)=3.2	 Value(S)=3.0
10	 Value(S)=5.0	 Value(S)=4.4	 Value(S)=3.2
14	 Value(S)=5.0	 Value(S)=4.4	 Value(S)=3.2

Figure 4.2. Second Example of Experiment 1

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.

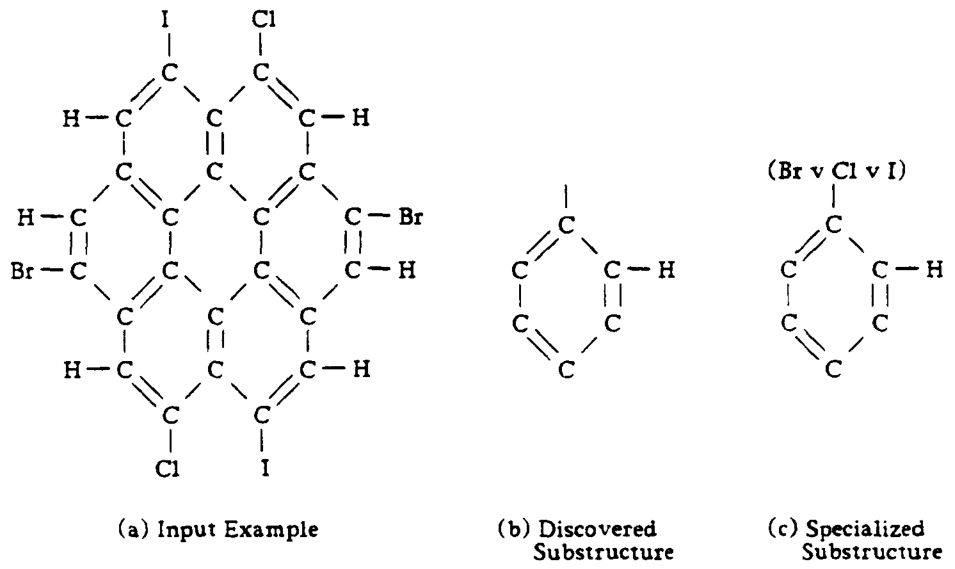


Figure 4.3. First Example for Experiment 2

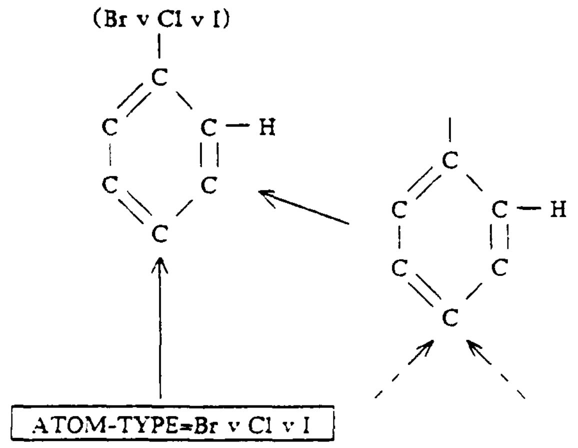


Figure 4.4. Background Knowledge Hierarchy After First Example

The second example is shown in Figure 4.5a. The example describes a derivative of the compound Triphenylene. The substructure background knowledge finds occurrences of both 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.

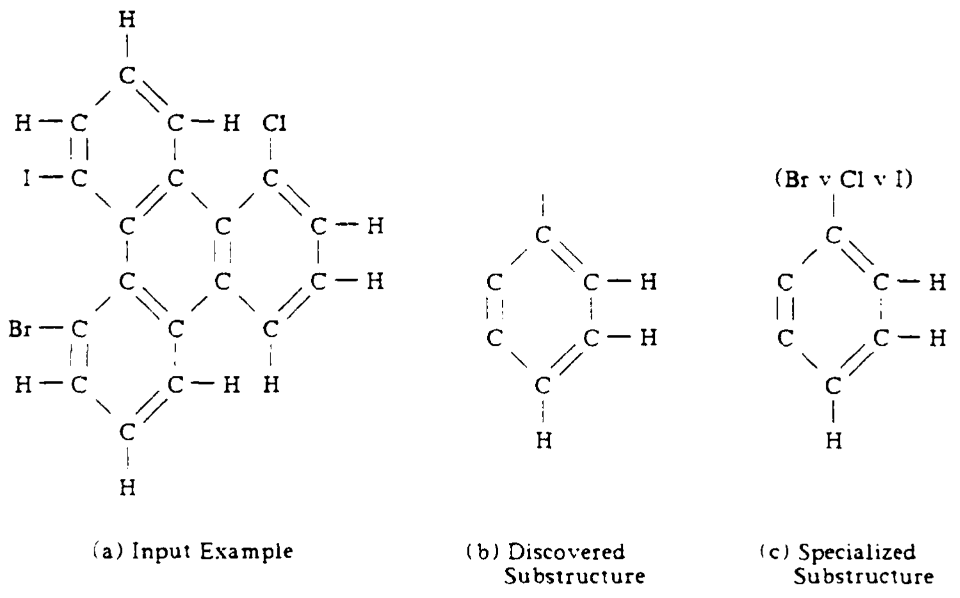


Figure 4.5. Second Example for Experiment 2

The background knowledge hierarchy containing the four substructures of this experiment is shown in Figure 4.6. As in Figure 4.4, the dashed arrows in Figure 4.6 represent the background knowledge hierarchy defining the discovered substructure of Figure 4.3b.

This experiment demonstrates SUBDUE's ability to utilize previously discovered substructures in subsequent discovery tasks. Without background knowledge, SUBDUE discovers the substructure of Figure 4.5b in the second example after considering 29 substructures. With the background knowledge, SUBDUE discovers the same substructure after considering only 5 substructures As knowledge of a domain increases, SUBDUE can discover more complex substructures in terms of the substructures already known. For each new example, SUBDUE applies the known substructures to the example and incorporate the resulting discovered substructures into the background knowledge hierarchy.

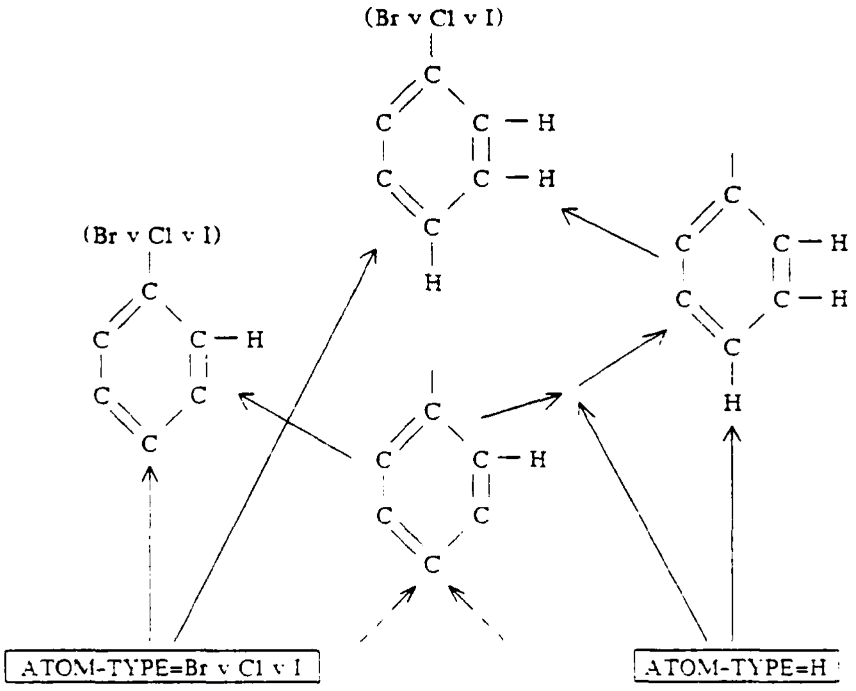


Figure 4.6. Background Knowledge Hierarchy After Second Example

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

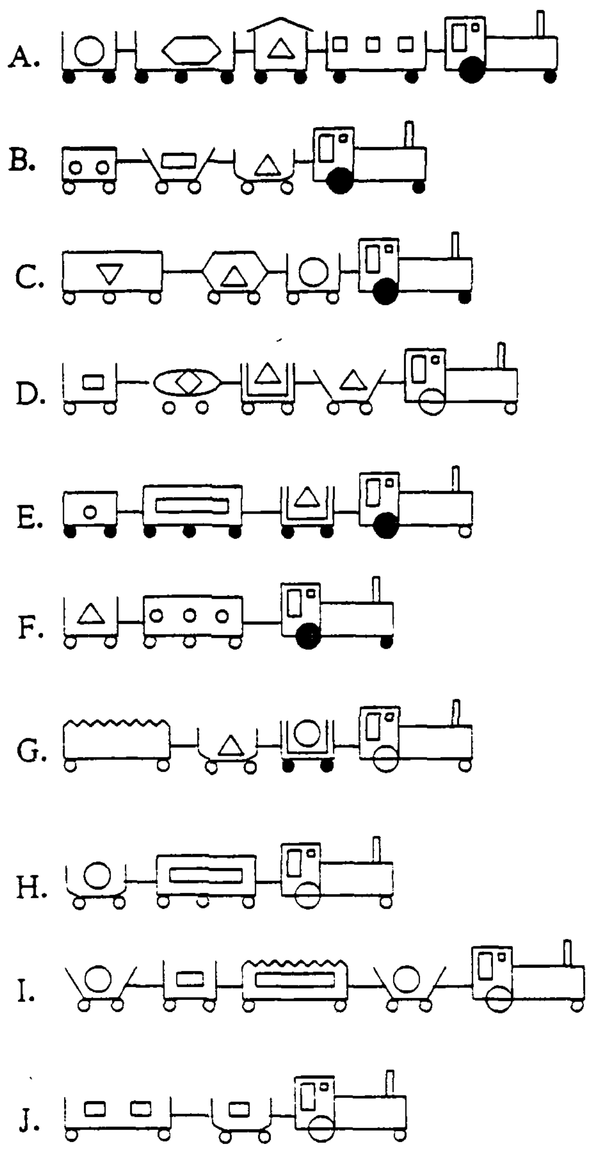


Figure 4.7. Example Trains for Experiment 3

following attributes: *car-shape*, *car-length*, *wheel-color*, *load-shape* and *load-number*. See Appendix A for the actual input specification for this experiment.

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

GDN. If CLUSTER/CA were able to use a heuristic like cognitive savings to augment the GDN with problem-specific attributes, discovering such clusterings would be easier and perhaps more efficient than the combination of the two systems. Using SUBDUE to focus a conceptual clustering system like CLUSTER/CA can produce better results than a system with less direction towards relevant attributes.

Substructure discovery in SUBDUE represents a method for suggesting new attributes on which to focus the conceptual clustering process. In this way, SUBDUE allows other machine learning classification systems to discover novel concepts based on attributes that may not have been considered by the learning system alone.

4.4. Experiment 4: Discovering Macro-Operators in Proof Trees

Experiment 4 illustrates one possible application of SUBDUE to other work in machine learning and planning. The application demonstrated by Experiment 4 is related to the task of discovering macro-operators in plans.

There is much related research on learning macro-operators from plans. In the STRIPS program [Fikes72], once a plan is constructed to perform a given task, the plan is retained as a macro-operator for use in future planning. By storing the macro-operator in a *triangle table*, subsequences within the original plan are also available as macro-operators. The *chunking* mechanism of SOAR [Laird86] offers another method for learning macro-operators. After a problem is solved, the proof tree used to solve the problem is retained, or *chunked* for use in future problem solving. If the proof tree involves previously learned *chunks*, the new *chunk* is defined hierarchically in terms of the old *chunks*. Explanation-based learning (EBL) [DeJong86, Mitchell86] provides a third example of research related to learning macro-operators. In EBL, a proof tree is generated that proves an example is an instance of the goal concept to be learned. This proof tree is generalized and retained as a macro-operator, or *schema* for use in future learning. As in SOAR, *schemas* can be defined hierarchically in terms of other *schemas*.

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:

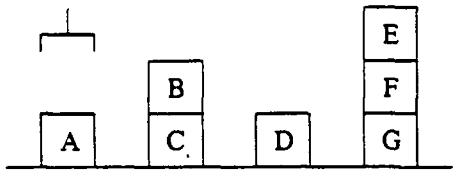
PICKUP(x)
Preconditions: ONTABLE(x), CLEAR(x), HANDEMPTY
Add: HOLDING(x)
Delete: ONTABLE(x), CLEAR(x), HANDEMPTY

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)

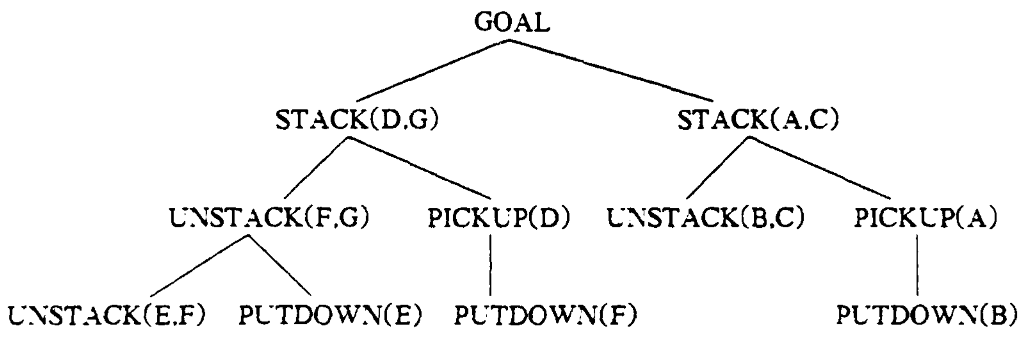
For this example, suppose the initial world state is as shown in Figure 4.8a, and the desired goal is in Figure 4.8b. The proof tree of operators to achieve the goal is shown in Figure 4.8c. With this proof tree as input, SUBDUE discovers the substructure shown in Figure 4.9. The



(a) Initial World State

[ON(A,C)][ON(D,G)]

(b) Goal



(c) Proof Tree

Figure 4.8. Proof Tree Example for Experiment 4

substructure represents a macro-operator for accomplishing a subgoal to stack a block X on another block Z when a block Y is already on top of block Z.

The macro-operators discovered by SUBDUE can be used in several ways. Replacing the occurrences of the macro-operator in the original proof tree by instantiations of the macro-operator can reduce the storage requirements of the schema constructed from the entire proof tree. Retaining the macro-operators might improve the performance of the explanation process in an EBL

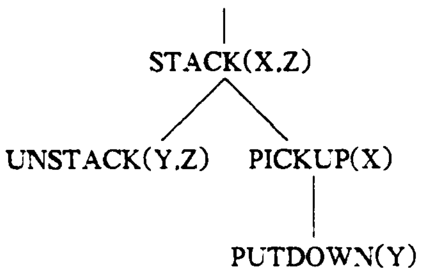


Figure 4.9. Discovered Macro-Operator

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.

All six examples were given to SUBDUE using the same 178 relations. After considering seven alternative substructures for 15 seconds of processing time, SUBDUE discovers the substructure concept of a benzene ring as shown on the left side of Figure 4.10c. The newly discovered substructure concept is then used to reduce the complexity of the original examples by replacing each occurrence of the benzene ring with a single relation, i.e., [BENZENE-RING(C1,C2,C3,C4,C5,C6)=T]. Using the reduced set of positive and negative examples, INDUCE then produces the concept on the right side of Figure 4.10c in 38 seconds of processing time. In Figure 4.10c the symbolic benzene rings represent the BENZENE-RING relation, not the complex structural representation used in the original descriptions of the examples.

By abstracting over the structure representing the benzene ring, SUBDUE allows INDUCE to discover the desired concept distinguishing the positive and negative examples: benzene rings are paired across one carbon atom in the positive examples, but not in the negative examples. INDUCE represents this concept in terms of the high-level benzene ring feature provided by SUBDUE. Furthermore, the processing time of SUBDUE and INDUCE combined (53 seconds) represents a

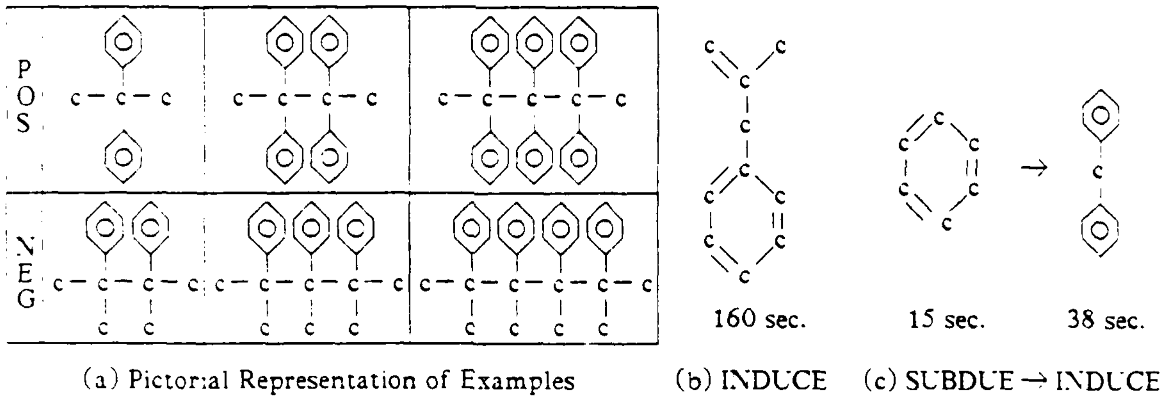


Figure 4.10. SUBDUE/INDUCE Example

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.

CHAPTER 5

RELATED WORK

This chapter presents related work on discovering substructure in examples. Describing examples in terms of their subparts suggests a gestalt approach to substructure discovery. Gestalt theory motivates several of the ideas behind the substructure discovery algorithm. In the area of machine learning, Winston's ARCH program contains mechanisms for discovering groups of objects in examples. More recent work in machine learning related to substructure discovery includes Wolff's SNPR program for language acquisition and Whitehall's PLAND system for discovering substructure in action sequences.

5.1. Gestalt Psychology

Many of the ideas in this thesis originated from work in gestalt psychology [Kohler47]. Gestalt theory identifies several underlying cognitive processes that humans use to perceive structure in a visual scene. In particular, two of the heuristics used for substructure evaluation (see Section 2.4) are derived from Wertheimer's *Principles of Organization* [Wertheimer39]. The compactness heuristic is a generalization of Wertheimer's *Factor of Closure*, and the connectivity heuristic is derived from Wertheimer's *Factor of Proximity*. Other research has shown that gestalt theory, particularly Wertheimer's principles, applies successfully to problems in the visual domain. For instance, Narasimhan proposes a *Syntactic Model* that applies gestalt phenomena to the analysis and description of pictures [Narasimhan63]. The model is used to detect alphabetic characters in pictures and linear elements in bubble chamber negatives. Also, Tuceryan and Ahuja apply results from the gestalt theory to the problem of finding perceptual structure in dot patterns [Tuceryan87]. Other psychological theories elaborate on the gestalt theory to arrive at more detailed explanations for the perceptual grouping abilities exhibited by humans [Treisman82]. The 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

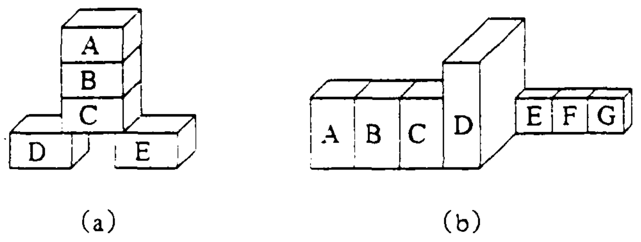


Figure 5.1. Sequence Termination Conditions

belong in the sequence. The revision process removes such objects according to three rules: terminate chains at junction points, break chains at size differences, and break chains at nonlinearities. As an example of the first rule, consider Figure 5.1a. Two sequences of SUPPORTED-BY relations are conjectured: A-B-C-D and A-B-C-E. However, the junction point at C causes the program to remove D and E from the sequences, resulting in one final sequence, A-B-C. Figure 5.1b demonstrates the applicability of the last two rules. The sequence of seven objects, A-B-C-D-E-F-G, connected by IN-FRONT-OF relations is broken into two sequences, A-B-C and E-F-G, because the objects of the original sequence are not collinear and differ in size. Upon completion of the revision process, the ARCH program uses the remaining sequences to describe the groups of objects as single concepts.

When searching for groups of objects with a common relation to another object, the ARCH program generates groups based on one common relation and then removes objects from the group to maintain a homogeneous set of objects. As an example of the procedure, consider Figure 5.2. The procedure begins by forming the *common-relationships-list*, a list of all relations possessed by more than half of the objects in the group. Objects A through E are considered as a possible group because they all possess a SUPPORTED-BY relation to object F. The relations of the candidate objects are:

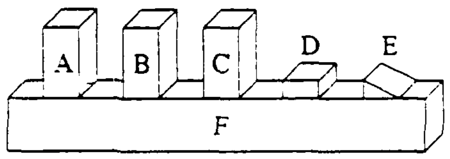


Figure 5.2. Common Relations Example

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

Next, the procedure removes those objects having a comparison measure less than 80 percent of the highest value. Thus, D and E are removed from the set of candidates. Object D is removed because of its uncommon size, and object E is removed because of its uncommon size and shape. The evaluation procedure repeats until none of the objects are eliminated from the candidate set. In the example of Figure 5.2, the resulting set contains objects A, B and C.

The substructure discovery procedure used by the ARCH program differs from SUBDUE in several ways. First, the methods employed by the ARCH program are designed specifically for the blocks world domain. For instance, the sequence discovery method looks for SUPPORTED-BY and IN-FRONT-OF relations only, and the sequence termination conditions depend on discrepancies in particular relations such as size and shape. SUBDUE's substructure discovery method is domain independent (assuming the domain is representable in first order predicate calculus).

Second, the ARCH program's method for discovering objects with common relations to another object represents a different approach to discovering such substructure. The ARCH program begins with a prototype object whose relations are kept on the *common-relationships-list*. The prototype object is modified as some objects are eliminated from consideration, while relations from other objects become more common in the remaining candidate objects. However, this procedure is somewhat arbitrary due to the *more than half* and *80 percent threshold* criteria used to add relations and drop objects, respectively. Alternatively, SUBDUE relies solely on the heuristic value of a substructure as the measure of "interestingness." Maintaining a prototype substructure is a more intelligent method for discovering an appropriate substructure; however, domain knowledge must be present to suggest such prototypes. SUBDUE simulates the use of prototypes by suggesting background knowledge substructures, but more work is needed to incorporate other types of domain knowledge for suggesting appropriate substructure prototypes.

Third, the ARCH program discovers substructures containing only one type of object. The objects in a sequence must all have the same size or shape. Although the objects in a grouping are not constrained to be the same type, the ARCH program prefers groupings whose objects are the 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 process: (1) formation of AND groupings, (2) formation of OR groupings, (3) choosing among groupings according to frequency and size, (4) recursion, (5) generalization, and (6) schema plus correction. Wolff's SNPR program [Wolff82] embodies all but the sixth principle. The third data compression principle plays a central role in SNPR by selecting the conjunctions, disjunctions, recursive structures and generalized structures that form a knowledge structure having maximum data compression capacity. Maximizing data compression is the same idea behind the cognitive savings heuristic employed by SUBDUE. This section briefly describes the SNPR program and compares the heuristic approaches of SNPR and SUBDUE.

The SNPR program utilizes the first five data compression principles to learn grammars from input texts. SNPR forms conjunctive (AND) groupings, or chunks, by detecting frequently used substrings within the input text. For example, the text, ABPQRABABPQRAB, may be reduced to *xyxxyx*, where $<x \rightarrow AB>$ and $<y \rightarrow PQR>$. Likewise, SNPR learns grammars with disjunctive (OR) groupings, such as $<x \rightarrow$ JOHNyMARY$>$ and $<y \rightarrow$ LOVES | HATES$>$ (where "|" means exclusive OR). These groupings are chosen according to their size and frequency within current and previous input texts. SNPR is capable of learning recursive grammars, such as $<x \rightarrow ABx>$. Generalization is accomplished by replacing the elements of an OR grouping with a disjunction of the elements. For example, the OR grouping <THE | ONE> would replace instances of <THE> and <ONE> in previously formed structures. Overgeneralizations are corrected through a process called *rebuilding* that removes elements from disjunctive structures when the particular instance was never seen. For instance, if the substring ABRCD was never seen in the input text, then the structure $<ABxCD>$, $<x \rightarrow P \mid Q \mid R>$ is reconstructed as $<AByCD>$, $<y \rightarrow P \mid Q>$.

The goal of the SNPR program is to find a grammar that generates the input text and provides the maximum amount of data compression. Wolff measures the data compression provided by a grammar by considering the amount of data compression provided by each production rule, or element, in the grammar. This measure is called the *compression value* (CV_e) of the element and is defined as

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

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 [Whitehall87] 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 heuristic to select among alternative substructures, and both systems utilize previously discovered substructures in subsequent discovery tasks. This section discusses the PLAND system and analyzes these similarities in more detail.

Input to the PLAND system consists of a sequence of primitive actions. Each action is related to the next action by a *follows* relation. The input sequences can be a list of complex actions, such as MOVE, PICKUP, PUTDOWN, or simply a list of letters similar to the input of Wolff's SNPR program described in the previous section. As mentioned above, PLAND searches the input for three types of substructure: sequences, loops and conditionals. Sequences are blocks of actions that appear repetitively throughout the input sequence. PLAND maintains sequences as partial macrops and promotes them to complete macrops only upon recommendation from background knowledge. Loops are sequences of actions appearing consecutively in the input sequence. Loops are expressed in a formal grammar syntax. For example, the loop in the input sequence, ABCBCEBCBCBCF, is expressed as (BC)*. Conditionals allow a choice of actions within a macrop. For example, within the input sequence, ABDCBECFBECBDCBDCG, PLAND discovers the macrop (B(D+E)C)*. Once primitive macrops are discovered, PLAND can discover more complex macrops in terms of these primitive macrops and construct a hierarchical representation of the input sequence.

PLAND processes the input sequence at different levels of abstraction (generalization), called *contexts*. For instance, the original input sequence is a context. After PLAND discovers macrops in the original input sequence, the sequence of actions described by the best macrops are replaced by single forms representing the macrops. These macrops act as single actions in another, more abstract context. Within a context PLAND considers alternative macrops according to *agendas*. An agenda contains information about where to look for a macrop within the input sequence.

At each level of processing, PLAND uses background knowledge to guide the search for macrops. High level background knowledge is used to determine which context to process next. Medium level background knowledge determines which agenda to process next. Low level background knowledge controls the type of macrops considered by PLAND. These background 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

Figure 5.3 demonstrates the results obtained by PLAND on a simple action sequence. In this example PLAND is run without background knowledge. The naming convention for the macrops, $M_{c,n}$, indicates that the macrop is the *nth* macrop discovered in context *c*. Within the first context, PLAND discovers three loop macrops: $M_{1,1} = (X)^*$, $M_{1,2} = (Y\ M_{1,1}{}^*)^*$, and $M_{1,3} = (M_{1,2}{}^*\ Z)^*$. Notice how PLAND builds a hierarchical structure by discovering new macrops in terms of recently discovered macrops. Along with each macrop is its cognitive savings value. PLAND replaces all occurrences of the best macrop with an atomic form, $M_{1,2}$ in this example. PLAND then creates a second context with the instantiated input sequence. In this context PLAND discovers only one macrop: $M_{2,1} = (M_{1,2}{}^*\ Z)^*$. After instantiating this macrop in the input sequence, PLAND finds no more interesting macrops in the sequence.

PLAND demonstrates the applicability of the cognitive savings heuristic to substructures different from those discovered by SUBDUE. Although PLAND is unable to retain the discovered substructures for use in subsequent action sequences, retention of the macrops as rules in the existing background knowledge does not seem difficult to implement. Instead, PLAND uses discovered macrops in subsequent discovery tasks within the same input sequence. Retention and instantiation of substructures not only across examples, but also across iterations of the discovery algorithm might improve SUBDUE's performance.

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 to generate new substructures, while the combination and cut methods add or remove large amounts of substructure to generate new substructures. A third dimension along which these methods may differ is the knowledge available for constraining the type and number of substructures generated. Background knowledge pertaining to the types of expansions or disconnections performed may improve the performance of the substructure generation methods.

Once the alternative substructures are generated, the substructure discovery system must select the more interesting substructures from among the alternatives. The proposed method of substructure selection chooses the best substructures according to four heuristics: cognitive savings, compactness, connectivity, and coverage. Several experiments demonstrate the applicability of these heuristics to the task of identifying interesting substructures.

The substructure generation and substructure selection processes form the nucleus of the substructure discovery algorithm. The algorithm performs a computationally-constrained best-first search through the generated substructures. The search is guided by the four heuristics. The result of the substructure discovery algorithm is the best substructure found within the amount of computation allotted to the algorithm. The discovery system can then perform several operations with this substructure. First, the substructure can be specialized by appending additional structure. Specialization serves to annotate the substructure with additional information about the context in which the substructure may be applicable. Second, the discovery system may retain both the original and specialized substructures for use in subsequent discovery tasks. The substructures are maintained in a hierarchy to preserve the hierarchical structure of the environment and to exploit previously learned substructures in the representation of newly discovered, more complex substructures. Third, the substructure can be used to simplify the original input examples by instantiating each occurrence of the substructure with a single form. The simplified description of the examples may then be passed to other learning systems. In addition, the substructure discovery process may be applied repetitively to further simplify the examples or to build a hierarchical interpretation of the examples in terms of their subparts.

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 substructure background knowledge must be extended to accommodate other forms of background knowledge and to allow flexible application of this knowledge. Promising future applications of the substructure discovery system include visual image processing, database organization and constructive feature formation.

6.2.1. Substructure Discovery and Instantiation

Currently, the substructure discovery algorithm relies solely on the four heuristics for directing the search toward interesting substructures. Although background knowledge may suggest substructures from which to begin the discovery process, this knowledge does not affect the operations performed by the algorithm. One potential improvement to the discovery algorithm is the utilization of background knowledge to intelligently constrain the substructure generation process. When expanding a substructure to generate new substructures, some expansions may be better than others. Background knowledge rules that prefer certain expansions to others may significantly reduce the number of substructures generated by the current exhaustive substructure generation process. Furthermore, these rules can be learned by monitoring the contributions of certain expansions to the discovery of interesting substructures.

Just as the substructure generation process can be constrained, the substructure selection process can also be constrained through the use of appropriate background knowledge and the application of new heuristics. Although the cognitive savings heuristic should play a central role in any substructure heuristic evaluation function, it and other heuristics by no means represent the only dimensions along which to measure the goodness of a substructure. For instance, by incorporating the substructure specialization module into the discovery process, the measure of the amount of specialization provided by a substructure could be intergrated into the heuristic evaluation function. This new discovery process may improve the quality of the discovered substructures over those resulting from the separate module approach. A better approach to the introduction of new heuristics is through background knowledge. Previously learned rules could 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., $<[ON(X,Y)=T]$ $[COLOR(X)\neq RED]>$), 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.

6.2.2. Background Knowledge

The substructure background knowledge in SUBDUE plays only a minor role in the discovery process. Currently, the background knowledge maintains discovered substructures, as well as user-supplied substructures and identifies which of these substructures occur in the input examples. The discovery process uses these substructures as starting points. The idea behind most of the proposed extensions to the substructure background knowledge is for the background knowledge to play an active role in the discovery process.

As mentioned in the previous section, the background knowledge should recommend not only exact substructures from which to begin the discovery process, but should also suggest heuristics, substructure generation constraints, and other substructures similar to the exact substructures. One possible method for incorporating such knowledge into the existing substructure hierarchy is to attach the knowledge to the node representing the substructure. When a substructure is identified by the discovery process, the attached knowledge augments the discovery algorithm with new heuristics, generation constraints, and alternative substructures. Thus, the background knowledge becomes a hierarchically arranged set of rules with the substructure definitions as the antecedents and the attached procedural knowledge as the consequents.

Along with the knowledge for directing the discovery process, the substructure nodes may also contain knowledge indicating the function of the substructure. This can be thought of as a separate plane of the background knowledge hierarchy. On one plane is the declarative substructure hierarchy, while a second plane contains the functional hierarchy. As in the declarative hierarchy, the functional hierarchy defines the function of a substructure in terms of more primitive substructure functionality already defined. However, in order for SUBDUE to learn functionality, the input examples must be recast to indicate the primitive function of their components. Such information may be difficult to add to the declarative definition of the examples.

Another proposed extension to the background knowledge is to perform only a partial ("fuzzy") match instead of the exact match. Without partial matching, the knowledge contained at 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 unnecessary detail. By instantiating both previously known and discovered substructures in the input, SUBDUE reduces the complexity of the input to a level that is tolerable to the other learning systems. Also, many current learning systems either do not add new features to the input examples or add only predefined features. SUBDUE offers a method for discovering interesting features within the input examples. These features can be added to the input examples so that the other system may learn concepts in terms of features not present in the original input. The results of experiments in Chapter 4 indicate that integrating SUBDUE into other learning systems may improve the overall performance of the learning method.

With the extensions discussed in the previous section, the SUBDUE system and the substructure discovery methodology will provide a powerful tool to aid other machine learning systems, as well as become a general method for discovering concepts in a richly structured environment.

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.

[Mogensen87] B. N. Mogensen, "Goal-Oriented Conceptual Clustering: The Classifying Attribute Approach," M.S. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, 1987. (Also appears as Technical Report UILU-ENG-87-2257)

[Narasimhan63] R. Narasimhan, "Syntactic Descriptions of Pictures and Gestalt Phenomena of Visual Perception," Technical Report No. 142, Digital Computer Laboratory, University of Illinois, Urbana, IL, July, 1963.

[Nilsson80] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Company, Palo Alto, CA, 1980.

[Palmer83] S. E. Palmer, "The Psychology of Perceptual Organization: A Transformational Approach," in *Human and Machine Vision*, J. Beck, B. Hope and A. Rosenfeld (ed.), Academic Press, New York, NY, 1983, pp. 269-339.

[Prather76] R. Prather, *Discrete Mathematical Structures For Computer Science*, Houghton Mifflin Company, New York, NY, 1976.

[Reingold77] E. M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[Shavlik88] J. W. Shavlik, "Generalizing the Structure of Explanations in Explanation-Based Learning," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, January 1988. (Also appears as UILU-ENG-87-2276, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Stepp86] R. E. Stepp and R. S. Michalski, "Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects," in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Morgan Kaufmann, Los Altos, CA, 1986, pp. 471-498.

[Stepp87] R. E. Stepp, "Machine Learning from Structured Objects," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 353-363.

[Treisman82] A. Treisman, "Perceptual Grouping and Attention in Visual Search for Features and for Objects," *Journal of Experimental Psychology: Human Perception and Performance 8*, 2 (1982), pp. 194-214.

[Tuceryan87] M. Tuceryan and N. Ahuja, "Extracting Perceptual Structure in Dot Patterns: An Integrated Approach," Technical Report UILU-ENG-87-2206, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, January 1987.

[Wertheimer39] M. Wertheimer, "Laws of Organization in Perceptual Forms," in *A Source Book of Gestalt Psychology*, W. D. Ellis (ed.), Harcourt, Brace and Company, New York, NY, 1939.

[Whitehall87] B. L. Whitehall, "Substructure Discovery in Executed Action Sequences," M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1987. (Also appears as Technical Report UILU-ENG-87-2256)

[Winston75] P. H. Winston, "Learning Structural Descriptions from Examples," in *The Psychology of Computer Vision*, P. H. Winston (ed.), McGraw-Hill, New York, NY, 1975, pp. 157-210.

[Witkin83] A. P. Witkin and J. M. Tenenbaum, "On the Role of Structure in Vision," in *Human and Machine Vision*, J. Beck, B. Hope and A. Rosenfeld (ed.), Academic Press, New York, NY, 1983, pp. 481-543.

[Wolff82] J. G. Wolff, "Language Acquisition, Data Compression and Generalization," *Language and Communication 2*, 1 (1982), pp. 57-89.

[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.

APPENDIX A

EXPERIMENT DATA

This appendix contains the input data and output traces for the experiments in Chapter 4. The experiments were run on a Texas Instruments Explorer. Input examples are given to SUBDUE through calls to the *DefExample* function:

(DefExample '(*objects relation-definitions relations*))

The *objects* argument is a list of all the object names referred to in the *relations* of the example. The *relation-definitions* argument is a list of relation definitions, where each definition is a list with two elements. The first element is the name of the relation, and the second element is T, if the order of the arguments to the relation is relevant, and NIL otherwise. Finally, the *relations* argument is a list of relations, where each relation is a list with three elements. The first element is the relation name. The second element is a list of object arguments to the relation, and the third element is the value of the relation. Multiple examples are defined through multiple calls to the *DefExample* function.

Once the input examples are defined, a call to the *subdue* function generates the output traces shown in this appendix. The *subdue* function has several keyword arguments:

:limit	The computational limit on the substructure discovery algorithm. Default is NIL, which sets the computational limit to half the number of relations in the current set of input examples. A nonzero, positive integer may be specified. The computational limit is the number of substructures considered by the algorithm.
:connectivity	A boolean value indicating whether or not SUBDUE should apply the connectivity heuristic to the cognitive savings, while evaluating a substructure. Default is T.
:compactness	A boolean value indicating whether or not SUBDUE should apply the compactness heuristic to the cognitive savings, while evaluating a substructure. Default is T.

: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.

A.1. Experiment 1

The purpose of Experiment 1 is to show how varying the computational limit affects the results of SUBDUE's heuristic-based discovery module. The experiment involves two examples, each run for four different values of the computational limit. The *DefExample* calls for the two examples and the output traces for each of the eight runs are shown below.

A.1.1. Input for First Example of Experiment 1

```
(DefExample
 '((t1 t2 t3 t4 s1 s2 s3 s4 c1 c2 c3 c4 c5 r1)
  ((on t) (shape nil))
  ((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 (c1) circle) (shape (c2) circle) (shape (c3) circle)
   (shape (c4) circle) (shape (c5) circle) (shape (r1) rectangle) (on (t1 s1) t) (on (s1 c1) 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) (on (s2 c2) t) (on (s3 c3) t)
   (on (s4 c4) t) (on (c5 r1) t))))
```

A.1.2. Output for First Example of Experiment 1: Limit = 4

```
> (subdue :limit 4)

Begin trace...

Parameters:    limit = 4          connectivity = t     compactness = t      coverage = t
               use-bk = nil       discover = t         specialize = nil     inc-bk = nil

Running substructure discovery...

Discovered the following 4 substructures in 0.23333333 seconds:

{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])}

{Substructure#3 :value 2.0813007 ([shape(object-0002)=circle] [on(object-0001,object-0002)=t])}
WITH OCCURRENCES:
{([shape(c1)=circle] [on(s1,c1)=t])}
{([shape(c2)=circle] [on(s2,c2)=t])}
{([shape(c3)=circle] [on(s3,c3)=t])}
{([shape(c4)=circle] [on(s4,c4)=t])}

{Substructure#2 :value 1.656963 ([on(object-0001,object-0002)=t])}
WITH OCCURRENCES:
{([on(t1,s1)=t])}
{([on(r1,t2)=t])}
{([on(t2,s2)=t])}
{([on(r1,t3)=t])}
{([on(t3,s3)=t])}
{([on(r1,t4)=t])}
{([on(t4,s4)=t])}
{([on(s1,c1)=t])}
{([on(s2,c2)=t])}
{([on(s3,c3)=t])}
```

```
{([on(s4,c4)=t])}
{([on(c1,r1)=t])}
{([on(c5,r1)=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])}
```

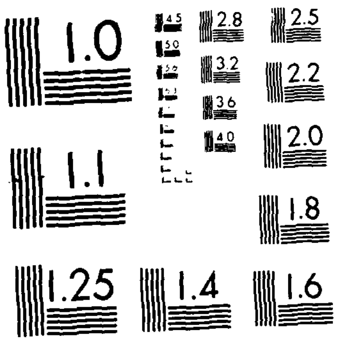
```
{([shape(t2)=triangle] [on(t2,s2)=t] [shape(s2)=square] [shape(c2)=circle] [on(s2,c2)=t])}
{([shape(t3)=triangle] [on(t3,s3)=t] [shape(s3)=square] [shape(c3)=circle] [on(s3,c3)=t])}
{([shape(t4)=triangle] [on(t4,s4)=t] [shape(s4)=square] [shape(c4)=circle] [on(s4,c4)=t])}

{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#10 :value 9.219512 ([shape(object-0016)=circle] [on(object-0016,object-0012)=t]
  [shape(object-0012)=rectangle] [on(object-0012,object-0008)=t] [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(c1)=circle] [on(c1,r1)=t] [shape(r1)=rectangle] [on(r1,t2)=t] [shape(t2)=triangle] [on(t2,s2)=t] [shape(s2)=square]
  [shape(c2)=circle] [on(s2,c2)=t])}
{([shape(c5)=circle] [on(c5,r1)=t] [shape(r1)=rectangle] [on(r1,t2)=t] [shape(t2)=triangle] [on(t2,s2)=t] [shape(s2)=square]
  [shape(c2)=circle] [on(s2,c2)=t])}
{([shape(c1)=circle] [on(c1,r1)=t] [shape(r1)=rectangle] [on(r1,t3)=t] [shape(t3)=triangle] [on(t3,s3)=t] [shape(s3)=square]
  [shape(c3)=circle] [on(s3,c3)=t])}
{([shape(o5)=circle] [on(c5,r1)=t] [shape(r1)=rectangle] [on(r1,t3)=t] [shape(t3)=triangle] [on(t3,s3)=t] [shape(s3)=square]
  [shape(c3)=circle] [on(s3,c3)=t])}
{([shape(c1)=circle] [on(c1,r1)=t] [shape(r1)=rectangle] [on(r1,t4)=t] [shape(t4)=triangle] [on(t4,s4)=t] [shape(s4)=square]
  [shape(c4)=circle] [on(s4,c4)=t])}
{([shape(c5)=circle] [on(c5,r1)=t] [shape(r1)=rectangle] [on(r1,t4)=t] [shape(t4)=triangle] [on(t4,s4)=t] [shape(s4)=square]
  [shape(c4)=circle] [on(s4,c4)=t])}
   :
   :

End trace.
```

A.1.5. Output for First Example of Experiment 1: Limit = 14

```
> (subdue :limit 14)

Begin trace...

Parameters:     limit = 14        connectivity = t     compactness = t     coverage = t
                use-bk = nil      discover = t         specialize = nil    inc-bk = nil

Running substructure discovery...

Discovered the following 14 substructures in 1.6166667 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#14 :value 10.327526 ([shape(object-0027)=square] [on(object-0027,object-0016)=t] [shape(object-0022)=circle]
  [on(object-0022,object-0012)=t] [shape(object-0016)=circle] [on(object-0016,object-0012)=t] [shape(object-0012)=rectangle]
  [on(object-0012,object-0008)=t] [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(s1)=square] [on(s1,c1)=t] [shape(c5)=circle] [on(c5,r1)=t] [shape(c1)=circle] [on(c1,r1)=t] [shape(r1)=rectangle]
  [on(r1,t2)=t] [shape(t2)=triangle] [on(t2,s2)=t] [shape(s2)=square] [shape(c2)=circle] [on(s2,c2)=t])}
{([shape(s1)=square] [on(s1,c1)=t] [shape(c5)=circle] [on(c5,r1)=t] [shape(c1)=circle] [on(c1,r1)=t] [shape(r1)=rectangle]
  [on(r1,t3)=t] [shape(t3)=triangle] [on(t3,s3)=t] [shape(s3)=square] [shape(c3)=circle] [on(s3,c3)=t])}
{([shape(s1)=square] [on(s1,c1)=t] [shape(c5)=circle] [on(c5,r1)=t] [shape(c1)=circle] [on(c1,r1)=t] [shape(r1)=rectangle]
  [on(r1,t4)=t] [shape(t4)=triangle] [on(t4,s4)=t] [shape(s4)=square] [shape(c4)=circle] [on(s4,c4)=t])}
```

AD-A197 778   DISCOVERING SUBSTRUCTURE IN EXAMPLES(U) ILLINOIS UNIV   2/2
AT URBANA COORDINATED SCIENCE LAB L B HOLDER MAY 88
UILU-ENG-88-2223 N00014-82-K-0186

UNCLASSIFIED   F/G 12/5   NL


1.0
4.5
5.0
5.6
6.3
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

```
{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])}
     :
     :
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-bk = nil       discover = t              specialize = nil         inc-bk = 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])}
```

```
{([connected(n04,n09)=t] [connected(n03,n04)=t])}
{([connected(n04,n05)=t] [connected(n03,n04)=t])}
{([connected(n04,n05)=t] [connected(n04,n09)=t])}
{([connected(n08,n09)=t] [connected(n04,n09)=t])}
{([connected(n09,n10)=t] [connected(n04,n09)=t])}
{([connected(n05,n10)=t] [connected(n04,n05)=t])}
{([connected(n09,n10)=t] [connected(n05,n10)=t])}
{([connected(n07,n08)=t] [connected(n06,n07)=t])}
{([connected(n08,n09)=t] [connected(n07,n08)=t])}
{([connected(n09,n10)=t] [connected(n08,n09)=t])}

{Substructure#4 :value 2.590909 ([connected(object-0003,object-0004)=t] [connected(object-0002,object-0003)=t]
  [connected(object-0001,object-0002)=t])}
WITH OCCURRENCES:
{([connected(n06,n07)=t] [connected(n01,n02)=t] [connected(n01,n06)=t])}
{([connected(n02,n07)=t] [connected(n01,n02)=t] [connected(n01,n06)=t])}
{([connected(n02,n03)=t] [connected(n01,n02)=t] [connected(n01,n06)=t])}
{([connected(n02,n07)=t] [connected(n06,n07)=t] [connected(n01,n06)=t])}
{([connected(n07,n08)=t] [connected(n06,n07)=t] [connected(n01,n06)=t])}
{([connected(n06,n07)=t] [connected(n02,n07)=t] [connected(n01,n02)=t])}
{([connected(n07,n08)=t] [connected(n02,n07)=t] [connected(n01,n02)=t])}
{([connected(n03,n08)=t] [connected(n02,n03)=t] [connected(n01,n02)=t])}
{([connected(n03,n04)=t] [connected(n02,n03)=t] [connected(n01,n02)=t])}
{([connected(n06,n07)=t] [connected(n02,n03)=t] [connected(n02,n07)=t])}
{([connected(n07,n08)=t] [connected(n02,n03)=t] [connected(n02,n07)=t])}
{([connected(n03,n08)=t] [connected(n02,n03)=t] [connected(n02,n07)=t])}
{([connected(n03,n04)=t] [connected(n02,n03)=t] [connected(n02,n07)=t])}
{([connected(n03,n08)=t] [connected(n07,n08)=t] [connected(n02,n07)=t])}
{([connected(n08,n09)=t] [connected(n07,n08)=t] [connected(n02,n07)=t])}
{([connected(n07,n08)=t] [connected(n03,n08)=t] [connected(n02,n03)=t])}
{([connected(n08,n09)=t] [connected(n03,n08)=t] [connected(n02,n03)=t])}
{([connected(n04,n09)=t] [connected(n03,n04)=t] [connected(n02,n03)=t])}
{([connected(n04,n05)=t] [connected(n03,n04)=t] [connected(n02,n03)=t])}
{([connected(n07,n08)=t] [connected(n03,n04)=t] [connected(n03,n08)=t])}
{([connected(n08,n09)=t] [connected(n03,n04)=t] [connected(n03,n08)=t])}
{([connected(n04,n09)=t] [connected(n03,n04)=t] [connected(n03,n08)=t])}
{([connected(n04,n05)=t] [connected(n03,n04)=t] [connected(n03,n08)=t])}
{([connected(n06,n07)=t] [connected(n07,n08)=t] [connected(n03,n08)=t])}
{([connected(n04,n09)=t] [connected(n08,n09)=t] [connected(n03,n08)=t])}
{([connected(n09,n10)=t] [connected(n08,n09)=t] [connected(n03,n08)=t])}
{([connected(n08,n09)=t] [connected(n04,n09)=t] [connected(n03,n04)=t])}
{([connected(n09,n10)=t] [connected(n04,n09)=t] [connected(n03,n04)=t])}
{([connected(n05,n10)=t] [connected(n04,n05)=t] [connected(n03,n04)=t])}
{([connected(n08,n09)=t] [connected(n04,n05)=t] [connected(n04,n09)=t])}
{([connected(n09,n10)=t] [connected(n04,n05)=t] [connected(n04,n09)=t])}
{([connected(n05,n10)=t] [connected(n04,n05)=t] [connected(n04,n09)=t])}
{([connected(n07,n08)=t] [connected(n08,n09)=t] [connected(n04,n09)=t])}
{([connected(n05,n10)=t] [connected(n09,n10)=t] [connected(n04,n09)=t])}
{([connected(n09,n10)=t] [connected(n05,n10)=t] [connected(n04,n05)=t])}
{([connected(n08,n09)=t] [connected(n09,n10)=t] [connected(n05,n10)=t])}
{([connected(n08,n09)=t] [connected(n07,n08)=t] [connected(n06,n07)=t])}
{([connected(n09,n10)=t] [connected(n08,n09)=t] [connected(n07,n08)=t])}
    :
    :
End trace.
```

A.1.8. Output for Second 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 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])}
```

```
{([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#8 :value 4.375 ([connected(object-0004,object-0009)=t] [connected(object-0009,object-0008)=t]
  [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(n07,n08)=t] [connected(n03,n08)=t] [connected(n02,n03)=t] [connected(n02,n07)=t] [connected(n06,n07)=t]
  [connected(n01,n02)=t] [connected(n01,n06)=t])}
{([connected(n08,n09)=t] [connected(n04,n09)=t] [connected(n03,n04)=t] [connected(n03,n08)=t] [connected(n07,n08)=t]
  [connected(n02,n03)=t] [connected(n02,n07)=t])}
{([connected(n09,n10)=t] [connected(n05,n10)=t] [connected(n04,n05)=t] [connected(n04,n09)=t] [connected(n08,n09)=t]
  [connected(n03,n04)=t] [connected(n03,n08)=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.1.10. Output for Second Example of Experiment 1: Limit = 14

```
> (subdue :limit 14)

Begin trace...

Parameters:     limit = 14       connectivity = t     compactness = t      coverage = t
                use-bk = nil     discover = t         specialize = nil     inc-bk = nil

Running substructure discovery...

Discovered the following 14 substructures in 120.71667 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#8 :value 4.375 ([connected(object-0004,object-0009)=t] [connected(object-0009,object-0008)=t]
  [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(n07,n08)=t] [connected(n03,n08)=t] [connected(n02,n03)=t] [connected(n02,n07)=t] [connected(n06,n07)=t]
  [connected(n01,n02)=t] [connected(n01,n06)=t])}
{([connected(n08,n09)=t] [connected(n04,n09)=t] [connected(n03,n04)=t] [connected(n03,n08)=t] [connected(n07,n08)=t]
  [connected(n02,n03)=t] [connected(n02,n07)=t])}
{([connected(n09,n10)=t] [connected(n05,n10)=t] [connected(n04,n05)=t] [connected(n04,n09)=t] [connected(n08,n09)=t]
  [connected(n03,n04)=t] [connected(n03,n08)=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

```
(DefExample
 '((h1 h2 h3 h4 h5 h6 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 (h1) hydrogen) (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 c18) 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))))
```

A.2.2. Output for First Example of Experiment 2

```
> (subdue :specialize t :inc-bk t)

Begin trace...

Parameters:     limit = 39        connectivity = t     compactness = t     coverage = t
                use-bk = nil      discover = t         specialize = t      inc-bk = t

Running substructure discovery...

Discovered the following 39 substructures in 406.93332 seconds:

{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])}
WITH OCCURRENCES:
{([single-bond(c01,i1)=t] [atom-type(c04)=carbon] [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] [atom-type(c04)=carbon] [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] [atom-type(c17)=carbon] [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] [atom-type(c08)=carbon] [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] [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])}
{([single-bond(c24,i2)=t] [atom-type(c21)=carbon] [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])}

{Substructure#36 :value 25.263159 ([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])}
WITH OCCURRENCES:
{([atom-type(c04)=carbon] [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])}
{([atom-type(c04)=carbon] [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])}
{([atom-type(c17)=carbon] [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])}
{([atom-type(c08)=carbon] [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])}
{([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(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])}

{Substructure#38 :value 22.780703 ([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,cl2)=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 BK...

Added the following substructures to BK:
```

```
Discovered: {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])}

Specialized: {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])}

End trace.
```

A.2.3. Input for Second Example of Experiment 2

```
(DefExample
  '((h02 h03 h04 h05 h08 h09 h10 h11 h12 cl br i c01 c02 c03 c04 c05 c06 c07 c08 c09 c10 c11 c12 c13 c14 c15 c16 c17 c18)
   ((single-bond nil) (double-bond nil) (atom-type nil))
   ((atom-type (cl) chlorine) (atom-type (h02) hydrogen) (atom-type (h03) hydrogen) (atom-type (h04) hydrogen)
    (atom-type (h05) hydrogen) (atom-type (i) iodine) (atom-type (h08) hydrogen) (atom-type (br) bromine)
    (atom-type (h09) hydrogen) (atom-type (h10) hydrogen) (atom-type (h11) hydrogen) (atom-type (h12) hydrogen)
    (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) (single-bond (c01 cl) t) (single-bond (c02 h02) t)
    (single-bond (c03 h03) t) (single-bond (c04 h04) t) (single-bond (c15 h05) t) (single-bond (c18 i) t)
    (single-bond (c11 br) t) (single-bond (c12 h08) t) (single-bond (c13 h09) t) (single-bond (c14 h10) t)
    (single-bond (c16 h11) t) (single-bond (c17 h12) t) (single-bond (c01 c06) t) (single-bond (c02 c03) t)
    (single-bond (c04 c05) t) (single-bond (c06 c07) t) (single-bond (c05 c10) t) (single-bond (c08 c09) t)
    (single-bond (c07 c11) t) (single-bond (c12 c13) t) (single-bond (c08 c14) t) (single-bond (c10 c15) t)
    (single-bond (c16 c17) t) (single-bond (c09 c18) t) (double-bond (c01 c02) t) (double-bond (c03 c04) t)
    (double-bond (c05 c06) t) (double-bond (c07 c08) t) (double-bond (c09 c10) t) (double-bond (c11 c12) t)
    (double-bond (c13 c14) t) (double-bond (c15 c16) t) (double-bond (c17 c18) t))))
```

A.2.4. Output for Second Example of Experiment 2

```
> (subdue :limit 10 :use-bk t :specialize t :inc-bk t)

Begin trace...

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

Checking BK for relevant substructures...

Found the following BK substructures:

{Substructure#0 :value 22.451612 ([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:
(([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,h12)=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])}
```

```
{([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,h11)=t] [single-bond(c15,h05)=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,h11)=t] [single-bond(c15,h05)=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(h05)=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,h10)=t] [single-bond(c13,h09)=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,h10)=t] [single-bond(c13,h09)=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,h08)=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,h04)=t] [single-bond(c03,h03)=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,h04)=t] [single-bond(c03,h03)=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,h02)=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 18.580645 ([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,h02)=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,h08)=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,h12)=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 241.81667 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(c18,i)=t] [atom-type(h05)=hydrogen] [atom-type(h12)=hydrogen] [single-bond(c17,h12)=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,h11)=t] [single-bond(c15,h05)=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]])}
 {([single-bond(c11,br)=t] [atom-type(h10)=hydrogen] [atom-type(h08)=hydrogen] [single-bond(c12,h08)=t]
  [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,h10)=t] [single-bond(c13,h09)=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]])}
{([single-bond(c01,cl)=t] [atom-type(h04)=hydrogen] [atom-type(h02)=hydrogen] [single-bond(c02,h02)=t]
  [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,h04)=t] [single-bond(c03,h03)=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]])}

 Substructure#4 :value 31.35484 ([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:
{([atom-type(h05)=hydrogen] [atom-type(h12)=hydrogen] [single-bond(c17,h12)=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,h11)=t] [single-bond(c15,h05)=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]])}
 ([atom-type(h10)=hydrogen] [atom-type(h08)=hydrogen] [single-bond(c12,h08)=t] [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,h10)=t] [single-bond(c13,h09)=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]])}
 ([atom-type(h04)=hydrogen] [atom-type(h02)=hydrogen] [single-bond(c02,h02)=t] [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,h04)=t] [single-bond(c03,h03)=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]])}

 Substructure#10 :value 26.034458 ([atom-type(object-0034)=carbon] [single-bond(object-0010,object-0034)=t]
  [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:
([atom-type(c08)=carbon] [single-bond(c08,c09)=t] [single-bond(c18,i)=t] [atom-type(h05)=hydrogen]
  [atom-type(h12)=hydrogen] [single-bond(c17,h12)=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,h11)=t] [single-bond(c15,h05)=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]])}
 ([atom-type(c06)=carbon] [single-bond(c06,c07)=t] [single-bond(c11,br)=t] [atom-type(h10)=hydrogen]
  [atom-type(h08)=hydrogen] [single-bond(c12,h08)=t] [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,h10)=t] [single-bond(c13,h09)=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]])}
 ([atom-type(c07)=carbon] [single-bond(c06,c07)=t] [single-bond(c01,cl)=t] [atom-type(h04)=hydrogen]
  [atom-type(h02)=hydrogen] [single-bond(c02,h02)=t] [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,h04)=t] [single-bond(c03,h03)=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]])}
```

```
:

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.

A.3.1. Input for Experiment 3

```
(DefExample                ;Train A
 '((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) black) (car-shape (car2) open-rectangle) (car-length (car2) long)
   (load-shape (car2) rectangle) (load-number (car2) three) (wheel-color (car2) black) (car-shape (car3) sloped)
   (car-length (car3) short) (load-shape (car3) triangle) (load-number (car3) one) (wheel-color (car3) black)
   (car-shape (car4) open-rectangle) (car-length (car4) long) (load-shape (car4) hexagon) (load-number (car4) one)
   (wheel-color (car4) black) (car-shape (car5) open-rectangle) (car-length (car5) short) (load-shape (car5) circle)
   (load-number (car5) one ) (wheel-color (car5) black) (infront (car1 car2) t) (infront (car2 car3) t)
   (infront (car3 car4) t) (infront (car4 car5) t))))

(DefExample                ;Train B
 '((car1 car2 car3 car4)
  ((car-shape nil) (wheel-color nil) (car-length nil) (load-shape nil) (load-number nil) (infront t))
  ((car-shape (car1) engine) (wheel-color (car1) black) (car-shape (car2) u-shape) (car-length (car2) short)
   (load-shape (car2) triangle) (load-number (car2) one) (wheel-color (car2) white) (car-shape (car3) open-trapezoid)
   (car-length (car3) short) (load-shape (car3) rectangle) (load-number (car3) one) (wheel-color (car3) white)
   (car-shape (car4) closed-rectangle) (car-length (car4) short) (load-shape (car4) circle) (load-number (car4) two)
   (wheel-color (car4) white) (infront (car1 car2) t) (infront (car2 car3) t) (infront (car3 car4) t))))

(DefExample                ;Train C
 '((car1 car2 car3 car4)
  ((car-shape nil) (wheel-color nil) (car-length nil) (load-shape nil) (load-number nil) (infront t))
  ((car-shape (car1) engine) (wheel-color (car1) black) (car-shape (car2) open-rectangle) (car-length (car2) short)
   (load-shape (car2) circle) (load-number (car2) one) (wheel-color (car2) white) (car-shape (car3) hexagon)
   (car-length (car3) short) (load-shape (car3) triangle) (load-number (car3) one) (wheel-color (car3) white)
   (car-shape (car4) closed-rectangle) (car-length (car4) long) (load-shape (car4) triangle) (load-number (car4) one)
   (wheel-color (car4) white) (infront (car1 car2) t) (infront (car2 car3) t) (infront (car3 car4) t))))

(DefExample                ;Train D
 '((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) triangle) (load-number (car2) one) (wheel-color (car2) white) (car-shape (car3) double)
   (car-length (car3) short) (load-shape (car3) triangle) (load-number (car3) one) (wheel-color (car3) white)
   (car-shape (car4) ellipse) (car-length (car4) short) (load-shape (car4) diamond) (load-number (car4) one)
   (wheel-color (car4) white) (car-shape (car5) open-rectangle) (car-length (car5) short) (load-shape (car5) rectangle)
   (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 E
 '((car1 car2 car3 car4)
  ((car-shape nil) (wheel-color nil) (car-length nil) (load-shape nil) (load-number nil) (infront t))
  ((car-shape (car1) engine) (wheel-color (car1) black) (car-shape (car2) double) (car-length (car2) short)
   (load-shape (car2) triangle) (load-number (car2) one) (wheel-color (car2) black) (car-shape (car3) closed-rectangle)
   (car-length (car3) long) (load-shape (car3) rectangle) (load-number (car3) one) (wheel-color (car3) black)
   (car-shape (car4) closed-rectangle) (car-length (car4) short) (load-shape (car4) circle) (load-number (car4) one)
   (wheel-color (car4) black) (infront (car1 car2) t) (infront (car2 car3) t) (infront (car3 car4) t))))

(DefExample                ;Train F
 '((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) black) (car-shape (car2) closed-rectangle) (car-length (car2) long)
   (load-shape (car2) circle) (load-number (car2) three) (wheel-color (car2) white) (car-shape (car3) open-rectangle)
   (car-length (car3) short) (load-shape (car3) triangle) (load-number (car3) one) (wheel-color (car3) white)
   (infront (car1 car2) t) (infront (car2 car3) t))))

(DefExample                ;Train G
 '((car1 car2 car3 car4)
  ((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) double) (car-length (car2) short)
   (load-shape (car2) circle) (load-number (car2) one) (wheel-color (car2) black) (car-shape (car3) u-shape)
   (car-length (car3) short) (load-shape (car3) triangle) (load-number (car3) one) (wheel-color (car3) white)
   (car-shape (car4) jagged) (car-length (car4) long) (load-number (car4) zero) (wheel-color (car4) white
(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 8.228033 ([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:
```

```
{([wheel-color(car2)=white] [infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one]
  [wheel-color(car3)=white])}
{([wheel-color(car2)=white] [infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one]
  [wheel-color(car3)=white])}
{([wheel-color(car1)=white] [infront(car1,car2)=t] [car-length(car2)=short] [load-number(car2)=one]
  [wheel-color(car2)=white])}
{([wheel-color(car2)=white] [infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one]
  [wheel-color(car3)=white])}
{([wheel-color(car3)=white] [infront(car3,car4)=t] [car-length(car4)=short] [load-number(car4)=one]
  [wheel-color(car4)=white])}
{([wheel-color(car4)=white] [infront(car4,car5)=t] [car-length(car5)=short] [load-number(car5)=one]
  [wheel-color(car5)=white])}
{([wheel-color(car2)=white] [infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one]
  [wheel-color(car3)=white])}
{([wheel-color(car2)=white] [infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one]
  [wheel-color(car3)=white])}
{([wheel-color(car1)=white] [infront(car1,car2)=t] [car-length(car2)=short] [load-number(car2)=one]
  [wheel-color(car2)=white])}
{([wheel-color(car3)=white] [infront(car3,car4)=t] [car-length(car4)=short] [load-number(car4)=one]
  [wheel-color(car4)=white])}
{([wheel-color(car4)=white] [infront(car4,car5)=t] [car-length(car5)=short] [load-number(car5)=one]
  [wheel-color(car5)=white])}
{([wheel-color(car1)=white] [infront(car1,car2)=t] [car-length(car2)=short] [load-number(car2)=one]
  [wheel-color(car2)=white])}

{Substructure#5 :value 7.4092503 ([infront(object-0008,object-0001)=t] [car-length(object-0001)=short]
  [load-number(object-0001)=one] [wheel-color(object-0001)=white])}
WITH OCCURRENCES:
{([infront(car1,car2)=t] [car-length(car2)=short] [load-number(car2)=one] [wheel-color(car2)=white])}
{([infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one] [wheel-color(car3)=white])}
{([infront(car1,car2)=t] [car-length(car2)=short] [load-number(car2)=one] [wheel-color(car2)=white])}
{([infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one] [wheel-color(car3)=white])}
{([infront(car1,car2)=t] [car-length(car2)=short] [load-number(car2)=one] [wheel-color(car2)=white])}
{([infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one] [wheel-color(car3)=white])}
{([infront(car3,car4)=t] [car-length(car4)=short] [load-number(car4)=one] [wheel-color(car4)=white])}
{([infront(car4,car5)=t] [car-length(car5)=short] [load-number(car5)=one] [wheel-color(car5)=white])}
{([infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one] [wheel-color(car3)=white])}
{([infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one] [wheel-color(car3)=white])}
{([infront(car2,car3)=t] [car-length(car3)=short] [load-number(car3)=one] [wheel-color(car3)=white])}
{([infront(car1,car2)=t] [car-length(car2)=short] [load-number(car2)=one] [wheel-color(car2)=white])}
{([infront(car3,car4)=t] [car-length(car4)=short] [load-number(car4)=one] [wheel-color(car4)=white])}
{([infront(car4,car5)=t] [car-length(car5)=short] [load-number(car5)=one] [wheel-color(car5)=white])}
{([infront(car1,car2)=t] [car-length(car2)=short] [load-number(car2)=one] [wheel-color(car2)=white])}
    .
    .
    .
End trace.
```

A.4. Experiment 4

Experiment 4 applies SUBDUE to the task of discovering macro-operators from a proof tree. The example for this experiment is a proof tree from the blocks world. The *DefExample* call for this example is shown, along with the resulting output of discovered macro-operators.

A.4.1. Input for Experiment 4

```
(DefExample
  '((g00 g01 g02 g03 g04 g05 g06 g07 g08 g09 g10 arga argb argc argd arge argf argg)
    ((subop t) (before t) (op-type nil) (arg-name nil) (stack-arg1 t) (stack-arg2 t) (unstack-arg1 t) (unstack-arg2 t)
     (pickup-arg t) (putdown-arg t))
((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 arge) t) (unstack-arg2 (g08 argf) t)
 (op-type (g09) putdown) (putdown-arg (g09 arge) 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] [bef re(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]
[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] [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])}
   :
   :
End trace.
```

A.5. Experiment 5

Experiment 5 demonstrates how SUBDUE may improve the performance of other machine learning programs. The examples for this experiment are taken from the organic chemistry domain. The *DefExample* calls for each example are shown, along with the resulting output.

A.5.1. Input for Experiment 5

```
(DefExample ;compound 1 (positive)
 '((c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15)
  ((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 (c6 c8) t) (single (c7 c8) t) (single (c8 c9) t) (single (c8 c10) t) (double (c10 c11) t) (single (c10 c12) t)
   (single (c11 c13) t) (double (c12 c14) t) (double (c13 c15) t) (single (c14 c15) t))))

(DefExample ;compound 2 (positive)
 '((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)
  ((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 (c6 c14) t) (single (c12 c15) t) (single (c13 c14) t) (single (c14 c15) t) (single (c15 c16) t)
   (single (c14 c17) t) (single (c15 c23) t) (double (c17 c18) t) (single (c17 c19) t) (single (c18 c20) t)
   (double (c19 c21) t) (double (c20 c22) t) (single (c21 c22) t) (double (c23 c24) t) (single (c23 c25) t)
   (single (c24 c26) t) (double (c25 c27) t) (double (c26 c28) t) (single (c27 c28) t))))

(DefExample ;compound 3 (positive)
 '((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 c35 c36 c37 c38 c39 c40 c41)
  ((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 (c6 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 c30) t)
   (single (c22 c36) t) (double (c24 c25) t) (single (c24 c26) t) (single (c25 c27) t) (double (c26 c28) t)
   (double (c27 c29) t) (single (c28 c29) t) (double (c30 c31) t) (single (c30 c32) t) (single (c31 c33) t)
   (double (c32 c34) t) (double (c33 c35) t) (single (c34 c35) t) (double (c36 c37) t) (single (c36 c38) t)
   (single (c37 c39) t) (double (c38 c40) t) (double (c39 c41) t) (single (c40 c41) t))))

(DefExample ;compound 4 (negative)
 '((c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18)
  ((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 (c6 c14) t) (single (c12 c15) t) (single (c13 c14) t) (single (c14 c15) t) (single (c15 c16) t)
   (single (c14 c17) t) (single (c15 c18) t))))

(DefExample ;compound 5 (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)
  ((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 (c6 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 c28) 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(c13,c15)=t] [double(c12,c14)=t] [single(c11,c13)=t] [double(c10,c11)=t] [single(c10,c12)=t])}
{([double(c13,c15)=t] [double(c10,c11)=t] [single(c14,c15)=t] [double(c12,c14)=t] [single(c10,c12)=t])}
{([double(c12,c14)=t] [double(c10,c11)=t] [single(c14,c15)=t] [double(c13,c15)=t] [single(c11,c13)=t])}
{([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(c11,c12)=t] [double(c8,c10)=t] [single(c9,c11)=t] [double(c7,c9)=t] [single(c7,c8)=t])}
{([double(c11,c12)=t] [double(c7,c9)=t] [single(c10,c12)=t] [double(c8,c10)=t] [single(c7,c8)=t])}
{([double(c8,c10)=t] [double(c7,c9)=t] [single(c10,c12)=t] [double(c11,c12)=t] [single(c9,c11)=t])}
{([double(c20,c22)=t] [double(c19,c21)=t] [single(c18,c20)=t] [double(c17,c18)=t] [single(c17,c19)=t])}
{([double(c20,c22)=t] [double(c17,c18)=t] [single(c21,c22)=t] [double(c19,c21)=t] [single(c17,c19)=t])}
{([double(c19,c21)=t] [double(c17,c18)=t] [single(c21,c22)=t] [double(c20,c22)=t] [single(c18,c20)=t])}
{([double(c26,c28)=t] [double(c25,c27)=t] [single(c24,c26)=t] [double(c23,c24)=t] [single(c23,c25)=t])}
{([double(c26,c28)=t] [double(c23,c24)=t] [single(c27,c28)=t] [double(c25,c27)=t] [single(c23,c25)=t])}
{([double(c25,c27)=t] [double(c23,c24)=t] [single(c27,c28)=t] [double(c26,c28)=t] [single(c24,c26)=t])}
{([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(c11,c12)=t] [double(c8,c10)=t] [single(c9,c11)=t] [double(c7,c9)=t] [single(c7,c8)=t])}
{([double(c11,c12)=t] [double(c7,c9)=t] [single(c10,c12)=t] [double(c8,c10)=t] [single(c7,c8)=t])}
{([double(c8,c10)=t] [double(c7,c9)=t] [single(c10,c12)=t] [double(c11,c12)=t] [single(c9,c11)=t])}
{([double(c17,c18)=t] [double(c14,c16)=t] [single(c15,c17)=t] [double(c13,c15)=t] [single(c13,c14)=t])}
{([double(c17,c18)=t] [double(c13,c15)=t] [single(c16,c18)=t] [double(c14,c16)=t] [single(c13,c14)=t])}
{([double(c14,c16)=t] [double(c13,c15)=t] [single(c16,c18)=t] [double(c17,c18)=t] [single(c15,c17)=t])}
{([double(c27,c29)=t] [double(c26,c28)=t] [single(c25,c27)=t] [double(c24,c25)=t] [single(c24,c26)=t])}
{([double(c27,c29)=t] [double(c24,c25)=t] [single(c28,c29)=t] [double(c26,c28)=t] [single(c24,c26)=t])}
{([double(c26,c28)=t] [double(c24,c25)=t] [single(c28,c29)=t] [double(c27,c29)=t] [single(c25,c27)=t])}
{([double(c33,c35)=t] [double(c32,c34)=t] [single(c31,c33)=t] [double(c30,c31)=t] [single(c30,c32)=t])}
{([double(c33,c35)=t] [double(c30,c31)=t] [single(c34,c35)=t] [double(c32,c34)=t] [single(c30,c32)=t])}
{([double(c32,c34)=t] [double(c30,c31)=t] [single(c34,c35)=t] [double(c33,c35)=t] [single(c31,c33)=t])}
{([double(c39,c41)=t] [double(c38,c40)=t] [single(c37,c39)=t] [double(c36,c37)=t] [single(c36,c38)=t])}
{([double(c39,c41)=t] [double(c36,c37)=t] [single(c40,c41)=t] [double(c38,c40)=t] [single(c36,c38)=t])}
{([double(c38,c40)=t] [double(c36,c37)=t] [single(c40,c41)=t] [double(c39,c41)=t] [single(c37,c39)=t])}
{([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(c11,c12)=t] [double(c8,c10)=t] [single(c9,c11)=t] [double(c7,c9)=t] [single(c7,c8)=t])}
{([double(c11,c12)=t] [double(c7,c9)=t] [single(c10,c12)=t] [double(c8,c10)=t] [single(c7,c8)=t])}
{([double(c8,c10)=t] [double(c7,c9)=t] [single(c10,c12)=t] [double(c11,c12)=t] [single(c9,c11)=t])}
{([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(c11,c12)=t] [double(c8,c10)=t] [single(c9,c11)=t] [double(c7,c9)=t] [single(c7,c8)=t])}
{([double(c11,c12)=t] [double(c7,c9)=t] [single(c10,c12)=t] [double(c8,c10)=t] [single(c7,c8)=t])}
{([double(c8,c10)=t] [double(c7,c9)=t] [single(c10,c12)=t] [double(c11,c12)=t] [single(c9,c11)=t])}
{([double(c17,c18)=t] [double(c14,c16)=t] [single(c15,c17)=t] [double(c13,c15)=t] [single(c13,c14)=t])}
{([double(c17,c18)=t] [double(c13,c15)=t] [single(c16,c18)=t] [double(c14,c16)=t] [single(c13,c14)=t])}
{([double(c14,c16)=t] [double(c13,c15)=t] [single(c16,c18)=t] [double(c17,c18)=t] [single(c15,c17)=t])}
{([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(c11,c12)=t] [double(c8,c10)=t] [single(c9,c11)=t] [double(c7,c9)=t] [single(c7,c8)=t])}
{([double(c11,c12)=t] [double(c7,c9)=t] [single(c10,c12)=t] [double(c8,c10)=t] [single(c7,c8)=t])}
{([double(c8,c10)=t] [double(c7,c9)=t] [single(c10,c12)=t] [double(c11,c12)=t] [single(c9,c11)=t])}
{([double(c17,c18)=t] [double(c14,c16)=t] [single(c15,c17)=t] [double(c13,c15)=t] [single(c13,c14)=t])}
{([double(c17,c18)=t] [double(c13,c15)=t] [single(c16,c18)=t] [double(c14,c16)=t] [single(c13,c14)=t])}
{([double(c14,c16)=t] [double(c13,c15)=t] [single(c16,c18)=t] [double(c17,c18)=t] [single(c15,c17)=t])}
{([double(c23,c24)=t] [double(c20,c22)=t] [single(c21,c23)=t] [double(c19,c21)=t] [single(c19,c20)=t])}
{([double(c23,c24)=t] [double(c19,c21)=t] [single(c22,c24)=t] [double(c20,c22)=t] [single(c19,c20)=t])}
{([double(c20,c22)=t] [double(c19,c21)=t] [single(c22,c24)=t] [double(c23,c24)=t] [single(c21,c23)=t])}

(Substructure#5 :value 60.33403 ([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(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,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(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,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(c24,c25)=t] [single(c28,c29)=t] [double(c26,c28)=t] [single(c24,c26)=t])}
{([double(c27,c29)=t] [single(c28,c29)=t] [double(c26,c28)=t] [single(c24,c26)=t])}
{([double(c24,c25)=t] [single(c28,c29)=t] [double(c27,c29)=t] [single(c25,c27)=t])}
{([double(c26,c28)=t] [single(c28,c29)=t] [double(c27,c29)=t] [single(c25,c27)=t])}
{([double(c32,c34)=t] [single(c31,c33)=t] [double(c30,c31)=t] [single(c30,c32)=t])}
{([double(c33,c35)=t] [single(c31,c33)=t] [double(c30,c31)=t] [single(c30,c32)=t])}
{([double(c30,c31)=t] [single(c34,c35)=t] [double(c32,c34)=t] [single(c30,c32)=t])}
{([double(c33,c35)=t] [single(c34,c35)=t] [double(c32,c34)=t] [single(c30,c32)=t])}
{([double(c30,c31)=t] [single(c34,c35)=t] [double(c33,c35)=t] [single(c31,c33)=t])}
{([double(c32,c34)=t] [single(c34,c35)=t] [double(c33,c35)=t] [single(c31,c33)=t])}
{([double(c38,c40)=t] [single(c37,c39)=t] [double(c36,c37)=t] [single(c36,c38)=t])}
{([double(c39,c41)=t] [single(c37,c39)=t] [double(c36,c37)=t] [single(c36,c38)=t])}
{([double(c36,c37)=t] [single(c40,c41)=t] [double(c38,c40)=t] [single(c36,c38)=t])}
{([double(c39,c41)=t] [single(c40,c41)=t] [double(c38,c40)=t] [single(c36,c38)=t])}
{([double(c36,c37)=t] [single(c40,c41)=t] [double(c39,c41)=t] [single(c37,c39)=t])}
{([double(c38,c40)=t] [single(c40,c41)=t] [double(c39,c41)=t] [single(c37,c39)=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(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(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,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(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,c26)=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,c27)=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,c28)=t] [double(c17,c18)=t] [single(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] [single(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.
```