# Fuzzy Substructure Discovery

**Lawrence B. Holder**
Dept. of Computer Science Engineering
University of Texas at Arlington
Arlington, TX 76019
holder@cse.uta.edu

**Diane J. Cook**
Dept. of Computer Science
University of South Florida
Tampa, FL 33620
cook@sol.csee.usf.edu

**Horst Bunke**
University of Berne
Langgassstr. 51, CH-3012
Berne, Switzerland
bunke@iam.unibe.ch

## Abstract

This paper describes a method for discovering substructures in data using a fuzzy graph match. A previous implementation of the SUBDUE system discovers substructures based on the psychologically-motivated criteria of cognitive savings, compactness, connectivity and coverage. However, the instances in the data must exactly match the discovered substructures. We describe a new implementation of SUBDUE that employs a fuzzy graph match to discover substructures which occur often in the data, but not always in the same form. This fuzzy substructure discovery can be used to formulate fuzzy concepts, compress the data description, and discover interesting structures in data that are found either in their pure form or in a slightly convoluted form. Examples from the domains of scene analysis and chemical compound analysis demonstrate the fuzzy discovery technique.

## 1 INTRODUCTION

Substructure discovery is the process of identifying concepts describing interesting and repetitive "chunks" of structure within structural descriptions of the environment. Once discovered, the substructure concept can be used to simplify the descriptions by replacing all instances of the substructure with a pointer to the newly discovered concept. The discovered substructure concepts allow abstraction over detailed structure in the original descriptions and provide new, relevant attributes for subsequent learning tasks.

The SUBDUE system [Holder, 1989] discovers substructure in structured data based on four heuristics: cognitive savings, connectivity, compactness and coverage. These heuristics are motivated from results in gestalt psychology [Wertheimer, 1939]. Two shortcomings of the SUBDUE system are the reliance on an expensive exact graph match and, therefore, the inability to match discovered substructures to similar, but non-identical, instances in the data. Because the heuristic value of a substructure greatly depends on the number of instances found in the data, the inability to match similar instances may prevent discovery of important substructures. This paper describes a less-expensive fuzzy graph match technique and its implementation within the SUBDUE architecture. The fuzzy graph match assigns a cost to the match between two graphs that can be transformed to a degree of match similar to the membership functions of fuzzy logic [Zadeh, 1973]. Using the fuzzy graph match, SUBDUE can identify slightly different instances of substructures and formulate the discovered substructures as fuzzy concepts.

Section 2 defines the term *substructure* and describes the SUBDUE substructure discovery system. Section 3 introduces inexact graph match and describes an efficient algorithm to find fuzzy subgraph isomorphisms and to determine the cost of such a match. Section 4 describes the integration of the inexact graph match into the SUBDUE system, and Section 5 illustrates the behavior of the system on two examples. Section 6 discusses the benefits and potential applications of fuzzy substructure discovery and suggests future directions.

## 2 SUBSTRUCTURE DISCOVERY

Structured data from the environment is represented as a graph within the substructure discovery system. Objects in the environment map to nodes or small subgraphs in the graph, and relationships between objects map to edges in the graph. A *substructure* is a connected subgraph within the graphical representation of

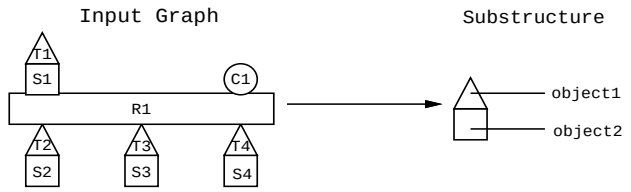


Figure 1: Example substructure

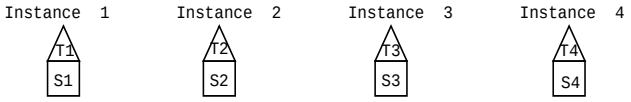


Figure 2: Instances of the substructure

the environment. This graphical representation serves as input to the substructure discovery system. Figure 1 shows a geometric example of such an input graph. The objects in the figure (e.g., `T1`, `S1`, `R1`) become labeled nodes in the graph, and the relationships (e.g., `on(T1,S1)`, `shape(C1,circle)`) become labeled edges in the graph.

An *instance* of a substructure in an input graph is a set of nodes and edges from the input graph that match, graph theoretically, to the graphical representation of the substructure. For example, the instances of the substructure in the example of Figure 1 are shown in Figure 2. A *neighboring edge* of an instance of a substructure is an edge in the input graph that is not contained in the instance, but is connected to at least one node in the instance. For example, the first instance has one neighboring edge: `on(S1,R1)`. An *external connection* of an instance of a substructure is a neighboring edge of the instance that is connected to at least one node not contained in the instance. Therefore, `on(S1,R1)` is also the only external connection of the first instance in Figure 2.

The substructure discovery algorithm used by SUBDUE is a computationally constrained best-first search guided by four heuristics. The algorithm begins with the substructure matching a single node in the graph. Each iteration through the algorithm selects the heuristically-best substructure and expands the instances of the substructure by one neighboring edge in all possible ways. The new unique generated substructures become candidates for further expansion within the best-first search paradigm. The algorithm searches for the heuristically best substructure until all possible substructures have been considered or the amount of computation exceeds the given limit. Due to the large number of possible substructures, the algorithm typically exhausts the allotted computation before considering all possible substructures. However, experiments in a variety of domains indicate that the heuristics perform well in guiding the search toward more promising substructures [Holder, 1988].

The four heuristics used by SUBDUE to evaluate a substructure are cognitive savings, compactness, connectivity and coverage. The first heuristic, *cognitive savings*, is the underlying idea behind several utility and data compression heuristics employed in machine learning [Minton, 1988; Whitehall, 1987; Wolff, 1982]. The cognitive savings of a substructure represents the net reduction in complexity after considering both the reduction in complexity of the input graph after replacing each instance of the substructure by a single conceptual entity and the gain in complexity associated with the conceptual definition of the new substructure. The reduction in complexity of the input graph can be computed as the number of instances of the substructure multiplied by the complexity of the substructure. Thus, the cognitive savings of a substructure $s$ for an input graph $G$ is computed as

$$\begin{aligned} &cognitive\_savings(s,G) \\ &\quad = complexity\_reduction(s,G) - complexity(s) \\ &\quad = complexity(s) * [\#instances(s,G) - 1] \end{aligned}$$

In SUBDUE's substructure discovery algorithm, the $complexity(s)$ is defined as the size ($\#nodes + \#edges$) of the graph representing the substructure. Since the instances may overlap in the input graph, $\#instances$ is the number of unique instances, which may be a fractional number.

The second heuristic, *compactness*, is a generalization of Wertheimer's *Factor of Closure*, which states that human attention is drawn to closed structures [Wertheimer, 1939]. A closed substructure has at least as many edges as nodes, whereas a non-closed substructure has fewer edges than nodes [Prather, 1976]. Thus, closed substructures have a higher compactness value. Compactness is defined as the ratio of the number of edges in the substructure to the number of nodes in the substructure.

$$compactness(s) = \frac{\#edges(s)}{\#nodes(s)}$$

The third heuristic, *connectivity*, measures the amount of external connection in the instances of the substructure. The connectivity heuristic is a variant of Wertheimer's *Factor of Proximity* [Wertheimer, 1939] that demonstrates the human preference for "isolated" substructures. Connectivity measures the "isolation"

of a substructure by computing the inverse of the average number of external connections over all the instances of the substructure in the input graph. The connectivity of a substructure $s$ with instances $I$ in the input graph $G$ is computed as

$$connectivity(s,G) = \frac{|I|}{\sum_{i \in I} |external\_conns(i)|}$$

The final heuristic, *coverage*, measures the fraction of structure in the input graph described by the substructure. Although cognitive savings measures the amount of structural, the coverage heuristic includes the relevance of this savings with respect to the size of the entire input graph. Coverage is defined as the number of unique nodes and edges in the instances of the substructure divided by the total number of nodes and edges in the input graph. Thus, the coverage of a substructure $s$ with instances $I$ in the input graph $G$ is computed as

$$coverage(s,G) = \frac{\#unique\_nodes(I) + \#unique\_edges(I)}{\#objects(G) + \#relations(G)}$$

Since the cognitive compression of our environment is one of the main goals in our search for repetitive structure, the cognitive savings heuristic is the main heuristic controlling the substructure discovery process. The compactness, connectivity and coverage heuristics refine the cognitive savings by increasing or decreasing the value to reflect specific qualities of the substructure. The cognitive savings has units of graph size; whereas, the other three heuristics are unitless ratios. The value of a substructure $s$ for an input graph $G$ is computed as the product of the four heuristics. Applying the heuristic evaluation to the substructure in Figure 1, $value = 15 * 3/2 * 1/3 * 20/37 = 4.054$.

## 3 FUZZY GRAPH MATCH

Although exact structure match can be used to find many interesting substructures, many of the most interesting substructures show up in a slightly different form throughout the data. These differences may be due to noise and distortion, or may just illustrate slight differences between instances of the same general class of structures. Consider the image shown in Figure 3. The pencil and the cube would make ideal substructures in the picture, but an exact match algorithm may not consider these as strong substructures because they rarely occur in the same form and orientation throughout the picture.

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

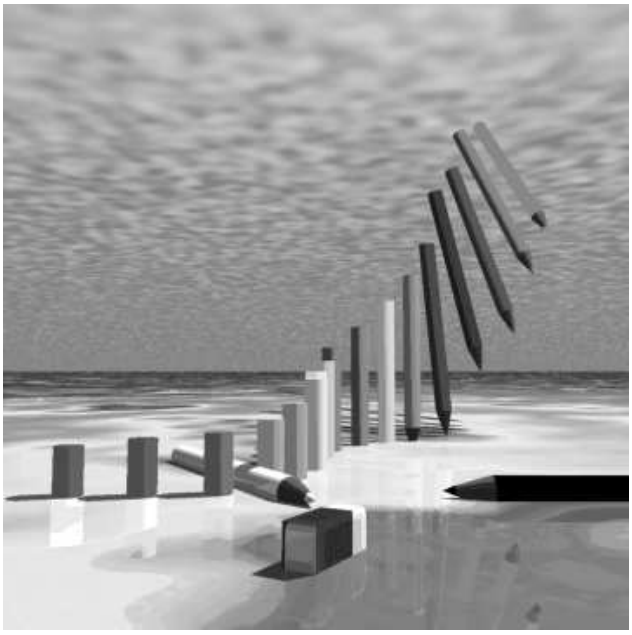

Figure 3: Scene analysis example

In this fuzzy match approach, each distortion of a graph is assigned a cost. A distortion is described in terms of basic transformations such as deletion, insertion, and substitution of nodes and edges. The cost for deleting (or inserting) a node with label $A$ is denoted by DELNODE($A$) (or INSNODE($A$)). The cost for substituting a node with label $A$ by a node with label $B$ is given as SUBNODE($A$, $B$). Similarly, SUBEDGE($a$, $b$) is the cost for an edge substitution. Defining costs for edge deletion and insertion is slightly more difficult since these transformations are dependent on the bordering nodes. When a node is deleted, then all of its outgoing and incoming edges vanish also. Conversely, when inserting a node, there are usually edges to be inserted also to properly embed this node in the rest of the graph. As a consequence, we define DELEDGE($a$) and INSEDGE($a$) as the costs for deletion and insertion of an edge with label $a$ while none of the bordering nodes are deleted or inserted, respectively. Specific substitution costs provide a means for the user to express a priori knowledge about the problem domain. Low costs should indicate high likelihood for a particular transformation and vice versa.

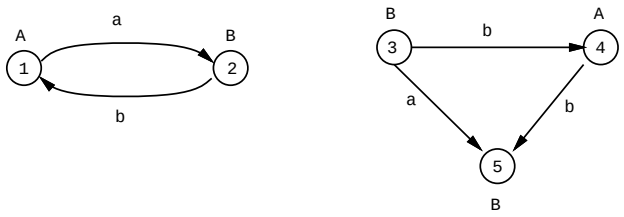


Figure 4: Two similar graphs $g_1$ and $g_2$.

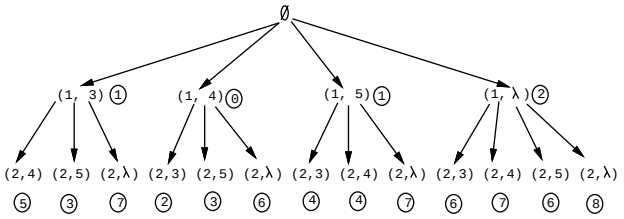


Figure 5: Search tree for computing the similarity between $g_1$ and $g_2$ in Figure 4.

A fuzzy graph match between two graphs $g_1$ and $g_2$ maps $g_1$ to $g_2$ such that $g_2$ is interpreted as a distorted version of $g_1$. Formally, a fuzzy graph match is a mapping $f : N_1 \rightarrow N_2 \cup \{\lambda\}$, where $N_1$ and $N_2$ are the sets of nodes of $g_1$ and $g_2$, respectively. A node $v \in N_1$ that is mapped to $\lambda$ (i.e., $f(v) = \lambda$), is deleted. That is, it has no corresponding node in $g_2$. Given a set of particular distortion costs as discussed above, we define the cost of a fuzzy graph match $cost(f)$, as the sum of the cost of the individual error transformations resulting from $f$. An example is shown in Figure 4. Assume our fuzzy subgraph match is $f(1) = 3$, $f(2) = 4$, and let

$$\begin{aligned}
&\text{SUBNODE(A, B)} = \text{SUBNODE(B, A)} = 1,\\
&\text{SUBEDGE(a, b)} = 1, \text{SUBEDGE(b, a)} = 2,\\
&\text{DELEDGE(a)} = \text{DELEDGE(b)} = 2,\\
&\text{DELNODE(A)} = \text{DELNODE(B)} = 2.
\end{aligned}$$

Then $\text{cost}(f) = \text{SUBNODE(A, B)} + \text{SUBNODE(B, A)} + \text{SUBEDGE(a, b)} + \text{DELEDGE(b)} = 5$.

Given graphs $g_1$ with $n$ nodes and $g_2$ with $m$ nodes, $m \geq n$, there are $\frac{m!}{(m-n)!}$ different fuzzy graph matches, each having a different cost, in general. We define the subgraph similarity of $g_1$ and $g_2$, $s(g_1, g_2)$, as one minus the fraction of the minimum cost fuzzy graph match between $g_1$ and $g_2$ over the size of the larger graph. That is,

$$s(g_1, g_2) = 1 - \frac{\min_f\{cost(f) \mid f \text{ is a fuzzy graph match between } g_1 \text{ and } g_2\}}{max(nodes_{g_1}+edges_{g_1}, nodes_{g_2}+edges_{g_2})}.$$

If $g_1$ is identical to $g_2$, then $s(g_1, g_2) = 1$. The less similar the two graphs are, the closer to 0 the similarity measure will become. If the cost of the match is greater than the size of the larger graph, $s(g_1, g_2)$ is defined to be 0.

Given $g_1$, $g_2$, and a set of distortion costs, the actual computation of $s(g_1, g_2)$ can be done by a tree search procedure. A state in the search tree corresponds to a partial match that maps a subset of the nodes of $g_1$ to a subset of the nodes in $g_2$. Initially, we start with an empty mapping at the root of the search tree. Expanding a state corresponds to adding a pair of nodes, one from $g_1$ and one from $g_2$, to the partial mapping constructed so far. A final state in the search tree is a match that maps all nodes of $g_1$ to $g_2$ or to $\lambda$. The complete search tree of the example in Figure 4 is shown in Figure 5. The numbers in circles in this figure represent the cost of a state. As we are eventually interested in the mapping with minimum cost, each state in the search tree gets assigned the cost of the partial mapping that it represents. Thus the goal state to be found by our tree search procedure is the final state with minimum cost among all final states. From Figure 5 we conclude that the minimum cost fuzzy graph match of $g_1$ and $g_2$ is given by the mapping $f(1) = 4$, $f(2) = 3$. The cost of this mapping is 2 and therefore the similarity between $g_1$ and $g_2$ is $1 - \frac{2}{3+3} = 0.667$. This similarity depends not only on $g_1$ and $g_2$, but also on the distortion costs.

The order of complexity of the fuzzy graph match is equivalent to that of exact graph match[1]; however, the fuzzy match offers the additional benefit of assigning a similarity measure to each possible mapping. Therefore, integrating the fuzzy match into SUBDUE incurs no additional computation cost, and allows SUBDUE to consider fuzzy instances of the discovered substructures.

## 4 FUZZY SUBSTRUCTURE DISCOVERY

Integration of the fuzzy graph match into SUBDUE implies that the graph match routine will now return a cost of the match instead of true or false. The higher the cost, the worse the match between the substructure and instance. This match cost requires changes in the heuristic formulas such that the values are weighted by the match cost. First, we define the instance weight $w$

[1] Bunke and Allermann [1983] provide a more complete discussion of the inexact graph match and its computational complexity.

of an instance $i$ of a substructure $s$ to be

$$w(i,s) = 1 - \frac{matchcost(i,s)}{size(i)}$$

where $size(i) = \#nodes(i) + \#edges(i)$. If the match cost is greater than the size, then $w(i,s) = 0$. Compared to the previous heuristics, the new heuristics, shown below, are a weighted average over the instances. In these formulas, $s$ is the substructure, $G$ is the input graph, and $I$ is the set of instances of $s$ in $G$. Since every subgraph in $G$ fuzzy matches a substructure to some extent, the set of instances of a substructure is filtered using a user-supplied upper-bound on the match cost of an instance. In the formula for coverage the $unique\_struct(i)$ of an instance $i$ is the number of nodes and edges in $i$ that have not already appeared in previous instances in the summation.

$$cognitive\_savings(s) = \left[\sum_{i \in I} w(i,s) * size(i)\right] - size(s)$$

$$compactness(s) = \frac{\sum_{i \in I} w(i,s) * \frac{\#relations(i)}{\#objects(i)}}{|I|}$$

$$connectivity(s) = \frac{|I|}{\sum_{i \in I} w(i,s) * |external\_conns(i)|}$$

$$coverage(s) = \frac{\sum_{i \in I} w(i,s) * unique\_struct(i)}{size(G)}$$

One purported benefit of substructure discovery is the ability to reduce the cognitive complexity of the input environment by replacing instances of a substructure with only a pointer to the substructure definition. However, now that instances of a substructure may no longer be exact matches, replacement is more difficult. In the exact match case, instances can be replaced with a single node representing the substructure with edges to nodes representing the original nodes of the substructure instance involved in external connections. Replacing fuzzy instances while preserving the ability to reproduce the original input graph involves additional edges to nodes describing the graph distortions required to produce the original instance from the substructure definition. Further fuzzy substructure discovery would ignore these distortion annotations. Assuming the substructure is sufficiently large and the number of distortions is small, this replacement process will still result in compression of the input graph. If exact reproduction of the input graph is not required, the distortion annotations can be removed from the graph, affording addition compression.

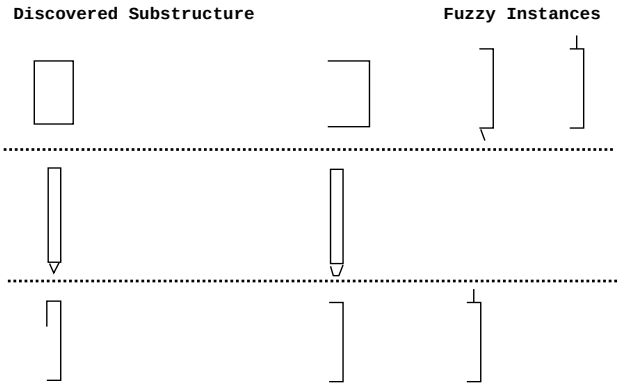


Figure 6: Substructures found in pencil scene

## 5 EXAMPLES

The first example comes from the domain of scene analysis and uses a graphical representation of the image in Figure 3. This representation consists of two types of edges (*edge* and *space*) and three types of nodes ($L$, $T$ and $A$). An *edge* edge represents a line in the image, and a *space* edge links the non-overlapping pencils together. The node labels come from the Waltz labeling [Waltz, 1975] of the junctions of the lines in the image, where $A$ stands for an arrow junction. At this stage, we have not included distance information in the graph representation, but such information is easy to add. Using this representation, a line drawing of the image would consist of rectangles (pencils stuck in the surface), partially occluded rectangles (overlapping pencils), and rectangles with triangles on the end (pencils with sharp points).

For this example, all graph distortion costs were set to one, and the threshold on the match cost was two (i.e., a fuzzy instance is no more than two distortions from the substructure). Figure 6 shows three substructures discovered by SUBDUE in the graphical image. Each substructure has several exact instances, but Figure 6 only shows the fuzzy instances. The first substructure is the rectangle with fuzzy instances that deviate from the rectangle by one line. The second substructure is the full, sharp pencil with one fuzzy instance being a pencil with a dull or hidden point. The third substructure forms a partially occluded pencil with two

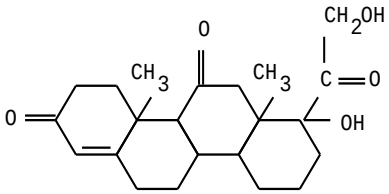


Figure 7: Cortisone

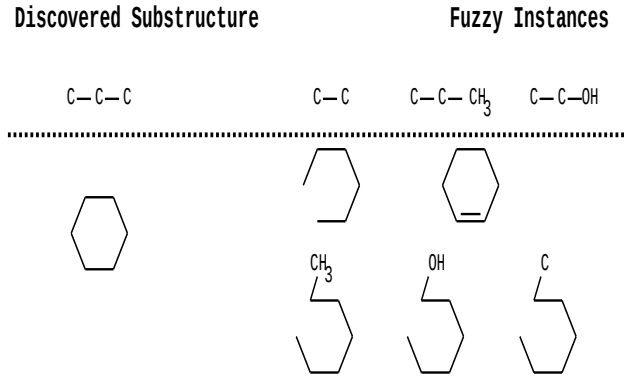


Figure 8: Substructures found in Cortisone compound

fuzzy instances deviating from the substructure by a single line. Thus, SUBDUE identified the major replicated components of the image along with similar, but not identical, instances. Less similar fuzzy instances might be included by increasing the upper bound on the match cost.

The second example comes from the domain of chemical structure analysis. Figure 7 shows the structural formula for the organic compound Cortisone. The graph representation of the compound consists of five types of node: C (carbon), O (oxygen), OH, $CH_3$, and $CH_2OH$; and two types of edges: single bond and double bond. The hexagons in Figure 7 have a single carbon at each vertex, and each side represents a single bond unless accompanied by a second parallel line.

As in the first example, all distortion costs were set to one, and the threshold on match cost was two. Figure 8 shows two substructures discovered by SUBDUE in the Cortisone example. The first is a three carbon chain connected by single bonds. The second substructure is a ring of six carbons connected by single bonds. In addition to these substructures, Figure 8 also shows some of the fuzzy instances matching the substructure. Each fuzzy instance differs from the substructure by one edge; therefore, the match cost is low.

As in the image example, SUBDUE identifies some highly repetitive substructures within the chemical compound. Again, increasing the match cost upper bound threshold would allow less similar instances. In the image example, the discovered substructures could now be used to re-express the image in terms of the three types of pencils. This would greatly reduce the complexity of the graph representation. The advantages in the chemical analysis domain would come from discovering previously-unknown molecules and reducing the cognitive complexity of the compound by abstracting over discovered molecules.

## 6 CONCLUSIONS

Substructure discovery allows the identification of interesting and repetitive chunks of structure in a structural representation of the environment. The substructures may be used as new concepts found in the environment and as a means of reducing the complexity of the representation by abstracting over instances of the substructure. However, the instances of a substructure found in the environment rarely occur in the exact same form. Adding the fuzzy graph match to the substructure discovery process incurs little computational penalty, allows the discovery of fuzzy concepts, and enhances the ability to compress the representation of the environment.

Fuzzy substructure discovery has numerous applications as evidenced by the examples in Section 5. One application is image analysis in which SUBDUE may find interesting substructures in the image corresponding to frequent similar objects in the environment. The instances can be replaced with a pointer to the substructure definition to effect image compression. The fuzzy graph match also has applications in image recognition for finding similar instances of a previously-discovered substructure in a new environment. Another application of fuzzy substructure discovery is in the domain of chemistry and molecular biology for finding interesting molecules or compounds that afford significant cognitive compression of the chemical compounds.

Further experimentation is necessary to validate the interestingness and accuracy of the discovered substructures. Interestingness could be validated by human expert approval of substructures discovered in real domains. Accuracy could be validated by quantifying the compression afforded by discovered substructures and attempting to discover substructures purposefully embedded in a large graph structure. Experimentation is also necessary to determine the effects of the distortion costs and match cost threshold on the

discovered substructures; especially since these parameters offer a means by which a user could supply the discovery process with domain-specific information.

One enhancement to SUBDUE will add the ability to weight the various heuristics so that the user can include domain-specific knowledge into the discovery process. Future directions will take advantage of a new capability in the fuzzy graph match to dynamically change graphs. In other words, if the graph represents a visual image from a camera, as the camera moves, the graph of the image changes as well. This capability will allow SUBDUE to dynamically discover substructures that did not appear in the original graph.

## References


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

Holder, L. B. 1988. Discovering substructure in examples. Master's thesis, Department of Computer Science, University of Illinois, Urbana, IL.

Holder, L. B. 1989. Empirical substructure discovery. In *Proceedings of the Sixth International Workshop on Machine Learning.* 133–136.

Minton, S. 1988. *Learning Search Control Knowledge: An Explanation-Based Approach.* Kluwer Academic Publishers.

Prather, R. 1976. *Discrete Mathemetical Structures for Computer Science.* Houghton Miffin Company.

Waltz, D. 1975. Understanding line drawings of scenes with shadows. In Winston, P. H., editor 1975, *The Psychology of Computer Vision.* McGraw-Hill.

Wertheimer, M. 1939. Laws of organization in perceptual forms. In Ellis, W. D., editor 1939, *A Sourcebook of Gestalt Psychology.* Harcourt, Brace and Company. 331–363.

Whitehall, B. L. 1987. Substructure discovery in executed action sequences. Master's thesis, Department of Computer Science, University of Illinois.

Wolff, J. G. 1982. Language acquisition, data compression and generalization. *Language and Communication* 2(1):57–89.

Zadeh, L. A. 1973. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics* 3:28–44.