

University of Montana

## ScholarWorks at University of Montana

---

Graduate Student Theses, Dissertations, &  
Professional Papers

Graduate School

---

1998

### An evolutionary hill-climbing approach to symbolic theory revision

Orest Jacob Pilskalns  
*The University of Montana*

Follow this and additional works at: <https://scholarworks.umt.edu/etd>

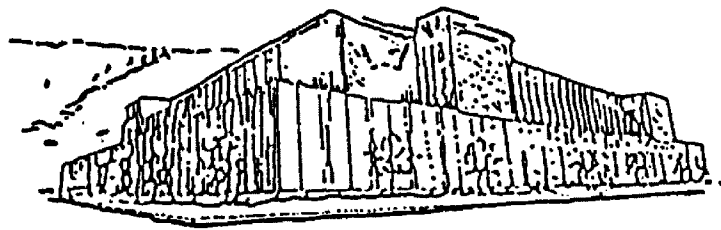
**Let us know how access to this document benefits you.**

---

#### Recommended Citation

Pilskalns, Orest Jacob, "An evolutionary hill-climbing approach to symbolic theory revision" (1998).  
*Graduate Student Theses, Dissertations, & Professional Papers*. 6607.  
<https://scholarworks.umt.edu/etd/6607>

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact [scholarworks@mso.umt.edu](mailto:scholarworks@mso.umt.edu).



Maureen and Mike  
**MANSFIELD LIBRARY**

\*

The University of **MONTANA**

---

Permission is granted by the author to reproduce this material in its entirety, provided that this material is used for scholarly purposes and is properly cited in published works and reports.

**\*\* Please check "Yes" or "No" and provide signature \*\***

Yes, I grant permission

No, I do not grant permission

Author's Signature

*Just Felder*

Date

*8-31-98*

Any copying for commercial purposes or financial gain may be undertaken only with the author's explicit consent.



**AN EVOLUTIONARY, HILL-  
CLIMBING APPROACH TO  
SYMBOLIC THEORY REVISION**

by

Orest Jacob Pilskalns

presented in partial fulfillment of the requirements

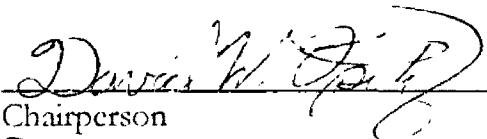
for the degree of

Master of Science

University of Montana

1998

Approved by:

  
Chairperson

  
Dean, Graduate School

8-31-98

Date

UMI Number: EP37408

All rights reserved

**INFORMATION TO ALL USERS**

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI EP37408

Published by ProQuest LLC (2013). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 - 1346

An Evolutionary, Hill-Climbing Approach to Symbolic Theory Revision (42 pp.)

Director: David W. Opitz *DWO*

This thesis presents an object-oriented, inductive learning system that is based on genetic algorithms and implemented in Java. A Genetic Algorithm (GA) is an optimization technique that many times can quickly and efficiently search global search spaces. However, their searching ability can suffer when making local refinements. Another shortcoming of GAs is their dependence of the initial population to contain the proper components to evolve the population into a more optimal state. Both drawbacks are addressed in this thesis. The learning technique applied by this system is a genetic algorithm with the traditional recombination and mutation operators and two independent procedures that may solve the GA's shortcomings. In order to effectively produce local refinements a hill-climbing procedure is used for local optimization that finds the best incremental change to an individual before placing the individual back into the population. In addition, a "domain theory" that represents an encapsulation of the current knowledge base about a task is used to create an initial population that contains the components necessary for an optimal solution. Results show an increase in overall performance of the GA by applying the hill climbing operator and the domain theory for generating an initial population.

# TABLE OF CONTENTS

Abstract.....	ii
List of Figures.....	iv
List of Tables.....	v
Acknowledgements.....	vi
1. Introduction .....	1
1.1 Genetic Algorithms: Some Strengths and Weaknesses .....	2
1.2 Evolutionary Models.....	3
1.3 Thesis Statement .....	4
1.4 Thesis Overview.....	5
2. The Inductive Learning System.....	6
2.1 A Domain-Theory Originated Genetically Modified Algorithm.....	6
2.1.1 The Language, Domain Theory, and Initial Population.....	7
2.1.2 Fitness and Fitness Proportional Selection .....	10
2.1.3 Mutation.....	11
2.1.4 Crossover.....	13
2.1.5 Hill Climbing.....	16
2.2 The Object-Oriented Design .....	17
2.2.1 The Class View .....	17
2.2.2 Dynamic Views.....	19
3. Results.....	23
4. Future and Related Work .....	27
5. Appendix A - Experimental Data Sets .....	28
Bibliography.....	36

## LIST OF FIGURES

<i>Number</i>	<i>Page</i>
Figure 2.1 - An Individual.....	9
Figure 2.2 - Mutation Deletion.....	12
Figure 2.3 - Mutation Addition.....	13
Figure 2.4 - Crossover.....	15
Figure 2.5 - Class View .....	18
Figure 2.6 - Dynamic View A .....	20
Figure 2.7 - Dynamic View B.....	21
Figure 3.1 - GA Mutation, Crossover, and Hill-Climbing Results .....	24
Figure 3.2 - GA and Hill-Climbing Results.....	25
Figure 3.3 - Size versus Crossovers .....	26
Figure A.1 - Gene Expression .....	29
Figure A.2 - Search by Signal .....	30



## LIST OF TABLES

<i>Number</i>	<i>Page</i>
Table 2.1 - DOGMA.....	7
Table 2.2 - Rules and Features.....	8
Table 2.3 - Crossover .....	14
Table 2.4 - Hill-Climbing.....	16
Table 3.1 - Inductive Learning Systems Compared.....	25
Table A.1 - Promoters Domain Theory.....	33
Table A.2 - Ribosome-Binding Sites.....	34

## ACKNOWLEDGMENTS

The arduous task of completing a graduate project is made possible through the efforts of not just one person, but through the help of an entire supporting cast. Therefore, I would like to thank my advisor David Opitz for providing the motivation, advice, and time to make this project possible. I would like to thank Alden Wright for his insightful comments and taking the time to be on my committee. I would like to thank Jim Jacobs for taking the time to be on my committee, for always being supportive of my educational endeavors, and for providing intellectual and financial undergraduate support. In addition, I would like to thank the entire faculty and staff of the Computer Science Department for their continual intellectual and financial support.

Most important of all, I would like to thank my wife Sasha for the love, encouragement, and help. Without her none of this would be possible.

## INTRODUCTION

Inductive learning algorithms are optimization techniques that learn from a set of labeled examples. A Genetic Algorithm (GA) is an optimization technique that can be used for inductive learning. GAs are an effective global search algorithm (Hart, 1994), however, building blocks are hard to find in complex problems (Forrest and Mitchell, 1996) and GAs are not effective at localizing (Hart, 1994). This thesis presents a new inductive learning system that addresses these deficiencies by using a hybrid GA/hill-climbing approach to refining background knowledge with a set of examples.

GAs, first introduced by Holland (1975), are global search and optimization methods. GAs employ an evolutionary filtering and refinement process resembling the biological phenomenon, first observed by Charles Darwin and Alfred Russel, referred to as *survival of the fittest*. Individual organisms in a population that are well adapted to their environment have a high survival rate and tend to reproduce more, those that are not as well adapted are more likely to perish. An individual that has the ability to thrive is known as being fit. New attributes are produced through biological mutations and reproduction of higher-fit individuals. Offspring of fit individuals will possibly inherit genetic components that are essential for increased survival, thus shifting the population to more optimal individuals.

GAs mimic its biological counterpart by taking a population of candidate solutions and evolving them into more-optimal solutions. Solutions are encoded as individuals or chromosomes, which are abstract representations of the

solution. Each solution or individual is subject to an evaluation function that assigns a fitness depending on how well the solution it encodes solves the problem at hand. In a traditional GA new and possibly better solutions are found by crossing over components or attributes of individuals producing new individuals. New individuals are also created by mutating an individual in a population. Mutation and crossover are referred to as genetic operators. A simple GA works as follows:

1. generate an initial population of individuals (candidate solutions)
2. calculate the fitness of each individual in the population
3. using a selection operator apply mutation and crossover to create a new population
4. go back to step 2

Each time this process is iterated it creates a new population referred to as a generation. An important aspect that has thus far been omitted is how individuals are chosen for reproduction. There are several methods used for selection. This thesis uses the most common method called fitness proportional selection, where individuals are chosen probabilistically proportional to their fitness.

### **1.1 GENETIC ALGORITHMS: SOME STRENGTHS AND WEAKNESSES**

The appeal of genetic algorithms is found in their simplicity and their ability to rapidly find solutions to certain difficult high dimensional problems (Forrest-Mitchell, 1993). GAs often are able to identify the most fit part of a large search space quickly and find a good solution (Hart, 1994). However, it has been

empirically demonstrated (Mitchell-Holland, 1994) that the search for good solutions, depending on the landscape of the search space, may take much longer than other optimization techniques such as hill climbing, and yet may still not provide an adequate solution.

Genetic algorithms perform well if the initial individuals in the population contain basic building blocks (Holland 1975, Goldberg 1989). A randomly created initial population may not contain the building blocks necessary for the algorithm to find good solutions; therefore, some method should be devised to ensure that the proper building blocks are available.

## 1.2 EVOLUTIONARY MODELS

Evolution suggests changes or adaptation of a species to the environment through the influence of genetic operators. The accepted model for biological evolution states that individuals may only inherit innate qualities known as the genotype or the genetic composition of an individual. This disallows the possibility of parents passing learned knowledge to a future generation. The phenotype is the combination of the genetic innate qualities as well as the attributes acquired in a lifetime. Jean Batiste de Lamarck proposed a different theory, *inheritance of acquired characteristics*, suggesting the phenotype might also be passed on to descendants. As an example, Lamarck suggests that a giraffe, through its lifetime may elongate its neck by reaching for leaves, and passes this acquired attribute onto its offspring. The Lamarckian model has been dismissed in the biological realm, however, GAs are in a simulated environment and are not constrained by biology. Thus, both models should be considered when constructing a GA.

### 1.3 THESIS STATEMENT

The inductive learning system developed in this thesis starts by utilizing a traditional GA's crossover and mutation operators (Koza, 1992). The actual structure is not represented by bit strings, but by propositional rules, similar to the style used by the programming language Prolog. In order to increase the GA's efficiency a hill-climbing optimization technique is used to supplement the GA's local refinement abilities. The hill climbing is similar to the Lamarckian evolution model in that individuals in the population keep the local refinements made by the hill climbing before being restored into the population.

A domain theory provides the knowledge base or the building blocks that are essential for a GA to find more optimal solutions quickly. The domain theory contains the available knowledge of the task to be learned and is encoded in the GA's propositional rule format.

The system uses both a test set and a validation set to measure the overall accuracy and the fitness of individuals in the population. The effectiveness of each genetic operator is measured and compared as well as the synergistic effect of the operators. In addition, the GA/hill climbing method presented in this thesis is compared to a naïve Bayes classifier and an artificial neural network.

**Thesis:** *The genetic algorithm is an effective global search technique while hill climbing as been shown to outperform genetic algorithms in local search spaces. Thus, combining hill climbing and traditional genetic operators should result in a more-optimal inductive learner. In addition, background knowledge in the form of*

*a rule base should provide the essential building blocks to aid the algorithm in finding more optimal solutions.*

#### **1.4 THESIS OVERVIEW**

The rest of the dissertation describes and empirically tests the genetic inductive learning system. The chapters will be arranged as follows:

- Chapter 2 describes in detail the encoding scheme for the individuals, the domain theory, and the genetic algorithm and all of its constituents. This chapter also discusses the object-oriented design of the entire system.
- Chapter 3 examines the many results of the inductive learning system and compares the inductive learning system to other inductive learning methods.
- Chapter 4 looks at the future work and related issues in genetic theory and inductive learning.
- Appendix A contains the domains used in testing and comparing the inductive learning system presented in this thesis.

## **THE INDUCTIVE LEARNING SYSTEM**

The inductive learning system is the entire software package that contains the genetic algorithm examined in this thesis. The system is built from interacting objects that the algorithm utilizes, and the algorithm itself is contained inside an n-fold cross validation test set environment. In addition, the software package sports a graphic user interface that contains visualizations of the genetic process. This chapter will be broken into two parts, the first describes the theoretical basis of the inductive learning system which includes a detailed look at the conceptual mechanics of the genetic algorithm, the language accepted by the GA, and the genetic operators. The second part of this chapter addresses the application of the theoretical model and the object-oriented design.

### **2.1 A DOMAIN-THEORY ORIGINATED GENETIC, MODIFIED ALGORITHM (DOGMA)**

The genetic algorithm, DOGMA, uses background knowledge, fitness proportional selection, crossover, mutation, and hill climbing to optimize a population of solutions. The learning ability in DOGMA can be segmented into three areas or methods. First, DOGMA learns from prior knowledge by utilizing the domain theory. Second, the genetic process we have previously discussed learns by passing useful information on to future generations through fitness proportional selection, crossover, and mutation. The third part in the learning algorithm optimizes individuals and passes on the optimized individual into the next generation. This could be considered Lamarckian evolution. The second



and third learning methods can be observed in isolation, however, DOGMA is the amalgamation of all three methods. Table 2.1 summarizes DOGMA.

Goal: Search for the most fit individual in a domain theory initialized population.

1. Set aside a validation set from the training instances.
2. Create each member of the initial population by randomly perturbing the domain theory (section 2.2.1)
3. Hill climb each initial individual (optional).
4. Evaluate the fitness of each population member.
5. Loop (until stopping criteria is reached)
  - a. Select individual(s) for reproduction by fitness proportional selection.
  - b. Create new individual(s) using mutation (section 2.1.3) or crossover (section 2.1.4).
  - c. Hill climb new individuals.
  - d. Evaluate the fitness of new individuals using the validation set.
  - e. Place new individuals into the population and probabilistically remove individuals from the population returning it to its original size.

Table 2.1

Before discussing the genetic operators, it is important to discuss DOGMA's encoding of individuals. Genetic algorithms traditionally have used bit strings to encode the solution to a task, however, DOGMA uses a propositional language. The next section describes the propositional language that DOGMA uses.

### **2.1.1 THE LANGUAGE, DOMAIN THEORY, AND INITIAL POPULATION**

In order to render a problem understandable to DOGMA, two types of information are necessary. The first type of information specifies the attributes of the training examples that are used by DOGMA to learn. The second set of

information encodes the rules that specify the domain theory or the knowledge base that defines the initial population.

Information, used in defining examples, is restricted to nominal features and special subclasses of nominal features. Nominal features are features that have all values specified, for example, the feature *color* may have three values red, yellow, and blue. Binary features are nominal features that have only two values true or false. Ordered features are nominal features that are totally ordered, for example, the feature *size* might be represented by the set (small, medium, large, very-large). Linear features are not presently incorporated into DOGMA, and will be discussed in Chapter 4 - Future and Related Work.

Information representing the domain theory takes the form of propositional rules. Propositional rules have a Boolean result, either negative or positive. The syntax of the language can best be demonstrated with the set of rules and the corresponding set of features and feature values in Table 2.2. Table 2.2 represents a hypothetical domain theory to determine if a day is good or bad for sailing. A tilde preceding a rule denotes the negation of a rule and is represented by a black line in Figure 2.1.

Propositional Rules	Features and Feature Values
Result: 1 of 2: (C,~D) C: 1 of 2: (f,E,h) D: 1 of 2: (g,h,f) E: 1 of 2: (f,g,i) f: windy = false g: outlook = sunny h: humidity = medium i: temperature = high	<b>outlook:</b> sunny, overcast, rain. <b>temperature:</b> high, medium, low. <b>humidity:</b> high, medium, low. <b>windy:</b> true, false.

Table 2.2

The rules in Table 2.2 that are denoted by lower case letters can be considered inputs. As an example, if we have an instance where the day that is not windy, rule f would be true. Rules can be formed through N-of-M propositions where at least N of the M antecedents has to be true for the propositional rule to be positive. Antecedents are the conditional members of the proposition, for example, in Table 2.2 *i* is antecedent of E. The rules usually contain a hierarchical structure that combines the effect of the input features. The rules and features in Table 2.1 can be represented graphically as a genetic individual (see Figure 2.1), where each node is a rule and the result is the rule that determines the output of the individual. The input nodes are called terminals and the others are referred to as intermediate nodes.

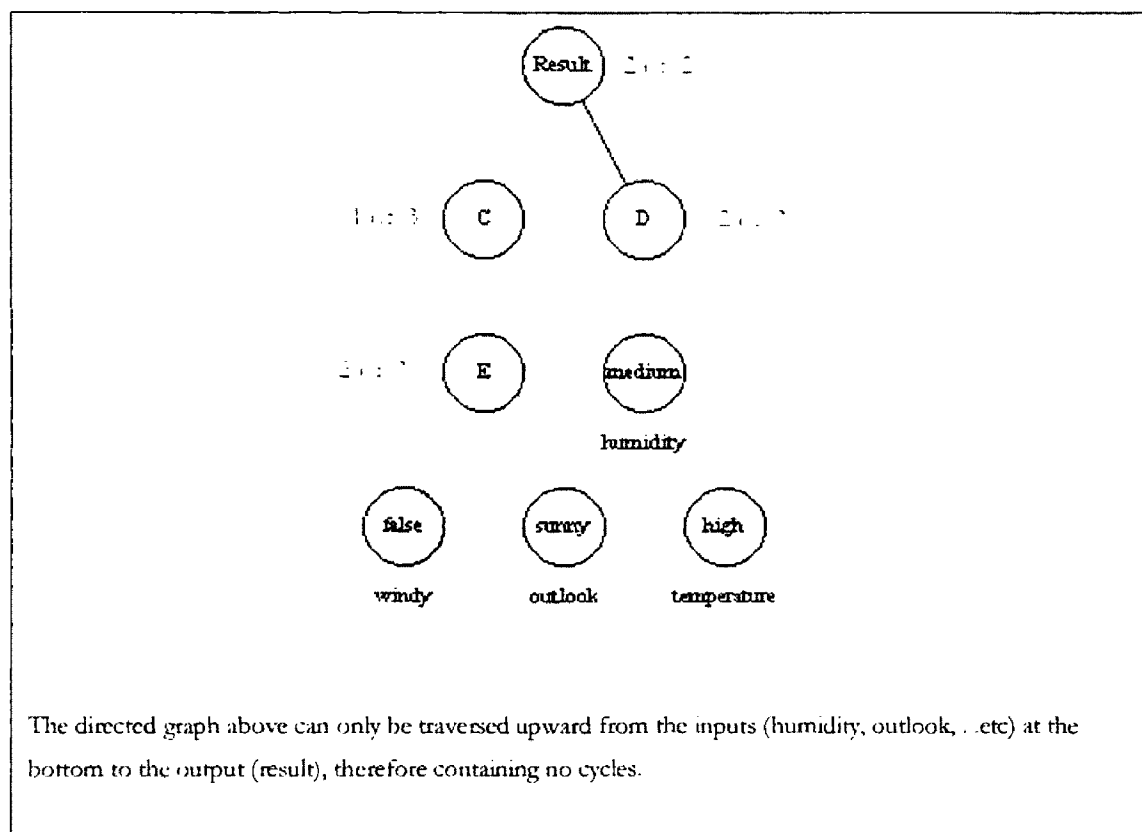


Figure 2.1

The graph must be directed and acyclic (i.e. without cycles), a restriction imposed by the language. The acyclic property allows for easy evaluation of the graph given an example.

The initial population is constructed by using the available domain theory and constructing an individual as in Figure 2.1. Currently<sup>1</sup>, the individual should have only one output node resulting in conclusions that are Boolean. To ensure diversity among the population the initial individual is randomly mutated each time an individual is created using the mutation operator as described in section 2.1.3.

### 2.1.2 FITNESS AND FITNESS PROPORTIONAL SELECTION

The fitness function evaluates the correctness of an individual by mapping an individual to a quantifiable numeric value. DOGMA measures fitness by taking a test set of examples and finding the number of examples that are correctly classified by the individual.

The fitness function provides a method of discovering the individuals that contribute possibly useful building blocks to future generations. Fitness proportional selection is a procedure that favors the selection of individuals for reproduction based upon the individuals fitness. Fitness proportional reproduction can be expressed with the following formula where  $f$  is the fitness function that maps an individual to a numeric value.

---

<sup>1</sup> See Chapter 4 - Future and Related Work

$$p_k = \frac{f(x_k)}{\sum_i f(x_i)}$$

Equation 2.1

The probability of choosing an individual  $x_k$ , is equal to the individual's fitness in relation to the sum of the population's fitness. Individuals that are selected to be parents for the genetic operations of crossover and mutation in step 5a of DOGMA are selected through fitness proportional selection. Individuals that are selected to be removed in step 5e are selected through an altered fitness proportional selection, where the least fit have the greatest possibility of being removed.

### 2.1.3 MUTATION

Mutation is the genetic operator responsible for adding diversity in a population and exploring new areas of the search space. New features and logical precepts can be added to an individual using mutation. The mutation operator randomly selects nodes and then adds or deletes antecedents of these randomly selected nodes. The structure of the individual may change due to the addition or deletion of new links, which are the connection between a node and its antecedent. This operator creates a new individual and possibly better solutions.

The mutation operator selects individuals through fitness proportional selection. It then selects a random number of nodes up to one-third the total number. Each selected node has the possibility of being altered by adding or deleting antecedents. Deletion is done by randomly choosing and removing an antecedent of a node. The number of antecedents changes so the N-of-M ratio also changes. The mutation operator changes N to match as closely as possible the ratio before M changed due to the deletion. For example, if a node needed 2-of-3

antecedents or 66% of the antecedents to be true before the deletion, then N would be set to one after the deletion. With N equal to one, 50% have to be true where as if N was chosen to be two, 100% would need to be true. Obviously, 50% is closer to 66% than 100%. If a node is no longer linked to any other nodes, it is deleted. Figure 2.2 demonstrates the deletion of two antecedents, notice that one node is deleted.

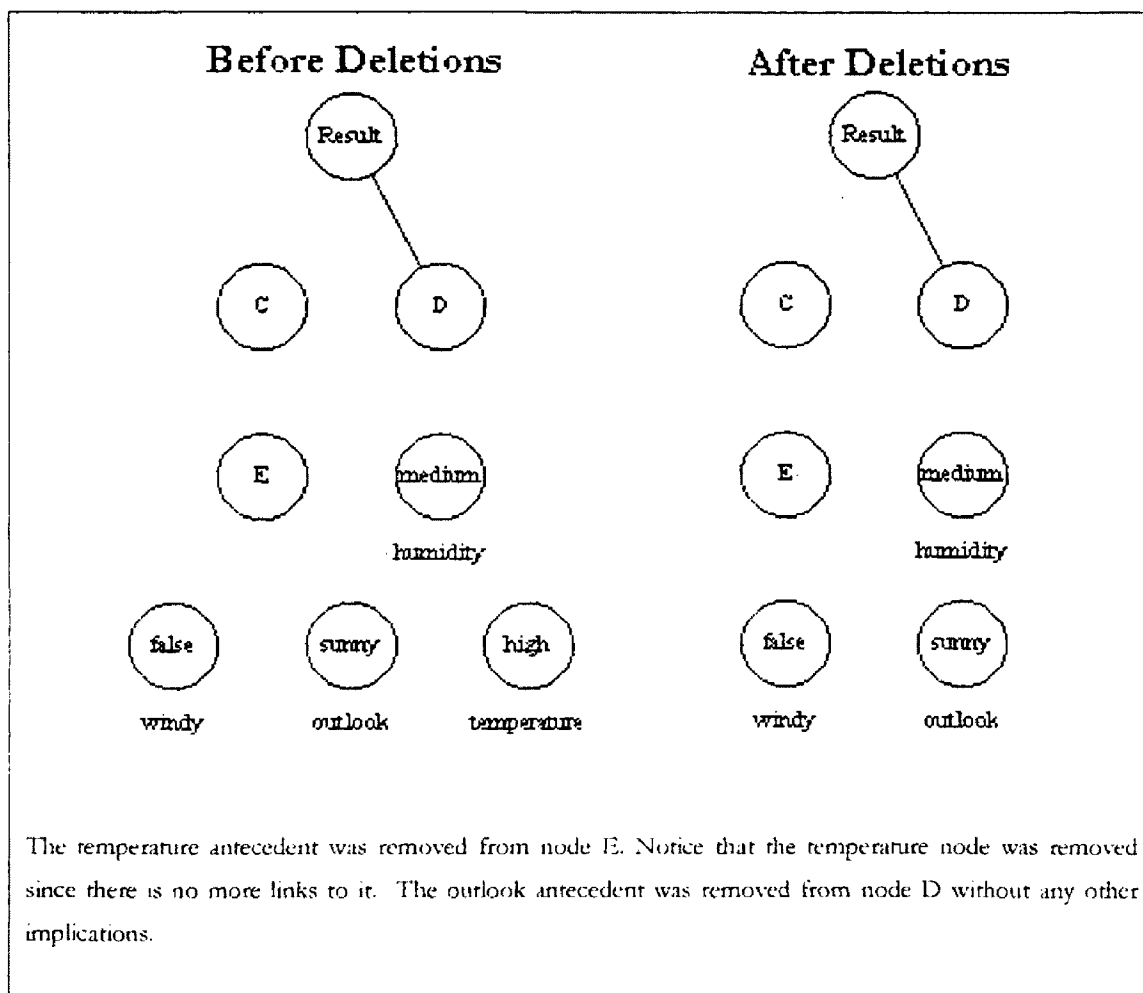


Figure 2.2

Adding new antecedents to nodes is another way of mutating an individual. Adding new links between existing nodes can be complex due to the possibility of circularity. To ensure that no circularity is introduced into a mutated

individual, the new antecedent can only be added if its longest path from the output is longer than the longest path of the node, that it will have as an antecedent. Figure 2.3 demonstrates two additions: a new link, and a new node.

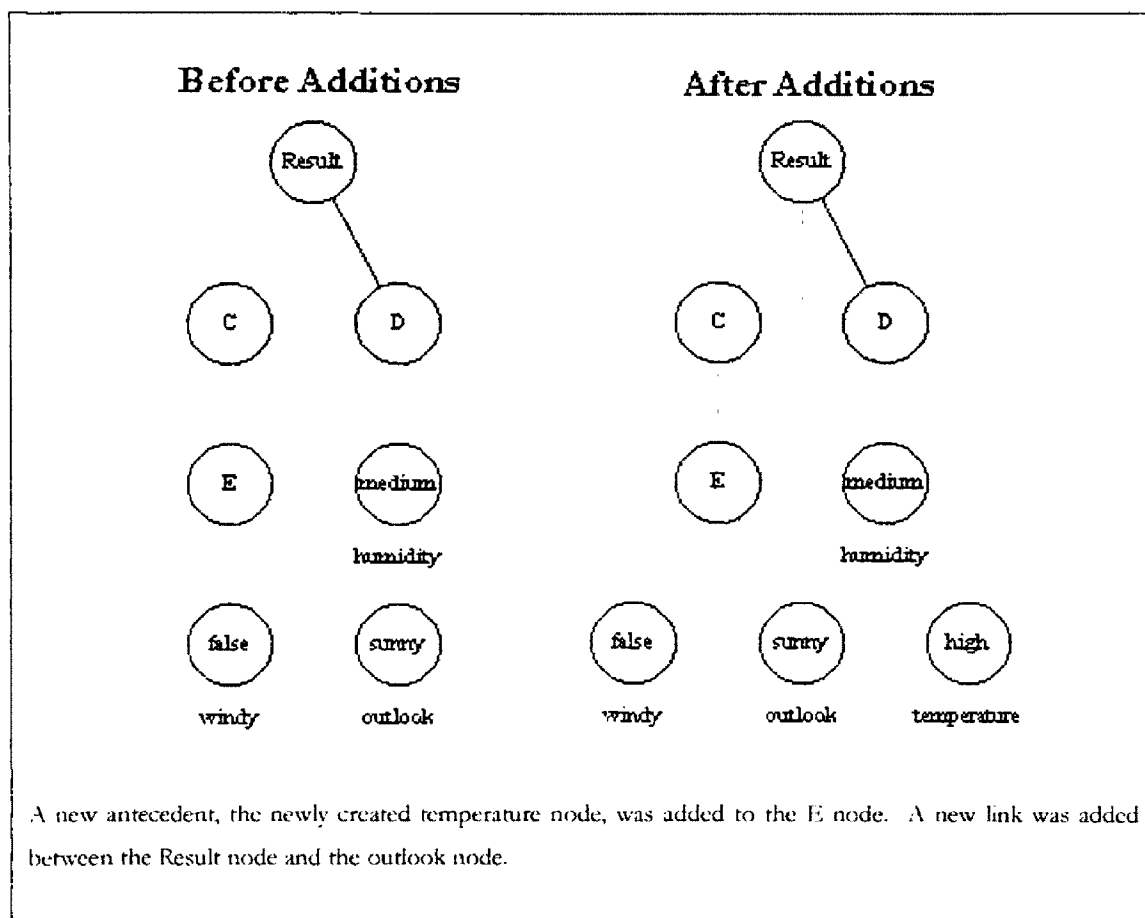


Figure 2.3

### 2.1.4 CROSSOVER

Crossover or recombination takes two parents found by fitness proportional selection and swaps sections of the individuals creating two new individuals. The crossover process is responsible for passing possibly good building blocks onto future generations. When discussing complex genetic operators, the terms parent and child often emerge referring to a nodes antecedent as a child and the node itself as the parent. There are varieties of methods for completing a genetic

crossover, the method employed by DOGMA is summarized in Table 2.3. Most methods can usually be divided into two distinct phases: (a) the division of the individuals and (b) the recombination of the individuals.

<p>Goal: To divide and recombine two individuals creating two new offspring.</p> <ol style="list-style-type: none"> <li>1. For each individual:             <ol style="list-style-type: none"> <li>a. Randomly select an intermediate node as a crossover point</li> <li>b. Recursively clone the nodes below the crossover point creating a branch</li> <li>c. Recursively delete the nodes below the crossover point creating a trunk</li> </ol> </li> <li>2. Create two new individuals by:             <ol style="list-style-type: none"> <li>a. Randomly selecting an intermediate node from each trunk as a reconnection point</li> <li>b. Create a link from the crossover node in the branch to the reconnection point in the trunk</li> </ol> </li> </ol>
--

Table 2.3

The division process randomly selects an intermediate node from a parent, which we will call the crossover node. The crossover node is then recursively cloned thus including the graph structure below the node. All links are kept intact except for those links that are dependent upon nodes that do not have the crossover node as an ancestor<sup>2</sup>. The cloned section of graph will be referred to as a branch. The crossover node is then recursively deleted--thus deleting any child nodes that are exclusively dependent upon the crossover node or its descendents. The graph that is left after the branch has been separated will be called the trunk. This is done to each parent individual creating two branches and two trunks.

The recombination process is direct. Simply take a branch and a trunk and reconnect them to create a new offspring. The reconnection takes place between a random intermediate node from the trunk and the crossover node from the branch. This is done for each branch and trunk creating two new individuals.

---

<sup>2</sup> An ancestor node is a node from which another node has descended from through the parent-child relationship.



This method helps preserve the entire sub-graph structure and thus keeps the basic building blocks from the domain theory mostly intact. Figure 2.4 demonstrates the crossover process.

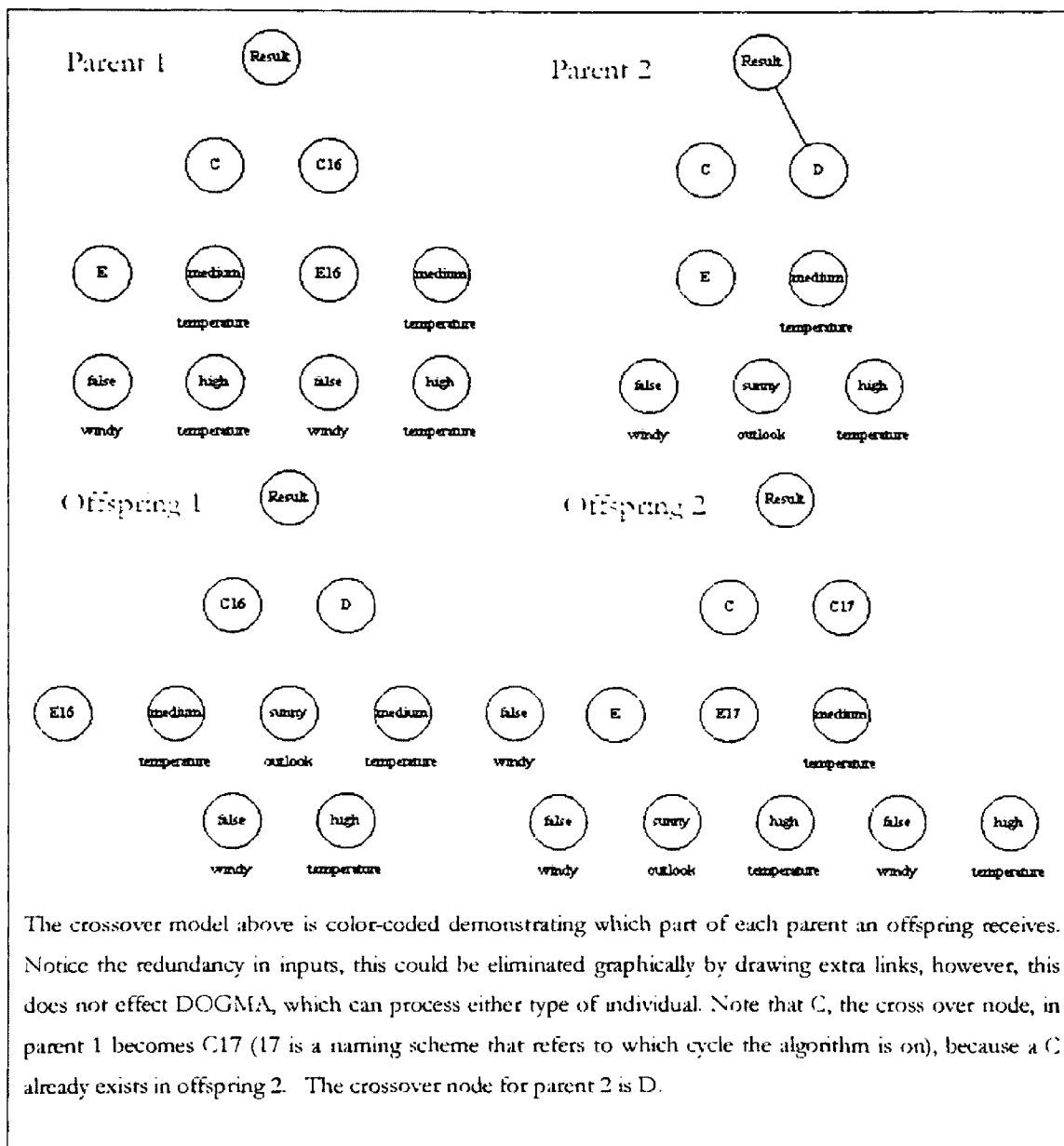


Figure 2.4

The new graphs that emerge from the crossover process could possibly use more connections between the newly added branch and the original trunk. This

connection process could be added into the crossover process, however, DOGMA is equipped with a hill climbing operator that can optimize the links in an individual.

### 2.1.5 HILL CLIMBING

Hill climbing is a greedy search algorithm that finds the optimal local improvement to an individual. Mutation and crossover provide a certain amount of global searching potential (Goldberg 1989), however, may not provide the refinements necessary to bring a population close to an optimal state. Hill climbing helps the entire population make small directional steps towards a local maximum, while mutation and crossover provide the large migratory leaps.

Hill climbing is accomplished by optimizing the linked structure within a DOGMA individual. Table 2.4 summarizes the hill climbing genetic operation.

<p>Goal: Optimize the linked structure and the N-of-M values in an individual.</p> <ol style="list-style-type: none"> <li>1. Randomly select an intermediate node to act as the node for optimization.</li> <li>2. Loop until output is reached. <ol style="list-style-type: none"> <li>a. Remove all parent links from the optimized node.</li> <li>b. Reconnect optimized node one link at a time to the target<sup>3</sup> nodes testing fitness and optimizing the N-of-M values for target nodes</li> <li>c. Select a parent of the node that is being optimized and set it as the new node for optimization</li> <li>d. go to the beginning of 2 and start cycle over</li> </ol> </li> </ol>
--

Table 2.4

A random intermediate node is picked from an individual. This node is then disconnected and reconnected incrementally to every node above<sup>4</sup> it in the directed graph. Each time it is reconnected to another node, a target node, the value of N in the N-of-M context is optimized and the fitness is measured. The

<sup>3</sup> Nodes that have a longest path that is shorter than the optimization node.

<sup>4</sup> Above implies the longest path is shorter.

maximum fitness determines the configuration that is chosen before continuing. The process is continued by randomly selecting a parent of the node just optimized and optimizing it until the root is reached.

An alternative hill climbing method is also implemented called full hill climbing. Full hill climbing optimizes every node in the individual, starting with the layer of nodes with the longest path. This method is obviously takes more time, however, is more vigorous in its search. The previous method discussed could be called partial hill climbing, because not all the nodes are visited.

## **2.2 THE OBJECT-ORIENTED DESIGN**

There are a variety of object-oriented modeling practices, symbols, terminology, and methods. Until recently, no standardization or common modeling language had been proposed. The Booch and OMT methods have been prominent in designing object-oriented models, but have fallen short in certain areas and contain disparate terminology and symbols. The Unified Modeling Language (UML) has encapsulated many of the same principles as the Booch and OMT methods and has standardized the terminology and symbols. Therefore, UML is the obvious choice for presenting the object-oriented design of the inductive learning system covered in this thesis. There will be no overview of UML terminology and symbols, however, there will be annotation when necessary. The entire UML domain is quite vast, so only the logical view will be presented in this section.

### **2.1.1 CLASS RELATIONSHIPS**

The logical view may contain several different diagrams; one of the most useful is the class diagram. The class diagram is a static model type that describes the system in terms of classes and relationships among the classes. One goal of the

class diagram is to define a foundation for other diagrams namely dynamic models. Figure 2.5 is a class diagram that describes the class relationships in the system without the detailed attributes of each class. The only relationships used in Figure 2.5 are association relationships. Association relationships represent a semantic connection between classes.

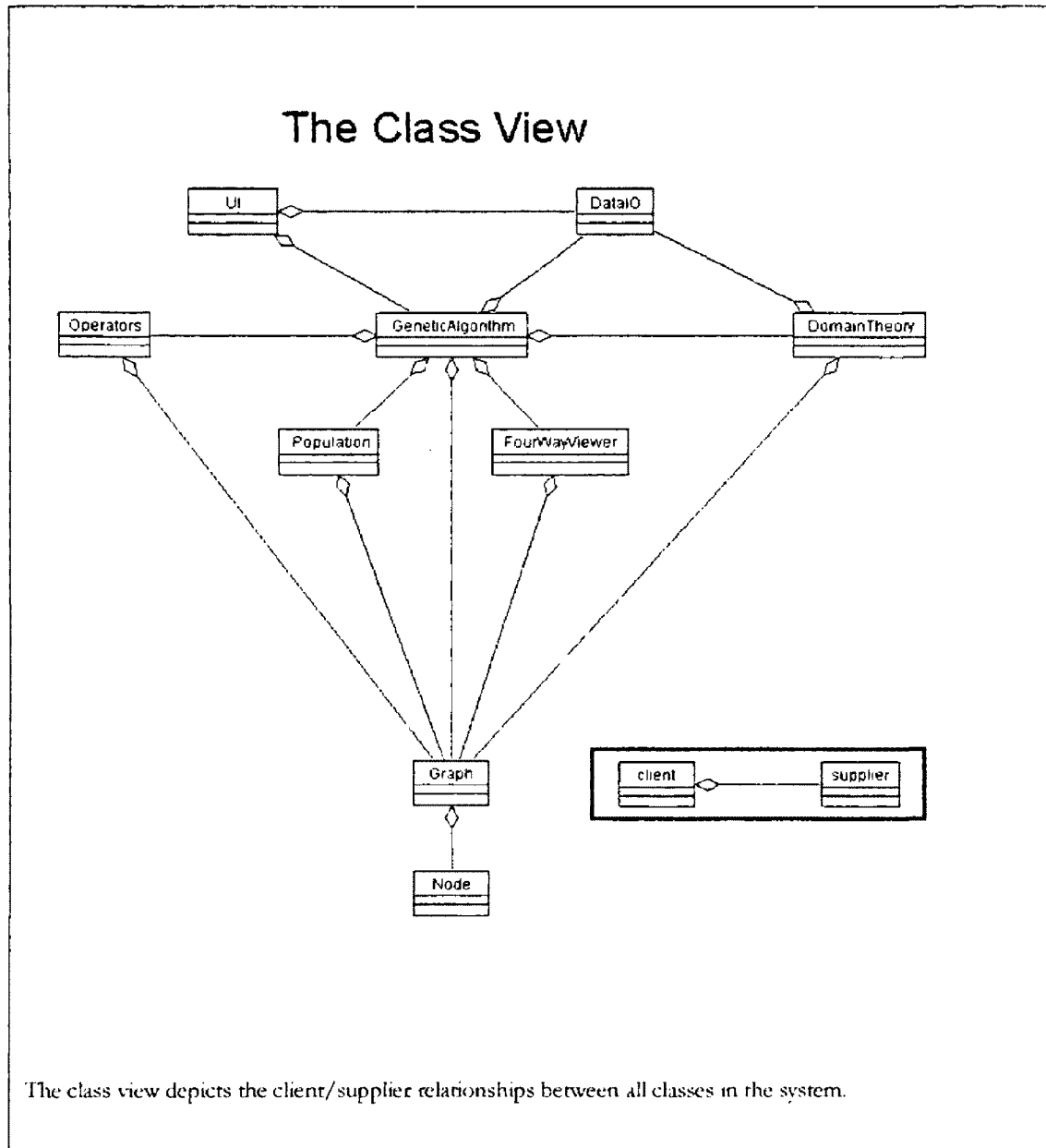


Figure 2.5

Association implies that a two-way relationship is present between two classes. However, Figure 2.5 shows a more restrictive association with client/supplier notation indicating only one of the classes accesses the other. The Node class is only accessed by the Graph class thus it is said to be contained by the Graph class.

### 2.1.2 The Dynamic View

The dynamic views rely on an understanding provided by the static class view. The distinction between the idea of a static class and the instantiation of the class resulting in an object must always be maintained, yet, a certain amount of flexibility should be allowed when discussing interactions of both. A description will be given of the class-object interactions concluding in sequence diagrams.

The main driver of the system is the UI class. The UI class, depending on the command line parameters, can run in GUI mode non-GUI mode. The UI has relationships with only two other classes the GeneticAlgorithm class and the DataIO class. The DataIO class is responsible for all file interactions. Only the UI class and the GeneticAlgorithm class have relationships with DataIO. The Genetic Algorithm class orchestrates the genetic process and directly or indirectly interacts with all other classes. The genetic individuals themselves are created from instantiations of the Node class and are represented through the Graph class. All genetic operators, such as hill climbing, crossover, fitness and mutation, are contained in the Operators class, which interacts with the Graph Class. The DomainTheory class is responsible for constructing an instantiation of the Graph class. The Population and FourWayViewer classes are simple containers and visual displays for instances of Graph classes.

The sequence diagram in Figure 2.6 is demonstrates how objects in the system are initially constructed. Figure 2.6 leaves out many of the details, yet delivers the

sequence of main events. The thread of control in the program can be traced by following the arrows. The dotted line indicates the lifeline where the object itself is represented by a vertical rectangle. The system is multi threaded thus two threads of control are present. The diagram indicates this with the notation "new thread" when the GeneticAlgorithm object is formed. The UI object still has a thread of control, but generates a new one with the construction of the GeneticAlgorithm object. The multi-thread of control is needed for the UI to be active to the user, yet allow the GeneticAlgorithm to run. Figure 2.6 primarily shows the construction of the initial objects such as the population and several individuals (Graphs).

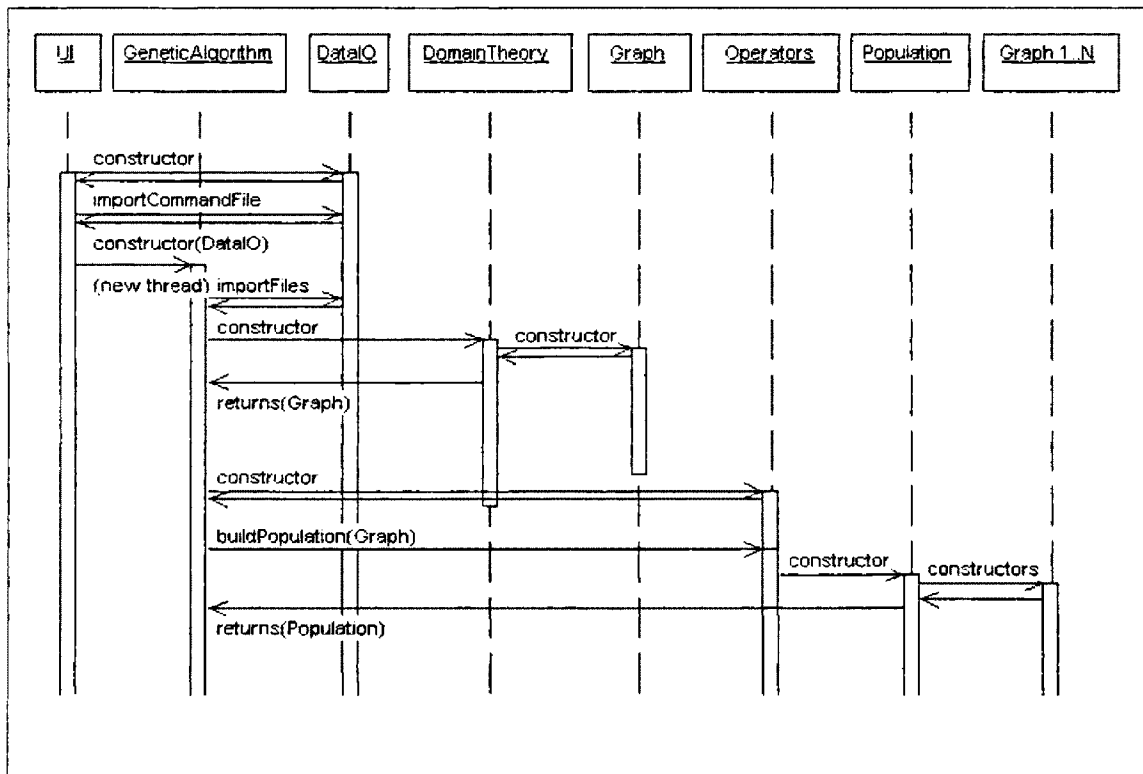


Figure 2.6

Once the population has been created, the FourWayViewer object is constructed and the genetic evolution process starts by the GeneticAlgorithm object making several method calls to the Operators object. The first method call to the

Operators object determines the fitness of each individual in the population. There is no reference to the data structures that are needed for the fitness function, this done to make the diagram readable. The next method call is to the fitness proportional selection function, which returns an individual, based on its previously calculated fitness. This is followed by method calls to either a mutation or crossover function followed by a hill climbing method. Figure 2.7 demonstrates this process by starting out where Figure 2.6 left off.

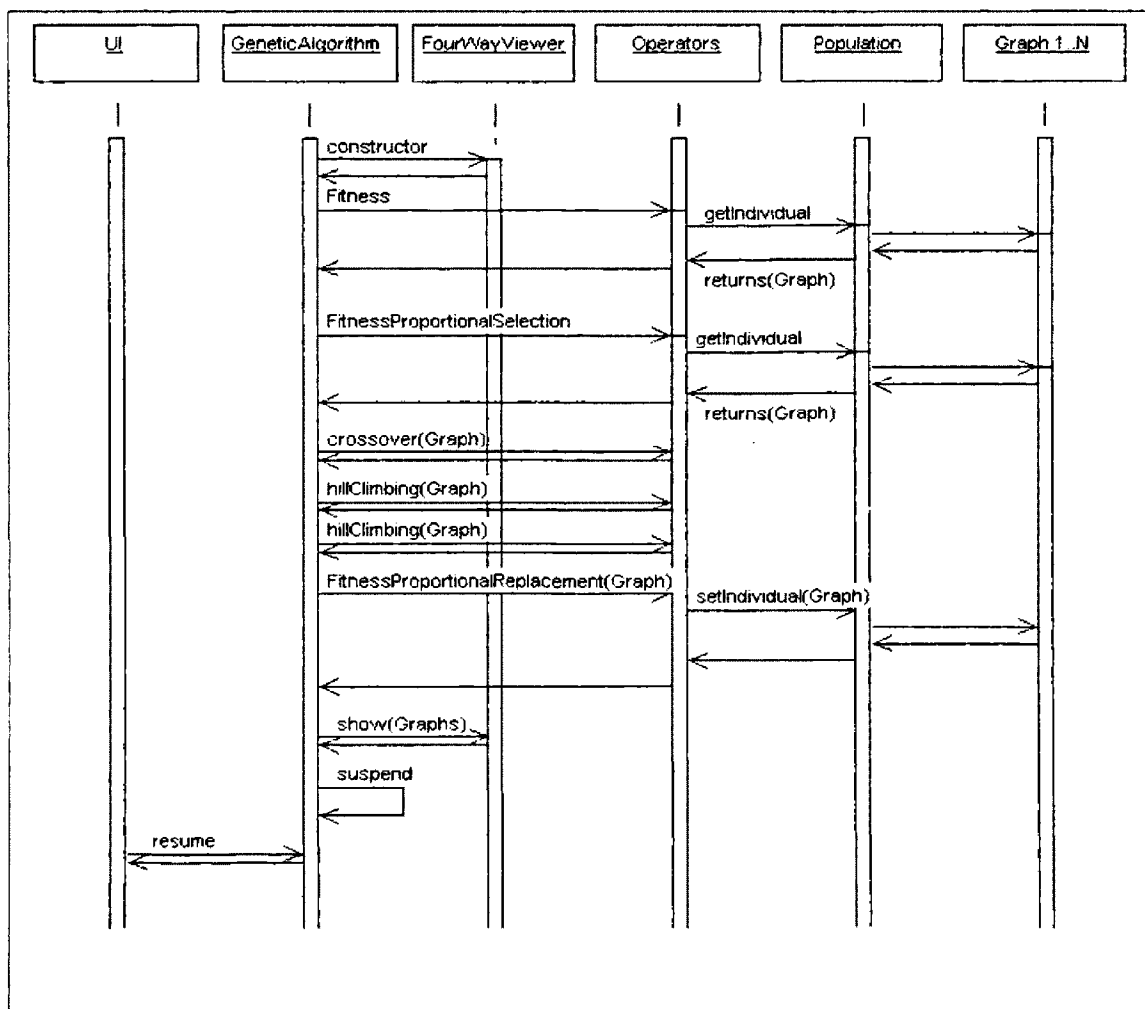


Figure 2.7

Notice that some objects are missing because they are no longer used by the system. Each time the GeneticAlgorithm finishes the genetic evolution

operations, it passes the graphs acted upon to the FourWayViewer and then suspends itself. The GeneticAlgorithm only continues when the UI, on a separate thread of control, sends a resume message.



## **THE RESULTS**

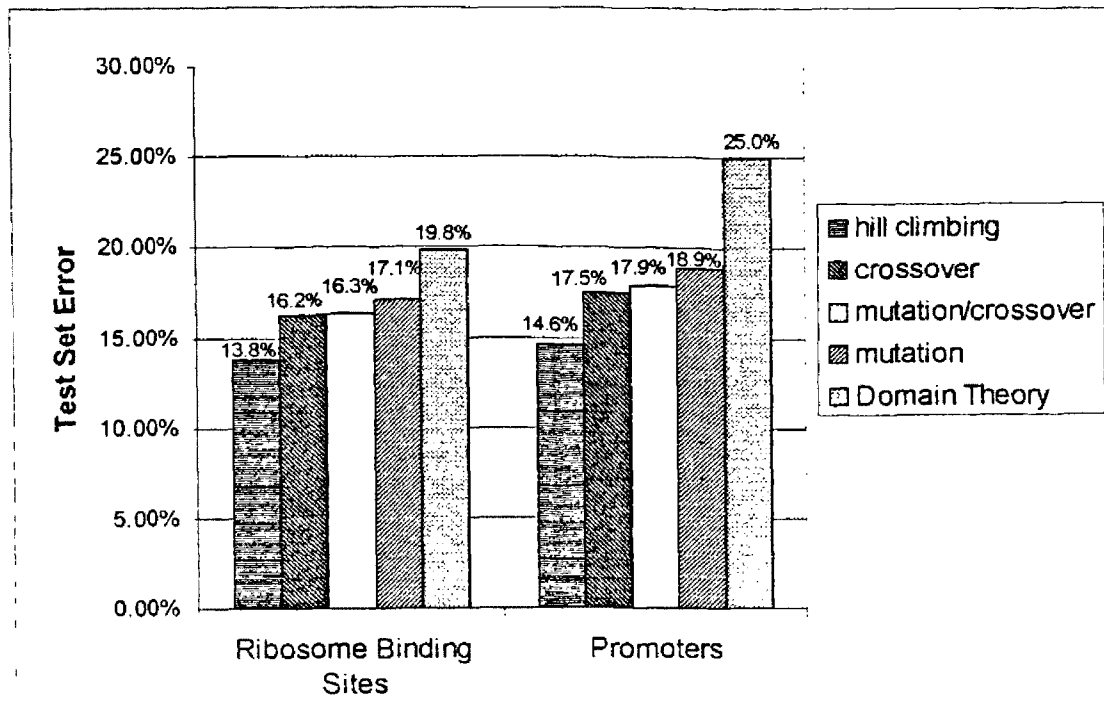
In this section, DOGMA is tested on two real-world problems from the Human Genome Project. The genetic operators of crossover and mutation are evaluated and compared to the hill-climbing operator. As a baseline, the accuracy of the domain theory for each problem is shown in the results. Operators will be observed independently and synergistically within the algorithm. In addition, DOGMA is compared to a Naïve Bayes Classifier and an Artificial Neural Network, which are other inductive learning algorithms.

### **3.1 COMPARING CROSSOVER, MUTATION, AND HILL CLIMBING**

This section compares DOGMA's crossover and mutation operators independently and then synergistically with hill climbing. The Ribosome Binding Sites (RBS) and the Promoters domains, both presented in Appendix A, are used for testing. The crossover operator is responsible for passing useful building blocks onto future generations, while mutation attempts to find novel solutions. Mutation is often considered a secondary operation that is only used sparingly (Goldberg, 1987). Figure 3.1 demonstrates the test set accuracy of the mutation and crossover operators independently and synergistically with a population of twenty.

The results presented are generated from ten-fold cross validation sets. Ten-fold cross validation divides the examples into ten sections and holds aside one section for testing accuracy, and allows the other nine sections to be used for training. This process is repeated ten times allowing each section to appear once

as a test set. DOGMA uses the nine sections to train by setting aside a test set



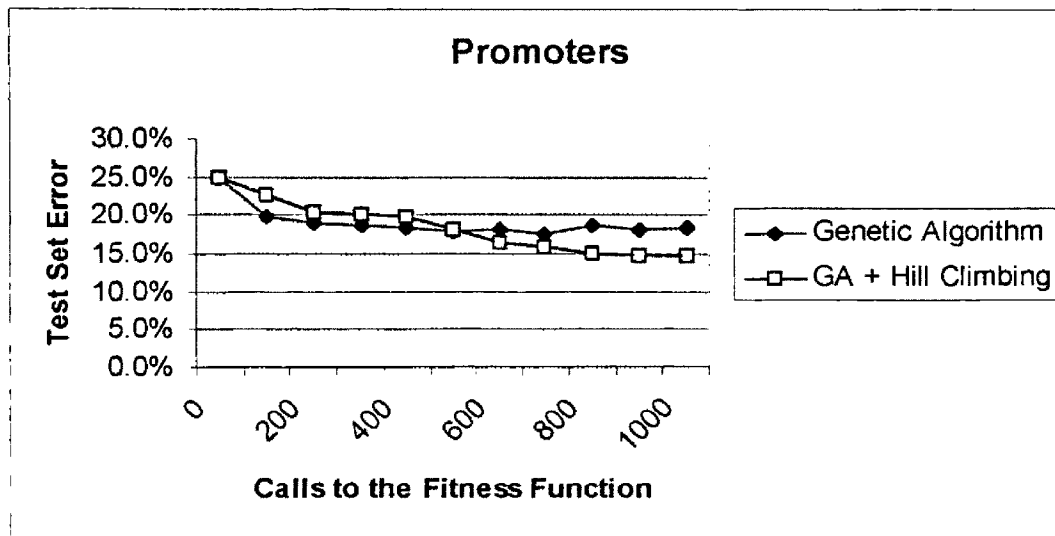
for fitness evaluation and using the remaining data for hill climbing.

Figure 3.1

The mutation/crossover algorithm uses an equal amount of mutation and crossover. Five hundred individuals are considered in each variation of the algorithm (not including the individuals seen by hill climbing). Figure 3.1 demonstrates that the crossover and mutation operators perform about the same. The hill climbing is added to the mutation/crossover algorithm by partially hill climbing every individual before adding it back into the population.

Hill climbing is time consuming due to the amount of individuals that must be considered to find the optimal incremental change. If the traditional genetic algorithm considered the same amount of individuals as a genetic algorithm with hill-climbing what would be the result? Figure 3.2 addresses this question by

tracking the number of calls to the fitness function by a traditional genetic



algorithm and a hill-climbing genetic algorithm.

Figure 3.2

Figure 3.2 demonstrates that the GA by itself quickly improves the test set error, however, it does not improve very quickly after the first 500 cycles. The GA in combination with the hill climbing does not improve as quickly, however, it maintains improvement as more individuals are considered.

One concern when creating populations of individuals is the size of the individuals after the crossover operator is applied. The crossover operator is designed to keep intact the basic building blocks. However, this could increase the size of individuals to a point where fitness is expensive to apply, and thus hill climbing a large individual could take a large amount of time. The average size of the initial individual is approximately equivalent to the size of the domain theory. Figure 3.3 shows the average size, in terms of nodes, of a population as a crossover operator is applied 100 times. The increase in size of individuals in the

population is noticeable, but does not appear to present a major problem since the increase seems to be linear.

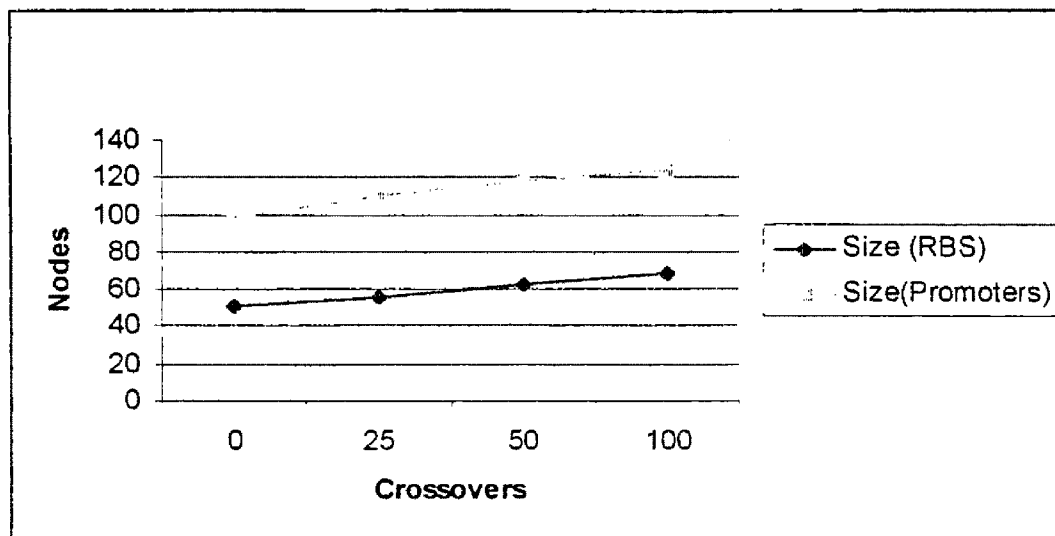


Figure 3.3

The GA with hill climbing outperforms a Naïve Bayes Classifier on the RBS and Promoters data sets. However, the GA with hill climbing is not yet able to compete with artificial neural networks. Table 3.1 demonstrates the results of DOGMA compared to a Naïve Bayes Classifier and a Neural Network.

	RBS	Promoters
Artificial Neural Network	9.7%	5.1%
Naïve Bayes Classifier	19.6%	24.6%
Genetic Algorithm with Hill Climbing	13.8%	14.6%

Table 3.1

## **FUTURE AND RELATED WORK**

The subject of genetic algorithms and hill climbing methods is immense. There are many different avenues to pursue. Three different categories of future and related work could be followed. First, the hill-climbing algorithm presented in this thesis could be optimized to a greater degree. Second, the language accepted by DOGMA could be expanded. Finally, a greater variation of genetic operators could be introduced.

Hill climbing aspects are of the most interest due to the possible increase in performance they provide. Making hill climbing practical, by decreasing the time constraints and by optimizing and exploring greedy search algorithms is necessary to conduct more expensive experiments. A compiled implementation language might be helpful in the testing phase.

The expressiveness of the current language accepted by DOGMA could be increased. Currently, DOGMA only accepts domain theories with Boolean outputs, this could be expanded to a larger number of outputs. In addition, DOGMA does not accept real valued features, in the future DOGMA could be modified to accept real values.

The power of the genetic algorithm could be increased by trying several different crossover methods. Several could be available, thus allowing the algorithm to use certain crossover methods that are effective in special landscapes or instances. The mutation operator could also be modified into several different variations and applied dynamically within the algorithm.

*Appendix A***EXPERIMENTAL DATA SETS**

The inductive learning system presented in thesis is tested on two real-world Human Genome problems. The domain theories and data sets are explained in this appendix (explanations and figures are mostly directly copied from Opitz, 1995).

**A.1 FINDING GENES IN DNA SEQUENCES**

The two domains in this appendix are important sub-problems in the computer analysis of DNA sequences. DNA is a linear sequence of four *nucleotides* - adenine, guanine, thymine, and cytosine - that are commonly abbreviated by the letters A, G, T, and C. Genes are subsequences of DNA that serve as blueprints for proteins, which in turn provide most of the structure, function, and regulatory mechanisms of cells and are thus the key building blocks of organisms. Researchers are currently sequencing large volumes of DNA; however, biologists are only able to study small sections of DNA at a time. Thus, the Human Genome Project (Cooper, 1994) will produce long runs of DNA that have not been analyzed biologically. Therefore, it is necessary to develop automated techniques that are able to find where genes occur in these unanalyzed sequences.

Figure A.1 illustrates the process of gene expressions. This process is broken into two phases transcription and translation. Transcription happens when the enzyme RNA - polymerase transcribes DNA into an RNA molecule called messenger RNA (mRNA). The enzyme does this by first binding to a DNA sequence, called a promoter that precedes the gene. It then transcribes the DNA

sequence into a similar RNA sequence, except that the nucleotide thymine is replaced with the nucleotide uracil (U).

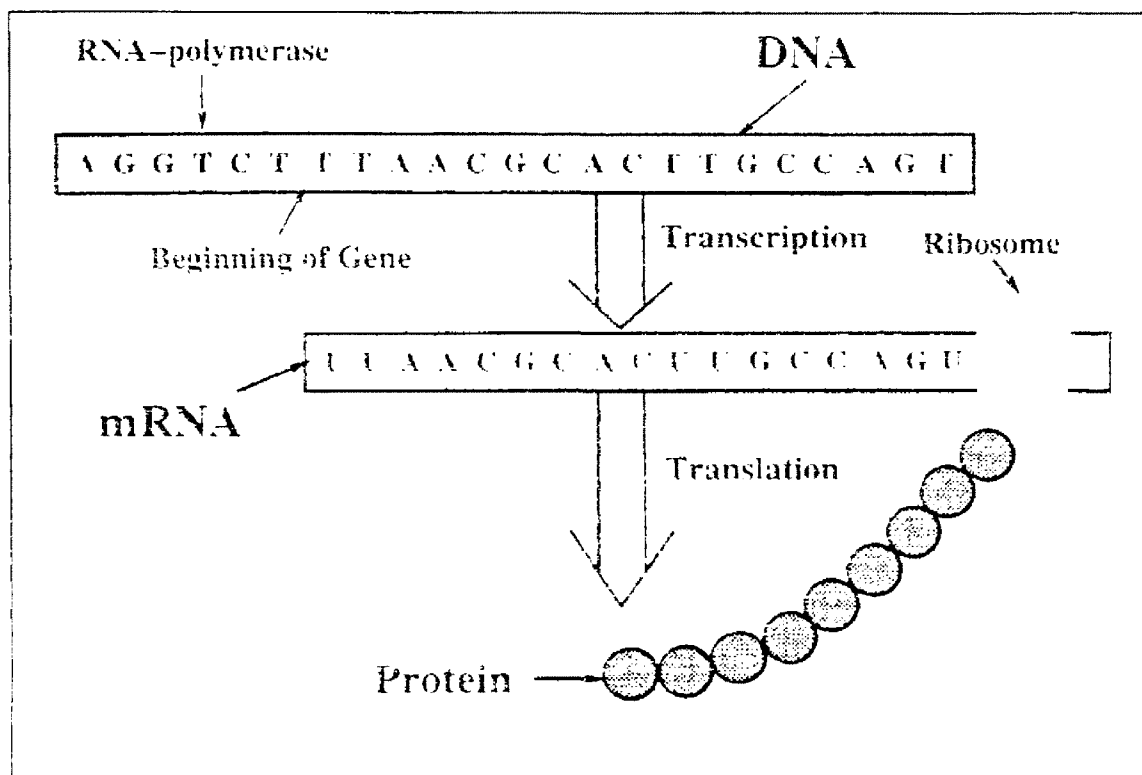


Figure A.1

Translation occurs when the ribosome molecule reads the mRNA strand and assembles a protein chain. One common approach to finding genes is called search-by-signal (Stormo, 1987). This approach works by trying to indirectly find genes through specific signals that are associated with gene expression. Not only are these signal detections important for finding genes, they are important in their own right to understand the mechanisms of gene expression. Figure A.2 illustrates how I represent the search-by-signal problems in a genetic algorithm. The genetic individual is given a fixed length window of DNA with the task of deciding if the desired signal is located at a fixed location in the window. A trained individual can then scan a DNA sequence, finding potential points of interest.

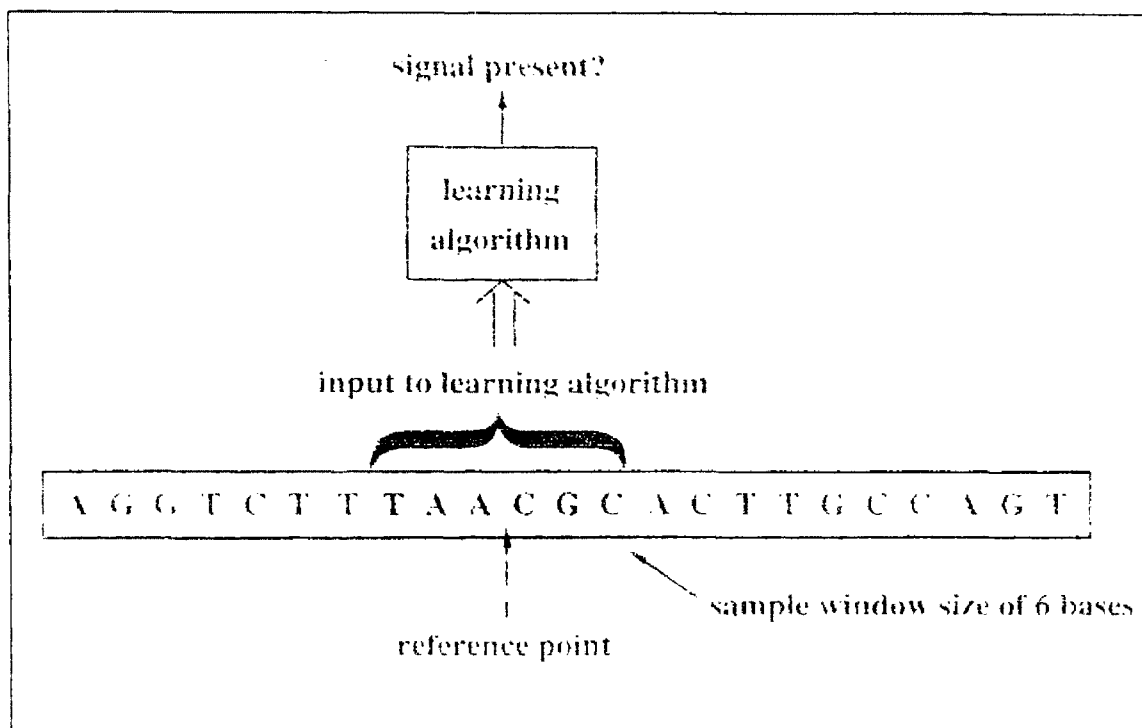


Figure A.2

The following sections describe the two search-by-signal domains that are important in finding genes: (a) promoter sites and (b) ribosome-binding sites. See Craven and Shavlik (1994a) for more details about these tasks. An expert (M. Noordewier) generated both of the data sets and domain theories from the biological literature. Before the domains are presented, the relevant notation is discussed in the next section.

### A.1.1 NOTATION

The domain theories presented in this section use a special notation for specifying location in a DNA sequence. In this notation, each location is numbered with respect to a fixed, biologically meaningful reference point. Negative numbers are locations preceding the reference point, while positive numbers are locations that follow this point. The following is an example:





consequent: -M of (antecedent -list).

For Example, "T: -2 of @-39'AGT"." means the consequent, T, is considered true if at least two of the three antecedents (i.e. location -39 is an A, location -39 is a G, and location -37 is a T\_ are satisfied).

### A.1.2 PROMOTER SITES

The first domain is that of recognizing promoter sites in a sequence of *E. coli* DNA. As stated above, promoters are short DNA sequences where the RNA-polymerase binds to the DNA. This site is located just "upstream" from where transcription begins; thus locating promoters helps locate genes

The data set contains 235 positive examples, and 702 negative examples. The reference point in this case is the transcription-initiation site. The input consists of 57 sequential nucleotides, starting at location -5- and ending at location +7. The negative examples are generated from a (putative\_ promoter-free head of the phage lambda that is 4977 bases long.

The approximately correct domain theory is shown in Table A.1 and contains 31 rules that M. Noordewier extracted from biological literature. Briefly, these rules are characterized by a region rich with A and T from location -19 to -35, the sequence CTTGACA starting at location -37, and finally another region rich with A and T directly preceding the reference location. The five promoter rules differ (a) in the type of nucleotide located near position -30 and (b) in the exact location of where the sequence TATAAT begins. The domain theory is overly specific; it correctly classifies all the negative examples, but only classifies two of the positive examples correctly. Nonetheless, the rules do capture significant information about promoters. This domain is available at the University of Wisconsin Machine Learning (UW-WI.) site via the World Wide Web.

```

promoter <- bend, minus_35, short_spacer, minus_10_15.
promoter <- bend, minus_35, short_spacer, minus_10_16.
promoter <- bend, minus_35, minus_10_17.
promoter <- bend, minus_35, long_spacer, minus_10_18.
promoter <- bend, minus_35, long_spacer, minus_10_19.

bend <- 4 of @-39="WWWWW".

minus_35 <- 6 of @-37="CTGACA".

short_spacer <- 3 of (homonuc1, homonuc2, homonuc3, homonuc4,
homonuc5, homonuc6, homonuc7, homonuc8).

long_spacer <- 3 of (heteronuc1, heteronuc2, heteronuc3,
heteronuc4, heteronuc5, heteronuc6, heteronuc7, heteronuc8).

homonuc1 <- @-30="RR".
homonuc2 <- @-29="RR".
homonuc3 <- @-28="RR".
homonuc4 <- @-27="RR".
homonuc5 <- @-30="YY".
homonuc6 <- @-29="YY".
homonuc7 <- @-28="YY".
homonuc8 <- @-27="YY".

heteronuc1 <- @-30="RY".
heteronuc2 <- @-29="RY".
heteronuc3 <- @-28="RY".
heteronuc4 <- @-27="RY".
heteronuc5 <- @-30="YR".
heteronuc6 <- @-29="YR".
heteronuc7 <- @-28="YR".
heteronuc8 <- @-27="YR".

minus_10_15 <- 5 of @-11="TATAAT".
minus_10_16 <- 5 of @-12="TATAAT".
minus_10_17 <- 5 of @-13="TATAAT".
minus_10_18 <- 5 of @-14="TATAAT".
minus_10_19 <- 5 of @-15="TATAAT".

melt <- 13 of @-15="XXXXXXXXXXXXXXXXXXXXXXXXXXXX".

```

Table A.1

### A.1.3 RIBOSOME-BINDING SITES

The second domain is the task of being able to recognize a ribosome-binding site (RBS). As previously shown in Figure A.1, RBSs are sites where the mRNA is translated into proteins. As stated in Section A1, the ribosome is a complex molecule that reads the mRNA strand to produce the proteins chain of amino acids.

The data set contains 366 positive examples and 1,511 negative examples. Each instance contains a sequence of 49 nucleotides with the point of reference being a ribosome-binding site. The inputs start at location -25, and since there is no location zero, end at location +24. The negative examples are generated from a head of the phage lambda that is 1559 bases long and not known include a ribosome-binding site. With an input window size of 49 bases, 1511 (partially overlapping) negative examples can be generated. The input sequences are defined in terms of the DNA nucleotides rather than the corresponding RNA nucleotides.

```
rbs <- tetranucleotide start-codon.

tetranucleotide <- agga-region.
tetranucleotide <- gagg-region.

start-codon <= @+13="ATG".
start-codon <= @+12="ATG".
start-codon <= @+11="ATG".
start-codon <= @+10="ATG".
start-codon <= @+9="ATG".
start-codon <= @+8="ATG".
agga-region <- @+2="AGGA".
agga-region <- @+1="AGGA".
agga-region <- @-1="AGGA".
agga-region <- @-2="AGGA".

gagg-region <- @+2="GAGG".
gagg-region <- @+1="GAGG".
gagg-region <- @-1="GAGG".
gagg-region <- @-2="GAGG".
```

Table A.2

Table A.2 shows the domain theory, extracted from the biological literature by M Noordeweier. It contains 17 rules which say that a ribosome-binding site contains two parts: (a) either the sequence AGGA or the sequence GAGG near the site, and (b) the start codon ATG beginning 8 to 13 nucleotides before the site. This domain is available at the University of Wisconsin Machine Learning (UW-WI) site via the World Wide Web.

## BIBLIOGRAPHY

- Back, T. *Evolutionary Algorithms in Theory and Practice*. Oxford Press, NY, 1996
- Coreman, T, Leiserson, C. and Rivest, R. *Introduction to Algorithms*. MIT Press, MA 1990
- Craven, M. and Shavlik J. Learning Symbolic Rules Using Artificial Neural Networks. In Proceedings of the Tenth International Conference on Machine Learning (pp.73-80), Amherst, MA. 1993
- Eriksson, H. and Penker, M. *UML Toolkit*. John Wiley and Sons, NY. 1998
- Flanagan, D. *Java In A Nutshell*. O'Reilly and Associates, CA, 1997
- Forrest, S. and Mitchell M. *What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation*, Machine Learning 13(2), 285-319, 1996
- Goldberg, D. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading MA: Addison - Wesley. 1989.
- Goodrich M. and Tamassia, R. *Data Structures and Algorithms in Java*. John Wiley and Sons. NY. 1998
- Hart, W. *Adaptive Global Optimization with Local Search*. Ph.D Thesis, University of San Diego, Department of Computer Science and Electrical Engineering. Technical Report 1281. 1994
- Holland, J. *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan, 1975.
- Kinnear, K. *Advances in Genetic Programming*, MA, MIT Press, 1994.
- Koza, J. *Genetic Programming*. Cambridge, MA: MIT Press, 1992
- Mars, P. Chen, J. and Raghu, N. *Learning Algorithms*, CRC Press, 1996
- Mitchell, M. and Forrest, S. *Genetic Algorithms and Artificial Life*. Sante Fe Institute Working Papers, NM.1993
- Mitchell, M. and Holland, J. *When Will a Genetic Algorithm Outperform Hill-Climbing*, In J.Cowan, G. Tesauro, and J. Alspector (Eds.) *Advances in Neural Information Processing Systems*, Volume 6, San Mateo, CA Morgan Kaufman. 1994

- Opitz, D. *An Anytime Approach to Connectionist Theory Refinement: Refining the Topologies of Knowledge-Based Neural Networks.* Ph. D. thesis, Computer Sciences Department, University of Wisconsin, Madison, WI. 1995.
- Stormo G. Identify Coding Sequences. In Bishop N. H. & Rawlings, C.J., editors. *Nucleic Acid and Protein Sequences Analysis: A Practical Approach.* IRL Press, Oxford England. 1987
- Towell, G. & Shavlik, J. *Knowledge-Based Artificial Neural Networks.* Artificial Intelligence, 70: 119-165. 1994.