APPLICATIONS OF MODERN HEURISTICS AND ADVANCED DATA MINING TECHNIQUES

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Kevin S. McCarty

December 2008

Major Professor: Milos Manic, Ph.D.

AUTHORIZATION TO SUBMIT THESIS

This thesis of Kevin McCarty, submitted for the degree of Master of Science with a major in Computer Science and titled "APPLICATIONS OF MODERN HEURISTICS AND ADVANCED DATA MINING TECHNIQUES" has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor _____ Date_____

Milos Manic

Committee Members _____ Date_____

Akira Tokuhiro

_____ Date_____

Miles McQueen

Department Administrator _____ Date_____

Mark L. Manwaring

Discipline's College Dean _____ Date_____

Howard Peavy

Final Approval and Acceptance by the College of Graduate Studies

_____ Date_____

Margrit von Braun

ABSTRACT

Applications of advanced data mining techniques have proven useful in addressing a wide range of research topics and problems. Data mining results, however, can be difficult to interpret and often mask important relationships with trivial ones. In particular, the Decision Tree, used for classification, prediction and association has a tendency to mask sparse data as it may not reach the information gain threshold required to generate a new node. Rule generation based upon Decision Trees also can be difficult to interpret without a proper contextual framework to base those rules upon. Fuzzy logic, effective in creating semantic precision by using partial contributions from multiple sets, applied to Decision Trees can make them both more precise linguistically and easier to understand. Fuzzy Type-2 extends fuzzy logic even further by providing a contextual framework within which a Decision Tree rule can be polymorphically derived. Use of these new contexts also allows for faster, set-based pruning of the tree, as opposed to traditional node searches.

Applications of data mining include intelligent controllers for autonomous vehicles. By maintaining a database of prior behavior, an autonomous vehicle can learn to follow and better anticipate moves by a lead vehicle. At times, however, when a given space is either too large or simply unknown, a vehicle might have to rely upon local search techniques in order to determine the most appropriate action for a given situation.

By combining traditional techniques with modern heuristics in combination with non-traditional constructs, even more powerful, effective or practical implementations are possible. This thesis presents applications of modern heuristics and algorithms used

to improve upon a traditional Data Mining Technique: the Decision Tree. Because Data Mining is often complemented with Local Search Techniques, this thesis looks at the effectiveness of a number of Local Search Techniques and explores improvements to Stochastic Hill Climbing, and Simulated Annealing in a Factory Scheduling Problem. Finally, applications, such as intelligent controllers often incorporate elements of Data Mining as well as local search. This thesis presents a practical method for the control of an autonomous vehicle. Applications of these techniques are demonstrated in examples showing significant reduction and simplification of .the Decision Tree, significant reduction in Local Search failure rates and an effective tracking algorithm.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the following individuals for their support and assistance throughout the academic process:

Professor Miles McQueen

Debbie McQueen

The author also wishes to thank the IDAHO NSF EPSCoR RII: Idaho Experimental Watershed Network for its generous support.

Finally, the author wishes to acknowledge the tremendous support received from Professor Milos Manic as advisor, editor and sounding board for ideas. It is hard to conceive that this thesis would even have been possible without his input, time and effort throughout the entire process.

The thesis is dedicated to Andrea, my wife and very best friend.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

From data classification to rule generation for intelligent controllers to local search routines for adversarial games, modern heuristics and advanced data mining techniques play an increasingly important role. Application of these techniques has proven useful in addressing a wide range of theoretical as well as practical issues [Cox 05], [Han 06], [Kliewer 00]. In addition, in items such as autonomous vehicles, all of these elements may well come into play. Driving around safely and appropriately requires application of numerous heuristics as the dynamics and rules can change with each new sensor input. Decision Trees, possibly derived from huge data warehouses can provide guidance for a given set of circumstances, while local search algorithms can help to find optimal solutions where the search space is simply too large to comprehensively explore.

However, problems persist in that certain traditional approaches in each of these areas are not necessarily the most practical or even best approaches given certain conditions [Guimar 07], [Mendon 97], [Fuering 99]. Better results are achievable by combining these traditional approaches with heuristics and other modifications. Among the modifications demonstrated here are the use of fuzzy logic and neural networks to enhance relevancy and overall performance.

Chapter 2 presents background material about the techniques described in this thesis: Fuzzy Type 1 Logic and Fuzzy Type 2 Logic, Neural Networks, Local Search Algorithms and Advanced Data Mining Techniques.

Chapter 3 examines a modification of the Decision Tree called CoFuH-DT for Contextual Fuzzy Hierarchies for Decision Trees; an algorithm for capturing intrinsic relationships among the nodes of a DT and based upon a proposed concept of Fuzzy Type 2 “contexts”. In a series of steps, this algorithm fuzzifies a Decision Tree, then overlays Fuzzy Type 2 contexts. The resulting Fuzzy Type 2 classification is then able to capture intrinsic relationships that are otherwise difficult to describe. The resulting Decision Tree is smaller and more precise for rule construction. Example tests demonstrate savings of multiple orders of magnitude in terms of nodes and applicable conditions.

Chapter 4 extends the work done in chapter 3 by presenting a method for generating the hierarchical contexts used by the CoFuH-DT technique. Contextual derivation from Decision Trees, CoT-DT, searches for useful classifications within a traditional Decision Tree, classifications which can then be applied in CoFuH-DT. CoT-DT is demonstrated along with examples showing significant improvement in data representation.

Because data mining techniques are often used in combination with other approaches such as local search algorithms or applications such as intelligent control, Chapter 5 discusses modern heuristics used in local search algorithm modifications and as a means of creating an inexpensive, yet effective tracking algorithm.

Regarding the former, tests are performed on local search algorithms and applied to a production scheduling problem. Two traditional local search techniques are modified using a set of heuristics resulting in a significant increase in the success

rate. A complete application demonstrating all of the techniques along with the algorithms used is described and comparisons of results are shown.

As to the latter, tracking algorithm, a simple and elegant configuration is demonstrated, called FLoST, Fuzzy Line of Sight Tracking, based on inexpensive line-of-sight devices controlled by a heuristic to determine direction and speed of a follower. Unlike alternative approaches where the follower needs to undergo a complex training process, the FLoST-based follower primarily relies upon a human leader to provide direction. This allows for a much simpler and less expensive implementation while still being able to match or exceed the effectiveness of the autonomous design under similar conditions. Three boundary cases of lead vehicle maneuvers, circle, spiral and weave, are presented to show the efficacy of this approach.

Chapter 2

BACKGROUND

2.1 FUZZY LOGIC

2.1.1 Fuzzy Logic Type 1

Traditional systems are designed to make decisions based upon the truth or falsehood of a specific condition or value:

```
If X=A then
        DoSomething
Else
        DoSomethingElse                                    (1)
```

While this approach is fine for many applications, there are situations where having hard, or, in fuzzy terms, “crisp” decision boundaries can lead to difficulties [Hanss 05], [Cox 05]. Consider, for example, the cruise control on a car. Suppose the driver wishes to cruise at 60 miles per hour and sets the cruise control to 60. Now suppose the internal cruise mechanism has three settings: ACCELERATE, which applies gas to speed the car up, BRAKE which applies the brakes to slow it down and NEUTRAL which equates to a no-operation. The control logic might then be as follows:

```
If SPEED < 60 mph then
        ACCELERATE
Elseif SPEED = 60 then
        NEUTRAL
ElseIf SPEED > 60 then
        BRAKE                                                    (2)
```

When the car travels at less than 60 mph, it speeds up and greater than 60 mph it slows down. Problems can occur, however, at or around the 60 mph speed marker. If the car is still accelerating at 59.99 mph, residual acceleration will result in the car exceeding 60 mph, in which case it will begin to apply the brake, perhaps bringing the car under the 60 mph threshold again, which will result in another round of acceleration and braking, making for a very inefficient system. A smart designer might decide to improve the system by creating degrees of acceleration and braking such as HARD_ACCELERATION, MEDIUM_ACCELERATION, SOFT_ACCELERATION and HARD_BRAKE, MEDIUM_BRAKE and SOFT_BRAKE attempting to mitigate problems at the 60 mph speed marker. The designer decides to create 5 mph zones on each side so the control logic might look like this:

```
If SPEED <= 60 - 10 mph then
        HARD_ACCELERATE
ElseIf SPEED <= 60 - 5 mph AND SPEED > 60 - 10 mph then
        MEDIUM_ACCELERATE
ElseIf SPEED < 60 mph AND SPEED > 60 - 5 mph then
        SOFT_ACCELERATE
ElseIf SPEED = 60 then
        NEUTRAL
ElseIf SPEED > 60 AND SPEED <= 60 + 5 then
        SOFT_BRAKE
ElseIf SPEED > 60 + 5 AND SPEED <= 60 + 10 then
        MEDIUM_BRAKE
ElseIf SPEED > 60 + 10 then
        HARD_BRAKE                                          (3)
```

This approach would serve to smooth out performance but there would still be problems at the various boundaries, such as when transitioning from HARD_BRAKE to MEDIUM_BRAKE or SOFT_ACCELERATE to NEUTRAL to SOFT_BRAKE. The ride would be rough and uneven. Adding even more degrees of acceleration and braking would help, but would also introduce a great deal more complexity into the system.

Another approach might be to come up with a smooth, continuous function, such as a linear or Cosine function; but often such simple representations do not come

close to approximating complex, real-world systems such as a cruise control. More complex polynomial functions are possible, but require significant computer resources and are often difficult to derive for more than a small number of dimensions.

There are other problems as well. Describing a HARD_BRAKE as a single value, say deceleration of exactly -10 ft/s2 provides no description for neighboring values such as deceleration of -9.9 ft/s^2. Speedometers often give imprecise readings so any given speed reading is likely to be high or low to some degree. To make a useful, reliable and consistent controller noise, ambiguity and uncertainty must be taken into consideration; but to do that requires a level of complexity that may be unattainable. This has to do with the fact that much of the real-world phenomena an individual, or machine, is likely to encounter are imprecise [Cox 95]. Dealing with imprecise phenomena using precise means can be computationally expensive, if not impossible [Hanss 05]. What is needed is a way to describe imprecise, or "uncertain" characteristics such that complexity is kept to a minimum.

Fuzzy Logic, introduced by Lofti Zadeh in 1965, attempts to deal with these complexities by introducing a level of imprecision or "uncertainty" in place of crisp values [Cox 94]. This uncertainty takes the form of a "fuzzy set", $\widetilde{F_1}$ which consists of the set of all $\mu(x)$ where μ is Fuzzy Type 1 membership function that determines a degree of membership, or truth, from 0 to 1, for a given element x in some range of values X.

$$\widetilde{F_1} = \{x, \mu(x) |\ \forall x \in X, \mu(x) \subseteq [0,1]\} \tag{4}$$

The fuzzy set, $\widetilde{F}_1$, as well as the membership function, μ, depend upon the knowledge of one or more domain "experts" who define boundaries and rules to create a suitable approximation of the desired result [Cox 95]. In Boolean logic, the truth, or membership, value only consists of the values 0, indicating false, or 1, indicating true, while a fuzzy logic value can consist of the values 0, indicating no membership, 1, indicating full membership, or any value in between. As a result, fuzzy members can have membership in not just one, but potentially many fuzzy sets, with the degree determined by μ, hence their inherent "fuzziness".

For example, a person providing a description of a real-world object such as:

THE MAN IS TALL

might consider the description TALL to mean someone who is more than 6 feet in height. The traditional Boolean description would look like this:

6 feet IS TALL (Tall = True)

6 feet IS NOT SHORT (Short = False) (5)

However the same person would likely not consider it entirely accurate to classify someone who is 5 feet 11.5 inches as SHORT, nor someone who is 5 feet 11.9999 inches. While it may seem counterintuitive, because language and real-world phenomena are often imprecise, describing objects in imprecise/fuzzy terms often leads to a corresponding *increase* in precision as well as a reduction in both ambiguity and complexity of systems.

Take, for example, a gentleman who is 5 feet 11.5 inches in height. While he may not be TALL, he is clearly not SHORT either. Instead, he might be considered by most to be TALL to a degree, but not completely, and also SHORT to a degree, though only marginally. In fuzzy terms, 5 feet 11.5 inches might look like this:

$$\mu_{Tall}(5'11.5") = 0.95$$

$$\mu_{Short}(5'11.5") = 0.05 \qquad (6)$$

The fuzzy description is both precise and much more consistent with traditional perception. The expert designing a fuzzy description of a person's height would create a pair of overlapping fuzzy sets to describe a person's height [Cox 94]. The resulting description might say that a person under 4 feet in height is SHORT, while a person over 6 feet is TALL, but between 4 and 6 feet, a person has gradations of both as shown in Figure 2 - 1.

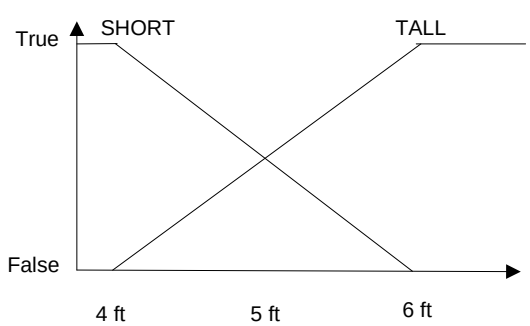

Figure 2 - 1, A basic, fuzzy description of a person's height

The shapes in Figure 2 - 1 are trapezoidal, but fuzzy sets can consist of many kinds of "shapes", such as a triangle and Bell Curve, among others.

The only restriction to fuzzy logic membership lies at the end points; fuzzy logic is an extension of traditional Boolean logic so at the endpoints 0, 1, any fuzzy μ must produce exactly the same value as its Boolean counterpart. In the case described above, 6 feet = TALL:

$$\mu_{Tall}(6') = 1 \Rightarrow \text{6 feet IS TALL}$$

$$\mu_{Short}(6') = 0 \Rightarrow \text{6 feet IS NOT SHORT} \qquad (7)$$

which satisfies the restriction and holds to equation (5).

To further extend Boolean logic, Zadeh introduced fuzzy set operators [Cox 94], taking the traditional AND, OR, NOT operators and creating the fuzzy equivalents μ_{min}, μ_{max}, $u_{\sim,}$ indicating μ*'s* complement, where:

$$\mu_{min}(x, y) = \min(\mu(x), \mu(y))$$

$$\mu_{max}(x, y) = \max(\mu(x), \mu(y))$$

$$\mu_{\sim}(x) = 1 - \mu(x) \qquad (8)$$

These fuzzy operators reduce to Boolean equivalents at the endpoints 0 and 1, as demonstrated by the following truth table comparing the Boolean AND with the fuzzy operator μ_{min}:

Value A	Value B	A AND B	μ_{min}(A, B)
1	1	1	min(1, 1) = 1
1	0	0	min(1, 0) = 0
0	1	0	min(0, 1) = 0
0	0	0	min(0, 0) = 0

Table 2 - 1, Fuzzy and Boolean AND Truth Table

Fuzzy sets also have modifiers, called "hedges" which serve to strengthen or relax a given fuzzy set. The net effect is to either increase or decrease the gradient of a fuzzy set [Cox 94]. Generally these modifiers employ commonly used linguistic terms such as VERY (increase gradient) or SOMEWHAT (decrease gradient) and have an effect similar to that shown in Figure 2 - 2 and Figure 2 - 3.

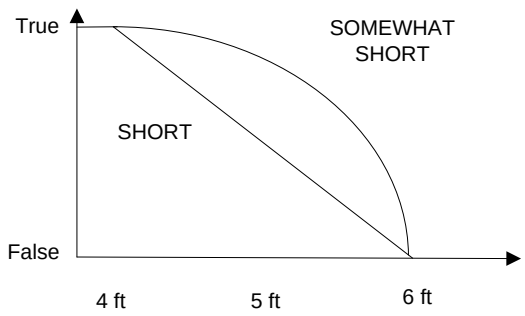

Figure 2 - 2, Fuzzy description of a person's height using hedge SOMEWHAT.

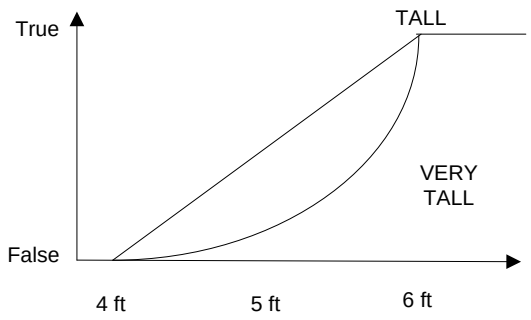

Figure 2 - 3, A basic, fuzzy description of a person's height using hedge VERY.

As would be expected, a person who is to some degree SHORT is to a greater degree SOMEWHAT SHORT, while a person who is to some degree TALL is to a lesser degree; VERY TALL.

Taking another look at the example of the cruise control, the crisp implementation of acceleration described would look like a stair-step pattern of choices.

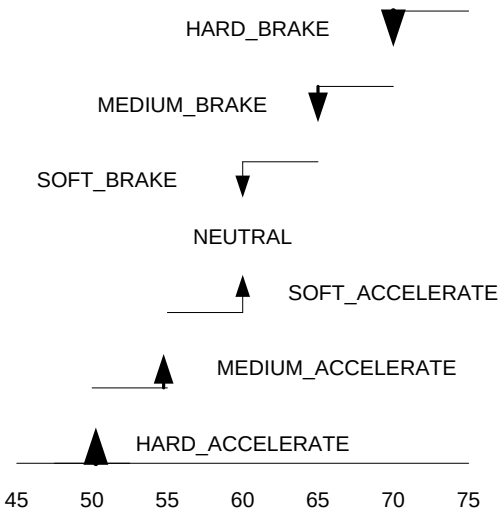

Figure 2 - 4, A stair-step, crisp implementation.

While a fuzzy-based implementation instead would create a series of overlapping fuzzy sets.

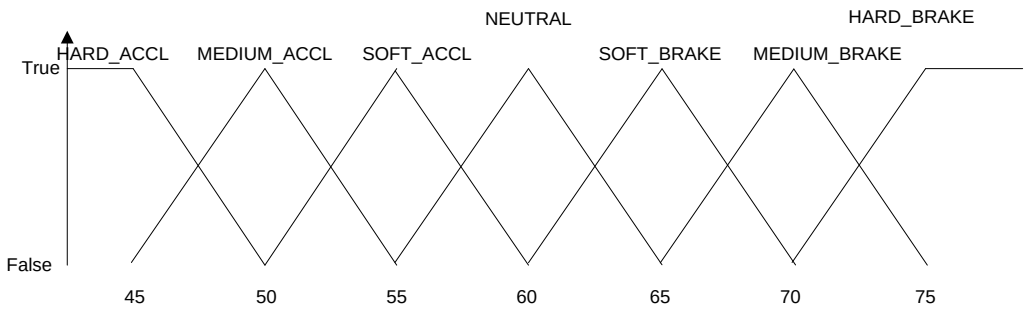

Figure 2 - 5, Constructing fuzzy sets from speed ranges.

Any acceleration/braking would be a fuzzy function taking into account membership in one of the fuzzy sets described below:

$$\mu_{Action}(\text{SPEED}) = \mu_{Hard\_Accl}(\text{SPEED}) + \mu_{Medium\_Accl}(\text{SPEED}) + \mu_{Soft\_Accl}(\text{SPEED}) + \mu_{Neutral}(\text{SPEED}) + \mu_{Soft\_Brake}(\text{SPEED}) + \mu_{Medium\_Brake}(\text{SPEED}) + \mu_{Hard\_Brake}(\text{SPEED}) \quad (9)$$

The resulting implementation, instead of a fragmented stair-step, would look more like a smoother sigmoid, or S-Curve as shown in Figure 2 – 6..

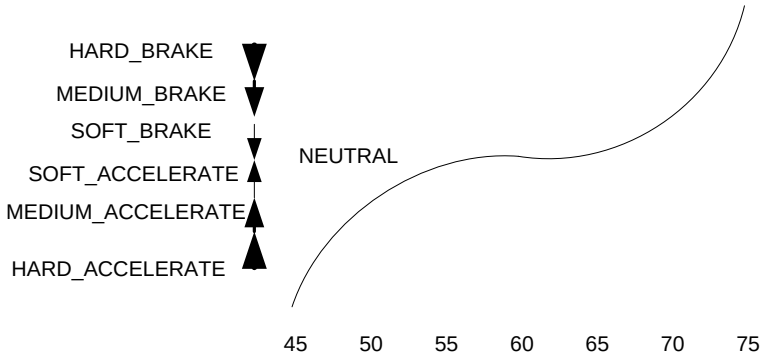

Figure 2 - 6, A smooth fuzzy implementation

In many cases Fuzzy logic provides a number of advantages over traditional, crisp implementation [Hanss 05], [Cox 94]:

1. The ability to model complex systems

   Fuzzy logic can be used to represent very complex systems with many diverse elements.

2. Semantic precision

   Fuzzy logic can describe states and actions in a way that is both more precise and easily understood.

3. Cooperative modeling

   Fuzzy logic can incorporate the opinions of multiple experts into a single unified model

4. Reduced complexity

   Fuzzy logic can be used to approximate complex equations as well as a myriad of diverse interactions.

5. Improved handling of uncertain values and noise

Precision can be difficult to obtain if data is noisy or sensors are inaccurate. By being able to relax a specific value into a range of values, Fuzzy logic can allow for variances without having to compromise accuracy. As such, it can do a better job of handling and using less-than-reliable values, such as an inaccurate speedometer, as well as other types of noise that can affect a model.

6. Improved handling of possibilities

A policeman trying to determine if a person is driving recklessly would asses such factors as speed, weaving, and road conditions. Fuzzy logic treats each of these as fuzzy sets, e.g. SPEED (HIGH, NORMAL, LOW), WEAVE (HIGH, NORMAL, LOW), ROAD (GOOD, NORMAL, POOR) with a fuzzy function $\mu_{Reckless}$ that describes membership in the RECKLESS category. The relationship between SPEED, WEAVE, ROAD and the possibility of recklessness, as indicated by membership in RECKLESS, is direct and easy to understand. A crisp expert system would otherwise have to employ a sophisticated array of conditionals which may provide an adequate answer, but does little to tell us about the intrinsic relationship between the individual components.

2.1.2 Fuzzy Logic Type 2

Fuzzy Type 1 logic has been proven to be very useful for implementation in a wide array of difficult problems. However, there are plenty of issues that Fuzzy Type 1 logic has difficulty handling [Mendel 02]:

1. Experts can disagree on meaning of linguistics terms.

   "Cold", "Warn" and "Hot" can have different meanings to experts living in different regions of the world, such as Barrow, Alaska, Hilo Hawaii, and Phoenix, Arizona. Creating a thermostat that tries to maintain a "Warm" temperature in each region turns into a complex problem as midpoints, endpoints and ranges of fuzzy sets can vary dramatically.

2. Fuzzy sets work best on continuous data.

   Histograms containing non-continuous data can pose a problem for a Fuzzy Type 1 implementation.

3. Data can contain noise beyond the ability of Fuzzy Type 1 to handle easily.

   All of these "uncertainties" can influence the ability of a membership function to come up with an appropriate solution.

Fuzzy Type 2 extends Fuzzy Type 1 by adding another dimension of "uncertainty" to the existing Fuzzy Type 1 construction. Recall in the previous section the Fuzzy Type 1 set $\widetilde{F_1}$, described as a union of a range of values X and a fuzzy membership function μ:

$$\widetilde{F_1} = \{x, \mu(x) | \ \forall x \in X, \mu(x) \subseteq [0,1]\} \tag{10}$$

Fuzzy Type 2 creates a new fuzzy set $\widetilde{F_2}$ which is the union of a new membership function μ_2 applied to members of $\widetilde{F_1}$:

$$\widetilde{F_2} = \{((x, \mu(x)), \mu_2(x, \mu(x)) | \forall x \in X, \forall \mu(x) \subseteq [0,1]\} \tag{11}$$

This new fuzzy dimension "relaxes" the original Fuzzy Type 1 set, generating a transformation into a new Fuzzy Type 1 set for more generic problem solving. As such it is able to compensate for many the shortcomings of Fuzzy Type 1. Consider the problem of the definition of "Warm". Experts may disagree on the precise mid-point or range of "Warm" but they are likely to have a consensus on "Warm" being at or around some statistical measure; for example, the daily mean. While it is not possible to construct a Fuzzy Type 1 set to a suitable "Warm" range for all climates, using Fuzzy Type 2, "Warm" can now come to mean the average temperature plus and minus one standard deviation. Now the Fuzzy Type 1 set for "Warm", under Fuzzy Type 2 becomes location-dependent and can adjust its members, as well as its corresponding membership values, as necessary to fit the appropriate "expert" definition.

Take for example an airplane taking passengers from to various points in the United States. It has a fuzzy thermostat. However, the airplane wants everybody to be most comfortable, or "Warm" depending upon the location they fly to. In Barrow, Alaska, where temperatures tend to remain below freezing for long periods, a fuzzy representation of "Hot", "Warm", and "Cold" might look like Figure 2 - 7.

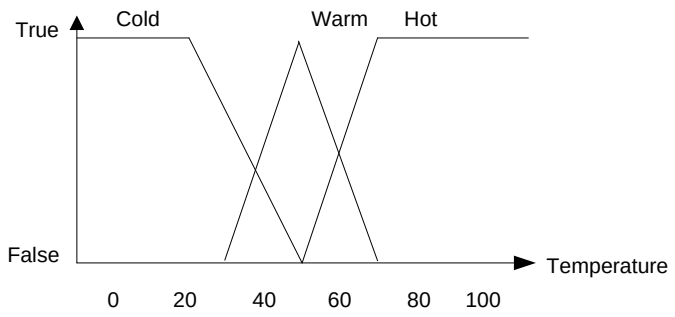

Figure 2 - 7, Fuzzy sets for a thermostat in Barrow, Alaska.

In Phoenix, Arizona, where temperatures tend to remain above freezing for long periods, a fuzzy temperature gage might use a representation like Figure 2 - 8.

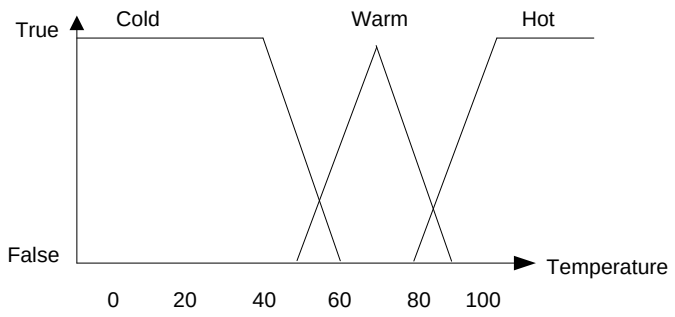

Figure. 2 - 8, Fuzzy sets for a thermostat in Phoenix.

Notice that a move from Barrow to Phoenix requires a corresponding shift in the fuzzy sets designated for “Cold”, “Warm” and “Hot” in order to accommodate the differing Fuzzy Type 1 definitions for each location. Fuzzy Type 2 allows the resulting Fuzzy Type 1 definitions the flexibility to change according to requirements, but remain internally consistent.

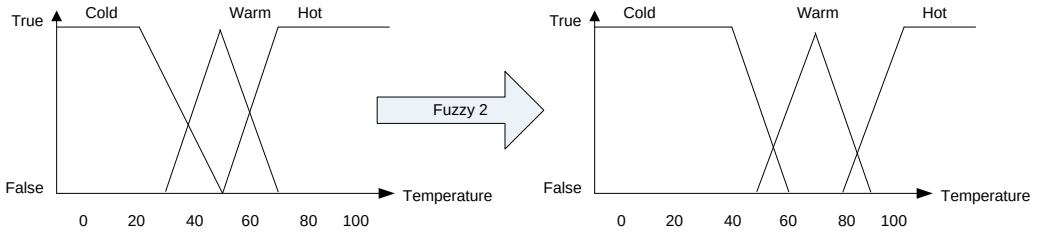

Figure 2 - 9, A Fuzzy Type 2 transition of Fuzzy Type 1 sets.

It is the new dimension of uncertainty provided by Fuzzy Type 2 that accomplishes this.

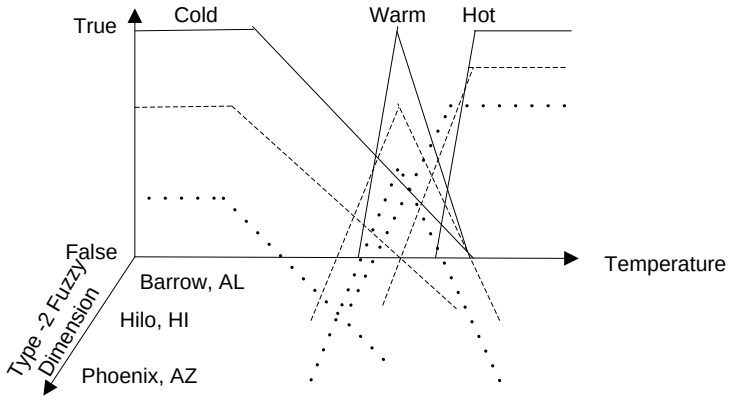

Figure 2 - 10, Fuzzy Type 2 dimension of uncertainty added to Fuzzy 1.

Just as in Fuzzy Type 1, Fuzzy Type 2 must be consistent with Boolean logic at the endpoints and must use the same Zadeh operators as Fuzzy Type 1.

2.2 NEURAL NETWORKS

Neural networks arose from a study of biological neural systems, although only superficial similarities exist [Schalk 97]. The concept of a neural network centers around a single unit, called a neuron, that receives input from one of more sources. In the biological version, a series of fine structures called dendrites collect signals and transmit them to the neuron's cell body. Each neuron has an inhibitor which serves as a threshold for the incoming signal. If the incoming signal is sufficiently strong to overcome the inhibitor, the cell "fires" or initiates a chemical process, creating a signal that gets passed through an axon to other dendrites of other cells [Zurada 92]. In the artificial version, dendrites serve as inputs which are summed and passed to an algorithm called an Activation Function that serves as the threshold as demonstrated in Figure 2 - 11.

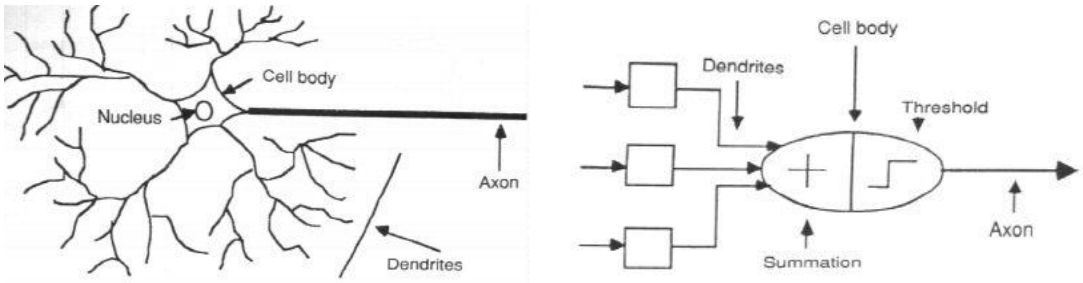

Figure 2 - 11, A biological neuron and its computer-based equivalent.

The input consists of a value along with "weights" and in the artificial neuron is combined with all other inputs and a bias. The total is then processed by the activation function, which outputs one of two values along the Axon, or output. This new model is shown in Figure 2 - 12.

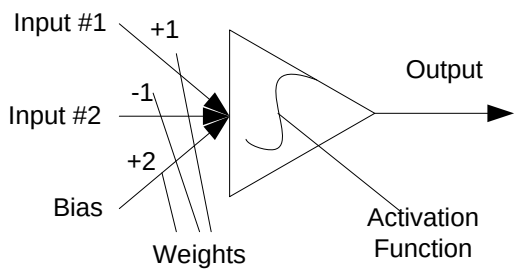

Figure 2 - 12, A Simple Artificial Neuron.

The output values allow a neuron to "classify" inputs into one of two categories. The classification allows a single neuron to distinguish characteristics between points in n-dimension space. By adjusting the weights on the individual inputs and bias, the neuron can "learn" to behave in a given fashion, that is, change the way it classifies a given point.

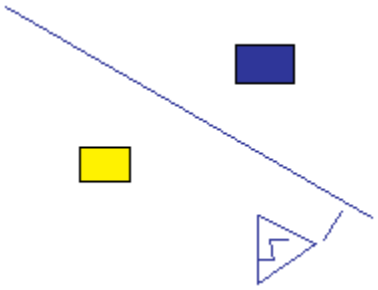

Figure 2 - 13, Single neuron separates two square patterns.

Much like their biological counterparts, artificial neurons can learn by example, but can also "explore" a space in a process called unsupervised learning [Wang 06], [Zurada 92].

Whereas a single neuron can distinguish or separate points into one of two categories, multiple neurons working together can be trained to recognize very sophisticated patterns [Ben-Gan 06]. These neurons working collectively make up the neural network as demonstrated in Figure 2-14.

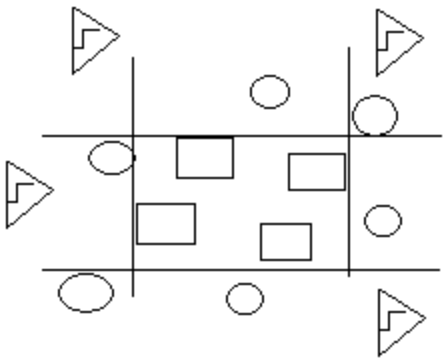

Figure 2 - 14, Neurons working together separate squares from circles.

Neural networks come in a variety of configurations such as feed-forward, feedback, single-layer and multi-layer and display a variety of behaviors. Some, such as the Kohonen Network,s are highly effective at relating clusters of data points. Others such as Error Back Propagation are useful at discovering non-linear classifications. Still others, such as Hopfield and Counter Propagation Networks are very effective at learning and associating images and like patterns.

Neural networks are often very successful at identifying patterns that are often too complex for human inspection or other artificial techniques [Ben-Gan 06]. Once identified, the neural network acts as the “expert” for that particular pattern and is able to discern that pattern from among other sources of data.

Typical uses for a neural network are classification, regression and prediction [Han 06]. Classification is achieved by “training”, either supervised or unsupervised, a network’s weights so that the resulting output value is able to identify patterns which meet classification criteria. Regression is achieved by having a neural network modify itself in order to describe a sequence of known values. Prediction is achieved by taking a known pattern and extrapolating its behavior forward in time.

Neural networks are used in many commercial applications from image, character and voice recognition to medical diagnosis, stock market prediction and data mining [Yu 06], [Han 06].

2.3 LOCAL SEARCH TECHNIQUES

When investigating an unknown state space, a software agent must engage in some form of search. For a space with a small number of states, a comprehensive search is the preferred method for discovering the goal state. A system simply looks at all of the possible permutations and selects the best one. For state spaces that are sufficiently large, there may be either insufficient time or computing resources, making a comprehensive search impractical. In such cases a solution may still be found using limited resources. This is accomplished by starting at a random point and using a neighborhood search technique called Local Search [Russell 03], [Martin 07].

Instead of trying to examine all possible states, a local search algorithm limits its search to neighboring states. Examination of the "neighborhood" by a local search algorithm will often yield a gradient which can be followed to other neighboring states with the prospect that this may eventually lead to a goal state or local maximum. This type of search can greatly reduce the cost of computer resources and search time but may also fail in its mission to reach a goal state. Success or failure depends upon the state space and the type of local search algorithm used.

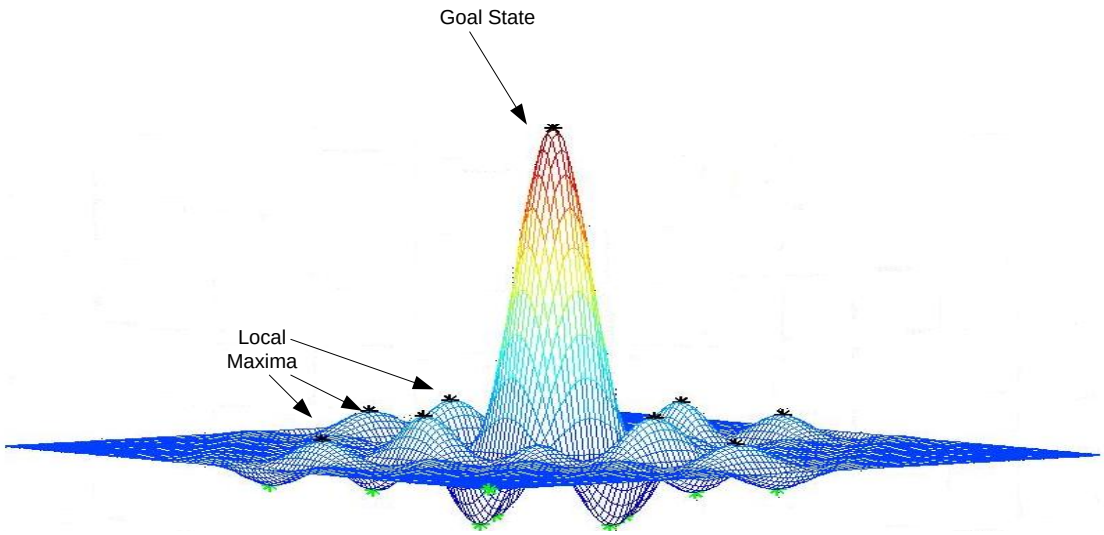

Figure 2 - 15, Typical space with a global (goal state) and local maxima.

Some "greedy" local search algorithms, such as Hill Climbing, simply follow the gradient to its end. Others, such as Simulated Annealing, introduce some random movement to better their chances of finding a goal state without becoming trapped in a local maximum. Still others, such as Tabu search combine a random search with a memory to map areas searched and avoid them if they prove unfruitful [Martin 07], [Zheng 06].

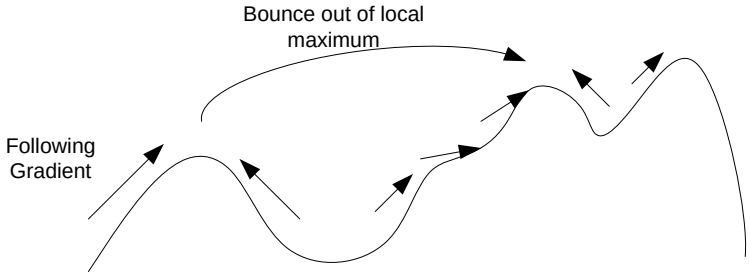

Figure 2 - 16, Local Search following gradient and "bounce" out of local maximum.

Local search algorithms can be combined with additional heuristics to improve their effectiveness for a given search problem.

2.4 ADVANCED DATA MINING TECHNIQUES

With each passing day, the amount of data collected continues to increase. As data sources grow, it becomes more and more difficult to derive meaningful information via traditional human query and search mechanisms. Traditional reporting often gets overwhelmed in minutiae while key relationships remain hidden [Kita 02], [Fu 07].

Database technology has evolved to meet these challenges with an ever greater and more advanced array of storage mechanism and query tools. Relational databases dominate the business landscape surrounded by data warehouses, high-speed connections and high-density hard-drives [Han 06], [Qimming 99]. Anyone looking for a specific answer to a specific question, such as who made the most purchases of a widget last month, only has to submit the appropriate query to retrieve it.

Problems often arise, however, in trying to determine what questions are appropriate. Datasets can become so large that even finding out where to begin presents significant challenges. Basic questions that answer “who” or “how many” do not easily lead to the more pressing and useful question of “why”. Common queries often fail to define important relationships or criteria, such as how to distinguish a “good” customer from a “poor” one or at what times are customers more receptive to certain promotions. Nor can traditional queries easily draw associations, such as products that tend to be sold together, either in one purchase or subsequent purchases [Han 06], [Ben-Gan 06], [Witten 05].

In addition, important information for certain questions may not be easily formulated. With databases containing thousands of dimensions, it can be a daunting

challenge to determine which, if any, hold useful information. Manually researching these large datasets is very costly, given the vast quantities of information, and human comprehension is often too limited to recognize many of the more fruitful patterns that exist from among thousands of possible attributes.

Advanced Data Mining Techniques are a class of algorithms designed to search large databases and provide answers to vague and difficult questions by identifying relationships and patterns in the underlying data [[Han 06], [Ben-Gan 06], [Cox 05]. They generally operate as a semi-directed or completely automated process, analyzing large datasets looking for useful information. The results can take the form of data clusters, Decision Trees, histograms, graphs, lift charts and other presentations that distill complex relationships into a readable form. The purpose of data mining is to take a large series of data points and derive information from which relevant, important, and heretofore unknown knowledge can be gained. This new knowledge can then be used for competitive advantage through the creation of rules, or simply as a way to understand and predict the behavior of a complex system [Haru 05], [Adom 01].

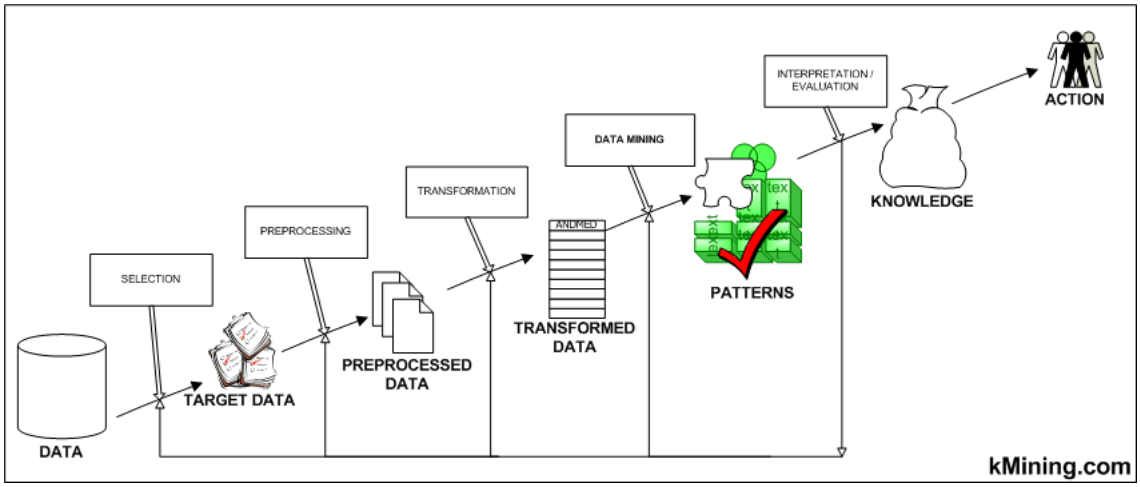

Figure 2 - 17, Data Mining Process.

Data mining starts with data, usually in the form of a large relational database [Han 06]. This database may then be transformed into a data warehouse or data cube through a process called ETL (Extraction, Transformation, Loading) which attempts to create "clean" data free from errors and missing values and formatted in a way to make it easier to mine [Ben-Gan 06]. Because of the tight relationship between databases and data mining, large vendors of database, such as Microsoft, SAP, Oracle, and IBM all offer data mining tools to go with their database products. Once the data is put into a more friendly form, data mining tools begin the process of creating mining models and extracting useful information.

Data Mining Techniques fall into a number of categories:

1. Classification, where data points are related by classification criteria.

   Among classification techniques include Decision Trees, Neural Networks, Baysian Networks, Rule-based systems, Support Vector Machines, K-Means and Fuzzy C-Means.

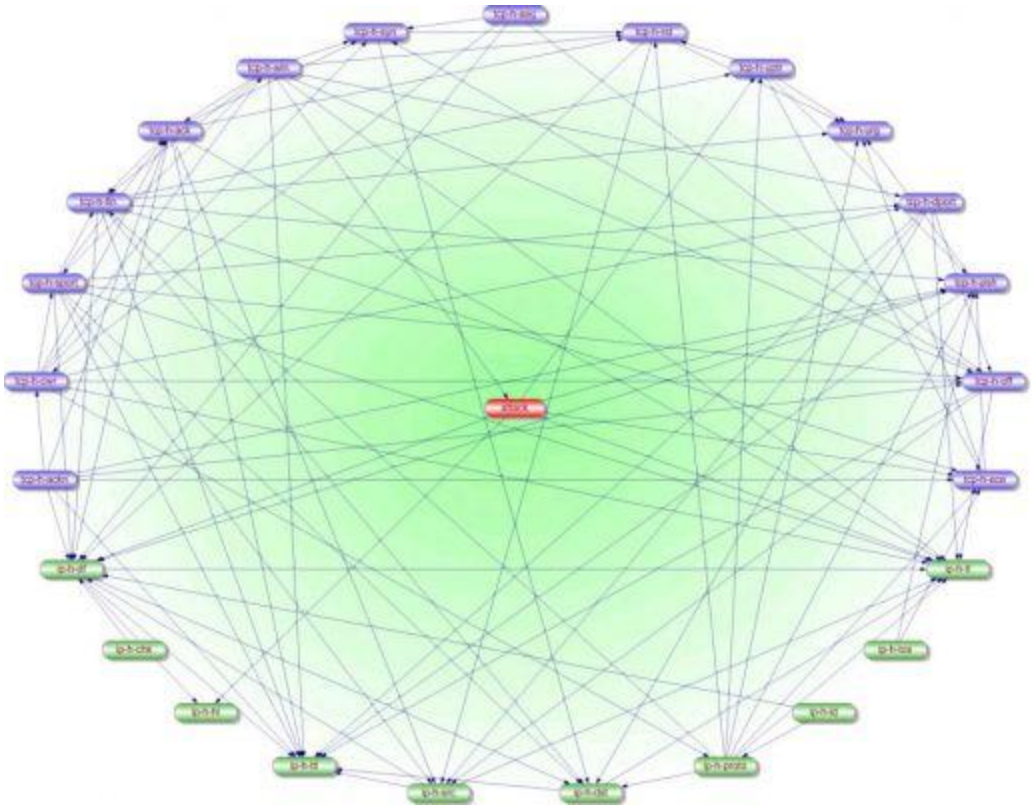

Figure 2 - 18, Classification of relations among multiple elements.

2. Association, where relationships are drawn between objects.

   Among association techniques include Association Rules, Decision Trees, Baysian Networks, and K-Nearest Neighbor.

Figure 2 - 19, Amazon.com association links other book titles to a book purchase.

3. Prediction, where past behavior is extrapolated to anticipated future actions.

Classification techniques can also serve as predictors.

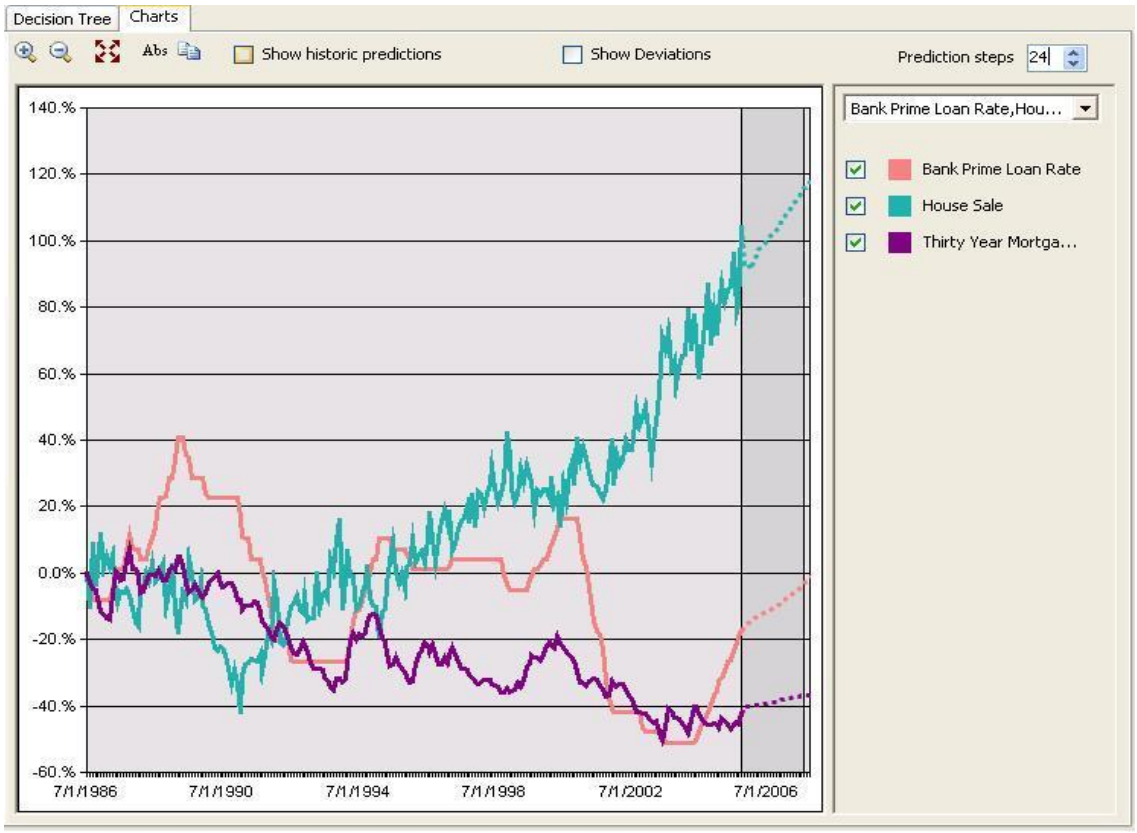

Figure 2 - 20, Predictions (far right dotted lines) from existing data patterns

4. Regression, where "common" characteristics are established to explain past behavior.

Among regression techniques are Neural Networks, Linear and Non-Linear regression techniques and Fuzzy-Set based approaches.

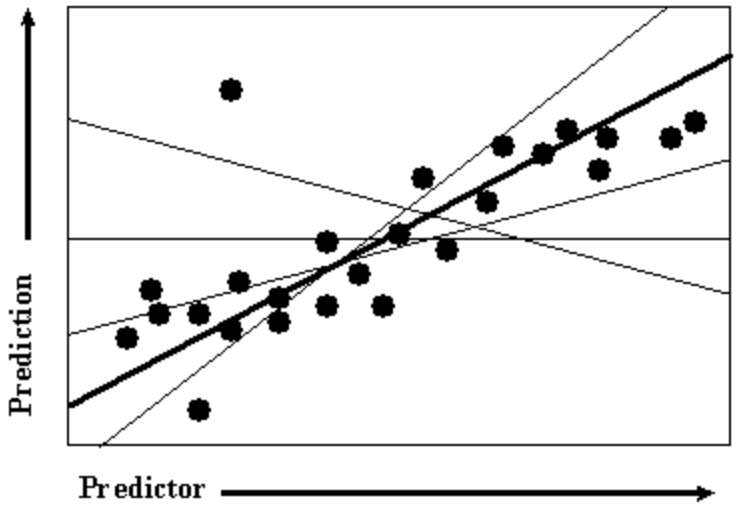

Figure 2 - 21, Linear Regression attempts over a series of data points.

5. Time Series, where activity is group and trends established according similar periods of time in a given time sequence.

Time series techniques involve a the creation of periodic "time slices" which often factor into account seasonal or other significant time-related events, such as weekends or a holiday period like the Christmas Season, in order to create trend-based views that compare similar time periods or time spans or contrast with other data values. Figure 2 – 22 shows a Time Series Analysis of the relationship between U.S. Nominal GDP and the yield of the 5-Year U.S. Treasury Note. A close inspection of the year-over-year Treasury rate and of the rate of change of Nominal GDP changes shows a lagging correlation between the two with Nomincal GDP as the lead. Time Series makes this otherwise vague relationship much more clear.

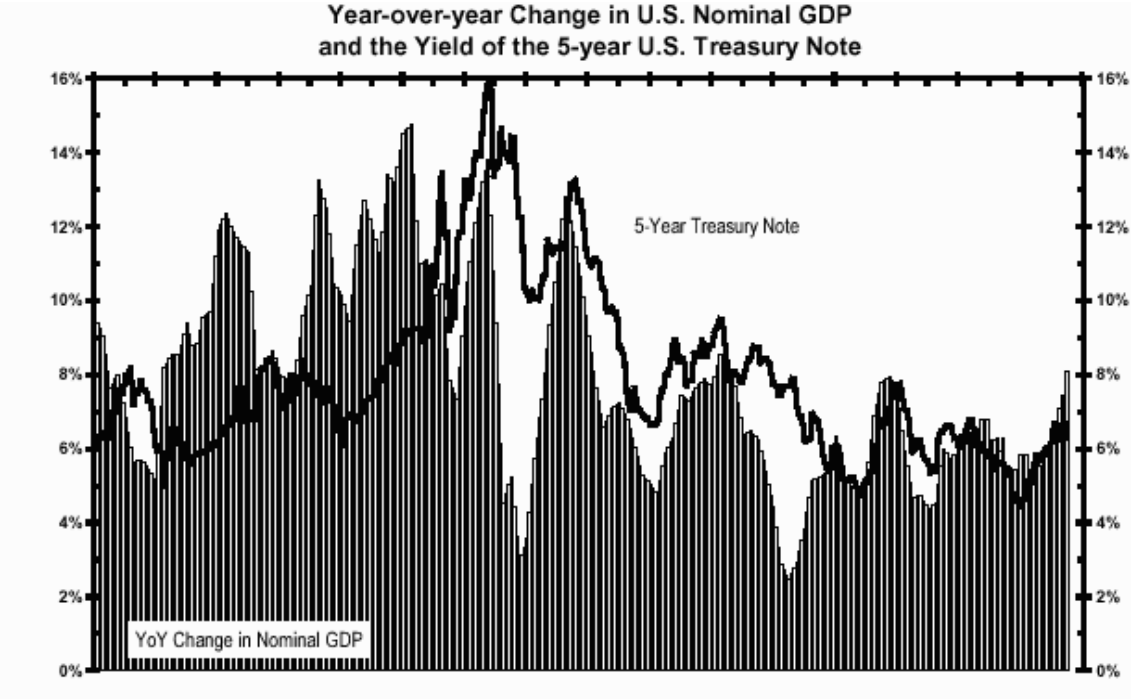

Figure 2 - 22, Time series analysis of US Nominal GDP vs. 5-year Treasury Note.

The list above is by no means complete. Each implementation attempts to "mine" the data in a particular way in order to discover the knowledge hidden there. Success often depends upon the characteristics of the data and the types of information to be mined. For that reason, successful data miners often employ multiple techniques to both confirm previous findings as well as obtain a more comprehensive "picture" of the underlying data.

Chapter 3

CONTEXTUAL FUZZY TYPE-2 HIERARCHIES FOR DECISION TREES (COFUH-DT) – AN ACCELERATED DATA MINING TECHNIQUE

This chapter presents a technique for modifying a Decision Tree using Fuzzy Type 1 & 2 operations. The resulting contextual tree is substantially smaller and semantically more meaningful. The results were presented at the IEEE HIS08, Krakow, Poland, May 2008

3.1 INTRODUCTION

Suppose the vehicle described in Chapter 1 has to deal with a complex environment. It may have a Decision Tree providing rules on how to react to certain obstacles, what distance to maintain, or how quickly to accelerate. These rules may need to deal with all kinds of factors such as weather, terrain, and other vehicles, all of which contribute to the overall decision process.

What if, on the other hand, there was an environment feature that overwhelmed the decision-making process so that most or all other factors had little to no relevance. Suppose, for instance, the vehicle was on a steep slope. Turns to the right or left might result in a rollover so any instruction or rule set involving right or left turns suddenly loses all meaning. Having to search and prune a Decision Tree with millions of nodes or to create a special branch specifically for this possibility requires significant resources. This chapter presents a solution to that and other problems faced by Decision Trees, particularly when used for data mining.

Organizations make extensive use of data mining techniques in order to define meaningful and predictable relationships between objects [Liu 07]. Retailers use these techniques to create recommender systems that seek to bring products and customers

together [Qiming 99], [Kita 02], [Fu 07]. Game designers employ them in order to create worthwhile and realistic adversaries. Zoologists use them to create environments in which animals can thrive. One of the most widely employed methods for data mining is the Decision Tree. The Decision Tree is created using algorithms, such as ID3, that take a set of data points and build a tree based upon the content therein [Zhao 06], [Li 03], [Li 02].

Typically a Decision Tree is viewed as a set of conditions and probabilities that, when combined, represent a node. Examining the tree usually means traversing it in a depth-first or breadth-first search, looking for nodes to prune in order to optimize the search. Instead, consider the Decision Tree as a set of elements and filters, or conditions. Each node represents a subset of its parent, created by applying one or more conditions to the parent set. The sequence of conditions represents the "path" to a given node.

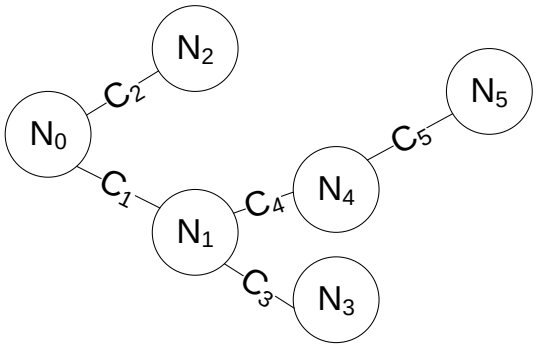

Figure 3 - 1, A basic, horizontal Decision Tree.

Hence, in Figure. 3 - 1, the Decision Tree node, N_1, represents a sample of data for which the condition C_1 is applied to N_0: $N_1 = C_1(N_0)$. N_3 then becomes the condition C_3 applied to its parent node, N_1: $N_3 = C_3(N_1) = C_3(C_1(N_0))$.

Generically, any given node, N_j, is the resulting set derived when applying its "path" condition C_{Nj} to its parent:

$$N_j = C_k(N_{jParent}) \qquad j=1,..,j_{max},\ k=1,..,k_{max} \tag{12}$$

where j_{max} is the number of nodes and k_{max} is the number of conditions. For any given node, one can determine the conditions, or "path" which lead to it and derive rules to apply this node "knowledge". This knowledge takes the form of probabilistic events specified within the node, such as a purchase or appearance of a threat. Rules can then associate an event with some set of conditions and dictate an appropriate action [Adom 01], [Kwan 02], [Dai 07].

DEFINE RULE *rule_name*
ON *event*
IF *condition*
DO *action* (13)

The conditions describe the relationships between node elements whether obvious, such as customers in a store, or more obscure, such as peanut butter and a bottle of cleaner; attempting to draw a meaningful relationship between them. For example:

IF Customer BUYS Computer THEN
Customer BUYS Printer 25% (14)

The above condition tells a store manager that a customer who buys a computer will also buy a printer 25% of the time. This indicates that there is a high likelihood that any given customer who buys a computer will also be interested in purchasing a printer. The manager may choose to act upon this information by bundling printers and computer together in a special to encourage more printer purchases. Using a Decision Tree, the manager now knows the probabilities for any given set of conditions and sales. With that information, he or she can create rules that stand a better chance of improving sales.

The CoFuH algorithm extends traditional Fuzzy Type 1 sets through the use of Fuzzy Type 2 hierarchies called "contexts". In doing so, it both simplifies the underlying data set as well as makes it more semantically precise under the higher-level, polymorphic, implication of its context. This is accomplished using fast, fuzzy-set based operators and rules that remove uninteresting data points that are "out of context" while enhancing what remains. The end result is a smaller, yet more precise and meaningful data set. This chapter demonstrates the application of this technique to the Decision Tree, taking a large tree, fuzzifying it and applying contexts so that the resulting tree is smaller by orders of magnitude, yet more meaningful. The Contextual Fuzzy Hierarchies algorithm for the Decision Tree (CoFuH-DT) is then used to quickly prune some sample Decision Trees and create a meaningful relationship between two very different objects, such as in the example case of a jar of peanut butter and a bottle of window cleaner.

This Chapter is organized as follows: Section 2 presents background, describing the previous work and issues; Section 3 presents the problem in detail; Section 4 presents the algorithm; Section 5 applies CoFuH-DT to a pair of example Decision Trees; Section 6 presents conclusions and future work.

3.2 PROBLEM STATEMENT

For data with many characteristics or non-intuitive ones, it can be difficult to build a manageable and meaningful tree because of the following:

1. Difficulty of analysis.

   For the manager of an online store, as an example, understanding the relationships between and among thousands of customers, each with their own tastes and preferences, and products, means having to analyze a Decision Tree with potentially millions of nodes. Simply creating and managing rules for such a large number of nodes requires substantial computer resources. OnLine Analytical Processing (OLAP) systems [Qiming 99] help to manage huge datasets but do little to address other issues.

2. Semantic differences.

   Experts often disagree in rule definition [Mendel 02], [Hanss 05]. For example, what differentiates a "good" customer from any other? Is a "bargain shopper" someone who *always* buys items that are on sale or someone who *only* buys items that are on sale?

3. Relationships may be dynamic.

   Some relationships between products change within a given context, e.g. turkey and cranberry sauce are closely associated in the United States during the Thanksgiving holiday but may not be closely related otherwise.

4. Relationships can vary over time.

In the summer, for example, a sleeping bag might be associated with a swimsuit, bug spray and a fishing pole; while in the winter that same sleeping bag may be more closely associated with a parka, snow shoes and gloves.

5. Decision Trees can be difficult to interpret.

   Many paths are of no use at all; for instance a node that says ALL BABIES ARE BORN TO PREGNANT WOMEN does not provide much useful information. Other paths may be too obscure to define readily. An example of this is that of a woman buying certain food items and cleaning supplies. In her mind, these items are closely related in the context of "monthly shopping". The Decision Tree *may* reflect this; however, to a retailer such an association may not be so obvious, thus looking more like an outlier.

In a real world situation involving many products and customers with differing tastes, the number of nodes in a Decision Tree with n dimensions is determined by the cross product of the number of elements e of each dimension d_i used to branch:

$$\textit{Total number of nodes in Decision Tree} = \prod_{i=1}^{n} d_i \qquad (15)$$

The store manager is probably going to be faced with very large Decision Tree.

Now suppose there is a node on the tree containing the woman's purchase of food and cleaning supplies. The system produces a rule to address the case of the peanut butter to window cleaner relationship:

```
DEFINE RULE PB_Cleaner
        ON Customer PURCHASE
IF PURCHASE is PeanutButter
        DO Recommend Window Cleaner                    (16)
```

This rule does little to describe to the manager the overall context of the purchase and how best to take advantage of this information because there is no natural or obvious relationship between the objects to assess. Simply adding these rules to an already existing rule set means having to manage a substantially larger number of rules. More rules lead to ever more complex relationships as well as greater difficulty deriving meaningful information from them.

Fuzzy Type 1 Decision Trees were created in an attempt to address some of these issues [Lee 03], [Wang 01] but run into difficulty dealing in areas where even the semantics themselves are called into question [Mendel 02]. In Fuzzy Type 1 form, Decision Trees simplify sets of nodes but do little to address the overall complexity of the tree itself.

Hybrid approaches [Liu 07], behavioral abstractions [Kita 02], [Haru 05], Online Analytical Mining (OLAM) [Qiming 99], [Adom 01], [Kwan 02] and multi-level association rules [Li 02], [Vlag 07] have also been devised to deal with these issues. While successful, these approaches consume significant computing resources and can end up creating numerous, multi-layer and often difficult to understand conditions. A modification of rule PB_Cleaner (16) to add a multi-level association and a monthly shopping hierarchy might end up looking like the following:

```
DEFINE RULE PB_Cleaner_Multi
    ON Customer PURCHASE
    IF PURCHASE is Peanut Butter
        AND SHOPPING_TYPE IS MONTHLY
        AND DAY IS First Saturday of Month THEN
    DO Recommend Window Cleaner
    OR
    IF PURCHASE is Window Cleaner
        AND SHOPPING_TYPE IS MONTHLY
        AND DAY IS First Saturday of Month THEN
    DO Recommend Peanut Butter                    (17)
```

An interpretation of this very simple rule is that peanut butter and window cleaner are somehow related, but the type of relationship is not easily discernible.

Unfortunately, typical real world situations are usually more complex. Relationships trying to account for many dimensions, dimension elements and corresponding Decision Tree nodes become more difficult to describe. As a result, rules themselves become more difficult to generate and understand. Data growth leads to significant growth in the corresponding Decision Tree but without the corresponding growth in usefulness.

Suppose the virtual store manager wishes to give his customers the best shopping experience possible. He has lots of statistics about past purchases and uses Decision Trees to breakdown the types of purchases his customers made. There are lots

of things he must take into account, such as how often they shop, what sort of things they buy when they come in, what sorts of other products might they be interested in and so on. His initial Decision Tree might consist of the following, Customer Type (CT), Product Type (PT), Relative Product Price (RPP), Day of Week (DW), Time of Year (TY), Customer Age (CA), Geographic Location (GL) as shown in Table 3-1.

Dimension	Sample Values
CT	normal, bargain, premium, bulk, impulsive
PT	food, cleaning, household…
RPP	bargain, normal, sale, premium
DW	Sunday, Monday, … Saturday
TY	Jan 1, Jan 2…Dec 31
CA	1, 2, 3, …100
GL	address, city, postal code

Table 3 - 1, Dimensions for a virtual store manager

Even with a small average number of elements, e.g. 10 per dimension, the total number of nodes generated from this configuration could run into the millions. In addition, a large percentage of these nodes, such as those focusing on time of year, contain very little useful information most of the time; but at other times become very important. Removing those uninteresting nodes may still leave a very large tree with a correspondingly large number of rules to manage. By fuzzifying the tree and overlaying strategic contexts according to the algorithm presented, the manager can reduce and transform the complexity of the generated rules to a more easily understood and manageable state.

3.3 CoFuH-DT – Contextual Fuzzy Type-2 Hierarchies For Decision Trees Algorithm

The CoFuH-DT algorithm presented in this section consists of the following two phases: deconstruction of a Decision Tree into datasets and filters, then fuzzification of both datasets and filters resulting in a series of fuzzy sets. Fuzzy Type-2 membership functions, representing one or more newly introduced "contexts" are applied to the sets; separating via fuzzy arithmetic those elements that are in context from those out of context. From the remaining fuzzy sets a smaller, in context Decision Tree is constructed as demonstrated by Figure 3 - 2.

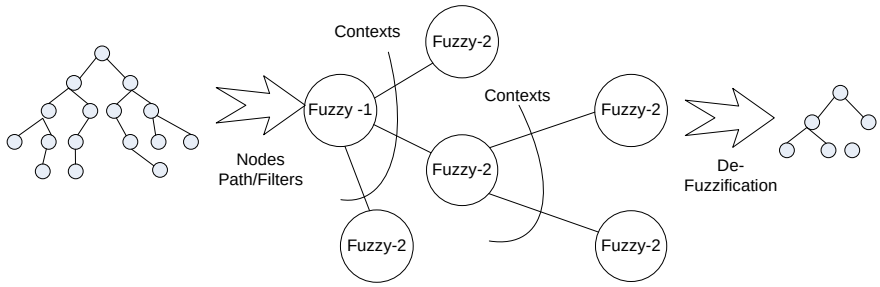

Figure 3 - 2, CoFuH-DT reduction of Decision Tree

The steps of the CoFuH-DT algorithm is as follows:

Step 1. Condition creation

Let $N_1..N_n$ be the set of nodes generated through data mining techniques such as ID3 [2-15], creating a Decision Tree for the original data set *D*.

$$N = \{N_1, N_2, \ldots, N_n\} \tag{18}$$

Now let $R_1..R_n$ be the set of rules generated by applying individual paths to each node to its data as demonstrated by Figure 3 - 3.

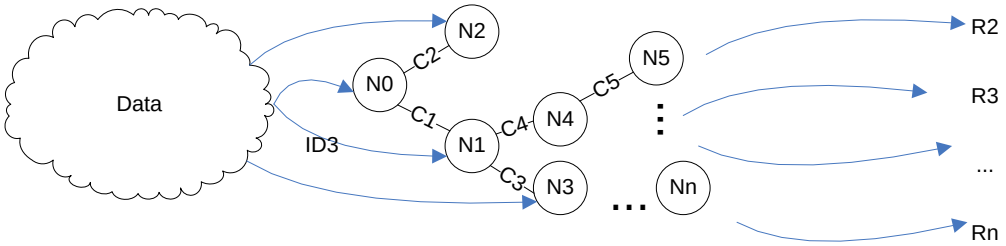

Figure 3 - 3, Rule Creation using Decision Tree

Step 2. Condition Normalization

Create a function *f* to normalize a set of conditions and corresponding rules C_R by mapping each C*i* to the range [0,1], then translating those values to a normalized set C_{norm}:

$$C_{norm} = \{f(C_i),\ C_i \in C_R,\ \ f(C_i) \in [0,1]\} \tag{19}$$

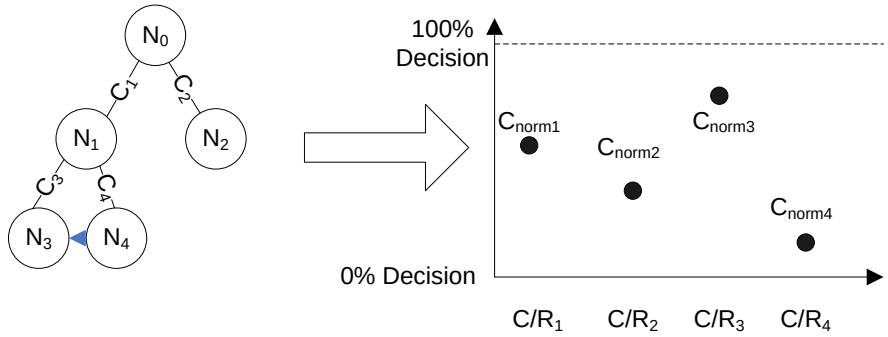

Figure 3 - 4, Normalization of a Decision Tree

Step 3. Condition fuzzification

Fuzzification of the normalized values occurs by extending those values using Fuzzy Type 1 membership functions and fuzzy hedges in order to ensure appropriate

representation, if necessary, across the entire set and thereby generate the Fuzzy Type 1 set μ_{Cnorm}.

Discrete points e.g. a decision whether to recommend purchases of certain foods such as bread, ham, etc. now become a series of fuzzy triangles as demonstrated in Figure 3 - 5 with the original crisp conditions represented as a series of ranges at the base of each triangle.

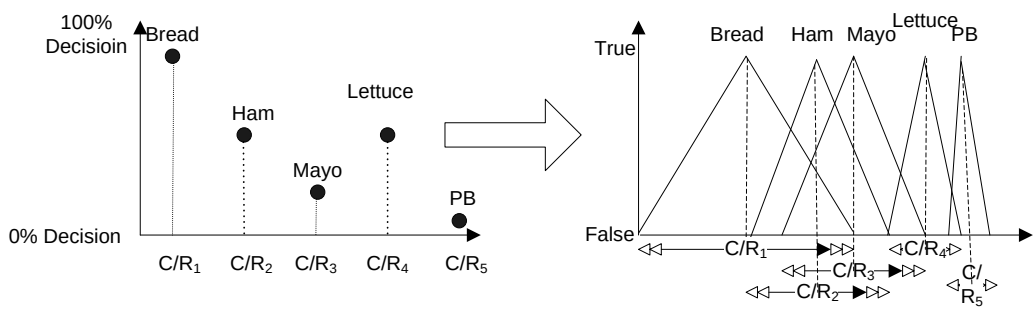

Figure 3 - 5, Fuzzifying customer's Decision Tree

In cases where there are multiple Boolean conditions for a node we can apply Zadeh's operators AND and OR for fuzzy unions and intersections for conditions $C_1 \ldots C_n$

$$\cap\, \mu_{C_i} = \max(\mu_{C_1}, \mu_{C_2}, \ldots, \mu_{C_n})$$

$$\cup\, \mu_{C_i} = \min(\mu_{C_1}, \mu_{C_2}, \ldots, \mu_{C_n}) \qquad (20)$$

Further, more extreme examples can make use of mean and weighted mean or other general algebraic operators [Cox 05].

Step 4. Context creation

Create fuzzy sets using a method such as that demonstrated in Chapter 4 describing "contexts" which group items that may or may not have a natural association but do relate within a given broader context. Contexts also can bring together elements of different clusters while at the same time preserving cluster identity as shown in Figure 3 - 6. For the Decision Tree, this has the effect of "pruning" all those nodes which fall out of context as demonstrated in Figure 3 – 7.

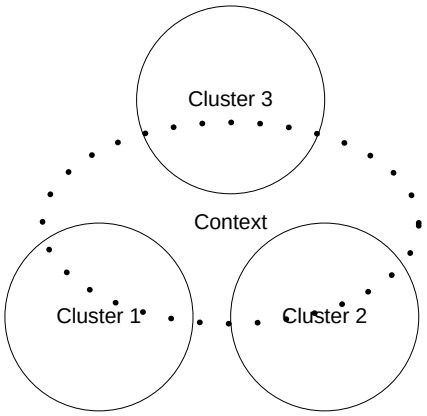

Figure 3 - 6, Context unifying 3 clusters

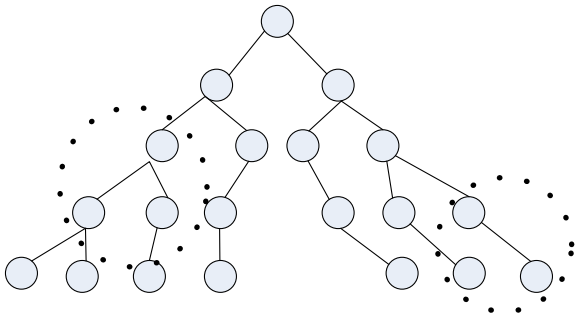

Figure 3 - 7, Nodes pruned by context

Using fuzzy, new dimensions of uncertainty are added, allowing new specifications to exist and altering existing ones. In the example of the woman doing her monthly shopping, the context and new dimension of uncertainty "monthly

shopping" alters the notion of both "food" and "cleaning supplies" by increasing membership in "food" for those items which are bought only occasionally while reducing it for others. At the same time, the context draws a link between food and cleaning supplies imposing a hierarchy of "monthly shopping" on top of both. Hence the resulting Fuzzy Type 2 set, "monthly shopping" produces a new set consisting of "monthly food" and "monthly cleaning supplies" whose original primary sets locally are still regarded as "food" and "cleaning supplies". The membership of any item in any base set, e.g. food, now assumes a more polymorphic representation dependent upon one or more contexts in which it happens to find itself.

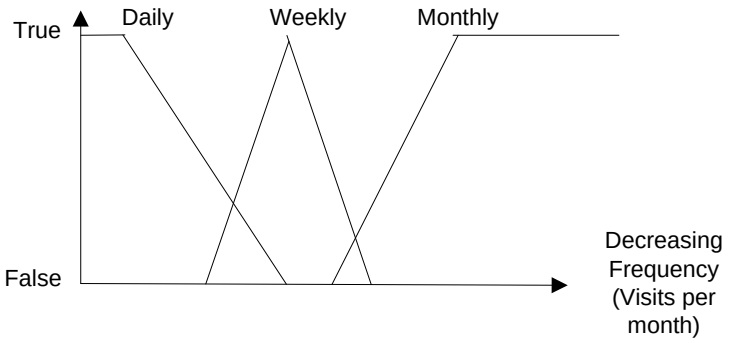

Figure 3 - 8, Context of shopping type.

Here the Fuzzy Type 2 context contains the values "daily", "weekly" and "monthly".

Adding additional dimensions is a matter of creating and applying other contexts. For example, suppose the manager wanted to take into account various holiday periods. Now new contexts such as "Thanksgiving" or "St. Patrick's Day" are overlaid onto the Decision Tree to create a potentially different representation for the nodes underneath.

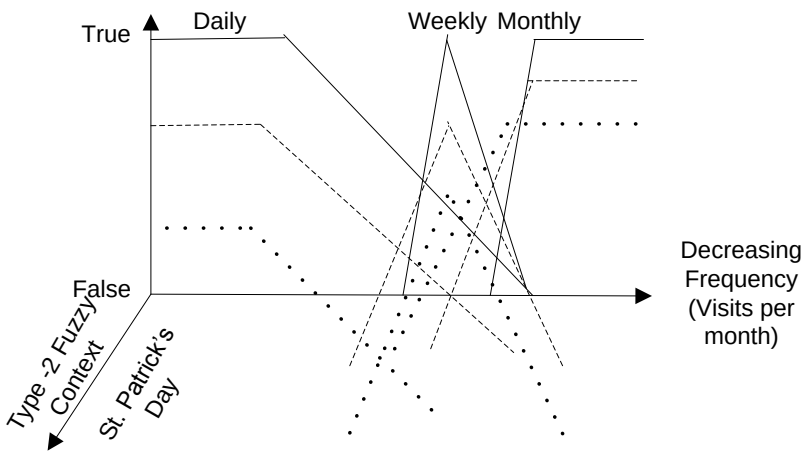

Figure 3 - 9, Fuzzy deformation under a context.

Step 5. Fuzzy Type 2 application of contexts to fuzzified conditions

Fuzzy Type 2 contexts extend the newly created Fuzzy Type 1 set by adding an additional dimension of uncertainty. The context creates a Fuzzy Type 2 set $\hat{C}$ [Mendel 02], whose members are the combination of the context functions over the original Fuzzy Type 1 membership functions over the original conditions shown in Equation (19). Applying the Zadeh product operator across the domain of $\hat{C}$ eliminates those sets and the underlying conditions which are "out of context". Setting appropriate minimum memberships thresholds can serve to further reduce the final result space R_C:

$$R_C = \cap \hat{C} \tag{21}$$

This has the desired effect of pruning those nodes completely out of context as well as marginalizing those elements which are only of minimal interest.

For the retailer with the customer doing monthly shopping, de-fuzzification of the remaining conditions yields a much smaller Decision Tree. In addition, by using

the context applied over the remaining conditions, the conditions take on new meaning within that context. The rule developed previously in Equation (16) can now be generalized to:

```
DEFINE RULE ShoppingType
        ON Customer PURCHASE
IF PURCHASE IS MonthlyContextItem THEN
        DO Recommend Other MonthlyContextItems                    (22)
```

This new rule is both simpler to implement as well as more descriptive and intuitive. It also takes into account the contextual components of the shopping trip, that of a regular monthly shopping day. In the case of the peanut butter and window cleaner, while distinct and very different types initially, they are united under the context of “monthly shopping”.

3.4 TEST EXAMPLES

The following test examples were used to demonstrate the effectiveness of the algorithm when applied to real world situations. Developing appropriate contexts and then applying them to the underlying dimension elements results in a significant decrease in the number of "in context" elements as well as the resulting Decision Tree.

Example 1. Trivial Case

In the trivial case where the context has no affect on the underlying fuzzy conditions, for example "monthly shopping" on a list of only monthly shopping items, no deformation occurs and any set operations and the set of rules reduces to that described in Equation (13).

Example 2. Woman in store

Suppose a woman customer comes into the virtual store to buy some groceries. The Decision Tree for this woman is based upon Table 3 - 1. A traditional Decision Tree would consist of 1.4 million potential nodes, depending upon the available data. Pruning the tree using standard methods requires traversing a large number of nodes, investigating each node for applicability. However, creating a context of "Monthly Shopping" (MS) and applying the fuzzification processes a number of things occur:

1. The "impulsive" customer type (CT) falls out of context as MS is considered planned, thus reducing the size of CT from 5 to 4.
2. Many of the product types (PT) that are considered impulse buys (e.g books, candy) or quickly perishable items (e.g. bread, lettuce) or irregular purchases (e.g nails),

daily purchases, weekly purchases and holiday items fall out of context reducing the size of the PT from 10 to 4.

3. Relative Product Price is unaffected by MS.

4. Since MS occurs on the weekend, Day of Week (DW) values Monday through Friday fall out of context reducing the DW dimension from 7 to 2.

5. Time of Year (TY) is unaffected

6. Customer Age (CA), the context MS usually involves heads of household which eliminates certain age categories such as “Under 10”, “Young Adult 10-20”, bringing the CA category from 10 to 8.

7. Geographic Location (GL) is unaffected.

Even more dramatic would be a context such as “Holiday - St. Patrick’s Day”. The types of products shoppers celebrating St. Patrick’s Day require comprise a very small group and the type of individual celebrating the holiday is likewise limited. The resulting Decision Tree is reduced considerably. The final node totals of customer Decision Trees for “Monthly Shopping” and “Holiday – St. Patrick’s Day” are shown in Table 3 - 2.

	Traditional DT	Ctx - Mnthly Shopping	Ctx - St. Patrick’s Day
Dimensions In Context	7	7	7
Elements In Context	51	42	24
Potential Nodes	$1.4x10^6$	$9.6x10^4$	1024

Table 3 - 2, Example 2 – Node Reduction Under Contexts

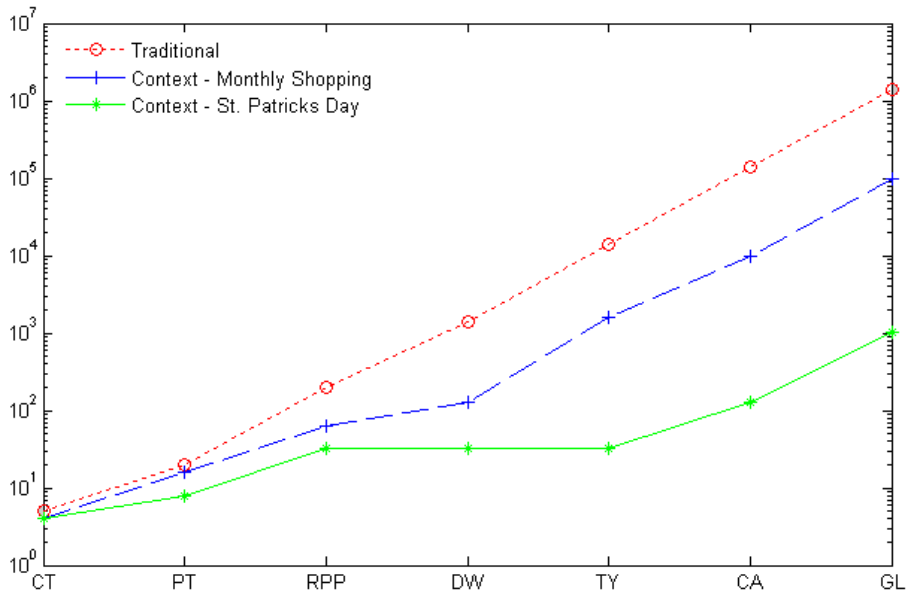

Figure 3 - 10, Node growth under normal conditions and contexts.

Example 3. Plant manager

The manager of a plant uses a Decision Tree to decide how to set up the production line, taking into account inventory, backlog, capacity and other dimensions. For sake of simplicity, limit to 10 elements per dimension. Creating holiday contexts allows the manager to tailor production to meet the changing demands as holidays come and go. Other contexts such as “Preferred Customer” and “Holiday Schedule”, quickly reduce the number of possibilities to a small number of “in-context” production options. An example is the “Preferred Customer” context, whose implementation eliminates all low priority, non-customer components, while the context “Holiday Schedule” eliminates those components not purchased or shipped during the holiday.

	Traditional DT	Ctx Pref. Customer	Ctx Holiday Sched.
Dimensions In Context	7	7	7
Elements In Context	70	27	33
Potential Nodes	$1x10^7$	4400	$3.6x10^4$

Table 3 - 3, Example 3 – Node Reduction Under Contexts

3.5 CONCLUSION

As shown in Examples 2 and 3, the use of contexts significantly reduces the number of "in context" dimension elements. In Example 2, the original number dropped from 51 to 42 to 24 for "St. Patrick's Day". The reductions were even more dramatic when applied to the number of potential nodes of the Decision Tree, dropping from $1.4x10^6$ down to 1024, resulting in a reduction of approximately 3 orders of magnitude.

Whether an e-commerce retailer, behavioral scientist, intelligent controller, or manager of a production plant; each relies upon Decision Trees to formulate rules for actions. However, outliers and large combinations of conditions can create difficult and confusing sets of rules that have limited applicability. Current solutions attempt to alleviate this problem through clever techniques or sheer brute force to derive meaning but have difficulty if relationships are numerous or non-intuitive.

The fuzzy methods demonstrated in this chapter improve upon these techniques by introducing new dimensions of uncertainty serving to both reduce the number and complexity of rules as well as tie non-intuitive relationships together within a larger meaningful context. The examples demonstrated many orders of magnitude improvement of subsequent Decision Tree construction over traditional methods.

Chapter 4

CONTEXTUAL DERIVATION FROM DECISION TREES (COT-DT) BASED ON ADVANCED DATA MINING TECHNIQUES AND INTELLIGENT CONTROL

This chapter presents an algorithm for the discovery of interesting contexts within Decision Trees. Applying these contexts with the CoFuH-DT technique reveals important information and rules that were previously hidden within the tree.

4.1 INTRODUCTION

Effective data mining requires the ability to quickly sift through mountains of data and extract meaningful kernels of knowledge [Han 06]. This new knowledge manifests in new rules for intelligent systems from e-commerce to intelligent controllers. There are a number of Advanced Data Mining Techniques such as Bayesian networks, Artificial Neural Network (ANN) classifiers, distance and fuzzy clustering techniques and others which are applied to the data in order to derive meaningful associations [Han 06], [Adom 01]. One of the more popular techniques is the Decision Tree.

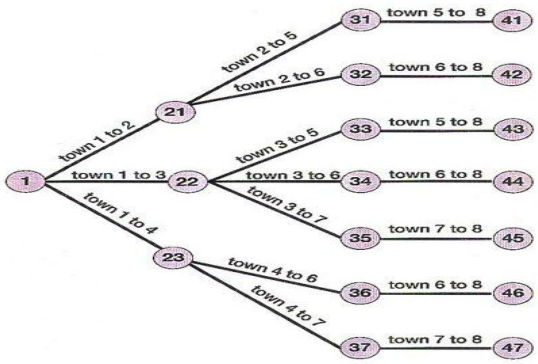

Figure 4 - 1, A typical Decision Tree used for data mining.

Decision Trees are built using techniques such as ID3 and C4.5 [Han 06], [Zhao 06]. These Decision Tree induction algorithms recursively "grow" the tree, starting from a single parent node containing a set of data, by selecting an attribute from among a set of candidate attributes. By using this attribute and distributing the data into smaller segments, new child nodes are generated, as demonstrated in Figure 4 - 2. ID3 uses a simpler notion of "information content" while C4.5 attempts to overcome the bias of uneven sampling by normalizing across attributes.

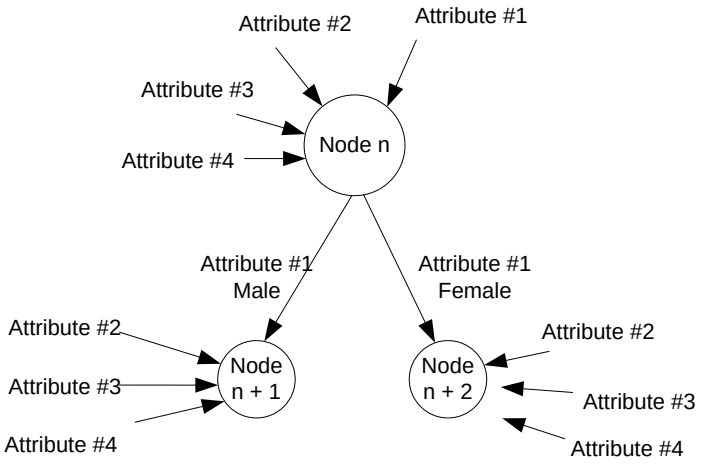

Figure 4 - 2, A Decision Tree node generation node with attributes.

The tree is grown in the following steps:

1. Determine appropriate information "threshold", designed to yield optimal "information content".
2. Choose attribute from among set of attributes with maximum "information gain"
3. If information gain from attribute exceeds threshold, create child nodes by splitting attribute accordingly [Sun 05].

ID3/C4.5 determines the maximum gain by choosing the attribute which will yield the most “information content” or clear differentiation of the data with a minimum amount of noise or randomness. If the gain is above a predetermined threshold, i.e. there is sufficient differentiation that new knowledge is likely then the node will produce one or more leaf offspring, with each leaf containing a subset of the parent node data partitioned along the attribute.

As a simple example of this technique, consider the node *n* in Figure 4 - 2 as representing a data sample with 100 college students, 50 male and 50 female. Now consider the Attribute #1 as *Gender*. *Gender* achieves maximum gain because it affects every data point and partitions the data into subsets of equal size. In contrast, a sample of 99 female students and 1 male student generates little gain.

As Figure 4 - 2. shows, by applying the Decision Tree algorithm to node *n*, 2 new nodes are generated in the tree along the *Gender* attribute, one for male and one for female.

This process continues recursively for each child node until no new nodes can be produced.

Decision Trees are a very effective tool for data mining [Han 06] but suffer from some drawbacks:

1. Noise

   Non-systemic errors in either the data or attributes can cause the induction method to generate spurious nodes, generating unnecessary complexity or creating a tree where meaningful paths are obscured [Yu 06], [Sun 05], [Zhao 06].

2. Large trees

   Large numbers of attributes or overly granular attributes can quickly grow trees to an unmanageable size. Initial pruning of trees by brute-force threshold limits creates a likelihood that meaningful but small relationships and non-intuitive relationships will be overlooked or skipped altogether [McCarty 08a].

3. Applicability

   The uncertain nature of both attributes and data often generate trees that have little applicability to real-world decision making [Zhao 06].

4. Slow to search

   Large tree searches for rule generation, using methods such as depth-first or breadth-first, are very expensive and time-consuming [Russell 03].

A number of approaches have been proposed to address these issues, such as using Fuzzy Trees [Zhao 06], introducing Support Vector Machines [Wang 06], or using the Contextual Fuzzy Type 2 Hierarchies for Decision Trees (CoFuH-DT) method [McCarty 08a]. The CoFuH-DT is fuzzification of the Decision Tree followed by application a Fuzzy Type 2 context. Under CoFuH-DT, Decision Trees can be pruned quickly via fuzzy set operators and understood in the context of polymorphic sets of rules.

However, in order for CoFuH-DT to be effective, contextual information must exist that can be applied to the Decision Tree. Simply running ID3 or C4.5 over the data is unlikely to produce anything but a more or less detailed tree; so a different, hybrid technique is required. Advanced Data Mining Techniques (ADMT) such as

Artificial Neural Networks (ANN) are an effective means of generating classifications and learning about patterns that may contain sparse or noisy data [Han 06]. As such ADMTs are an effective tool for generating a range of candidates for a Fuzzy Type 2 context.

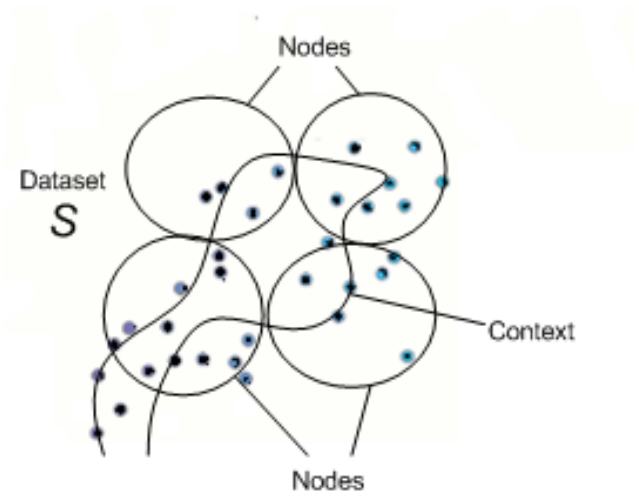

Figure 4 - 3, Context spanning several nodes

This chapter demonstrates application of several ADMTs to a Decision Tree generated from a sample data set. It shows how the resulting contexts are available for use by CoFuH-DT. This chapter is organized as follows: Section 2 introduces a typical data problem. Section 3 describes the various steps of applying an ADMT to the Decision Tree to generate contexts. Section 4 applies the algorithm to a sample data set to generate useful contexts. Section 5 presents the conclusions and future work.

4.2 PROBLEM STATEMENT

Consider a bank wanting to decide which customers represent the best credit risk. There are many types of customers with different income and backgrounds that present widely varying degrees of risk. Some will pay off their loans on time, others will be early, others late and still others will default. The loan officer decides to generate a profile of his customers using a Decision Tree. The attributes of the data used could end up looking like that of Table 4-1.

Attribute	Potential Values
Income	Many
Collateral	6
Age	7
Education	6
Occupation	Many
Children	5
Gender	2
Region	Many
Marital Status	4
# Cars	4
Owns Home	2
% Down Payment	5
Credit Score	Many

Table 4 - 1, Attributes of a typical customer.

By limiting the number of distinct ranges of values of *Income*, *Occupation*, *Region* and *Credit Score* to just 10, a Decision Tree could still have over 4 billion potential nodes. Making things even more difficult is that some values, like *Income* and *Credit Score*, have varying weights in lieu of other factors, such as the down payment and payment history. Other values, such as *Children* appear to have little relevance at all but may actually be very important in accurately assessing risk.

The loan officer wanting to create rules using the resulting Decision Tree is faced with a dilemma. He must choose between analyzing a huge tree in the hope of

gaining the necessary insight, or setting the information gain threshold high enough to reduce the tree to a manageable number of nodes. In the first case, resources and time required in order to process and analyze a huge base of nodes can be substantial. In the second case, by increasing the threshold for the Decision Tree algorithm, the resulting tree may be smaller and more manageable, but a lot of information could be lost in the process, potentially leaving the loan officer with no reliable way to measure a significant segment of the market.

CoFuH-DT presents a better alternative by combining the efficiency of the Decision Tree with the power of Fuzzy Type 2 contexts. Generating the contexts can be a difficult task, but is made easier through the use of ADMTs such as an Artificial Neural Network (ANN). This is accomplished by applying the ANN to the resulting datasets representing the nodes of the Decision Tree and thus generating a series of classifications, or contexts. These contexts can then be applied to the fuzzified Decision Tree using CoFuH-DT. The resulting Decision Tree is smaller, more semantically concise and appropriate to the situation but without the loss of information associated with traditional methods.

4.3 CoT-DT Algorithm

Contextual Derivation from Decision Trees (CoT-DT) works as follows: Consider the dataset *S*; applying ID3 or C4.5 or other algorithm to generate a Decision Tree produces a DT with number of nodes *M* with *N* leaf nodes. Each leaf node n_i of the set of all leaf nodes *N* contains a subset s_i of the dataset *S*.

$$\forall n_i \in N, f(n_i) = \{s_i \subset S\}, \ \cup s_i = S, i = 1, .., N \tag{23}$$

where $f(n_i)$ is a filter applying all the attributes of n_i against *S*. Then let the distance $f_d(n_i, n_{i+1})$ between any two nodes n_i, n_{i+1} be the number of intermediate nodes that must be traversed when traveling from n_i to n_{i+1} as demonstrated in Figure 4 - 4.

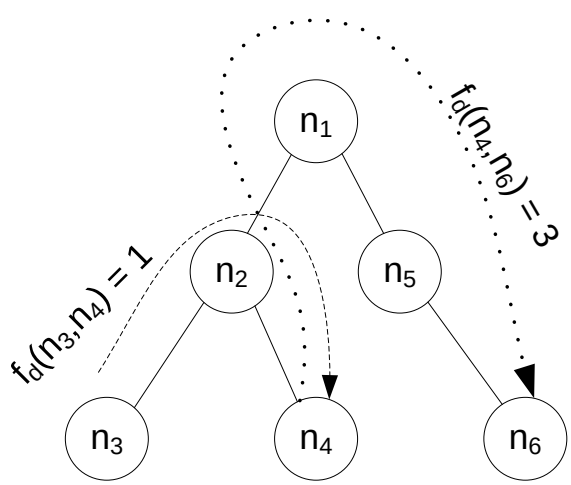

Figure 4 - 4, Calculating distance between nodes

Unlike traditional classification using ANNs or other ADMT, which seeks to create clusters of data based upon some measure of "closeness", context generation seeks to discover relationships that exist between sets of data within a given set of nodes. This is accomplished by examining the intersection of a particular classification

across a set of nodes. “Interestingness” is a function of the node and data characteristics for that classification.

Whenever the ADMT discovers a cluster that spans more than one node, a context is possible. The algorithm’s steps are as follows:

1. Decision Tree generation
2. Node selection
3. ANN classification
4. Context Evaluation and Creation

Step1. Decision Tree generation.

Use ID3, C4.5 or other algorithm as described in Equation (23) to determine the information threshold and generate the Decision Tree from the dataset *S* shown in Figure 4 - 5.

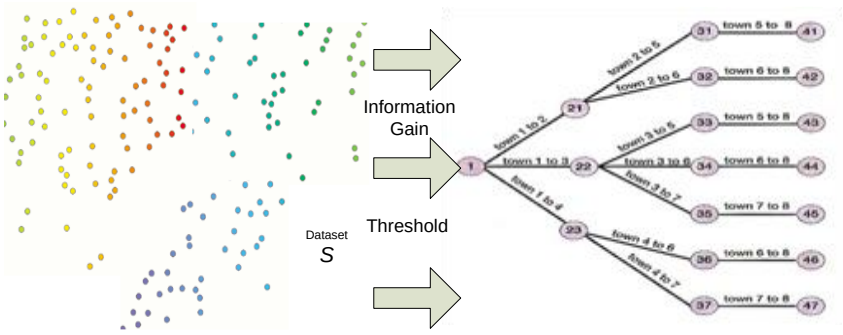

Figure 4 - 5, CoT-DT Step 1 - Creation of Decision Tree.

Node generation will depend upon how high or low the information threshold is set. The Decision Tree will contain *M* nodes and *N* leaf nodes.

Step 2. Node Selection.

Look at the Decision Tree from the point of view of a set-based operator. Each leaf n_i of the tree encompasses a subset $s_i \in S$ demonstrated in Figure 4 - 6.

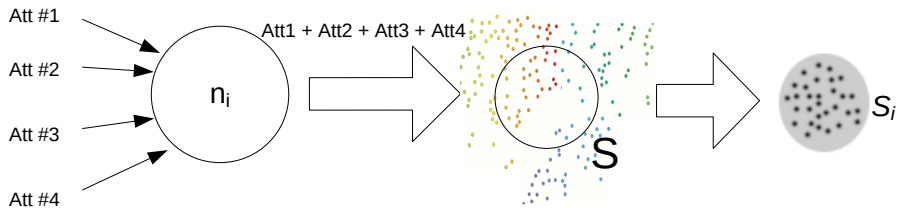

Figure 4 - 6, Nodes of Decision Tree produce subset s_i of original set S.

Figure 4 - 6. shows how the collection of attributes A of the leaf combine to create a filter that when applied to S, produces the data set s_i of the leaf.

$$\forall n_i \epsilon N,\ A_{n_i}(S) = s_i, i = 1, .., N \tag{24}$$

Node selection then combines s_i into subsets of S for analysis in Step 3.

Step 3. ADMT classification.

From the s_i created in Step 2, use, in this case, a multilayer, feed-forward, Error-Back Propagation Artificial Neural Network (EBP-ANN) to create a set of data clusters C as shown in Figure 4 - 7.

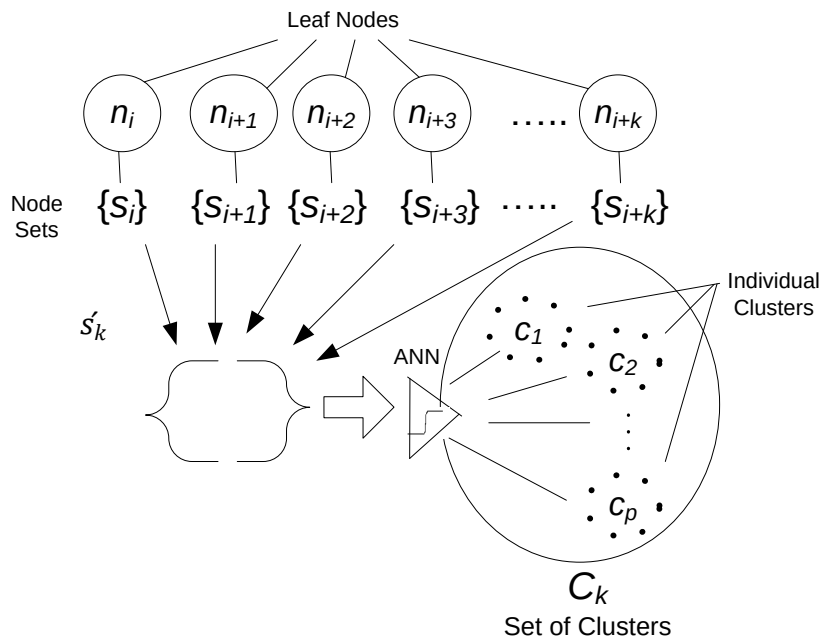

Figure 4 - 7, ANN classifier applied to leaf node sets produces clusters.

Each resulting cluster c_p in the set of generated clusters C_k represents a degree of "closeness" between a series of data points s'_k. s'_k represents the combination of leaf node s_i created in Step 2 and is a subset of S.

$$s'_k \subset S,\ g(s'_k) = \{c_p \mid \cup c_p = C_k\ p = 1,..,k\} \tag{25}$$

where $g(s'_k)$ is an ADMT such as an ANN that when applied to s'_k produces the set of clusters C_k.

Figure 4 - 8 demonstrates how cluster creation using an ANN combines subsets of a node set into one or more unique clusters.

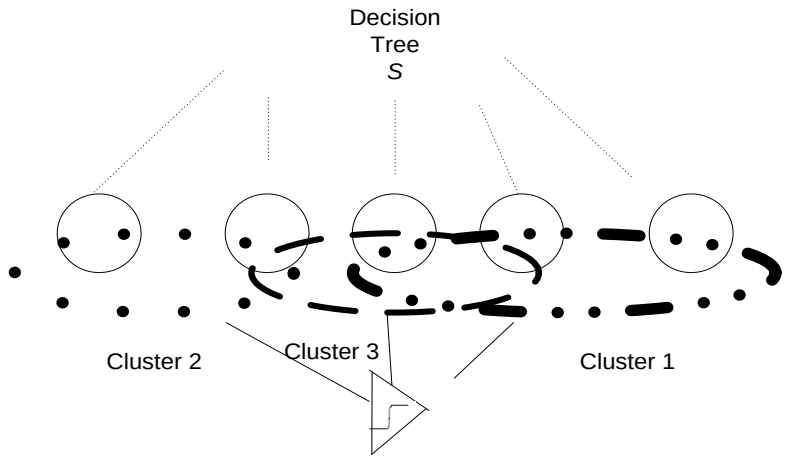

Figure 4 - 8. ANN cluster generation.

Step 4. Context Evaluation and Creation.

Compare each cluster $c_p \in C_k$ to each node n_i. Denote the non-empty intersection of c_p with each s_i in n_i as the element e_j.

$$e_j = s_i \cap c_p \, , e_j \neq \emptyset \tag{26}$$

The union of the node elements e_j over all or some subset of the leaf nodes N is called a cluster-span as shown in Figure 4 - 9.

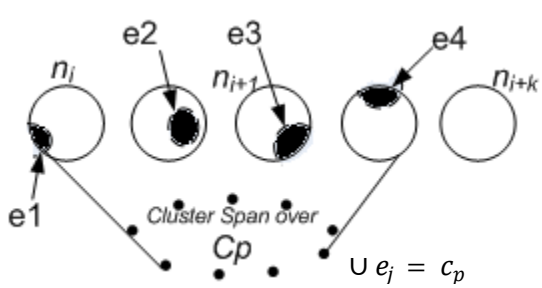

Figure 4 - 9. Cluster span over several nodes.

Each single node element e_j of the cluster span consists of a "coverage". Let $f_{dp}(e_j)$ represent the number of data points in e_j, and let $f_{dp}(s_i)$ represent the total number

of data points in the node's corresponding data set s_i. The coverage $f_{cvg}(e_j)$ is the ratio of the number of data points in e_j to the number of data points in s_i.

$$f_{cvg}(e_j) = \frac{f_{dp}(e_j)}{f_{dp}(s_i)} \tag{27}$$

Let $f_d(e_i,e_j)$ be the distance between the corresponding nodes for e_i and e_j as illustrated in Figure 4 - 4. Let $f_{dm}(e_j)$ represent the greatest distance between the node containing the element e_j and any other node in the cluster-span.

$$f_{dm}(e_i) = \max(f_d(e_i, e_j), \forall e_j \subset c_p,$$
$$i = 1,..,n,\ j = 1,..,n,\ p = 1,..,k \tag{28}$$

Further, let "interestingness" of an element $f_{int}(e_j)$ be a function of its coverage multiplied by its distance function.

$$f_{int}(e_j) = f_{cvg}(e_j) * f_{dm}(e_j) \tag{29}$$

In addition any cluster-span containing some non-empty set of elements $e_1..e_j$ also creates a "context", CT_i. The context is available to be fuzzified and used in CoFuH-DT.

$$CT_i = \cup\, e_j \tag{30}$$

Note that if a given context CT_i has only one element, the distance function for that element equals 0 as does the measure of interestingness for the context. The context may be particularly interesting but belonging to a single node it adds no new information to the Decision Tree. Hence for any given context CT_i to be "interesting" its corresponding cluster-span must have at least 2 elements. Interestingness of a entire context, F_{int} is the weighted) sum of the interestingness of its corresponding elements.

$$F_{int}\,(CT_i) = \sum_j \; w_j\, f_{int}\left(e_j\right),\; e_j \subset c_p \in C_k,\; j = 1,..,p, \text{ i=1,..,k} \qquad (31)$$

where w_j represents a given weight assigned to the corresponding e_j. Weights are a means to take into account the relative size or relevance of a node or to reduce the impact of noisy data.

As an example consider the following basic Decision Tree with four leaf nodes as shown in Figure 4 - 10. Each leaf node contains exactly 100 elements.

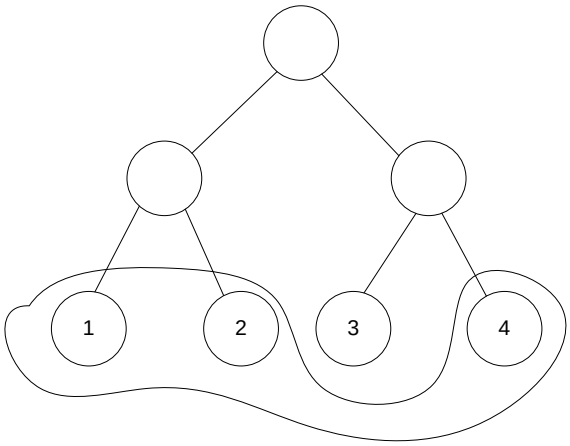

Figure 4 - 10. Sample Decision Tree with cluster-span.

Now consider a cluster-span which contains 50 elements from nodes 1 and 2 and another 25 elements from node 4. Assuming all nodes are weighted equally, by Equation (31), its corresponding context "interestingness" is calculated as follows:

$f_{int}(e_1) = 3 \text{ x } .5 = 1.5,$

$f_{int}(e_2) = 3 \text{ x } .5 = 1.5,$

$f_{int}(e_3) = 3 \text{ x}.25 = .75$

$$F_{int}\,(CT_i) = 1.5 + 1.5 + .75 = 3.75 \qquad (32)$$

Contexts with sufficient interestingness may now be employed with CoFuH-DT to perform fuzzy-set operations, classification and context pruning.

4.4 TEST EXAMPLES

A sample database provided for users of Microsoft's SQL Server 2005 Analysis Services contains approximately 60,000 purchase records. The dataset contains 12 relevant attributes, each with 2 to 70 possible values. The total potential size of the Decision Tree is $2x10^{10}$ nodes. The Microsoft Decision Tree induction algorithm is described as a proprietary hybrid algorithm based on a C4.5 algorithm combined with elements of CART (Classification And Regression Trees). Using the Microsoft Decision Tree induction algorithm to construct the Decision Tree resulted in 187 actual nodes.

Applying a standard back-propagation neural network to the dataset resulted in a number of potential contexts. Some of the more interesting contexts were based upon the customer's age and income. Creating fuzzy regions for both by breaking the span of ages into the fuzzy sets, YOUNG, MIDDLE-AGED, and OLD, and the span of income into the fuzzy sets POOR, LOWER-CLASS, MIDDLE-CLASS, UPPER-CLASS, and RICH generates a series of classifications.

From these classifications, two contexts, in particular, emerged with a high degree of "interestingness": RICH_AND_YOUNG and RICH_AND_OLD. Although they were sparse so coverage was low, they covered a number of distant nodes and thus were still quite interesting.

Each context showed a very high correlation between membership in the corresponding fuzzy region and high volume and high dollar purchases. Other cases, for example RICH_AND_MIDDLE-AGED, had a much lower correlation.

Two other ADMTs were also applied, a K-Means clustering algorithm and a Bayesian network. These also generated contexts. From the Bayesian network, there was a focus on marital status and no children while the K-Means added the dimension of home ownership. These contexts would be described as MARRIED_NO_CHILDREN (M-NC) and MARRIED_HOMEOWNER_NO_CHILDREN (M-HNC). Customers who were members of the contexts described all showed significantly higher predispositions to make more and/or higher value purchases than those who were not members.

Applying any of these contexts reduced the number of potential nodes on the original Decision Tree. These reductions were very dramatic due to the specificity of the context, making irrelevant or "out of context" many other attributes. Even though these contexts proved very significant, they were lost in the original Decision Tree generated with the commercial CART algorithm. Because the data was relatively sparse it fell below the threshold for information gain and was hence ignored in favor of more dense data.

A reasonable interpretation of the aforementioned contexts might be that younger buyers are more impulsive while older buyers are more secure in their finances than members in the middle group. Hence members of the two outside groups are more likely to take on greater and more premium discretionary purchases. Whatever the reason, a sales manager now has a collection of CoFuH-DT-based, semantically simple, yet powerful contexts with which to frame and generate rules for particular customers.

Finally, rule generation was made much simpler. In traditional rule generation, rules define an action for the set of conditions represented by a node [McCarty 08a], [Wang 06].

DEFINE RULE *rule_name*

ON *event*

IF *condition1*

AND condition2

…

AND condition

DO *action* (33)

Any situation described by the rule above may involve a great number of conditionals to accurately represent the large number of affected attributes and sub-conditions. However, as a result of CoT-DT combined with CoFuH-DT, generating a context-base rule is much simpler because the many disparate products and customers now belong to a single contextual category. For example a contextual rule based upon the context RICH_AND_YOUNG might look like this:

DEFINE RULE *RECOMMEND_PURCHASE*

ON CustomerPurchase

IF Customer IS RICH_AND_YOUNG

DO Recommend purchase PREMIUM_PRODUCT (34)

Use of CoT-DT, CoFuH-DT and Decision Trees is not limited to e-commerce applications. Intelligent controllers use a variety of methods to determine how to respond to their environment. Among them is the use of rules derived from Decision Trees.

A robotic land rover attempts to navigate a landscape with a myriad of environmental and physical obstacles and hazards. The faster it moves, the more quickly it must process all the various attributes and come to a good decision. However, there are times when certain factors become so overwhelming that a good decision only needs to take those most relevant factors into account while ignoring the others. Take the case where the land rover has to navigate a steep slope. Turning to the right or left greatly increases the possibility of a roll-over so virtually any decision which would involve such a turn is not a good one. It makes no sense to contemplate turning decisions or pursue decision branches which might be considered irrelevant when making a turn. At other times, outliers in behavior or actions which would in most cases be considered abnormal, suddenly become “normal” or preferred within a given context. For example suppose under low battery conditions the rover has an overriding need to seek a power source and may have to engage in any number of aberrant moves and behaviors to meet that goal.

CoFuH-DT/CoT-DT allows the rover to frame potential actions, such as might be required in a low battery condition, into a meaningful context as well as more quickly prune its Decision Tree, resulting in a more understandable set of rules.

Comparisons of Decision Trees using the aforementioned derived contexts are shown in Table 4-2. While the original Decision Tree (Org DT) had many potential

nodes, the Microsoft CART algorithm produced a tree with only 187 nodes. CoFuH-DT trees using the contexts RICH_AND_YOUNG/MIDDLE_AGED/OLD (EBP Cond 1) and RICH_AND_OLD (EBP Cond 2) resulted in much smaller trees but more significant in identifying buyers more likely to purchase. The same applies to a lesser extent for CoFuH-DT trees generated using Bayes and K-Means algorithms.

	Nodes	Avg. # Purch	Avg. $ Purch
Org DT	$2x10^{10}$	3.27	1588
MS SQL HDT	187	3.27	1588
EBP Cond 1	17	4.07	3343
EBP Cond 2	13	4.0	1537
Bayes	43	3.46	1839
K-Means	24	3.51	2000

Table 4 - 2, Node comparisons using various contexts

4.5 CONCLUSION

This chapter demonstrates two significant benefits of the Contextual Derivation from Decision Trees (CoT-DT) algorithm using Advanced Data Mining Techniques (ADMT):

The first benefit is that ADMT under CoT-DT can derive new contextual information from a fully-formed Decision Tree for use by Contextual Fuzzy Type-2 Hierarchies for Decision Trees (CoFuH-DT) rule generation. The second benefit of the CoT-DT approach is that it can be used to measure and validate the overall effectiveness of a Decision Tree induction algorithm. The more accurate or complete an algorithm, the fewer and less interesting contexts that are likely derivable. By the same token, CoT-DT can compensate for an ineffective algorithm by providing useful contexts for appropriate rule generation.

As demonstrated by experimental results of this chapter, the CoT-DT approach produced new and meaningful contexts. Viewing a Decision Tree within the narrow frame of a context reduced the in-context Decision Tree by many orders of magnitude over what was theoretically possible. After applying a commercial algorithm, CoT-DT was able to achieve an additional contextual reduction of over 90%.

Chapter 5

GENERAL APPLICATIONS OF MODERN HEURISTICS

This chapter presents heuristics used to modify traditional local search algorithms and enable autonomous vehicles to track, anticipate, and respond to moves from a human leader. A 99% decrease in a local search failure rate and a less expensive, yet more effective implementation of autonomous technology were obtained. Results were presented at IEEE ICIEA08 Conference, June 2008 and ETFA08 Conference, September 2008.

5.1 INTRODUCTION

Advanced Data Mining Techniques have many uses well beyond the data-centric applications described in preceding chapters. Decision Trees, for example, guide the actions of intelligent controllers such as those that might direct the path of an autonomous vehicle or serve as a computer opponent in a chess match. However, limitations of data mining often mean that real-world problems and applications require much more than a database of options or predictions, and as such rely on a much broader array of techniques to in order to meet a set of requirements. Some of these techniques, however, have limitations of their own. Among them are unacceptably high rates of failure and cost of implementation. This final chapter looks at ways heuristics can be used to address some of these limitations and solve real-world applications.

The first part of this chapter describes heuristics to modify various local search techniques called Descending Deviation Optimizations. While plenty of situations can be described with a modest-sized Decision Tree or database of rules, there are problems where the complete state space cannot be described, whether it is too large or simply unknown at the time. Data mining tools are of little use in this environment. Instead, there is a class of heuristics known as Local Search Algorithms (LSAs) which allow

exploration of the space locally, in the hopes of finding a global solution or maxima. While LSAs share many of the same applications as data mining and intelligent control, this chapter looks at an application of LSAs to solve an automation problem.

One of the very important aspects of factory automation is the efficient use of both time and space [Gao 07], [Chase 06]. Doing so requires the precise coordination of people, material and equipment in limited space boundaries in order to maximize throughput and minimize latency [GCI07]. However, having one optimal set layout is generally impractical as priorities often change during the course of a production cycle or significant events [Garcia 08]. For example, the breakdown or introduction of new equipment can significantly affect the production schedule.

Unfortunately, combinations of variables and constraints can quickly result in the factorial growth of the possible permutations to search beyond the practical ability of modern computer systems to thoroughly assess. An exhaustive search through a space of potential configurations becomes impractical. Problems like that of the production scheduling problem mentioned above belong to a generic class of problems called Constraint Satisfaction Problems (CSPs). CSPs often encompass a potential set of states for which the entire state space is beyond a system's ability to search comprehensively. CSPs belong to a class of combinatorial problems called NP for "Non-Deterministic Polynomial" for which a given solution can be found by a polynomial-time algorithm [Martin 07], [Sipser 06].

Local Search Algorithms (LSAs) have proven very useful for finding solutions to CSPs [Guimar 07]. LSAs compensate for a lack of universal awareness by starting at some beginning state then exploring neighboring states and testing for goal states along

the way [Russell 03]. This allows for a smaller requirement of resources as only neighboring states need to be stored or searched. If there are multiple goal states in the overall state space, then there is a significant probability that an LSA will discover one quickly. This makes LSAs the preferred method for solving CSPs such as the factory automation production scheduling problem [Liu 07]. However, there are many different LSA techniques and all have various issues particularly dealing with locally optimal but globally sub-optimal states called local maxima. Descending Deviation Optimizations addresses some of these issues by allowing LSAs to move away from local maxima in a controlled fashion so as to have a higher likelihood of finding global maxima.

The second section of this chapter takes a look at how to simplify a tracking algorithm. There are situations where it is necessary or desirable to be able to rely upon autonomous, machine-guided vehicles to perform certain tasks. It is not practical, for example, to send a human to explore the surface of Mars. Nor is it wise for a human to travel the crater of an active volcano.

In less extreme conditions, autonomous vehicles are still called upon to perform a variety of tasks. Unfortunately both the difficulty and cost often prove significant obstacles to implementation [Liu 06]. Innovative solutions have addressed this [Wall 02], but challenges in the form of urban terrain, road conditions, traffic “rules” and other obstacles continue to plague autonomous vehicles. Situations do exist, however, where the environment is tightly controlled enough not to require a fully autonomous solution but rather one which combines human leadership with the ability of a vehicle to follow a leader.

For example, it may be necessary for an individual to require the help of a machine to transport material from point to point. An example could be an airport, where a person has to move luggage across a busy terminal, or a factory, where inventory is moved from production to shipping through small corridors or a junkyard where tons of metal has to guided around piles of debris. More extreme cases might involve the removal of hazardous waste or movement through a dangerous area where it would be preferable to use automated vehicles in lieu of human resources. Whereas it might otherwise be prohibitively expensive or simply too dangerous to trust to a truly automated vehicle, a hybrid system, with a human leader and an array of mechanical followers could prove a practical alternative.

"Following" technology, as opposed to a purely autonomous one, doesn't require sophisticated decisions with respect to direction, speed, hazards, or road conditions and as such requires less sophisticated sensory hardware and software. Additional reliance upon the judgment of the human leader can also mitigate the impact of obstacles and other issues which can make the operation of a purely autonomous vehicle difficult and hazardous.

Other solutions for automated following have been proposed, for example, by combining CCD cameras and neural networks for pattern recognition [Omura 99], motion sensors, GPS systems and standard communications [Chang 91] for platooning. This thesis chapter demonstrates that combining line-of-sight devices and a fuzzy algorithm for following is superior to the first solution in that it avoids much of difficulties associated with noise in the patterns and superior to the latter solution in that it employs a simpler array of devices and logic. An autonomous follower, using FLoST

(Fuzzy Line of SighT) to mimic its human leader, presents a much more cost-effective solution and a potentially more effective one. Using the analogy of a mother "Duck" and her "ducklings", the FLoST algorithm guides a series of mobile, autonomous units to follow a lead vehicle (or "Duck") and each other from a predetermined distance, mimicking both velocity and, to a greater or lesser degree depending upon conditions, direction traveled.

The proposed alternative technique relies upon a series of rapid angular scans to achieve location and distance measurements to the lead target. Technologies for line-of-sight tracking have been in use in both the commercial and military sectors for many years in various devices [Leigh 97], [Michaud 06], [Hsu 07], [Crump 07]. These devices detect an object (such as a hostile aircraft) and relay information such as distance, direction and speed to other units. Such devices, mounted upon and directing the movement of some sort of mobile platform, following a human or mechanical leader, can thereby creating some new utility.

This chapter is organized as follows: Section 5.2.1 presents a problem statement and a brief look at various local search techniques under consideration. Section 5.2.2 presents the Descending Deviation Optimizations steps. Section 5.2.3 presents the application of Descending Deviation Optimizations to Simulated Annealing and Stochastic Hill Climbing to solve the scheduling problem along with other test results for many of the traditional techniques.

Section 5.3.1 presents a simple scenario and a series of applications for the tracking algorithm presented in this paper. Section 5.3.2 presents the FLoST (Fuzzy Line of SighT) algorithm. Section 5.3.3 lists test scenarios along with a discussion of

the FLoST algorithm performance. Section 5.4 will present conclusions and future work.

5.2.1 Descending Deviation Optimization Techniques For Scheduling Problems

5.1.2 Background and Problem Statement

Factory automation scheduling must often take into account a myriad of competing elements ranging from floor space to personnel and equipment costs, production times to shipping and delivery priorities. Finding the best configuration means creating a schedule where conflicts and idle time are reduced to the greatest extent possible [Li 06], [Chase 06], [Ishi 99].

In a typical factory a number of different products are produced. Each product consists of components. Each component is made from raw materials or other components delivered to the factory or produced within it. Raw materials must be shipped in and moved across the factory floor from a receiving point to a production station navigating a maze through which other raw materials, components and products must also pass.

For testing purposes, consider a factory that produces 8 products. From raw material to shipped product there are 8 stations to pass through: receiving (R), component assembly (CA), quality inspection (QI), final assembly (FA), inventory control (IC) and storage (S), testing (T) and loading and shipping (LS).

The factory's goal is to try to reduce the time material is in the plant to the smallest possible amount. This means moving raw materials in and product out as quickly as possible. But the situation can be very dynamic, with changing schedules and priorities and product lines. In order to work as efficiently as possible the factory must be able to create a layout that will allow materials to move such that there is a limited amount of time spent waiting to move to the next station. Constraints include

minimum time spent at each station and minimum time to move between stations. The goal state is one in which all products were at a given station with no product waiting on another.

As there was no way to predict what the initial state would be, a random state generator was used to create 1000 random starting states to see how well the scheduling problem could be resolved to a goal state.

A number of Local Search Algorithms (LSAs) were used to see which could most consistently move from a random starting state to a goal state without being trapped by local maxima.

Heuristics determine where in the local neighborhood LSAs are to search as well as places to avoid. Many LSAs work to explore nearby maxima through a process of moving to successively more optimal states, hoping to encounter a global solution along the way [Martin 07], [Russell 03], [Mantawy 99]. The problem is that many problems are populated with localized maxima such that flowing nearby gradients can "trap" a LSA by leading it to a position which is not a global solution but in which all of its neighboring positions lead to a less optimal state as shown in Figure 5.1 - 1.

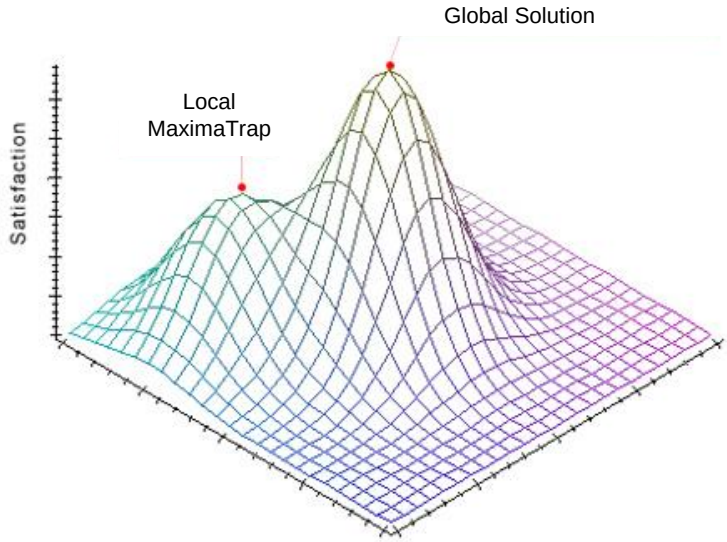

Figure 5.1 - 1, Comparison of Local/Global Maxima.

Some LSAs attempt to escape out of these local maxima through some sort of random “bounce” [Kurbel 98], [Pasias 04], [Kliewer 00], [Mendon 97] which moves an algorithm to a less optimal state but potentially into a location more capable of providing a solution as shown in Figure 5.1 - 2.

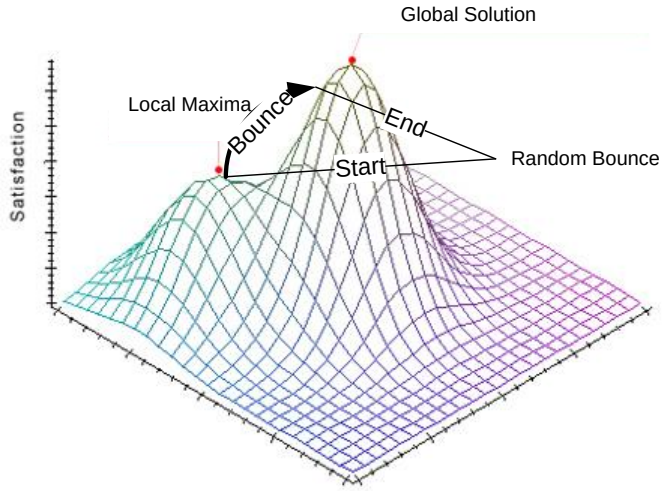

Figure 5.1 - 2, Bounce out of a Local Maxima Trap.

These random “bounces” are often not successful however, leading an algorithm away, rather than towards, a solution; expending time and computing resources in a fruitless search. Descending Deviation Optimizations (DDOs) tries to improve upon an LSAs ability to escape local maxima and find a goal state by restricting it movements somewhat in order to prevent it from moving to far away in any random direction from a potential goal state. This process works well in the scheduling problem presented in this paper because the state space contains a number of goal states spread throughout.

Local Search algorithms tried were as follows:

1. Hill Climbing
2. Stochastic Hill Climbing
3. Random Restart Hill Climbing
4. Simulated Annealing
5. Genetic Mutation
6. Minimum Conflicts Search
7. Tabu Search

The results are listed in Table 5.1 - 1.

Algorithm	Tries	Success	Failure	% Success
Hill Climbing	1000	141	859	14.1
Stochastic Hill climbing	1000	146	854	14.6
Random Restart Hill Climbing[1]	1000	866	134	86.6
Simulated Annealing [2]	1000	271	729	27.1
Genetic Mutation[3]	1000	229	771	22.9
Min Conflicts[4]	1000	919	81	91.9
Tabu Search[5]	1000	680	320	68.0

Table 5.1 - 1, Initial Results of LSA Testing

1. Random Restart declares failure after 100 restarts and no goal state
2. Simulated Annealing alpha set at .99, number iterations max set to 1000, temp set to 1000

3. Genetic Mutation population of 30 boards, sample of 2
4. Min Conflicts max iterations set to 100
5. Tabu Search max iterations set to 100

All of the techniques had some success in finding goal states, but the most successful required additional memory resources (Minimum Conflicts Search, Tabu Search) or a “lucky” combination of start states (Random Restart Hill Climbing) in order to succeed. With limited resources, it would be more advantageous to implement a different strategy using one of the other techniques and the Descending Deviations Optimization.

5.1.3 The Descending Deviation Optimizations Technique

Steps in the Descending Deviation Optimization (DDO) Implementation are then as follows:

Step 1. DDO-LSA generates a potential random choice. If the choice leads to a goal state then declare success.

Step 2. DDO-LSA choice is compared to the DDO threshold. If the choice moves the algorithm beyond that threshold, then choice is rejected and algorithm selects another random choice and tests again until a choice if found or all choices are tested.

Step 3. If choice is accepted, the optimal threshold reduced by a predetermined amount and the algorithm moves to Step 1.

As an example of the DDO technique consider a common local search technique: Simulated Annealing.

Simulated Annealing (SA), named after a process in metallurgy whereby metals are successively heated and cooled, implements a succession of random "bounces" that slowly diminish over time [Kliewer 00], [Mendon 97]. SA's pseudo-random selection method measures a random pick against a slowly descending de-optimization threshold. The algorithm allows a large range (nearly random) set of choices early on, getting progressively more restrictive in favor of better choices with each iteration. Since the range of options is greater in the beginning, it will have a tendency to explore more maxima and is correspondingly more likely to find one that is a global solution. SA is able to explore a relatively wide range of possibilities when compared to other algorithms and does a comparatively good job of finding global maxima compared with

other local search techniques. However this can be computationally expensive. In addition, the algorithm can have a tendency to be lead hopelessly astray by a succession of less than optimal choices as demonstrated in Figure 5.1 - 3.

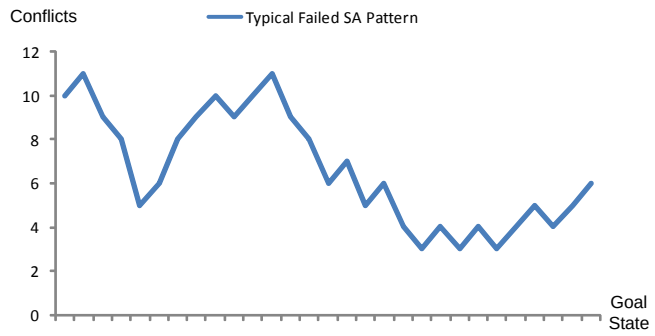

Figure 5.1 - 3, Pattern in which SA fails to find a solution

The DDO approach to SA takes the original SA implementation and adds the following optimizations:

1. An artificial, decreasing ceiling is imposed on the allowable number of conflicts. The DDO threshold is a function of the square root of the temperature variable. This prevents the solution from going from a lower state to a much higher state late in the process via a series of small, negative changes demonstrated in Figure 5.1 - 4. With each iteration the DDO threshold forces the SA to explore a smaller and smaller range of randomizations, hopefully to move it more quickly to the goal state.
2. Some versions of SA pick a value and may or may not use it depending upon whether or not it exceeds some “fitness” value. In this case, all the local potential moves are tested. Any move which would cause a no-operation to occur is thrown out of the sample of choices so that each iteration produces only those values that meet the fitness criteria.

3. During the screening process, if a particular choice is found that reaches the goal state, use that choice automatically so the process ends in success.

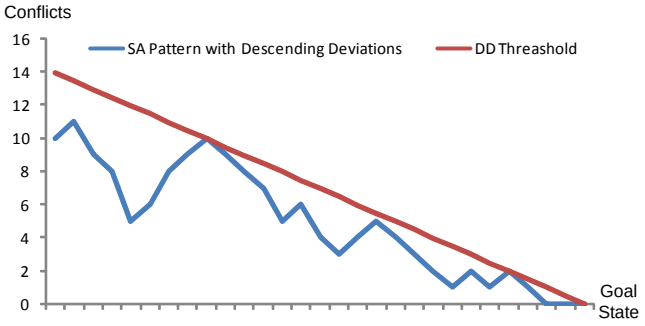

Figure 5.1 - 4, Simulated Annealing with Descending Deviations

5.1.4 Test Examples

In order see how effective DDOs are, the best and worst LSAs were chosen for implementation; Simulated Annealing (SA) as the best performing and Stochastic Hill Climbing (SHC) as the worst. Traditional Hill Climbing does not utilize a random component so was excluded.

Stochastic Hill Climbing (SHC) is a variant of the traditional Hill Climbing in which not the steepest ascent is picked but any ascent is eligible, dictated by a probability assigned to each option [Russell 03]. The probability is dependent to some degree upon the steepness of the ascent.

DDO-SHC works exactly like the traditional SHC until it gets "stuck", at which point it "bounces" the solution to a nearby, less optimal state and again applies the original strategy. The "bounces" are gradually lessened in height or until they disappear at which time if a global solution is not reached, the strategy fails.

The DDO-SHC and DDO-SA were added to the suite of LSAs and tested against the scheduling problem. The results of the modified LSAS are listed in Table II.

Algorithm	Tries	Success	Failure	% Success
DDO-Stochastic Hill Climbing[1]	1000	253	747	25.3
DD-Simulated Annealing[2]	1000	993	7	99.3

Table 5.1 - 2, Results of Modified Local Search Algorithm Testing.

1. DD-Hill rescued 121 failures, threshold set at 5
2. Simulated Annealing/DD-Simulated Annealing alpha set at .99, number iterations max set to 1000, temp set to 1000

In both cases DDO modifications to the original LSAs resulted in significant improvements in the LSAs ability to avoid local maxima and find a global solution. The DDO-SHC success rate nearly doubled (14.6% to 25.3%) while the DDO-SA achieved an almost fourfold (27.1% to 99.3%) rate increase to the point it was nearly perfect and better than any of the traditional LSAs tried.

There were 2 additional benefits as well for the DDO-SA algorithm. Despite the additional overhead imposed by the DDO, the increased success rate resulted in 20% fewer iterations overall for the given 1000-test cycle. In addition, the algorithm also displayed a lesser tendency to “wander around” or be lead astray by a series of bad choices. This resulted in both more successes and lead to a net time reduction of over 35% to complete 1000 iterations, also resulting in a large net decrease in computational resources required.

5.2 LINE-OF-SIGHT TRACKING BASED UPON MODERN HEURISTICS APPROACH

This remainder of this chapter presents heuristics to enable autonomous vehicles to track, anticipate and respond to moves from a human leader. These heuristics for less expensive, yet more effective implementation of autonomous technology. The results were presented at IEEE ICIEA08, June 2008.

5.2.1 Introduction

Intelligent control is one area that can make extensive use of both data mining and local search techniques. Using a Decision Tree, for instance, a controller can turn a database of information into a mechanism to determine various actions. In the event a database is not available, Local Search techniques allow it to explore the neighborhood for the most optimal course of action.

Among other things, intelligent controllers are in use today to drive autonomous vehicles. There are situations where it is necessary or desirable to be able to rely upon autonomous, machine-guided vehicles to perform certain tasks. On the surface of Mars, for example, it is not practical to send a human driver; while the crater of an active volcano may be practical but deemed too hazardous, but not all conditions are this extreme.

Regardless of the situation, there is usually significant difficulty and cost to autonomous implementation [Liu 06]. Innovative solutions have addressed this [Wall 02] but challenges in the form of urban terrain, road conditions, traffic “rules” and other obstacles continue to plague autonomous vehicle operation.

An alternative exists for those situations where the environment is tightly controlled enough not to require a fully autonomous solution and safe and practical

enough for human participation. In this situation an approach is possible which combines human leadership with the ability of a vehicle to follow.

For example, it may be necessary for an individual to require the help of a machine to transport material from point to point. An example could be an airport, where a person has to move luggage across a busy terminal, or a factory, where inventory is moved from production to shipping through small corridors or a junkyard where tons of metal needs to be guided around piles of debris. More extreme cases might involve the removal of hazardous waste or movement through a dangerous area where it would be preferable to use automated vehicles in lieu of human resources. Whereas, it might otherwise be prohibitively expensive or simply too dangerous to trust to a truly automated vehicle, a hybrid system, with a human leader and an array of mechanical followers could prove a practical alternative.

"Following" technology, as opposed to a purely autonomous one, doesn't require sophisticated decisions with respect to direction, speed, hazards, or road conditions and as such requires less sophisticated sensory hardware and software. Additional reliance upon the judgment of the human leader can also mitigate the impact of obstacles and other issues which can make the operation of a purely autonomous vehicle difficult and hazardous.

Other solutions for autonomous following have been proposed, for example, by combining CCD cameras and neural networks for pattern recognition [Omura 99], motion sensors, GPS systems and standard communications [Chang 91] for platooning. This chapter demonstrates that combining line-of-sight devices and a fuzzy algorithm for following has advantages to the first solution in that it avoids many of the

difficulties associated with noise in the patterns. It also has advantages to the latter solution in that it employs a simpler array of devices and logic. An autonomous follower, using FLoST to mimic its human leader, presents a much more cost-effective solution and a potentially more effective one.

FLoST is a novel, fuzzy arithmetic based algorithm. Using the analogy of a mother "Duck" and her "ducklings", the algorithm guides a series of mobile, autonomous units to follow a lead vehicle (or "Duck") and each other from a predetermined distance, mimicking both velocity and, to a greater or lesser degree depending upon conditions, direction traveled.

The proposed alternative technique relies upon a series of rapid angular scans to achieve location and distance measurements to the lead target. Technologies for line-of-sight tracking have been in use in devices in both the commercial and military sectors for many years in various devices [Leigh 97], [Michaud 06]. These devices detect an object, such as a hostile aircraft, and relay information, such as distance, direction and speed, to other units. Such devices, mounted upon and directing the movement of some sort of mobile platform, following a human or mechanical leader, can create some new utility.

The remainder of this chapter is organized as follows: Section 5.2.2 presents a problem statement with simple scenario and a series of applications for the algorithm presented in this chapter. Section 5.2.3 presents the FLoST, for Fuzzy Line of SighT, algorithm. Section 5.2.4 lists test scenarios along with a discussion of the FLoST algorithm performance. Section 5.2.5 will present conclusions.

5.2.2 Problem Statement

For the purpose of demonstration, consider a lead vehicle ("Duck") and *N* followers ("ducklings"). Based on the FLoST algorithm presented in this paper, each "duckling" becomes the "Duck" to the subsequent "duckling", hence any configuration of 1 "Duck" to *N* "ducklings" is possible.

This technique has application in many areas where the movement of material is impractical for human agents. Consider the following scenario: an earthquake damages a chemical plant. Highly explosive chemicals must be moved to another facility immediately but it is deemed unsafe for anyone to be near the chemicals while in transit. Applying FLoST in this scenario, the chemicals are loaded onto a series of FLoST-equipped vehicles following a lead vehicle with a human driver. The lead vehicle, or "Duck", is a heavily armored vehicle able to protect its human driver from the effects of a blast. The followers, or "ducklings", are FLoST transports.

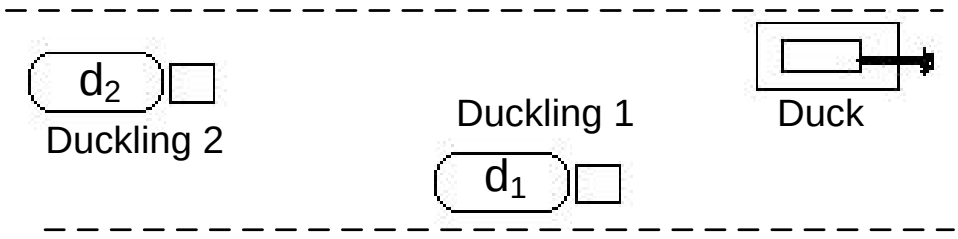

Figure 5.2 - 1, 2-D surface, example with 1-Duck and 2 ducklings.

Typically the "Duck" will proceed in a determined, but not constant direction, and will not have to back-track at any point. The road surface may contain obstacles to move around, but otherwise allow the "ducklings" to maintain line-of-sight to the "Duck". "Ducklings" themselves are "daisy-chained" such that the "duckling" in front will serve as its follower's respective "Duck". It is reasonable to assume that the

“Duck” will not intentionally try to evade the “ducklings” so its movement will be fairly consistent, though it may be necessary, at times, for more drastic maneuvers.

5.2.3 FLoST (Fuzzy Line-of-Sight Tracking) algorithm

The FLoST algorithm will be presented on a generic problem of *N* "ducklings" following a "Duck", as illustrated by Figure 5.2 - 2:

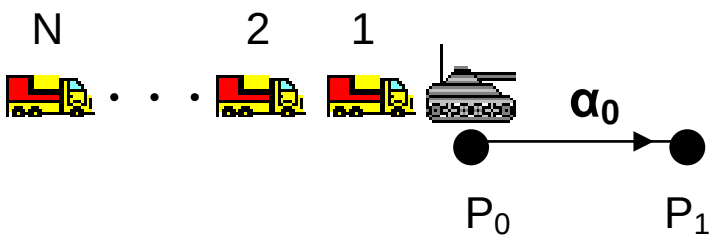

Figure 5.2 - 2, Duck and ducklings at start.

The heuristic of the FLoST algorithm allows each "duckling" to follow the "Duck" as it proceeds from point to point on its journey. Each "duckling" accomplishes this by maintaining a knowledge base of "Duck" behavior. At each point, the "duckling" records the relative distance, ΔP, and relative direction change, $\Delta\alpha$, of the "Duck" in its knowledge base, allowing it to determine its absolute position and direction as well as its desired velocity.

The "duckling" then applies the FLoST heuristic to better predict future behavior of the "Duck's" human driver as a function of its past behavior. In this example, the "duckling" creates three speed "zones" called Slow (*S*), Normal (*N*), and Fast (*F*), which are overlapping measures of the maximum speed of the "Duck" and used to assign direction changes, $\Delta\alpha$, along with an average direction change α_{Avg}, as shown in Figure. 5.2 - 3.

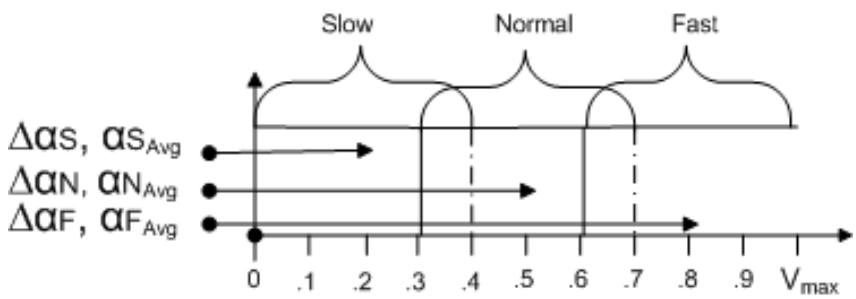

Figure 5.2 - 3, Direction changes and averages in Duck speed zones

The "duckling" then assigns each Δα to its respective zone or zones if it lies within an overlap:

$$\Delta\alpha_i \in \begin{cases} S, & V_i \le .4V_{max} \\ N, & .3V_{max} \le V_i \le .7V_{max} \\ F, & V_i \ge .6V_{max} \end{cases} \tag{35}$$

where V_{max} is the maximum velocity of the "Duck". By taking the mean over all Δα in a given zone, the duckling calculates the α_{Avg}, which is then used to generate a turn rate coefficient for that zone. In this example the coefficients are:

$$TR_x = \tfrac{1}{2}\alpha_{xAvg} + \tfrac{1}{2}\alpha_{Last},\ x \in \{S, N, F\} \tag{36}$$

where α_{Avg} is the average α for a given zone; α_{Last} is the last measurement taken for that zone; and TR_x is a component used in a fuzzified function to calculate the search.

At the beginning of this scenario, the "Duck" starts at location P_0, followed by two "ducklings". The "Duck" will then proceed over time Δt in a direction and speed indicated by the angle α_0 to the point P_1 as shown in Figure 5.2 - 4. The first "duckling"

will orient itself on and proceed to P_0, then using the FLoST algorithm, the duckling will begin its first scan for the "Duck"using as its first search angle, Θ, the vehicle Maximum Turn Rate (MTR) based upon the fuzzy equation developed by Wu, Zeng, Chaing and Lee [Wu 05] for a given vehicle.

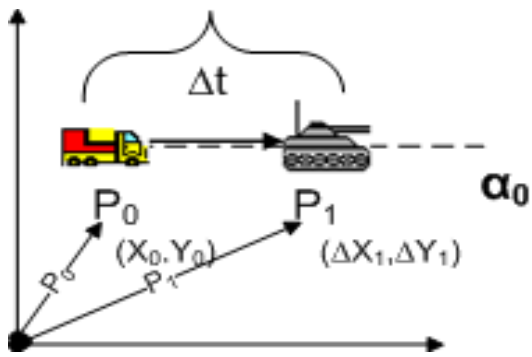

Figure 5.2 - 4, Duck proceeds to first point

As the "duckling" moves from point to point, it updates its knowledge base of "Duck" behavior with information derived from each new "Duck" point. From the original speed zones, the duckling creates fuzzified versions of Slow (*S*), Normal (*N*), and Fast (*F*) that range from 0 to the maximum velocity of the Duck (V_{max}) as shown in Figure 5.2 - 5.

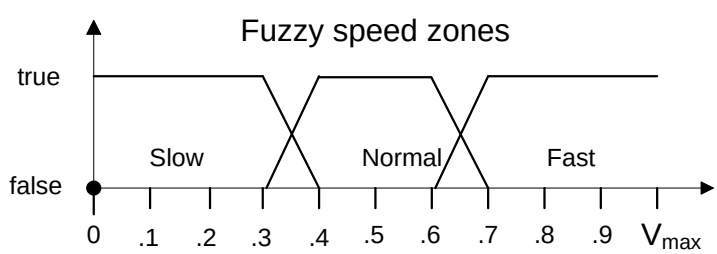

Figure 5.2 - 5, Fuzzified speed of Duck.

The duckling applies the FLoST algorithm using the following steps:

Step 1. Scan for and locate "Duck". Apply FLoST to determine the search angle Θ, calculated as follows:

$$\Theta = Vf * TR = [Vs \quad Vn \quad Vf] * \begin{bmatrix} TR_s \\ TR_n \\ TR_f \end{bmatrix} \tag{37}$$

where Θ has a minimum value of 1 degree. In the absence of any points for any TR_x, use the vehicle Maximum Turn Rate (MTR).

The search angle is drawn using the +/- Θ offset from the current angle α_i The search continues until either the "Duck" is located or it is determined the "Duck" is lost as illustrated by Figure 5.2 - 6.

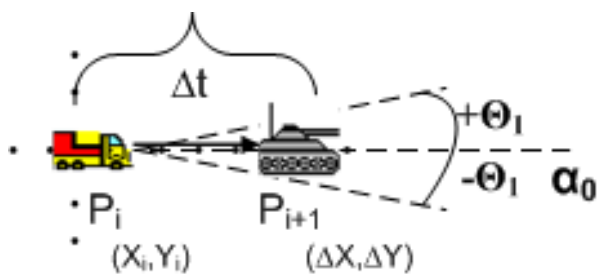

Figure 5.2 - 6, Scan for Duck at P_1.

Step 2. Use information about Duck's relative position to determine next point: P_1, and a new direction α_1.

The "duckling's" rangefinder provides a relative distance from the "Duck", ΔP, and the traversal mechanism provides the relative direction Δα. The new location P_1 and the new vector α_1 are defined as:

$$P_{i+1} = P_i + \Delta P_i \tag{38}$$

$$\alpha_{i+1} = \alpha_0 + \Delta\alpha \tag{39}$$

Step 3. Calculate the velocity of "Duck" to P_1 as:

$$V_i = |P_i - P_{i-1}| / \Delta t_i, \qquad \Delta t_i = t_i - t_{i-1} \tag{40}$$

Step 4. Update the knowledge base as described by Equation. (35) & (36).

Step 5. Adjust course and speed and then proceed to "Duck's" new known location.

As opposed to just applying the Maximum Turn Rate, MTR, the FLoST TR_x will usually generate a smaller search area than MTR. In this way, the "duckling" (d) can concentrate its search in the area the "Duck" (D) appears to be headed as illustrated in Figure 5.2-7.

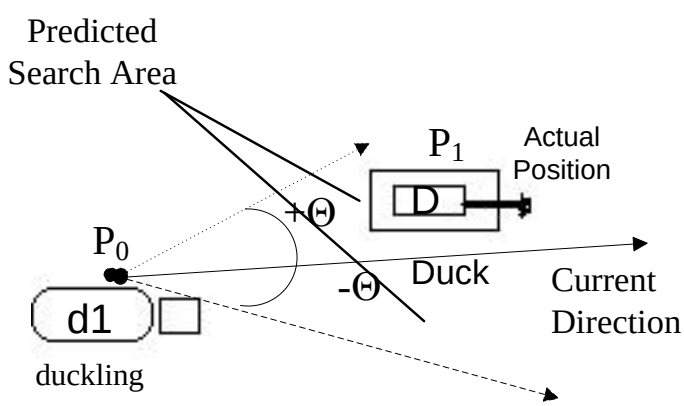

Figure 5.2 - 7, Choosing a search area.

The major hardware components of the "duckling" consist primarily of an array of Line-of-Sight (LoS) sensors mounted on elements that can traverse the search area. A sample configuration is illustrated in Figure 5.2 - 8. These LoS mechanisms feed

speed, direction and distance information to the navigation system which adjusts the "duckling's" movement accordingly.

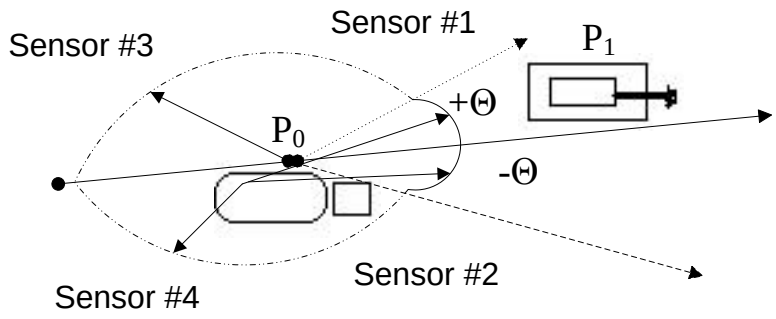

Figure 5.2 - 8, Duckling searching with sensor array

Using the FLoST algorithm, the "duckling" follows the "Duck" from point to point, adjusting course and speed and updating its knowledge base of prior actions as shown in Figure 5.2 - 9. The algorithm attempts to minimize the search area whenever possible, adjusting the search angles to account for variations in "Duck" movement at each iteration. Minimizing search angles allows for a more rapid scan rate which both reduces the search area and enables the "duckling" to make course and speed corrections on a more frequent basis thus reducing the chance for future misdirection.

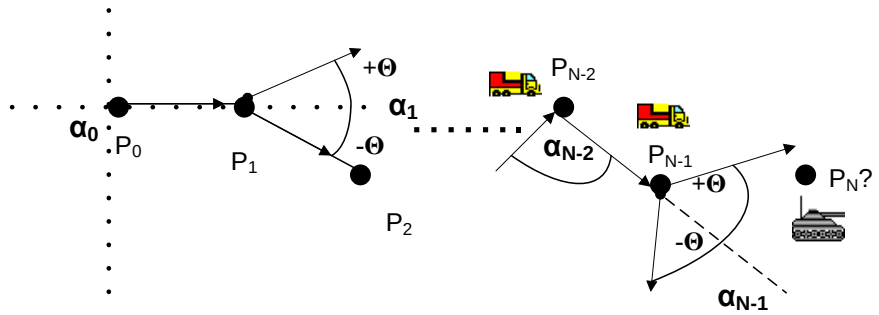

Figure 5.2 - 9, Two ducklings using FLoST to follow Duck

This is possible when the actions of the "Duck" are generally consistent and relieves the "duckling" of having to take into account considerations regarding terrain,

speed, safety, and obstacles. These, and other factors, can be very difficult and expensive to implement in an automated system. However, such factors are much more easily "inferred" by training the duckling to follow the behavior of the "Duck" precisely. Because human behavior is likely to vary at differing speeds and times, FLoST implements a series of fuzzy speed zones TRs. Because transitions are loosely defined, variances in behavior are more easily tolerated.

5.2.4 Test Examples

Examples consist of both trivial cases and non-trivial cases. Trivial cases arise when the following three instances occur: velocity is zero; the change of direction is zero; and the sample time is zero. In each case, the search angle collapses. These trivial examples will not be discussed any further. Non-trivial cases arise when the "Duck" moves in one direction for a period of time, establishing a knowledge base of very small direction changes, then executes a maximum turn in one direction or the other. For the "ducklings" this behavior is unexpected and will require additional iterations in order to learn and adapt to this new pattern. The performance of FLoST will be discussed using three boundary cases, representing three scenarios of sudden or unusual trajectory of "Duck" movements. These cases are: the "Duck" is going in tight circle; the "Duck" is going in spiral; the "Duck" is weaving back and forth.

The first boundary case, where the "Duck" moves in a tight circle using the maximum turn rate (MTR), is shown in Figure 5.2 - 10.

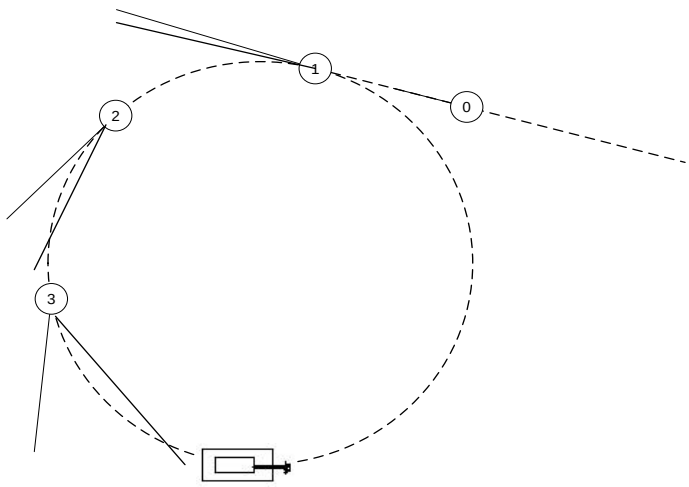

Figure 5.2 - 10, Duck moves in tight circle

As the duckling has “learned” only small movements to this point, it will have to learn very different, new behavior in order to generate a reasonable search rate coefficient TR.

While a traditional search uses the full MTR to establish the search angle, the coefficients used by the “duckling” create a tight angle initially, growing larger with each sampling. While the standard search will always capture the “Duck” in its primary scan, the “duckling” will require many samples in order to generate a large enough search angle. A way to compare the two methods is to examine how efficient each is in its primary scan. This can be accomplished by comparing the scan Θ for search technique:

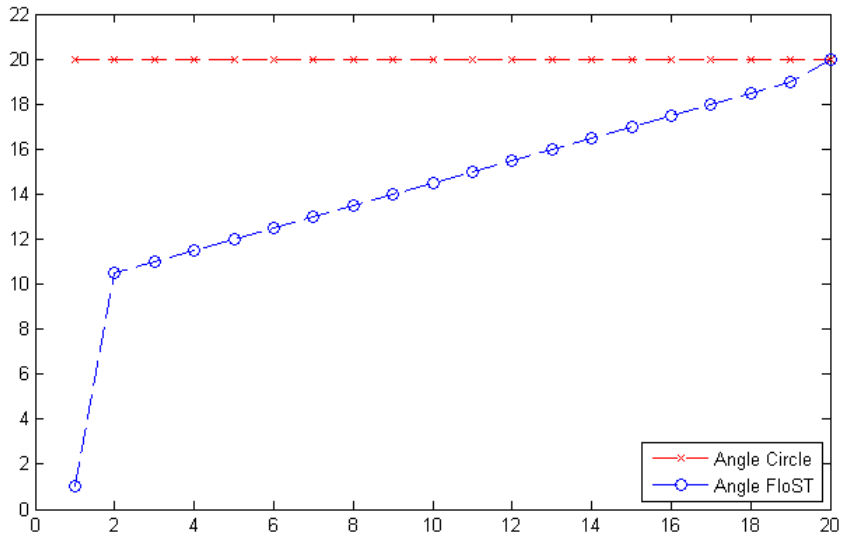

Figure 5.2 - 11, Comparison of FLoST vs Maximum Turn Rate Search

For TR = 20° the FLoST algorithm will begin at a significant disadvantage as it tries to unlearn previous behavior with the first measurement being off by a factor of 20. Very quickly, however, the FLoST adjusts the angle of search based upon the last recorded change in direction enabling it to rapidly approximate the increased difference between the directions of "Duck" and "duckling" as demonstrated in Figure 5.2 - 11. Problems of major directional change are not limited to FLoST; extreme maneuvers cause difficulty in other fuzzy tracking algorithms as well [Chan 95], [Cheng 01].

The second boundary case is when the "Duck" moves in a widening spiral.

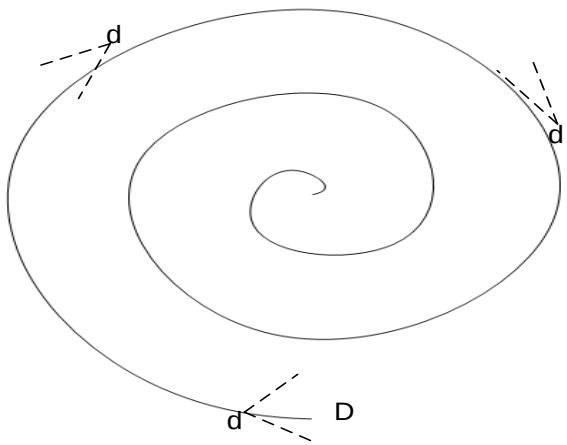

Figure 5.2 - 12, Ducklings follow Duck in spiral.

The spiral motion starts as extreme as the circle but gradually reduces the turn rate as it moves outward. In this example, the spiral starts out with an initial turn of 20° and loses a degree every two samples. It does not take long for the FLoST algorithm to catch up in this case as indicated by the graph in Figure 5.2 - 13 comparing the search areas:

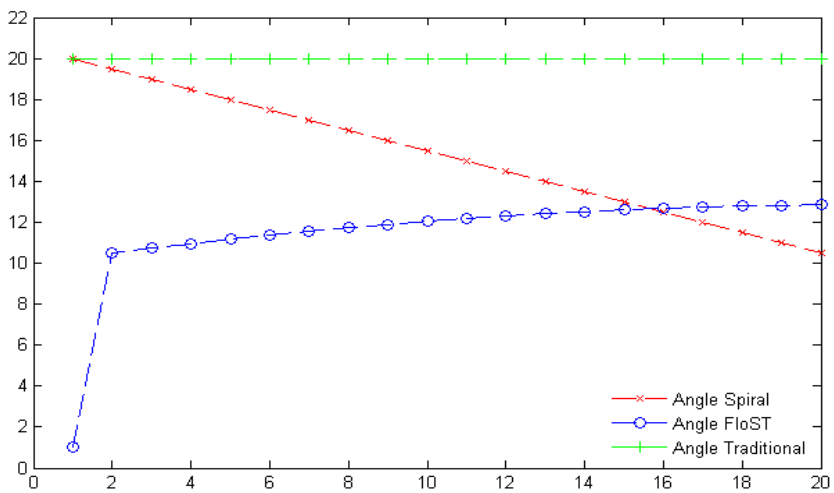

Figure 5.2 - 13, Comparison of FLoST vs Maximum Turn Rate Search in a spiral.

As with the tight circle, the "duckling" starts off at a significant disadvantage. However, the FLoST search angle quickly "catches up". It surpasses the efficiency of a

brute-force MTR calculation after a series of iterations allows it to expand and adapt, then narrow its search to accommodate the slowly degrading turn of the spiral.

A third boundary case is the weave. Like the circle, the weave utilizes maximum turn rate MTR, but in alternating directions.

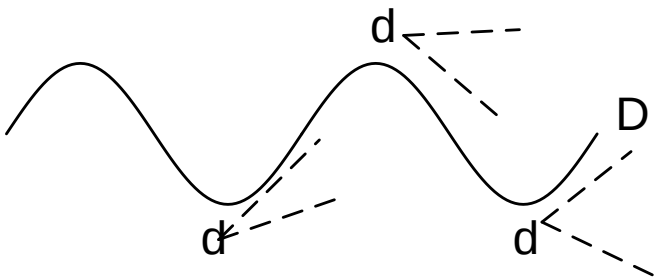

Figure 5.2 - 14, Ducklings follow Duck in weave pattern.

The weave attempts to perform the same extreme maneuver as the circle, although inertia in one direction will hinder its ability to exploit the full MTR in the other direction.

With an initial turn of 20° and subsequent weaves of 19°, FLoST quickly adapts to create a useful Θ as indicated by the graph in Figure 5.2 - 15.

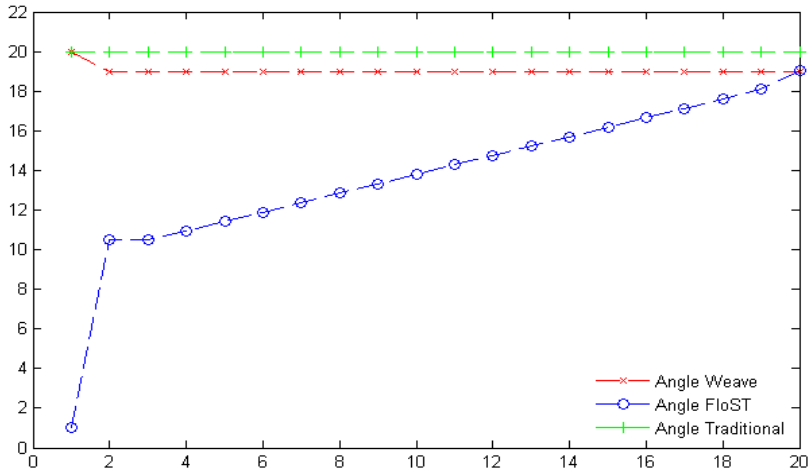

Figure 5.2 - 15, Comparison of FLoST vs Maximum Turn Rate Search in a weave

Despite some initial trouble at boundary extremes, FLoST does very well under less extreme, more “normal” operating conditions.

It is reasonable to assume a human operator will not deliberately attempt to evade a “duckling” or drive in an extreme circle, weave or spiral for any significant length of time. “Normal” operation then consists of relatively gentle turns at higher speeds or extreme turns at low speeds followed by sequences of relatively straight paths. Under these conditions, the FLoST “duckling” expands or narrows its search angle to compensate, relative to the fuzzy zones (slow, normal, and fast) determined during operation. Using the fuzzy zones further optimizes the “normal” case since larger degree turns are more safely accomplished at slower speeds, and is reflected in the autonomous “duckling’s” knowledge base. The “duckling” then knows that the slower speed has a greater incidence of wide turns and will adjust its search parameters accordingly.

In the original scenario of the chemicals that need to be moved to a safe location, workers could load the dangerous chemicals and be well away from harm while the “Duck” calls the “ducklings” to marshal. The “Duck” driver can navigate a complex path at a safe speed knowing the automated “ducklings” will mimic the course and speed very precisely. Once at an area where they can be unloaded, the “ducklings” are dispersed automatically to safe areas and processed as needed.

5.3 CONCLUSION

The results demonstrate the importance of applying additional heuristics to traditional algorithms. In the case of Local Search Algorithms, the advantage of Descending Deviation Optimizations when applied to Stochastic Hill Climbing and Simulated Annealing in the production line scheduling problem of factory automation was significant. Compared with the traditional approaches, DDO-Optimized versions were not only more successful overall in finding a solution, but in the case of DDO-SA were less likely to engage in fruitless searches due to a lengthy series of bad choices. This resulted in a significant net time reduction and large net decrease in computational resources required. DDO-SA also outperformed all other Local Search Algorithms tested, including those heavily dependent upon memory resources and “luck”.

In the case of the autonomous vehicle, notions of an unmanned successful tracking system usually involve a complex array of devices and software and the assumption that no human direction is available. However, as this thesis demonstrates, a simpler combination of human intelligence and machine algorithm could prove a worthwhile alternative. The performance of FLoST, the algorithm presented in this paper, is discussed for three boundary cases, “Duck” moving in circle, “Duck” moving in spiral and “Duck” weaving back and forth, in which the FLoST algorithm quickly and successfully adjusted the search parameters to compensate.

Chapter 6

FINAL CONCLUSIONS AND FUTURE WORK

The preceding chapters presented methods and algorithms to improve the effectiveness of Decision Trees, Local Search Algorithms and Intelligent Controllers. Taken together or separately, these techniques constitute very powerful systems with extensive capabilities. However, the results shown demonstrate that these capabilities can still be greatly improved through application of the techniques detailed in this thesis. Using Contextual Fuzzy Type-2 Hierarchies for Decision Trees, CoFuH-DT, combined with Contextual Derivation From Decision Trees, CoT-DT, Decision Trees were made smaller by many orders of magnitude; simpler, faster and more easily described. Rule generation moved from a cumbersome sequence of If-Then statements to a much cleaner polymorphic construction. Descending Deviation Optimizations applied to Local Search Algorithms reduced failure rates by up to 99%. Finally, using FLoST resulted in an autonomous vehicle both relatively inexpensive but still quite effective.

Future work needs to be done in each of the areas. Artificial Neural Networks, (ANNs), were shown in Chapter 2 to be a well-suited for classification. In Chapter 4, once such ANN, the Error Back Propagation ANN or EBP-ANN, was used to generate significant contexts for a Decision Tree. However ANNs are very diverse with a wide range of capabilities. The EBP-ANN used to generate contexts for CoT-DT is only one of a number of possibilities. Future work involving the effectiveness of other ANN implementations, such as Kohonen Self-Organizing Maps, needs to be explored. In

addition, other classifiers such as Support Vector Machines and Bayesian Networks also show promise for context generation.

While the Decision Tree was the primary focus of this thesis, both CoFuH and CoT have potential for generalization across a wide range of Advanced Data Mining Techniques, particularly classifiers, such as K-Means, Bayesian, Fuzzy c-Means and others. Generalizing the CoFuH and CoT algorithms to provide a more complete and generic framework for other ADMTs is a goal.

For Local Search Algorithms, future work needs to be done to apply the DDO method to other LSAs to see what improvements are possible elsewhere. It appears possible that any of the traditional LSAs which utilize some randomization, for example, Genetic Mutation, can be improved. Also to be tested is how DDO-Simulated Annealing will perform in place of traditional Simulated Annealing for other classes of NP-problems such as neural network optimizations, adversarial game play, intelligent control and chip layout, and whether results will be comparable to those achieved in the factory scheduling problem.

Further work also is necessary to improve FLoST prediction and accuracy. This may be possible by incorporating a neural network based approach. Extending the FLoST to allow “ducklings” to properly couple and decouple from a train, sort themselves out and avoid conflicts with other “ducklings” would greatly improve the applicability and overall usefulness of FLoST. Finally there need to be processes to allow the “Duck” and “duckling” to respond and reestablish contact in the event line-of-sight tracking fails.

REFERENCES

[Liu 06] D.K. Liu, X. Wu, A.K. Kulatunga, G. Dissanayake, *Motion Coordination of Multiple Autonomous Vehicles in Dynamic and Strictly Constrained Environments*, IEEE Conference on Cybernetics and Intelligent Systems, June 2006.

[Wall 02] R.W. Wall, J. Bennett; G. Eis, *Creating a Low-Cost Autonomous Vehicle*, Annual Conference of the IEEE Industrial Electronics Society, Nov. 2002.

[Omura 99] Y. Omura, S. Funabiki, T. Tanaka, *A Monocular Vision-Based Position Sensor Using Neural Networks for Automated Vehicle Following*, IEEE 1999 International Conference on Power Electronics and Drive Systems, PEDS'99, July 1999.

[Chang 91] K.S. Chang, W. Li, P. Devlin, A. Shaikhbahai, P. Varaiya, J.K. Hedrick, D. McMahon, V. Narendran, D. Swaroop, J. Olds, *Experimentation With a Vehicle Platoon Control System*, Vehicle Navigation and Information Systems Conference, 1991, Oct. 1991.

[Leigh 97] C.J. Leigh-Lancaster, B. Shirinzadeh, Y.L. Koh, *Development of a Laser Tracking System*, Fourth Annual Conference on Mechatronics and Machine Vision in Practice, 1997.

[Michaud 06] F. Michaud, P. Lepage, P. Frenette, D. Létourneau, N. Gaubert, *Coordinated Maneuvering of Automated Vehicles in Platoons*, IEEE Transactions On Intelligent Transportation Systems, December 2006.

[Wu 05] S. Wu, G. Zeng, H. Chiang, T. Lee, *Application of Optimal Fuzzy Tracking Control for Automotive Driving*, 2005 IEEE International Conference on Man and Cybernetics Systems Oct. 2005.

[Hanss 05] M. Hanss, M., *Applied Fuzzy Arithmetic*, New York: Springer-Verlag, 2005.

[Kato 02] S. Kato, S. Tsugawa, K. Tokuda, T. Matsui, H. Fujii, *Vehicle Control Algorithms for Cooperative Driving With Automated Vehicles and Intervehicle Communications*, IEEE Transactions on Intelligent Transportation Systems, September 2002.

[Wu 00] S. Wu, C. Lin, *Continuous Time-Invariant Optimal Fuzzy Tracker Design Based on Local Concept Approach*, The Ninth IEEE International Conference on Fuzzy Systems, May 2000.

[Chan 95] K.C. Chan, V. Lee, H. Leung, *Robust Target Tracking Using a Fuzzy Filter*, IEEE International Conference on Systems, Man and Cybernetics, Oct. 1995.

[Wu 02] S. Wu, C. Lin, *Global Optimal Fuzzy Tracker Design Based on Local Concept Approach*, IEEE Transactions on Fuzzy Systems, April 2002.

[Cucch 99] R. Cucchiara. M. Piccardi, A. Prati, N. Scarabottolo, *Real-time Detection of Moving Vehicles*, IEEE International Conference on Image Analysis and Processing, Sept. 1999.

[Cheng 01] C. C. Cheng, *Fuzzy Tracking Method With a Switching Grey Prediction for Mobile Robot*, IEEE International Conference on Fuzzy Systems, Dec. 2001.

[Hsu 07] H Chia-Hung Hsu, A. Liu, *A Flexible Architecture for Navigation Control of a Mobile Robot*, IEEE International Conference on Systems, Man and Cybernetics; May 2007.

[Crump 07] D. Crump, D.L. Livingston, *Robot Navigation Using Sensors With Fuzzy Characteristics*, IEEE SoutheastCon, March 2007.

[Liu 07] D. Liu, C. Lai, W. Lee, *A Hybrid of Sequential Rules and Collaborative Filtering for Product Recommendation*, IEEE International Conference on E-Commerce Technology and the IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, July 2007.

[Qiming 99] C. Qiming, U. Dayal, M. Hsu, *A Distributed OLAP Infrastructure for E-Commerce*, IFCIS International Conference on Cooperative Information Systems, 1999.

[Kita 02] M. Kitayama, R. Matsubara, Y. Izui, *Application of Data Mining to Customer Profile Analysis in the Power Electric Industry*, IEEE Power Engineering Society Winter Meeting, Jan. 2002.

[Fu 07] K. Fu, D. Geng, *Using the Data Mining Approach to Determine the Product Preferences of Target Customers*, IEEE International Conference on Service Systems and Service Management, June 2007.

[Zhao 06] J.Zhao, Z. Chang, *Neuro-Fuzzy Decision Tree by Fuzzy ID3 Algorithm and Its Application to Anti-Dumping Early-Warning System*, IEEE International Conference on Information Acquisition, April 2006.

[Li 03] F. Li, J. Sun, X. Wang, *Analysis on the Fuzzy Filter in Fuzzy Decision Trees*, IEEE International Conference on Machine Learning and Cybernetics, Nov. 2003.

[Li 02] N. Li, Y. Xiao-Wei, Z. Cheng-Qi, Z. Shi-Chao, *Product Hierarchy-Based Customer Profiles for Electronic Commerce Recommendation*, International Conference on Machine Learning and Cybernetics, Nov. 2002.

[Adom 01] G. Adomavicius, A. Tuzhilin, *Using Data Mining Methods to Build Customer Profiles*, Computer, Volume 34, Issue 2, Feb 2001.

[Kwan 02] I.S.Y. Kwan, *A Mental Cognitive Model of Web Semantic for E-Customer Profile*, IEEE International Workshop on Database and Expert Systems Applications, Sept. 2002.

[Dai 07] M. Dai, Y. Huang, *Data Mining Used in Rule Design for Active Database Systems*, IEEE International Conference on Fuzzy Systems and Knowledge Discovery, June 2007.

[Mendel 02] J. M. Mendel, R. I. Bob John, *Type-2 Fuzzy Sets Made Simple*, IEEE Transactions on Fuzzy Systems, April 2002.

[Lee 03] J. Lee, J. Sun, L. Yang, *A Fuzzy Matching Method of Fuzzy Decision Trees*, IEEE International Conference on Machine Learning and Cybernetics, Nov. 2003.

[Wang 01] X-Z. Wang, D.S. Yeung, E.C.C. Tsang, *A Comparitive Study on Heuristic Algorithms for Generating Fuzzy Decision Trees*, Transactions on Systems, Man and Cybernetics, April 2001.

[Haru 05] C. Haruechaiyasak, C. Tipnoe, S. Kongyoung, C. Damrongrat, N. Angkawattanawit, *A Dynamic Framework for Maintaining Customer Profiles in E-Commerce Recommender Systems*, IEEE International Conference on e-Technology, e-Commerce and e-Service, April 2005.

[Vlag 07] D. van de Vlag, A. Stein, *Incorporating Uncertainty via Hierarchical Classification Using Fuzzy Decision Trees*, Transactions on Geoscience and Remote Sensing, Jan. 2007.

[Cox 05] E. Cox, *Fuzzy Modeling and Genetic Algorithms for Data Mining and Exploration*, Morgan Kaufmann Publishers, 2005.

[Witten 05] I. H. Witten, E. Frank; Data Mining, *Practical Machine Learning Tools and Techniques*; Morgan Kaufmann Publishers, 2005.

[Robak 03] S. Robak, A. Pieczynski, *Employing Fuzzy Logic in Feature Diagrams to Model Variability in Software Product-Lines*, IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, April 2003.

[Ke 03] W. Ke, Z. Senqiang, J.M.S. Yeung, Y. Qiang, *Mining Customer Value: From Association Rules to Direct Marketing*, IEEE International Conference on Data Engineering, March 2003.

[Yutao 05] G. Yutao, J.P. Muller, *A Personalized Product Recommendation Algorithm Based on Preference and Intention Learning*, IEEE International Conference on E-Commerce Technology, July 2005.

[McCarty 08] K. McCarty, M. Manic, *Line-of-Sight Tracking Based Upon Modern Heuristics Approach*, IEEE International Conference on Industrial Electronics and Applications, June 2008.

[Shun 04] W. Sung-Shun, L. Mei-Ju, *Personalized Product Recommendation in E-Commerce*, IEEE International Conference on e-Technology, e-Commerce and e-Service, March 2004.

[Lee 99] K. Lee, K. Lee, J. Lee, *A Fuzzy Decision Tree Induction Method for Fuzzy Data, International Fuzzy Systems Conference Proceedings*, August 1999.

[McCarty 08a] K. McCarty, M. Manic, *Contextual Fuzzy Type-2 Hierarchies for Decision Trees (CoFuH-DT) – An Accelerated Data Mining Technique*, IEEE International Conference on Human System Interaction, May 2008.

[Han 06] J. Han, M. Kamber; *Data Mining Concepts and Techniques*, 2nd Ed, Morgan Kaufmann Publishers, 2006.

[Tatuzov 06] A. Tatuzov, N. Kurenkov, W. Lee, *Neural Network Data Clustering on the Basic of Scale Invariant Entropy*, IEEE International Conference on Neural Networks, July 2006.

[Yu 06] L. Yu, S. Wang, K.K. Lai, *An Integrated Data Preparation Scheme for Neural Network Data Analysis*, IEEE Transactions on Knowledge and Data Engineering, Feb. 2006.

[Sun 05] J. Sun, X. Wang; *An Initial Comparison on Noise Resisting Between Crisp and Fuzzy Decision Trees*; 4th International Conference on Machine Learning and Cybernetics, August 2005.

[Schalk 97] R. J. Schalkoff, *Artificial Neural Networks*, McGraw-Hill, 1997.

[Wang 06] X. Wang, Z. Shi, C. Wu, W. Wang, *An Improved Algorithm for Decision-Tree-Based SVM*, Proceedings of the 6th World Congress on Intelligent Control and Automation, June 2006.

[Russell 03] S. Russell, P. Norvig, *Artificial Intelligence – A Modern Approach, 2nd Ed.*, Prentice Hall 2003.

[Wang 06] Z.P. Wang, S. S. Ge, T. H. Lee, X. C. Lai, *Adaptive Smart Neural Network Tracking Control of Wheeled Mobile Robots*, IEEE International Conference on Control, Automation, Robotics and Vision, Dec. 2006.

[Hsu 02] C. Hsu, H. Huang, D. Schuschel, *The ANNIGMA-Wrapper Approach to Fast Feature Selection for Neural Nets*, IEEE International Conference on Systems, Man, and Cybernetics, April 2002.

[Fuering 99] T. Fuering, J. Buckley, Y. Hayashi, *Fuzzy Neural Nets Can Solve the Overfitting Problem;* IJCNN, July 1999.

[Wang 02] C. Wang, G. Chen, S. Ge, D. Hill, *Smart Neural Control of Pure-Feedback Systems*, International Conference on Neural Information Processing, Nov. 2002.

[Martin 07] I. Martinjak, M. Golub, *Comparison of Heuristic Algorithms for the N-Queens Problem*, International Conference on Information Technology Interfaces, June 2007.

[Sipser 06] M. Sipser, *Introduction to the Theory of Computation, 2nd Ed*, Thompson Course Technology 2006.

[Gao 07] M. Gao, J. Tian, *Path Planning for Mobile Robot Based on Improved Simulated Annealing Artificial Neural Network*, International Conference on Natural Computation, 2007.

[Mantawy 99] A.H. Mantawy, Y.L Abdel-Magid; *Integrating genetic algorithms, tabu search, and simulated annealing for the unit commitment problem*; IEEE Transactions on Power Systems, 1999.

[Zheng 06] S. Zheng, W. Shu, L. Gao, *Task Scheduling using Parallel Genetic Simulated Annealing Algorithm*, IEEE International Conference on Service Operations and Logistics, and Informatics, 2006.

[Kurbel 98] K. Kurbel, B. Schneider, K. Singh, *Solving Optimization Problems by Parallel Recombinative Simulated Annealing on a Parallel Computer-An Application to Standard Cell Placement in VLSI design*, IEEE Transactions on Systems, Man, and Cybernetics, 1998.

[Pasias 04] V. Pasias, D.A. Karras; R.C. Papademetriou, *Traffic Engineering in Multi-Service Networks Comparing Genetic and Simulated Annealing Optimization Techniques*, IEEE International Joint Conference on Neural Networks, July 2004.

[Kliewer 00] G. Kliewer, S. Tschoke, *A General Parallel Simulated Annealing Library and Its Application in Airline Industry*, International Parallel and Distributed Processing Symposium, 2000.

[Mendon 97] P.R.S Mendonca, L.P Caloba; *New Simulated Annealing Algorithms*, IEEE International Symposium on Circuits and Systems, 1997.

[Liu 07] B. Liu, L. Wang, Y.H. Jin; *An Effective PSO-Based Memetic Algorithm for Flow Shop Scheduling*; IEEE Transactions on Systems, Man, and Cybernetics, February 2007.

[Guimar 07] F.G. Guimaraes, F. Campelo, H. Igarashi, D.A. Lowther, J.A. Ramirez, *Optimization of Cost Functions Using Evolutionary Algorithms With Local Learning and Local Search*, IEEE Transactions on Magnetics, 2007.

[Li 06] S. Li, Y. Li, Y. Liu, *Effects of Process Planning Upon Production Scheduling Under Concurrent Environment*, World Congress on Intelligent Control and Automation, 2006.

[Chase 06] R.B. Chase, F.R. Jacobs, N.J. Aquilano, *Operations Management for Competitive Advantage, 11th Ed*, McGraw-Hill, 2006.

[Ishi 99] H. Ishibuchi, T. Murata, *Local Search Procedures in a Multi-Objective Genetic Local Search Algorithm for Scheduling Problems*, International Conference on Systems, Man, and Cybernetics, 1999.

[Ishi 03] H. Ishibuchi, T. Yoshida, T. Murata, *Balance Between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling*, IEEE Transactions on Evolutionary Computation, 2003.

[Garcia 08] M.E. Garcia, S. Valero, E. Argente, A. Giret, V. Julian, *A FAST Method to Achieve Flexible Production Programming System*, IEEE Transactions on Systems, Man, and Cybernetics, 2008.

[Tang 05] Z. Tang, J. MacLennan; *Data Mining with SQL Server 2005*; Wiley Publishing, 2005, pp 2-20

[Cox 94] E. Cox; *The Fuzzy Systems Handbook*; Academic Press, 1994 pp 1-20

[Zurada 92] J. M. Zurada, *Introduction to Artificial Neural Systems*, West Publishing Company, 1992

[Harinath 06] S. Harinath, S.R. Quinn, *Professional SQL Server Analysis Services 2005 with MDX*, Wiley Publishing, 2006

[Cox 95] E. Cox, *Fuzzy Logic for Business and Industry*, Charles River Media, 1995

[Ben-Gan 06] I. Ben-Gan, D. Sarka, R. Wolter, *Inside Microsoft SQL Server 2005: T-SQL Programming*, Microsoft Press, 2006

[Morgan 08] S. Morgan, *Programming Microsoft Robotics Studio*, Microsoft Press, 2008

APPENDIX A

LOCAL SEARCH ALGORITHMS PSEUDO-CODE

A number of local search algorithms (LSAs) were studied and programmed for simulation in the C# programming language. What follows is pseudo-code for the individual algorithms.

Terms:

Schedule – The current schedule

ScheduleConfig – A possible alternative schedule (could be NULL) derived by a slight modification of existing schedule

schedule_state – A value indicating whether Schedule is in goal (success) state, a failure state or intermediate state

Candidates – An array of possible ScheduleConfigs

Element – Competing elements that make up a Schedule

Option – One particular element configuration

Track – Flow of a given option

bounce_condition – Occurs when a LSA has exhausted all search possibilities and is "trapped" in a condition that is not a goal state

bounce – A randomization of the current state to another state

Conflict – Condition where two elements require the same resource at the same time

LSA#1 - Hill Climbing

The Hill Climbing algorithm is a greedy algorithm which tries to move from a state to the lowest possible conflict state. The pseudo-code outline of this approach is as follows:

```
Hill Climbing(Schedule) returns schedule_state
Loop
    schedule_state = AssessSchedule(Schedule)
    if schedule_state = success/failure
        return schedule_state
    else
        ChangeSchedule(Schedule, GetNextHillClimbConfig(Schedule))
        UpdateConflicts(Schedule)
End Loop

GetNextHillClimbConfig(Schedule) returns ScheduleConfig
    for each ScheduleConfig in Schedule
        if Conflicts(ScheduleConfig) < Conflicts(Schedule) AND
    Conflicts(ScheduleConfig) <= LowestConflict
                AddToCandidates(ScheduleConfig)
                LowestConflict = Conflicts(ScheduleConfig)
        for each ScheduleConfig in Candidates
                if Conflicts(ScheduleConfig) > LowestConflict
                        Remove(ScheduleConfig)
   return random(ScheduleConfig) from Candidates
```

Strength of this approach is simple implementation and minimal resources required. Weakness is that it has a great tendency to be attracted to and trapped in a local maximum.

LSA #2 - Stochastic Hill Climbing

Stochastic Hill Climbing is a variant of the hill climb in which not the steepest ascent is picked but any ascent is eligible, dictated by a probability assigned to each option. The probability is dependent to some degree upon the steepness of the ascent. The pseudo-code outline of this approach is as follows:

```
Stochastic Hill Climbing(Schedule) returns schedule_state
Loop
    schedule_state = AssessSchedule(Schedule)
    if schedule_state = success/failure
        return schedule_state
    else
        ChangeSchedule(Schedule, GetLowerScheduleConfig(Schedule))
        UpdateConflicts(Schedule)
End Loop

GetLowerScheduleConfig(Schedule) returns ScheduleConfig
    for each ScheduleConfig in Schedule
        if Conflicts(ScheduleConfig) < Conflicts(Schedule)
                AddToCandidates(ScheduleConfig) * Conflicts(Schedule) –
            Conflicts(ScheduleConfig)
   return random(ScheduleConfig) from Candidates
```

Strength of this approach is that it does allow an algorithm to escape a local maximum. Weakness is that it still has a great tendency to be attracted to and trapped in a local maximum.

LSA #3 - Stochastic Hill Climb using Descending Deviation Optimizations

DDO-Stochastic Hill Climb works like the standard stochastic hill climb until it gets "stuck"; at which point it "bounces" the solution to a nearby, less optimal state and again applies the standard stochastic hill climb. The "bounces" are gradually lessened in height until they disappear; at which time if a global solution is not reached, the strategy fails. Pseudo-code implementation is as follows:

```
DD-Stochastic Hill Climbing(Schedule) returns schedule_state
Loop
     schedule_state = AssessSchedule(Schedule)
     if schedule_state = success/failure
          return schedule_state
     if neither state
          ChangeTile(Schedule, GetNextHillClimbTile(Schedule))
          UpdateConflicts(Schedule)
      If bounce_condition(Schedule)
          Bounce(Schedule)
End Loop

GetLowerScheduleConfig(Schedule) returns ScheduleConfig
     for each ScheduleConfig in Schedule
          if Conflicts(ScheduleConfig) < Conflicts(Schedule)
                    AddToCandidates(ScheduleConfig) * Conflicts(Schedule) –
               Conflicts(ScheduleConfig)
   return random(ScheduleConfig) from Candidates

Bounce(Schedule, height)
          ScheduleConfig = PickScheduleConfig(Random(Schedule.Option),
Random(Schedule.Track), height)
          MoveScheduleToScheduleConfig(ScheduleConfig)


PickScheduleConfig(Track, Option, height)
     for each ScheduleConfig in Schedule.Option
          if Conflicts(ScheduleConfig) <= height
                    AddToCandidates(ScheduleConfig)
   return random(ScheduleConfig) from Candidates
```

Strengths of this approach is that it will never do worse (or take longer) that the standard stochastic hill climb, but has the ability to escape local maxima. The weakness of this approach is when a global maximum is not nearby, the algorithm may not be able to “bounce” far enough to find it and hence will still fail, albeit with more effort.

LSA #4 - Random Restart Hill Climbing

The Random Restart is another variant on the Hill Climb, except in this case, once a local maximum is discovered, the process does a random scramble and tries again. The assumption is that a Random Restart will eventually produce an initial configuration that will allow it to reach a goal state. While the probably of success is essentially one, meaning that if given enough retries the Random Restart will eventually randomize its way to a solution. The implementation presented in this thesis limits the number of tries to 100 restarts, so as to not go on for too long. Pseudo-code implementation is as follows:

```
Random Restart Hill Climbing(Schedule) returns schedule_state
Loop
    schedule_state = AssessSchedule(Schedule)
    if schedule_state = success/failure
        return schedule_state
    if neither state
        ChangeScheduleConfig(Schedule,
GetNextHillClimbScheduleConfig(Schedule))
        UpdateConflicts(Schedule)
     If restart_condition(Schedule)
        Restart(Schedule)
End Loop

GetNextHillClimbScheduleConfig(Schedule) returns ScheduleConfig
    for each ScheduleConfig in Schedule
        if Conflicts(ScheduleConfig) < Conflicts(Schedule) AND
    Conflicts(ScheduleConfig) <= LowestConflict
                AddToCandidates(ScheduleConfig)
                LowestConflict = Conflicts(ScheduleConfig)
        for each tile in Candidates
                if Conflicts(tile) > LowestConflict
                        Remove(ScheduleConfig)
   return random(ScheduleConfig) from Candidates

Restart(Schedule)
        Element = Random(Schedule.Elements)
        ScheduleConfig = Random(Element.Track)
```

MoveElementToScheduleConfig(ScheduleConfig)

Strengths of this approach is that the algorithm should theoretically be able to find any solution if it exists. The weakness is that if global maxima are sparse and local maxima are dense, the algorithm could take a very long time to find a solution.

LSA #5 - Simulated Anneling (SA)

Simulated Annealing introduces a pseudo-random selection method in that the best choice is not necessarily used but it is not random either. The algorithm allows a large range, or nearly random, set of choices early on, getting progressively more restrictive in favor of better choices as the algorithm iterates. Eventually, the algorithm will work in much the same fashion as the Hill Climb, but since the range of options is greater in the beginning, it will have theoretically explored more maxima and is correspondingly more likely to find a global one. Psuedo-code implementation is as follows:

```
Simulated Annealing (Schedule) returns schedule_state
Loop
    schedule_state = AssessSchedule(Schedule)
    if schedule_state = success/failure
        return schedule_state
    if neither state
        ChangeScheduleConfig(Schedule, GetNextSAScheduleConfig(Schedule))
        UpdateConflicts(Schedule)
End Loop

GetNextSAScheduleConfig(Schedule) returns ScheduleConfig
    ScheduleConfig = RandomScheduleConfig(Schedule)
    if Conflicts(ScheduleConfig) < Conflicts(Schedule) OR
Annealing(Conflicts(ScheduleConfig)) <= Probability
        return ScheduleConfig
     else
        return null
```

The strengths of this approach is that it explores a pretty wide range of possibilities and does a better job of finding global maxima than the other Hill-Climb variants (except for the Random Restart). The weakness is that this process requires much more processing power and time.

LSA #6 - Genetic Mutation (GM)

Genetic Mutation attempts to emulate the characteristics of random selection. The first step is to create a "population" of candidates and select a small sample from that population; in Chapter 5, the sample size was two. The "fittest" of the sample survives, while the remaining members "mutate" based upon a probability and "cross-breed" with the best member, exchanging some of their attributes with those of the best member. Then the sample is returned to the population and a new sample is drawn. Pseudo-code implementation is as follows:

```
GeneticMutation (Schedule) returns schedule_state
        ScheduleCollection = random collection of Schedules + Schedule
Loop
    schedule_state = AssessSchedule(Schedule)
    if schedule_state = success/failure
        return schedule_state
    if neither state
        Sample = Sample(ScheduleCollection)  // pull a pair of Schedules
        NewSchedule = Fittest(Sample)        // get the Schedule with the least conflicts
        Crossover(NewSchedule, Sample – NewSchedule)
        Mutate(Sample – NewSchedule)
        UpdateConflicts(NewSchedule)
End Loop

Crossover(Schedule, ScheduleCollection)
    for each SampleSchedule in ScheduleCollection
        replace random Options in SampleSchedule with Options from Schedule

Mutate(ScheduleCollectioin)
        for each SampleSchedule in ScheduleCollection
                If random < threshold
                        Move random Element to random Track
```

The strengths of this approach is that it explores many different states and doesn't generally focus on any given local maxima. The weakness is that it performs a

lot of useless mutations, often moving around local maxima for a number of iterations instead of spending fewer cycles on a more direct approach. In many cases, the mutation also put potential schedules in a worse state. A DDO version might be to limit the range of mutations so that they produce only better schedules or, at least, no worse or marginally worse. This might serve to avoid a lot of effort that would otherwise be devoted to useless comparisons later on.

LSA #7 - Min Conflicts

The Min Conflicts approach takes apart the space option by option seeking the minimum for each. This allows for a multi-pronged approach to greedy search that appears to be very effective in this case. In this implementation, the directional sweep of rows alternated from left-to-right and right-to-left, so as to not bias the heuristic to any given direction. Pseudo-code implementation is as follows:

```
MinConflicts(Schedule) returns schedule_state
Loop
    schedule_state = AssessSchedule(Schedule)
    if schedule_state = success/failure
        return schedule_state
    if neither state
        for each column in Schedule
        MoveElementToBestTrack(ScheduleConfig)
        UpdateConflicts(Schedule)
End Loop

MoveElementToBestTrack(ScheduleConfig)
    for each Track in Option
        if Conflicts(ScheduleConfig(Track, Option)) < Conflicts(Element)
                Element.ScheduleConfig = ScheduleConfig(Track, Option)
```

The strengths of this approach are that is generally finds a solution in only a few iterations and is fairly easy to implement. The weakness is that may be susceptible to looping conditions as changes propagate and undo/redo various configurations that are interdependent.

LSA #8 - Tabu Search

Tabu Search combines elements of the Hill Climb with a local "memory" of previous configurations. Instead of necessarily picking the best solution, it simply picks from among as good or better solutions. The memory prevents the algorithm retrying paths that have already proven fruitless. Pseudo-code implementation is as follows:

```
TabuSearch (Schedule) returns schedule_state
Loop
    schedule_state = AssessSchedule(Schedule)
    if schedule_state = success/failure
        return schedule_state
    if neither state
        conflicts = Conflicts(Schedule)
        ScheduleConfig = GetALowerScheduleConfig(Schedule, conflicts)
        if ScheduleConfig = null return failure
        else AddToAlreadyUsed(ScheduleConfig)
        UpdateConflicts(Schedule)
End Loop

GetLowerScheduleConfig(Schedule, conflicts) returns ScheduleConfig
    For each ScheduleConfig in Schedule
        If Conflicts(ScheduleConfig) <= conflicts
        AddToCandidates(ScheduleConfig)

    For each ScheduleConfig in Candidates
        If AlreadyUsed(ScheduleConfig)
            Remove(ScheduleConfig)

   return random(ScheduleConfig) from Candidates
```

The strengths of this approach is that it explores many potential paths. The weakness is that it requires significant memory for large search problems (potentially configuration size times # bits to represent configuration times possible configurations) so may not be practical. One alternative is to limit the number of prior representations

stored in memory but this adds the additional complication that looping conditions can still be encountered if the loop exceeds the size of the tabu array.

LSA #9 – Simulated Annealing Using Descending Deviation Optimizations

DDO-Simulated Annealing is based upon traditional Simulated Annealing (SA) but attempts to correct some of the inherent problems associated with traditional SA. First, traditional SA can, through a series of bad randomizations, move to a progressively worse state or "wander" too far away off a gradient path to recover. DDO prevents this by imposing an artificial limit on the algorithm's ability to wander. A second problem with traditional SA is that it may pick a direction that is so bad that it fails the threshold test, resulting in a "no operation" (no-op). DDO-SA forces a selection from only those actions that are valid so a no-op doesn't occur. The third problem is that traditional SA can randomize to the goal state but not select it due to the number of possible states, of which the goal state is only a member. DDO-SA does a scan of all possible states looking specifically for the goal state. If the goal state is found the algorithm choose that and completes successfully. Pseudo-code implementation is as follows:

```
Simulated Annealing (Schedule) returns schedule_state
Loop
    schedule_state = AssessSchedule(Schedule)
    if schedule_state = success/failure
        return schedule_state
    if neither state
        ChangeScheduleConfig(Schedule, GetNextSAScheduleConfig(Schedule))
        UpdateConflicts(Schedule)
End Loop

GetNextSAScheduleConfig(Schedule) returns ScheduleConfig
    ScheduleConfig = RandomScheduleConfig(Schedule)
    if Conflicts(ScheduleConfig) < Conflicts(Schedule) OR
Annealing(Conflicts(ScheduleConfig)) <= Probability
        return ScheduleConfig
     else
        return null
```

```
RandomScheduleConfig(Schedule) returns ScheduleConfig
    For each ScheduleConfig in Schedule
        if ScheduleConfig = goal_state
          return ScheduleConfig

        if Conflicts(ScheduleConfig) < Conflicts(Schedule) OR
        (Annealing(Conflicts(ScheduleConfig)) <= Probability AND
        Conflicts(ScheduleConfig) <= DDO threshold)
        AddToCandidates(ScheduleConfig)
  Loop
  return random(ScheduleConfig) from Candidates
```

This approach carries with it all the positives of tradition SA, with one exception. In limiting the algorithm's ability to wander, finding a goal state in a state space that is sparsely populated may prove more difficult. One other limitation of this approach is that each iteration requires a complete scan of all possible current configurations. This could be somewhat expensive, SA +O(n), where n is # of possibilities. However, when compared to the original approach it might also be faster as fewer, relatively more expensive, iterations are needed to reach a solution. The scan could be optimized further by added elements to Tabu Search to prevent rescanning.

APPENDIX B

DATABASE, DATA WAREHOUSE AND DATA MINING TEST SOFTWARE

Microsoft's SQL Server 2005 Developer Edition provided the platform for the relational database along with the data warehouse, data cube, and the data mining test software. The test database was the Adventure Works sample database also provided by Microsoft. The database consists of sales records from a fictitious bicycle shop called AdventureWorks and contains data pertaining to customers, products and product sales, employees, vendors and other related information. Sales take place both in-store and on the web and are tracked separately. The schema is too big to show in its entirety but a partial schema is shown in Figure B - 1.

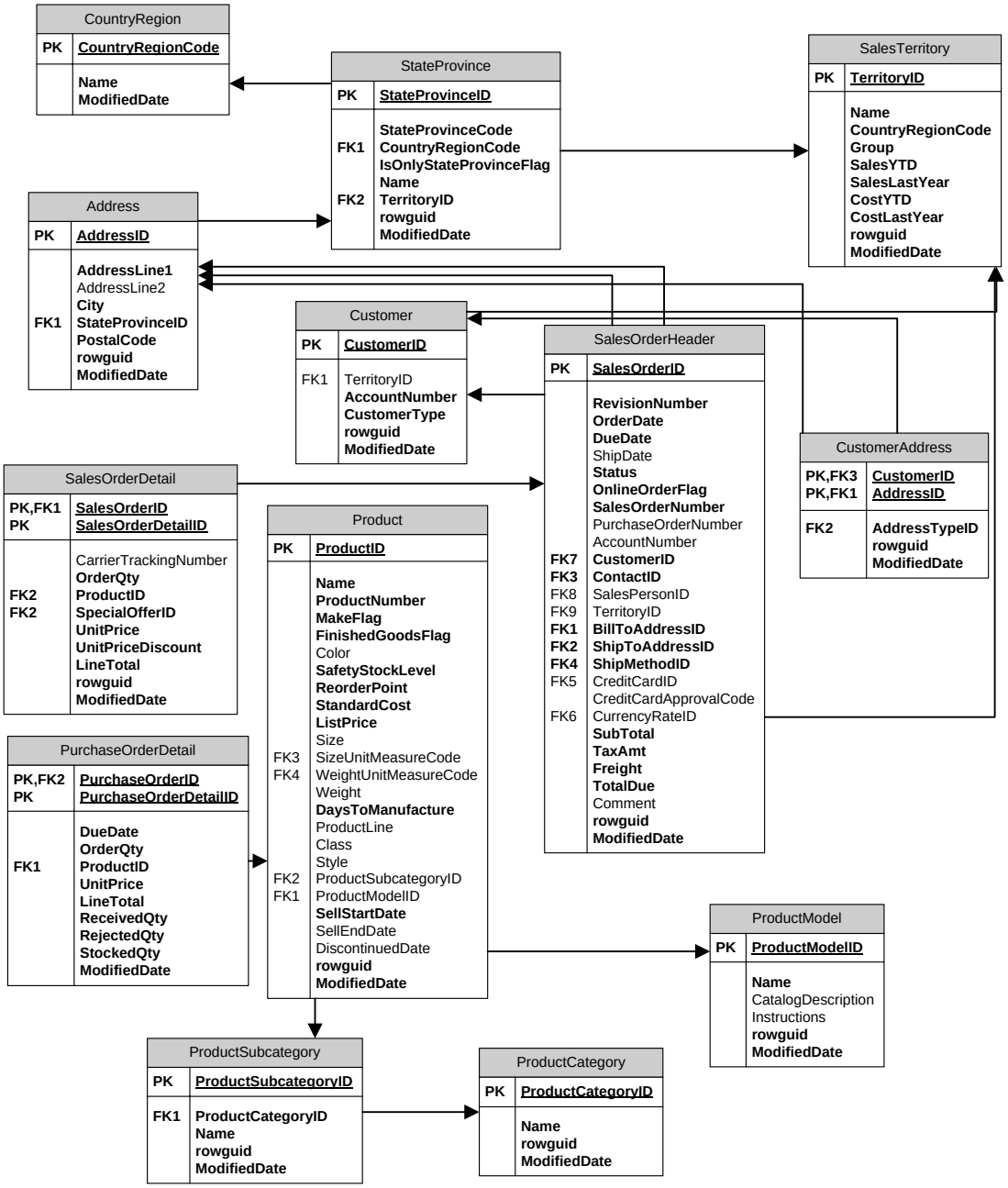

Figure B - 1, Partial Schema of AdventureWorks sample database

Using SQL Server 2005 Analysis Services, the relational database was processed into a data cube to speed up certain queries and also for use as the staging area for data mining. Data mining consisted of applying provided Bayes, Decision Trees, Clustering and Neural Network algorithms to the data cube. The Decision Tree

was generated using a proprietary process combing CART techniques with C4.5. The Artificial Neural Network used is a Back Propagation Network with one hidden layer. Mining models were then generated for each algorithm. The resulting Decision Tree (totaling 184 nodes) mining model implementation is shown in Figure B-2 and the Bayes mining model implementation is shown in Figure B - 3. The Neural Network mining model does not have a graphical representation. Instead it lists a set of variables with favorability ratings. These ratings determine how closely (or loosely) a given condition is associated with a result, demonstrated in Figure B - 4.

Figure B - 2, The Decision Tree mining model

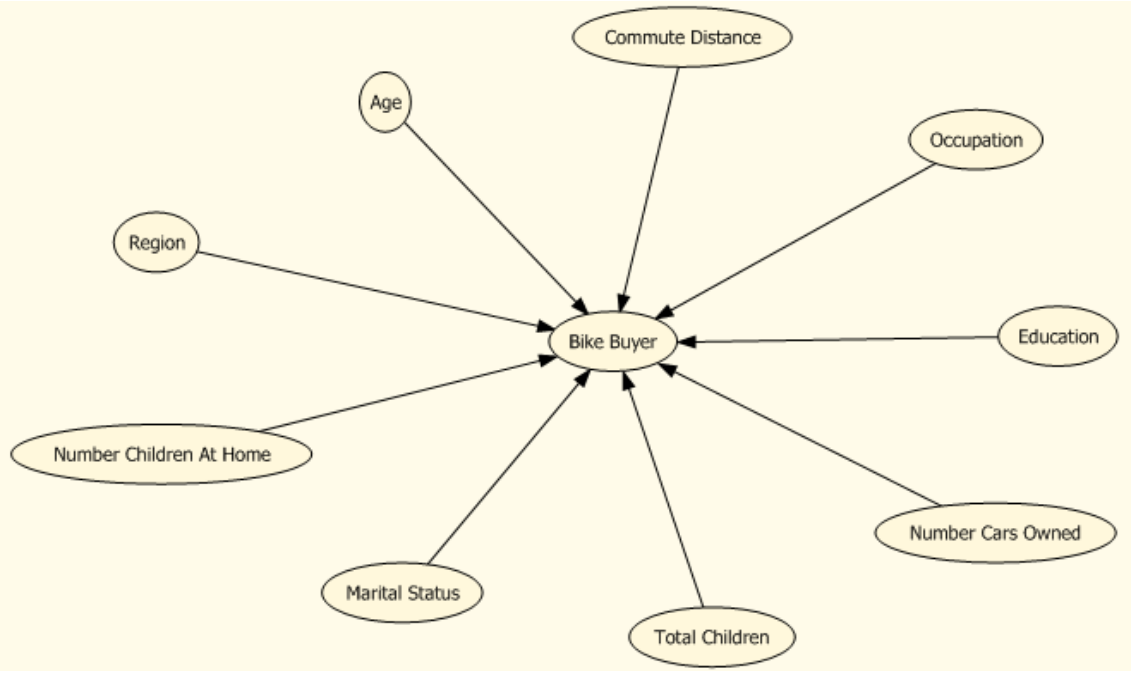

Figure B - 3, The Bayes mining model

Variables:

Attribute	Value	Favors 0	Favors 1
Age	67 - 70		
Commute Distance	10+ Miles		
Age	70 - 86		
Region	Pacific		
Number Cars Owned	4		
Number Children At Home	3		
Total Children	5		
Occupation	Manual		
Number Children At Home	4		
Number Cars Owned	0		
Age	32 - 37		
Education	Partial High School		
Yearly Income	79081.658 - 154160.683		
Commute Distance	5-10 Miles		
Age	49 - 55		
Age	61 - 67		
Region	North America		
Number Cars Owned	3		
Total Children	4		
Yearly Income	10000.000 - 35529.898		
Commute Distance	0-1 Miles		

Figure B - 4, The Neural Network mining model