

# Applications of Contextual Fuzzy Operators and Fuzzy Extensions for Polymorphism, Data Mining and Analysis

A Dissertation

Presented in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Kevin S. McCarty

December 2014

Major Professor: Milos Manic, Ph.D.

## AUTHORIZATION TO SUBMIT DISSERTATION

This dissertation of Kevin McCarty, submitted for the degree of Doctor of Philosophy with a major in Computer Science and titled “Applications of Contextual Fuzzy Operators and Fuzzy Extensions for Polymorphism, Data Mining and Analysis,” 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, Ph.D.

Committee  
 Members: \_\_\_\_\_ Date \_\_\_\_\_  
 Robert Heckendorn, Ph.D.

\_\_\_\_\_ Date \_\_\_\_\_  
 Sergiu-Dan Stan, Ph.D.

\_\_\_\_\_ Date \_\_\_\_\_  
 Vijay Dialani, Ph.D.

Department  
 Administrator: \_\_\_\_\_ Date \_\_\_\_\_  
 Gregory Donohoe, Ph.D.

Discipline’s  
 College Dean: \_\_\_\_\_ Date \_\_\_\_\_  
 Larry Stauffer, Ph.D.

Final Approval and Acceptance

Dean of the College  
 of Graduate Studies: \_\_\_\_\_ Date \_\_\_\_\_  
 Jie Chen, Ph. D.

## ABSTRACT

System behaviors and processes are often influenced to a large degree by external conditions. As those conditions change, so must any corresponding behavioral response. Driving a car from point A to point B when the road is straight and smooth is a dramatically different operation than if the surface is a sheet of ice with twists and turns. Even more challenging is when both surfaces are part of the same problem; for example, when a car must navigate a patch of “black ice” on the highway. Significant changes in the nature of a problem, such as when a surface changes from asphalt to ice, are referred to as Situational Discontinuity Problems (SDPs) and present a distinct challenge to computer scientists and programmers.

First, behaviors which might be appropriate for past problem states are inappropriate for future states. Second, data discontinuities make it difficult to describe all-encompassing actions without a “frame of reference”. Even well-known problems such as simple noise and outliers pose problems for processes. Third is the general uncertainty surrounding SDPs in or near transition states where multiple approaches are equally desirable.

Traditional approaches to SDPs involve generalizing an existing algorithm. This has the undesired side-effect of greatly increasing complexity while also providing a diminishing ability to further extend the algorithm. Fuzzy Contexts, or Fuzzy Logic Type-C with Fuzzymorphism, is an architecture designed to allow process extensions to solve SDPs or other disparate problems while also minimizing complexity and diminishing returns.

This dissertation explores prior work and methods used to solve SDPs and their limitations. It then introduces the notion of Fuzzy Contexts and Fuzzymorphism in a novel software framework supported with database extensions, a database-driven and XML-based configuration language, combined with evolutionary and local search techniques. It further explores the use of the framework and the Type-C architecture in various processes from data mining to robotic control in order to improve upon existing techniques.

## ACKNOWLEDGEMENTS

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

Miles McQueen

Debbie McQueen

Alice Allen

and, of course, the members of my committee, whose time and input are greatly appreciated.

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 dissertation would even have been possible without his input, time and effort throughout the entire process.

## **DEDICATION**

This dissertation is dedicated to Andrea, my wife, and very best friend.

To Marie McCarty, who gave me life and so much more.

Finally to Dr. Richard McCarty, my model in all things.

## TABLE OF CONTENTS

<b>Authorization to Submit Dissertation</b> .....	ii
<b>Abstract</b> .....	iii
<b>Acknowledgements</b> .....	iv
<b>Dedication</b> .....	v
<b>Table Of Contents</b> .....	vi
<b>List of Figures</b> .....	xi
<b>List of Pseudo-code</b> .....	xvii
<b>List of Tables</b> .....	xviii
<b>Chapter 1: Introduction</b> .....	1
1.1: Motivation .....	1
1.2: Objective .....	3
1.3: Dissertation Structure and Contributions .....	4
<b>Chapter 2: Background</b> .....	6
2.1: Problems that could use a little context .....	6
2.2: Using Fuzzy Logic .....	9
2.2.1: Fuzzy Type 1 .....	9
2.2.2: Fuzzy Logic Type 2 .....	10
2.2.3: Nonstationary Fuzzy Sets .....	11
2.2.4: Polymorphic Fuzzy Signatures .....	12
2.2.5: Fuzzy Clustering .....	13
2.3: Other Techniques for Contextual Problem Solving .....	15
2.3.1: Local Search Techniques .....	15
2.3.2: Evolutionary Computation .....	16
2.3.3: Software Engineering and Architecture Issues .....	16

2.3.4: Neural Networks .....	22
2.3.5: Radial Basis Functions .....	23
2.3.6: Subsumption.....	25
2.3.7: Dynamic Programming .....	26
2.3.8: Hybrid Systems .....	27
<b>Chapter 3: Fuzzy Contexts</b> .....	<b>29</b>
3.1: The Need for Context .....	29
3.1.1: The Situational Discontinuity Problem .....	32
3.2: The Fuzzy Logic Foundation.....	34
3.3: Overview of Fuzzy Contexts in Detail .....	39
3.3.1: How Fuzzy Contexts extend Fuzzy Logic .....	43
3.3.2: What is Fuzzymorphism?.....	45
3.3.3: Advantages of Fuzzy Contexts and Fuzzymorphism.....	47
3.3.4: Is there a Packing Problem? .....	53
3.4: Implementation of Basic Fuzzy Framework and Architecture.....	55
3.4.1: Introduction .....	55
3.4.2: Need for a General Fuzzy Framework to Support Type-C .....	58
3.4.3: Framework Architecture .....	59
3.4.4: Structural Changes to Support Type-C .....	64
3.5: Guidelines for Creating Fuzzy Contexts .....	66
Step 1: Determine Default Context if Necessary .....	66
Step 2a: Define Center .....	66
Step 2b: Define Inner Shape.....	68
Step 2c: Define Outer Shape .....	68
Step 2d: Define Fuzzy Membership Function.....	68

Step 2e: Label the Context .....	70
<b>Chapter 4: Fuzzy Contexts Extensions and Enhancements</b> .....	<b>72</b>
4.1: Architectural Enhancements .....	72
4.1.1: Enhancement #1 – Adding Fuzzy Type-2 Support .....	72
4.1.2: Enhancement #2 – Adding Fuzzy Type-C Support .....	73
4.1.3: Enhancement #3 – Adding Relational Database Support .....	76
4.2: Functional Enhancements .....	81
4.2.1: Enhancement #4 – Adding Memetic-Based Optimization .....	81
4.2.2: Enhancement #5 – Adding Unsupervised Learning Algorithm to Discover New Fuzzy Contexts .....	87
4.3: Structural Enhancements .....	89
4.3.1: Enhancement #6 - Algorithm Definition Language .....	89
4.3.2: Enhancement #7 - The Wizard Tool .....	97
4.3.3: Enhancement #8 - Hierarchical Technique for Diverse Contextual Dynamic Programming and Optimization .....	102
4.4: Test Examples .....	110
4.4.1: Basic Framework Tests .....	110
4.4.2: Context Tests .....	115
4.4.3: Other Enhancement Tests .....	125
4.5: Chapter Summary and Comparative Analysis .....	143
4.5.1: TCF vs. Other Frameworks .....	143
4.5.2: TCF vs. Traditional Fuzzy Type-1, Type-2 and other approaches .....	143
4.5.3: How TCF reduces the problem of over-fitting in ANNs .....	145
<b>Chapter 5: Work in Progress</b> .....	<b>148</b>
<b>Chapter 6: Conclusion</b> .....	<b>150</b>



Summary of Contributions .....	150
Final Note .....	152
<b>References</b> .....	153
<b>Appendices</b> .....	163
Appendix A: Minor Contributions.....	163
Minor Contribution #1: Contextual Fuzzy Hierarchies for Decision Trees (CoFuH-DT) – An Accelerated Data Mining Technique .....	163
Minor Contribution #2: Contextual Derivation From Decision Trees (CoT-DT) Based on Advanced Data Mining Techniques and Intelligent Control.....	180
Minor Contribution #3: Applications of Heuristics to Local Search Algorithms .....	196
Appendix B: Fuzzy Logic Primer.....	207
B.1: Fuzzy Logic Type 1 .....	207
B.2: Fuzzy Logic Type 2 .....	229
B.3: Nonstationary Fuzzy Sets .....	240
B.4: Polymorphic Fuzzy Signatures .....	244
B.5: Hybrid Fuzzy Systems.....	245
B.6: Fuzzy Clustering.....	246
Appendix C: Local Search Techniques .....	251
Appendix D: Neural Networks .....	254
Appendix E: Evolutionary Computation .....	257
E.1: Introduction.....	257
E.2: Natural Selection.....	259
E.3: Recombination .....	260
E.4: Mutation.....	262
Appendix F: Relational Database Concepts .....	263

F.1: Introduction .....	263
F.2: The Relational Model.....	267
F.3: Codd’s Rules for Relational Databases.....	269
Appendix G: Advanced Data Mining Techniques.....	278
Appendix H: Local Search Algorithms Pseudo-Code .....	284
H.1: Introduction .....	284
H.2: LSA #1- Hill Climbing.....	284
H.3: LSA #2 - Stochastic Hill Climbing .....	286
H.4: LSA #3 - Stochastic Hill Climb using Descending Deviation Optimizations .....	288
H.5: LSA #4 - Random Restart Hill Climbing.....	290
H.6: LSA #5 - Simulated Annealing (SA) .....	292
H.7: LSA #6 - Genetic Mutation (GM).....	294
H.8: LSA #7 - Min Conflicts.....	295
H.9: LSA #8 - Tabu Search .....	297
H.10: LSA #9 - Simulated Annealing Using Descending Deviation Optimizations .....	299
H.11: LSA #10 – Memetic Mutation.....	302
H.12: LSA #11 – Genetic Mutation Using Descending Deviation Optimizations .....	303
Appendix I: Database, Data Warehouse and Data Mining Test Software .....	305
<b>Publications</b> .....	310
Journal Publications.....	310
Peer-Reviewed Conference Publications.....	312
Book Chapters .....	321
Master’s Thesis.....	322
<b>Glossary of Terms</b> .....	324

## LIST OF FIGURES

Figure 1 - Grunion by the thousands swarm the beach during a Grunion Run.....	2
Figure 2 - Different Problems within a Problem Space .....	7
Figure 3 - Type-2 Fuzzy Sets.....	11
Figure 4 – Crisp and Fuzzy Clusters .....	14
Figure 5 - A computer-based neuron .....	22
Figure 6 - A simple artificial neuron.....	22
Figure 7 – A Radial Basis Function using a Gaussian.....	24
Figure 8 – Radial Basis Function Network.....	25
Figure 9 – Subsumption in a Robot .....	26
Figure 10 - Different Problems within a Problem Space .....	31
Figure 11 - Solving a Problem Space Using Multiple Approaches .....	34
Figure 12 - Fuzzy Sets as Triangles.....	35
Figure 13 - Type-2 Fuzzy Sets.....	37
Figure 14 – A Typical Fuzzy Controller .....	38
Figure 15 – Deconstructing an eye into $\rho$ , $\sigma$ , $\tau$ .....	39
Figure 16 – Simple Fuzzy Context .....	40
Figure 17 – Polymorphic action SPEAK for DUCK vs. DOG vs. CAT .....	46
Figure 18 - Adding Fuzzy Rules Results in Diminishing Returns.....	48
Figure 19 - Overlap of Techniques for a Scheduler.....	48
Figure 20 - Solving a Problem Space Using Multiple Approaches .....	50
Figure 21 - Fuzzy Context Architecture .....	51
Figure 22 - Points in Fuzzy Clusters .....	52
Figure 23 - Type-C membership over a Problem Space.....	53
Figure 24 a, b – Circles in a rectangular container leads to “Packing Problem” .....	54
Figure 25 – Example Problem Space.....	54
Figure 26 – Fuzzy Contexts overlap and cover a problem space.....	55
Figure 27 – Triangle-shaped fuzzy membership function .....	60
Figure 28 – Fuzzy Framework Architecture .....	64
Figure 29 – A Patch of Ice on a Road .....	67

Figure 30 – Simple Gaussian .....	69
Figure 31 – Gaussian with Fuzzy Hedge .....	69
Figure 32 – Gaussian with Extreme Fuzzy Hedge.....	70
Figure 33 – Original Fuzzy Framework Architecture with Type-2 Extensions.....	73
Figure 34 – A Vehicle Ready to Navigate a Maze.....	75
Figure 35 – DB Set Operations Contrasted with Application and Database Cursor Operations .....	80
Figure 36 – Sample Back and Forth Corrections Navigating Maze .....	82
Figure 37 - Determining Peaks and Valleys .....	83
Figure 38 - Peaks with Plateaus .....	83
Figure 39 - Ordered Peaks and Valleys after Processing.....	84
Figure 40 - Creating an Optimal Gradient .....	84
Figure 41 - Set Operation to Isolate True Peaks and Valleys .....	87
Figure 42 – Robot Car Unable to Navigate a Barrier.....	87
Figure 43 – An Unsupervised Technique to create a new Context.....	89
Figure 44 - A Fuzzy Function Definition in XML.....	91
Figure 45 - Combining a Term and Fuzzy Function to Create a Fuzzy Set.....	91
Figure 46 - Defining a Fuzzy Variable in XML .....	92
Figure 47 - Defining an FIS Using the Fuzzy Modeling Language.....	94
Figure 48 – Fuzzy Wizard Variables Page.....	98
Figure 49 – Defining a Membership Function using the Wizard.....	99
Figure 50 - A Fuzzy Function Definition in XML.....	99
Figure 51 – Windows for building a Type-1 Fuzzy Inference System .....	101
Figure 52 - Local Peak Prevents Discovery of Global Solution .....	104
Figure 53. Genetic Mutation .....	105
Figure 54 - Robot in a Maze .....	110
Figure 55 - Robot Navigating Maze Using Framework-Based T1 FLC.....	112
Figure 56 - Robot Navigating Maze Using Framework-Based T2 FLC.....	113
Figure 57 (a) - T1-FIS and 53 (b) – GT2-FIS .....	114
Figure 58 – T1-FIS overlaid with GT2-FIS .....	115
Figure 59 – T1-FIS interlaced with GT2-FIS .....	115

Figure 60 - New maze with terrain features.....	117
Figure 61 – Navigating Terrain Features (a) Transitioning to Canoe (b), Helicopter (c), Submarine Vehicle (d) .....	118
Figure 62 – Comparison Growth T1-FIS vs Contexts .....	120
Figure 63 – Vehicle Unable to Navigate Barrier .....	122
Figure 64 – Using Contexts to Navigate Barriers in Maze .....	123
Figure 65 – Comparing T1-FIS (left and top) and T1-FIS (right and bottom) with Contextual Subsumption for (a) left leg of maze, (b) top leg of maze, (c) bottom leg of maze.....	124
Figure 66 – Comparing GT2-FIS (left and top) and GT2-FIS (right and bottom) with Contextual Subsumption for (a) left leg of maze, (b) top leg of maze, (c) bottom leg of maze .....	124
Figure 67 – A solution for the 8-Queens problem .....	127
Figure 68 - A Robot Car Navigating a Maze .....	130
Figure 69 - A Robot Car Navigating a Maze, Typical FIS – Type 1 FLC .....	133
Figure 70 - A Robot Car Navigating a Maze, Optimized T1-FIS.....	133
Figure 71 - Bad FIS cause robot car to hit barrier.....	134
Figure 72 - Bad FIS repaired.....	134
Figure 73 - Robot in a Maze .....	135
Figure 74 – Algorithm Learning Contexts.....	136
Figure 75 – The Fuzzy Wizard Tool.....	137
Figure 76 – XML for a Type-1 FIS.....	140
Figure 77 – General Type-2 FIS in Wizard .....	141
Figure 78 – Database Schema for a Fuzzy Configuration .....	142
Figure 79 - A basic, horizontal Decision Tree .....	164
Figure 80 - CoFuH-DT reduction of Decision Tree.....	170
Figure 81 - Rule Creation using Decision Tree .....	171
Figure 82 - Normalization of a Decision Tree .....	171
Figure 83 - Fuzzifying customer’s Decision Tree.....	172
Figure 84 - Context unifying 3 clusters.....	173
Figure 85 - Nodes pruned by context.....	173
Figure 86 - Context of shopping type. ....	174

Figure 87 - Fuzzy deformation under a context .....	174
Figure 88 - Node growth under normal conditions and contexts.....	177
Figure 89 - A typical Decision Tree used for data mining.....	180
Figure 90 - A Decision Tree node generation node with attributes .....	181
Figure 91 - Context spanning several nodes .....	183
Figure 92 - Calculating distance between nodes.....	186
Figure 93 - CoT-DT Step 1 - Creation of Decision Tree .....	187
Figure 94 - Nodes of Decision Tree produce subset $s_i$ of original set $S$ . .....	187
Figure 95 - ANN classifier applied to leaf node sets produces clusters.....	188
Figure 96 - ANN cluster generation.....	189
Figure 97 - Cluster span over several nodes .....	190
Figure 98 - Sample Decision Tree with cluster-span.....	191
Figure 99 - Comparison of Local/Global Maxima.....	198
Figure 100 - Bounce out of a Local Maxima Trap.....	199
Figure 101 - Pattern in which SA fails to find a solution .....	203
Figure 102 - Simulated Annealing with Descending Deviations.....	204
Figure 103 - A basic, fuzzy description of a person's height.....	211
Figure 104 - Fuzzy description of a person's height using hedge SOMEWHAT. ....	213
Figure 105 - A basic, fuzzy description of a person's height using hedge VERY. ....	213
Figure 106 A stair-step, crisp implementation.....	214
Figure 107 - Constructing fuzzy sets from speed ranges.....	214
Figure 108 - A smooth fuzzy implementation .....	215
Figure 109 – A crisp membership function.....	218
Figure 110 – A Fuzzy Membership Function .....	218
Figure 111 – Trapezoid and Gaussian shaped membership functions.....	219
Figure 112 – A left trapezoid .....	220
Figure 113 - $\mu_A(x)$ .....	222
Figure 114 - $\mu_B(x)$ .....	222
Figure 115 – Operations on T1 Fuzzy Sets Using Minimum T-Norm Operator.....	223
Figure 116 – Operations on T1 Fuzzy Sets Using the Maximum T-Conorm Operator.....	224
Figure 117 – Typical Fuzzy Inference System .....	225

Figure 118 – Fuzzification/Defuzzification Process .....	228
Figure 119 - GT2 Fuzzy Set <b>A</b> .....	234
Figure 120 – GT2 Fuzzy Set <b>B</b> .....	234
Figure 121 – The resulting Union operation of GT2 Fuzzy Sets <b>A</b> and <b>B</b> .....	235
Figure 122 – The resulting Intersection operation of GT2 Fuzzy Sets <b>A</b> and <b>B</b> .....	236
Figure 123 - Fuzzy sets for a thermostat in Barrow, Alaska.....	240
Figure 124 - Fuzzy sets for a thermostat in Phoenix.....	241
Figure 125 - A Fuzzy NFS transition of Fuzzy Type 1 sets.....	242
Figure 126 - Fuzzy NFS dimension of uncertainty added to Fuzzy 1 .....	242
Figure 127 - Polymorphic Fuzzy Signature Tree.....	244
Figure 128 – Contrasting K-means and Fuzzy C-means clusters .....	247
Figure 129 – Queens Arranged on Chessboard with no Conflicts.....	251
Figure 130 - Typical space with a global (goal state) and local maxima.....	253
Figure 131 - Local Search following gradient and “bounce” out of local maximum. ....	253
Figure 132 - A biological neuron and its computer-based equivalent. ....	254
Figure 133 - A simple artificial neuron.....	255
Figure 134 - Single neuron separates two square patterns. ....	255
Figure 135 - Neurons working together separate squares from circles.....	256
Figure 136 – A View of Evolution.....	257
Figure 137 – Predators adapt to a preponderance of green beetles.....	259
Figure 138 – Beetles adapt to Predators Preferences by changing color .....	260
Figure 139 – Recombination of Genotypes .....	261
Figure 140 – Genetic Mutation .....	262
Figure 141 – A Hierarchical Database Design.....	265
Figure 142 – SQL Server Management Studio .....	269
Figure 143 – SQL Server 2012 System Tables.....	270
Figure 144 – 5 part identifier VMWAREBOX.FuzzyContexts.dbo.Algorithm.AlgorithmId	272
Figure 145 – Schema view listing primary key constraints .....	276
Figure 146 - Data Mining Process. ....	279
Figure 147 - Classification of relations among multiple elements. ....	280
Figure 148 - Amazon.com association links other book titles to a book purchase.....	280

Figure 149 - Predictions (far right dotted lines) from existing data patterns .....	281
Figure 150 - Linear Regression attempts over a series of data points. ....	282
Figure 151 - Time series analysis of US Nominal GDP vs. 5-year Treasury Note. ....	283
Figure 152 - Partial Schema of AdventureWorks sample database .....	306
Figure 153 – A Decision Tree in SQL Server Analysis Services .....	308
Figure 154 - The Bayes mining model.....	309
Figure 155 - The Neural Network mining model.....	309



## LIST OF PSEUDO-CODE

Pseudo-code 1 - A Generalized Contextual Approach .....	8
Pseudo-code 2 – Steps for Dynamic Programming .....	27
Pseudo-code 3 – Implementing a Context .....	33
Pseudo-code 4 - Determine a Context Contribution .....	74
Pseudo-code 5 - Determining the Optimal Path.....	82
Pseudo-code 6 - Set-based algorithm to find peaks and valleys .....	85
Pseudo-code 7 – Construct a Fuzzy Inference System from Modeling Language .....	95
Pseudo-code 8 - Create an Algorithm from the ADL .....	96
Pseudo-code 9 – Steps for Building a Type-1 Fuzzy Inference System .....	97
Pseudo-code 10 – Build Fuzzy Inference System from ADL.....	100
Pseudo-code 11 - Finding which algorithms solve a particular problem.....	107
Pseudo-code 12 - Finding most suitable algorithm for a particular problem.....	108
Pseudo-code 13 – Navigate Through a Maze .....	111
Pseudo-code 14 – Testing Algorithms against a problem.....	126
Pseudo-code 15 – Fuzzy C-means algorithm.....	250
Pseudo-code 16 – Hill Climbing Algorithm .....	285
Pseudo-code 17 – Stochastic Hill Climb Algorithm .....	287
Pseudo-code 18 – Improved Stochastic Hill Climb Using DD Technique .....	289
Pseudo-code 19 – Random Restart Hill Climbing Algorithm .....	291
Pseudo-code 20 – Simulated Annealing .....	293
Pseudo-code 21 – Genetic Mutation Algorithm .....	294
Pseudo-code 22 – Min Conflicts Algorithm .....	296
Pseudo-code 23 – Tabu Search Algorithm.....	298
Pseudo-code 24 – Improved Simulated Annealing Algorithm Using DD Technique .....	300
Pseudo-code 25 – Memetic Mutation Algorithm.....	302
Pseudo-code 26 – Improved Genetic Mutation Algorithm Using DD Technique.....	304

## LIST OF TABLES

Table 1 – Application vs. Database Comparative Analysis – 10,000 rows/objects.....	78
Table 2 – Application vs. Database Comparative Analysis – 100,000 rows/objects.....	78
Table 3 – Application vs. Database Comparative Analysis – 1,000,000 rows .....	79
Table 4 – Application vs. Database Comparative Analysis – 10,000,000 rows .....	79
Table 5 – Simple Comparison T1-FIS/GT2-FIS.....	114
Table 6 - Comparing Traditional and Contextual Performance.....	116
Table 7 - Extended Traditional vs Contextual Performance.....	116
Table 8 - Comparing Standalone and Contextual T1-FIS.....	120
Table 9 - Comparison of Traditional vs. Subsumptive (Inverted Context) Approaches .....	124
Table 10 - Comparing Algorithms for the 8-Queens Problem.....	128
Table 11 - Ranking Solutions to the 8-Queens Problem.....	129
Table 12 - Comparative Analysis T1-FIS vs. T2-FIS.....	130
Table 13 - Ranking T1-FIS vs. T2-FIS .....	131
Table 14 - Comparative Analysis and Ranking of Sorting Algorithms.....	132
Table 15 - Dimensions for a virtual store manager.....	169
Table 16 - Example 2: Node Reduction Under Contexts.....	177
Table 17 - Node Reduction Under Contexts.....	178
Table 18 - Attributes of a typical customer.....	184
Table 19 - Node comparisons using various contexts.....	195
Table 20 - Initial Results of LSA Testing .....	200
Table 21 – Steps for Descending Deviation Optimization Technique.....	201
Table 22 - Results of Modified Local Search Algorithm Testing.....	205
Table 23 - Fuzzy and Boolean AND Truth Table.....	212

## CHAPTER 1: INTRODUCTION

### 1.1: MOTIVATION

Imagine getting a phone call from some government official, “Congratulations, you have just won 1 million dollars! Buy a plane ticket and fly out here to collect your money.” How would you feel? Suppose the location was Washington DC and the dollars were US. Could you book a flight fast enough? How much risk, be it money, time or anything else, would you be willing to take to collect your paycheck? Probably a lot, because a million dollars US IS a lot of money. Suppose, however, the call was from Zimbabwe where a million Zimbabwe dollars is worth about \$2800 today [XE 12]. Would you still go? Now suppose the million dollars is US, but you can only collect if you are willing to give up your sight for the rest of your life. What would your decision be? What would your decision be if you were already blind? Each of these scenarios involves the same “million dollars” but the reactions can be quite different due to the context in which the decision is made.

We live in a world of context. As living creatures we respond to stimuli, but as thinking beings we take that a step further and are able to consider the “context” or environment in which we operate. Whether it is a reference to time, such as the season; or temperature, such as hot or cold; or even our state of mind, our actions and behavior are driven by the contexts around us. Contexts guide our decisions, shape our viewpoints and even affect our moods. Contexts can be highly intimate and personal, such as your family, or include a much larger association, such as a city or even a nation. Contexts can be singular, like a warm day; or form hierarchies of contexts, such as relatively warm period of the day.

More interestingly, a behavior which might appear wildly variant under one context is perfectly normal under another. Consider the behavior of the California Grunion. For most of its life it behaves like any other fish [CaGov 14] swimming the shallow waters just offshore from the California coast. However, during the spring and summer months if the phase of the moon and the tides are just right, grunion will leave the safety of the water for the shore to spawn. For brief periods the females dig themselves into shallow holes while the males curve around them. Thousands of spectators gather during these times to view the

paradoxical event; paradoxical because they come to see behavior that is highly usual for fish, yet completely understandable when taken in context of the need to spawn.



**Figure 1 - Grunion by the thousands swarm the beach during a Grunion Run**

Contexts can play a similar role in computing. Because real-world information and processes are messy and often ambiguous, contexts can help to make the unusual understandable to a human user or an algorithm. Contexts can also allow higher-level processes to switch between algorithms when a particular approach is clearly warranted or transition among multiple options when it isn't. A duck is a duck, but how it *moves* depends upon whether it is in the water, on land, in the air or somewhere in between. Likewise a computer process should be able to switch between different algorithms as the *context* of the problem changes. As we ask more and more of computing devices, like robots, we need to develop the frameworks and tools necessary to allow them operate seamlessly in and among the differing contexts they are likely to encounter. Like a duck swimming or running faster before taking flight or slowing down to approach a pond or patch of grass, computing

problems must also be able to identify and handle these transitional phases when moving from one problem space to another.

So what do we mean by “context” in the context (pun intended) of a running process? Dictionary.com [Random 13] states a context is, “the set of circumstances or facts that surround a particular event, situation, etc.” In the world of computers, there is a similar idea known as the *state*, which describes a particular process at a point during its execution [Sipser 12]. For purposes of this dissertation, consider the *state* to be the internal condition of a process component, such as a robot car or a database query. The *context*, on the other hand, refers to its external environment: for the car it might be the road it is traveling on and for the database query a set of query conditions.

## 1.2: OBJECTIVE

The goal of this dissertation is to introduce and demonstrate the novel concept of Fuzzy Contexts, or Fuzzy Type-C. It will take existing key concepts of Fuzzy Logic and other techniques and build upon them to describe the concepts and architecture. In addition, it will demonstrate how “Contextual Thinking” and a concept called Fuzzymorphism can help bridge human-computer interactions in a linguistically meaningful way. Once these ideas are established, the reader will be guided through the implementation of an advanced Fuzzy Framework.

Using the framework and test examples, the dissertation will demonstrate the following:

1. What Fuzzy Logic Type-C is and how it can be used.
2. What Fuzzymorphism is and how Fuzzy Contexts achieve Fuzzymorphism.
3. How Fuzzy Logic Type-C fits within general algorithm techniques and advantages for Type-C implementations as compared to various alternatives.
4. How Fuzzy Logic Type-C is distinguished from Fuzzy Logic Type 1 or Type 2, or other traditional hybrid and non-hybrid algorithms.
5. How Fuzzy Logic Type-C can be used as a software framework for implementations

6. How a Type-C framework can use a relational database and to achieve better performance and a simpler architecture over a non-database implementation.
7. How to implement dynamic algorithms and algorithm definitions in support of a Type-C application.
8. Applications of Fuzzy Contexts in solving a diverse spectrum of problems.

Via the framework the dissertation will apply *fuzzified* contexts as a way to both describe and solve certain classes of computing problems and show how these contexts can improve certain processes by providing a hierarchical framework to seamlessly combine different algorithmic approaches. The dissertation will also describe a technique to perform unsupervised decomposition of a problem space into contexts.

Finally the dissertation will explore other potential applications of contexts, such as contextual pruning of fuzzy decision trees and optimizations of local search techniques.

### **1.3: DISSERTATION STRUCTURE AND CONTRIBUTIONS**

The dissertation is organized as follows:

- Chapter 2 delves into the background of the techniques and problems relevant to this research topic. It presents a brief literature review of prior solutions, approaches and some of the resulting issues raised. In particular, the review discusses Fuzzy Logic and its variants, local search techniques, evolutionary algorithms, object-oriented design and architecture, advanced data mining techniques and the role of relational databases in application frameworks.
- Chapter 3 describes Fuzzy Contexts, related constructs and their implementation from previously published works and new research and how Type-C provides advantages over traditional approaches. The contribution in this chapter is the novel concept of Fuzzy Contexts and how it extends and is differentiated from traditional Fuzzy Logic and hybrid systems approaches:
  - Defining the notion of Situational Discontinuity Problems and the types of problem spaces for which a Type-C solution would be beneficial.
  - Presenting prior research in this area.

- Giving a description of Fuzzy Contexts and how they might be used to solve SDPs and some of the advantages they offer over traditional approaches.
- Chapter 4 presents a software framework developed from previously published works and as part of this dissertation. The contributions in this chapter are the framework and definition language and its many enhancements
  - Describing a novel software framework supporting Fuzzy Type-1, Type-2 and Type-C implementations.
  - Adding relational database extensions
  - Providing a Fuzzy Definition Language/Algorithms Definition Language and Configuration Tools for Type-1 and Type-2 Definitions and Implementations.
  - Describing a database-based memetic algorithm and fitness function
  - Presenting a set-based technique to determine peaks and valley in a dataset
  - Presenting a technique for adaptive Type-C context creation
  - Presenting brute-force technique for generic algorithm determination and optimization
  - Comparative analysis with other techniques
- Chapter 5 presents work in progress.
- Chapter 6 presents the conclusion of the research and results of this dissertation and suggests directions for future work.
- The appendices present more detailed descriptions of the techniques and tools discussed in Chapters 3, 4. In addition, the Minor Contributions sections present some published and unpublished research involving contextualization of fuzzy decision trees and an optimization technique for certain local search algorithms.

## CHAPTER 2: BACKGROUND

### 2.1: PROBLEMS THAT COULD USE A LITTLE CONTEXT

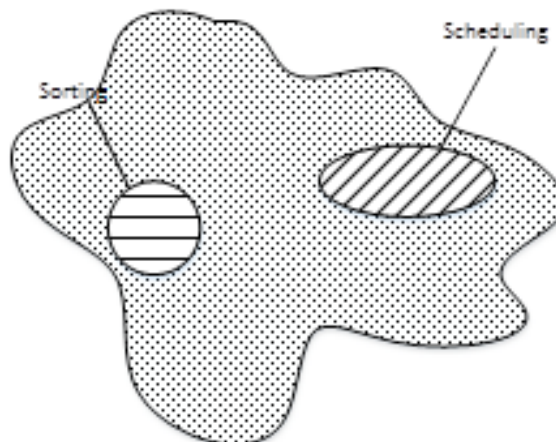
Among the many things computer processes do is to solve problems. They do this by using algorithms. An algorithm is defined as “any well-defined computational procedure that takes some value or set of values as input and produces some value or set of values as output” [Cormen 09]. In the decades since the introduction of the first digital computer, many kinds of algorithms were developed in order to solve specific classes of problems. These algorithms cover a range of diverse requirements from properly landing on the surface of another planet to generating the appropriate tone on your computer when you accidentally lean on your keyboard. Each was designed with a specific purpose in mind. For instance, when faced with a list of names, a developer may decide it necessary to sort them in alphabetical order. To do this efficiently requires using an efficient sorting algorithm such as a quicksort.

Quicksort is very useful for sorting problems but would it be an appropriate application for a scheduling problem? Not likely according to [Sipser 12] simply because it only knows how to operate properly within its own narrow set of requirements. Hence, other algorithms are needed as problems and requirements change. In the beginning, when computers were the size of school buses and little more than glorified calculators [Wiki 14], doing a single process with a single algorithm at a time was generally all that was possible. Today, however, the school bus-sized computer is a massively parallel system with thousands of processor cores. Computers the size of a grain of sand have more processing power than the first generation behemoths and are working their way ever more deeply into our everyday experience. As a result, computers are asked to do more and more of the tasks we used to do for ourselves. The glorified calculator is transforming itself into an extension of the human mind and body.

Therein lies the difficulty. Computers are not living things, but we are expecting them to understand and react to their environment in the manner of living things. Put another way, we want computers to understand and apply contexts like we do. Like a duck, for example, that swims on top of the water and waddles on land and flies through the air, our computers need to be able to navigate varying terrain features and obstacles, play differing levels of



competition and know we really like pancakes for breakfast, but not for dessert. Imagine a domain of problems in which some solutions are best served using a sorting algorithm and others using a scheduling algorithm as shown in figure 2.



**Figure 2 - Different Problems within a Problem Space**

It is not uncommon in a larger domain to see different problem spaces overlap. Consider, for example, the braking system on a car. Newer model cars employ “anti-lock” brakes which must tackle two separate problem spaces in particular: one being slowing down in response to a “brake” request, the other being to maintain control of the car by ensuring the wheels continue to rotate and not “lock”. When speed is slow and braking is gentle there is no need for the “anti-lock” function, but when speeds are high and braking is hard the problem spaces begin to overlap and transitions must occur until the car is able to stop.

Even within a given problem space, circumstances can arise which introduce “situational discontinuity”. Situational discontinuity occurs when the problem space, for example, a road, contains occurrences which change the resulting problem significantly, such as hitting a patch of ice. Because the subsequent behavior must be so different, it is effectively the same as having to address a different problem space altogether. Living creatures are naturally well-equipped to adapt to Situational Discontinuity Problems (SDPs). A duck, for example, swims in the water but waddles on land and flies through the air. Human beings sweat when it is hot and shiver when it is cold and so on.

In the artificial world, however, handling SDPs becomes a matter of using different algorithms or generalizing a single approach, which is not easy. In data mining, for example, there are many algorithms used to find interesting information from huge, often disparate data sets [Han 11]. An experienced data miner needs to be familiar with Decision Trees, Neural Networks, Linear Regression and a whole host of other algorithms, each of which has advantages depending upon the underlying patterns in the data [MacLennan 08]. Even something as simple as a fuzzy thermostat may have some rules for temperature, others for humidity and still others for time of day in order to handle many different demands for climate control.

Intuitively it seems obvious that different classes of problems require different approaches, but the problem with SDPs is that they tend to be ambiguous; hence it can be difficult to determine when an SDP has occurred and what to do about it. A fuzzy controller trying to navigate a maze must already deal with a number of navigation problems without also having to negotiate obstacles such as ice and potholes that it may or may not encounter. Ideally there would be a generalized contextual approach capable of handling all the underlying SDPs encountered; one that was efficient, easy to understand and implement. Pseudo-code for such an approach might look something like the following pseudo-code:

### **Pseudo-code 1 - A Generalized Contextual Approach**

**Algorithm:** TYPE-C\_EVALUATE (contexts, tuple)  
**Inputs:** *contexts*, a set of fuzzy contexts in which each context represents a problem scenario, such as ice, potholes, smooth, etc.  
           : *tuple*, set of values representing measurements or state of process  
**Output:** crisp result  
**Begin**  
 1 Test each context to see if it is valid for this state  
 2 FOR EACH valid context:  
 3     Determine the corresponding weighting of this context  
 4     Determine the membership value for this context  
 5     Run the corresponding context algorithm against *tuple*  
 6     Combine algorithm results, weight and membership values  
 7     Add to final result  
 8 NEXT context  
 9 return final result  
**End**

This approach allows for a decomposition of a large problem space into a series of smaller ones with presumably simpler solutions and is useful to avoid the complexity problems of generalized algorithms [Mendis 10].

## 2.2: USING FUZZY LOGIC

### 2.2.1: Fuzzy Type 1

Describing the behaviors of complex systems present many challenges for the software architect and developer. Foremost among them is the ability to model behaviors that are by their very nature imprecise [Zadeh 65]. In the “crisp” world, this is a particularly difficult task since even a small number of inputs requires a complex equation in order to create a smooth, continuous result. In particular, crisp solutions have difficulty properly describing behavior at boundaries [Cox 94].

Discontinuity at the boundaries must be smoothed in order for the function to prevent hyper-oscillation around those values. Fuzzy logic, also known as Type-1 Fuzzy Logic, introduced by Lofti Zadeh [Zadeh 65], [Taheri 06], [Zadeh 08] addresses these problems by approximate, rather than precise, descriptions for terms and allows for polyvalent membership definitions.

As compared to a crisp controller, a fuzzy controller allows for greater linguistic precision in describing a complex system behavior while at the same time relaxing precision around the boundary points and elsewhere. This is done through the use of a membership function  $\mu$  whose output, instead of the traditional FALSE (0) and TRUE (1) allows for output of 0, 1 and all values in between. Thus, for a domain  $D$

$$\mu(x) \rightarrow [0, 1], x \in D \quad (2.1)$$

A fuzzy membership function defines a fuzzy set  $f_s$ , which can be described using a linguistic term. A fuzzy set  $f_s$  is then a set of ordered pairs

$$f_s \equiv \{(x, \mu(x)) | x \in D\} \quad (2.2)$$

A fuzzy set can take any convex shape, with each fuzzy set depending upon its membership function. Fuzzy algorithms are very good at approximating complex polynomials. They also provide stronger mechanisms for handling noise and uncertainty along with variations among “expert” definitions than their crisp cousins [Roychowdhury 98]. However, fuzzy logic also has limitations that pose new problems. Whereas the crisp algorithm has difficulty with the discontinuity at a boundary, likewise a fuzzy algorithm has trouble handling large “contextual” changes such as those that occur in SPDs.

As a crisp solution could be improved by adding additional temperature tiers [Cox 94], likewise an SPD could be improved by the addition of fuzzy rules. However, adding tiers makes the temperature algorithm significantly more complex; likewise the addition of fuzzy rules adds significant additional complexity to a fuzzy solution [Mendis 10].

The underlying problem within fuzzy systems, and more generally, all approaches, is that certain problem domains are more solvable using certain approaches than others. Within each of these specific problem areas often lies even more specific issues which require ever more specialized techniques.

Type-2 Fuzzy Logic was designed to address such issues [Mendel 02], but with only limited success. For instance [Linda 11b], demonstrates how a Fuzzy Type-1 controller was superior navigating around corners but inferior to a Fuzzy Type-2 controller navigating smoother surfaces. Even within a particular problem domain, one configuration of a Fuzzy Inference System (FIS) will be superior for handling a simple maze while another FIS is more appropriate elsewhere for obstacles.

For a more extensive treatment of Fuzzy Logic Type-1, see the appendix at the back of this dissertation.

### 2.2.2: 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 a number of issues with Fuzzy Type 1 logic [Mendel 02]:

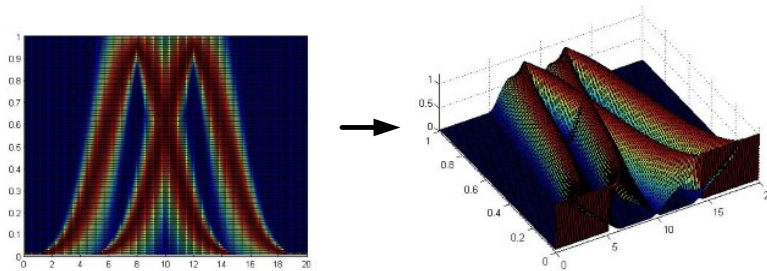
1. Experts can disagree on meaning of linguistics terms.

2. Fuzzy sets work best on continuous data.
3. Data can contain noise beyond the ability of Fuzzy Type 1 to handle easily.

Fuzzy Type-2 introduces uncertainty into the fuzzy sets themselves, in effect relaxing the boundaries of the membership function  $\mu_2$ , so in contrast to Equation 2.1:

$$\mu_2 = \{((x, \mu), \mu_2(x, \mu)) | \forall x \in D, \mu \in [0, 1]\} \quad (2.3)$$

Note also that output of  $\mu_2$  is also member of the set  $[0, 1]$ . Whereas a Type-1 fuzzy set is a 2-dimensional object, Type-2 fuzzy sets are surfaces as shown in figure 3.



**Figure 3 - Type-2 Fuzzy Sets**

A Type-2 fuzzy inference system is useful in dealing with problems such extensive noise or smoothing out erratic behaviors that plague Type 1 controllers.

For a more extensive treatment of Fuzzy Logic Type-2 see Appendix B at the back of this dissertation.

### 2.2.3: Nonstationary Fuzzy Sets

Nonstationary Fuzzy Sets (Fuzzy NFS) introduces the notion of variability of fuzzy sets over some dimension such as time, location, or even noise [Garibaldi 08]. They do this by giving the resulting Fuzzy Type 1 definitions the flexibility to change according to requirements, but remain internally consistent at a higher level within the overall Fuzzy

Inference System (FIS). In other words, the fuzzy sets change depending upon the value of the external dimension, but the fuzzy variables and rules remain unaffected.

The Fuzzy NFS  $nf_s$  is described as

$$nf_s = \int_{d \in D} \int_{x \in X} \mu_{fs}(d, x) / x / d \quad (2.4)$$

Where  $d$  is some value along a dimension of the problem domain  $D$  and  $x$  is a tuple or point within the set of possible inputs  $X$ . Nonstationary Fuzzy Sets provides the dynamic fuzzy membership function transform, or perturbation function (and resulting sets) able to accommodate significant changes to the problem space over that dimension. The perturbation function is simply responsible for adjusting the underlying membership functions as needs change. The Fuzzy NFS is then able to generate a variable FIS to handle changes in the problem space which otherwise might cause difficulties to a static Type-1 or Type-2 FIS.

For a more extensive treatment of Nonstationary Fuzzy Sets see Appendix B at the back of this dissertation.

#### 2.2.4: Polymorphic Fuzzy Signatures

Finally, Polymorphic Fuzzy Signatures (Fuzzy PFS) describe a multidimensional fuzzy tree of fuzzy sets where each leaf contains a specific fuzzy membership function [Mendis 10]. A problem domain is recursively decomposed into a hierarchy of subdomains, each with corresponding meta-information about its attributes. This occurs until decomposition ends at a leaf node. Each leaf is assigned a unique fuzzy inference system.

Fuzzification occurs by recursively traversing the branches and testing each node's meta-information to determine whether input has membership along that path. This process generates a candidate collection leaf nodes with positive membership. The resulting collection of leaf nodes' output is combined using traditional fuzzy functions such as *max* and *min*. The polymorphic fuzzy signature is described as [Mendis 08]:

$$\mu_{sig}: X \rightarrow [c_i]_{i=1}^k (\equiv \prod_{i=1}^k c_i) \quad (2.5)$$

$$\text{where } c_i = \begin{cases} [c_{ij}]_{j=1}^k; & \text{if } \mathbf{branch} (k_i > 1) \\ [0, 1]; & \text{if } \mathbf{leaf} \end{cases}$$

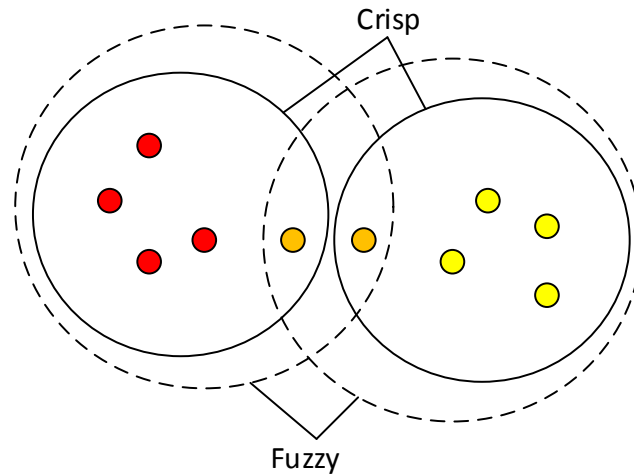
As a result, Polymorphic fuzzy signatures allows one to break down an SDP into smaller, easier to describe, components, each with its own FIS and attribute signature. FIS outputs are combine and fuzzified, with the resulting defuzzification using traditional methods.

For a more extensive treatment of Polymorphic Fuzzy Signatures see Appendix B at the back of this dissertation.

### 2.2.5: Fuzzy Clustering

Clustering is a technique used to groups sets of objects which are similar [Han 11]. Among the many techniques for this is Fuzzy Clustering. In general clustering techniques seek to create clusters with attributes such that they can assign every object or point to an individual cluster. In some clustering techniques, such as K-means, clusters have a centroid and a radius and points that belong to the cluster lie within the radius distance of the centroid. Similar to crisp logic, any individual point is either a member or NOT a member of any particular cluster.

Fuzzy C-means clustering, however, seeks to relax the crisp rules of K-means to allow points to belong to multiple clusters [Hung 11]. The difference is shown in figure 4.



**Figure 4 – Crisp and Fuzzy Clusters**

Fuzzy C-means is similar to K-means in that they both create clusters with centroids and a radius. The sum of their memberships also adds up to 1, but points in fuzzy C-means have membership in the range of 0 to 1 rather than just 0 or 1 [Gauge 11]. This gives a points in fuzzy C-means more flexibility in their associations as well as better handling of outliers and noise [Tsai 11].

The goal of fuzzy C-means is to minimize an objective function (Matteucci, 2012), such as:

$$J_m = \sum_{i=1}^N \sum_{j=1}^C u_{ij}^m \|x_i - c_j\|^2, \mathbf{1} \leq m < \infty \quad (2.6)$$

where  $m$  is any real number greater than 1

$u_{ij}$  is the degree of membership of  $x_i$  in the cluster  $j$

$x_i$  is the  $i$ th multi-dimensional data point in the domain

$c_j$  is the multi-dimensional centroid of the  $j$ th cluster

Fuzzy C-means techniques have been used in hybrid algorithms to handle the switching and contributions of the appropriate processes dynamically [Palm 98]. This is because fuzzy metrics provide a useful way to determine how “relevant” an algorithm is at a particular point.



Key to creating fuzzy c-means clusters is the “fuzziness” factor, a constant, greater than 1 (1 being the hard K-means number), which determines the fuzziness of the clusters. The higher the number, the greater the number of clusters any individual point may belong.

For a more extensive treatment of Fuzzy Clustering see Appendix B at the back of this dissertation.

## **2.3: OTHER TECHNIQUES FOR CONTEXTUAL PROBLEM SOLVING**

### **2.3.1: Local Search Techniques**

For state spaces that are sufficiently large, there may be either insufficient time or computing resources, making a comprehensive search for a solution 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 09].

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 [Martinjak 07].

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 [Martinjak 07], [Zheng 06].

Local search algorithms can be combined with additional heuristics to improve their effectiveness for a given search problem. Local search algorithms are discussed more fully with pseudo code examples in appendix C and H.

### 2.3.2: Evolutionary Computation

Evolutionary biology is the movement from simple to more complex life-forms over time [Dawkins 06]. What makes this movement possible is the growing specialization and interdependency with each new evolutionary step; allowing a complex organism to perform a greater variety of functions than its simpler predecessor. Instead of a simple life-form having to perform all the necessary functions of life such as motion, reproduction, eating, etc., the complex form consists of a series of subsystems, each of which is responsible for a particular life-sustaining function. Freed from having to worry directly about exactly how to accomplish a task, such as moving from point A to point B; the complex organism is able to focus on more abstract tasks, such as finding dinner.

The evolution of computing has progressed in a similar fashion. Ever since the introduction of the earliest computing devices engineers have sought to emulate evolution by developing more complex and interdependent computing subsystems. Standalone vacuum-tube based behemoths programmed by switches [Augarten 84] have evolved into tiny devices such as the smart phone connected to a vast array of applications, systems and services existing all over the globe. Freed from having to worry about such things as packet construction and encryption, data transmission, protocols and error-handling, query construction, data storage and retrieval, the cell-phone user is also able to focus on more abstract tasks, such as finding dinner.

### 2.3.3: Software Engineering and Architecture Issues

Developing any complex software requires the application of software engineering principles and architecture. Software engineering and architecture have been a primary concern of software developers and managers since the first software “bugs” were discovered [Augarten 84] but more so today. The primary reason is that much or most of the “expense” of a production system lies in its software [Pressman 09]. Perhaps more importantly, the expense often is greatly dependent upon the quality of the underlying software [McConnell 96]. There are unfortunately many stories about huge cost overruns due to poor software quality or worse, projects which simply failed. Most notable recently were the failure of 4 states’ attempts to create exchanges for the Affordable Care Act [Haberkorn 13] to the tune of

474 million dollars. In many cases the problem is not that developers lacked the skill to build quality software, rather they lacked a methodology to do so. Software engineering seeks to introduce a discipline to software development in order install better “habits” among software developers. Studies have shown that certain software engineering practices result in both better quality of software as well as reduced development time [McConnell 04], [McCarthy 95].

Software architecture is the foundation upon which software applications are built. Much like the architecture of a building provides the structure for the building itself, the software architecture lays the groundwork for an application or suite of applications at varying levels of complexity. Software architecture complements software engineering in that it establishes the framework upon which software engineering can express itself.

Software engineering and architecture are major disciplines and too large to be expounded upon in this dissertation; rather the author will discuss important practical issues heard, read about or experienced firsthand over decades in the industry.

#### *2.3.3.1: Software Engineering*

Most important is the issue of complexity. Many newer software developers, having long been recognized as top of their class, tend to equate complexity with cleverness. Case in point: the author, many years ago, working some full-text indexing/retrieval software managed to come up with a C++ routine requiring a triple pointer. He was quite proud at the moment, actually having been clever enough to conceive such a process and it worked well enough. The problem was it added unnecessary complexity to an already difficult piece of software. In time he had to say good-bye to that triple pointer as the software was refactored into something less clever, but much more maintainable. Lesson learned.

What developers often don’t realize is that the clever process they concocted can serve as an impediment to making upgrades or just trying to explain what it going on. Many figure they won’t forget what they did, until they do, or have no real consideration for the poor schlub who has to maintain that nasty bit of code later. Let’s rejoin the author some 10 years after the “triple pointer incident” working on a very complex priority queue for a card-swipe system. He worked hard to make this rather intricate routine as easy to understand as

possible. Fast forward almost 10 years later and he gets a call to make a minor change to this routine. 15 minutes after loading the project, he was able to locate where the change had to occur, what the change had to be, make the change, compile and verify the change worked – after not looking at the code in nearly 10 years. Easiest \$250 he ever made. Lesson indeed learned.

Software complexity also has a number of unintended consequences as well. Consider two very different software development platforms: Matlab and Visual Basic. Matlab, from MathWorks, has been around for roughly 30 years but remains in many respects a “fringe” tool [Moler 04] outside of science and academia. Matlab is a very powerful tool and capable of doing almost anything software can do. It has a huge set of powerful toolkits that allow it to do even more. Pick any business IT shop at random, however, and you will be hard-pressed to find a single developer who has used Matlab. Visual Basic, on the other hand is widely used in business applications but almost non-existent in the laboratory or university. Why the discrepancy? The answer has to do with complexity.

Introduced in 1991, within two years it became a staple in business software and by 1995 it was *the* rapid application development (RAD) tool pushing aside other, more capable development tools, such as Turbo Pascal, Turbo C++, PowerBuilder, Visual C++, Magic and others.

So what made Visual Basic so special, so different, so much *better* than the tools it replaced? It was definitely cheap compared to some, such as PowerBuilder, but more importantly, it was *easy*. Microsoft had built a tool which completely abstracted the underlying windows application programming interface (API) to a simple drag and drop, visual canvas. Under older environments, creating a simple application to pop up a “Hello World!” message box was not an easy process. First one had to build the “window” frame, hook into the Windows OS messaging system, configure a button, attach an event handler to the button, attach the button to the frame, add the message and build and display the message box – and don’t forget the code to clean up the application, turning off the events, discarding used memory, etc. In all it might entail a couple hundred lines of code along with the requisite debugging and tweaking. With Visual Basic, you built the same application by doing the following:

1. Launch the IDE.

2. Drag a button from the toolbox, drop it somewhere on the canvas.
3. Double click the button to go to a code page.
4. Add the following code: “MsgBox(“Hello World!”)

It would not be unreasonable to take an entire morning to build the HelloWorld application the old way, but using Visual Basic it could be done in a couple of minutes. Why? Because Visual Basic hid all the complexity of Windows behind the IDE. Developers were liberated from having to deal with Windows API in the same way programmers were liberated in the past from binary code (with machine language), then machine languages (with compilers and higher-level languages). It was a different paradigm and number of custom applications exploded. Anybody with a little smarts could build real UI-based applications using any number of tools that arrived on the heels of VB, including Delphi, Java, .NET and others. The introduction of these tools led to easier-to-use applications with corresponding productivity gains that led to robust growth throughout the roaring (19) 90's.

What about Matlab? Like Visual Basic, Matlab hides complexity, in this case functions such as matrix multiplication and inversion, differential equations, etc. Building software to do complex calculations in a language like Visual Basic is *hard*, but in Matlab it is *easy* (relatively speaking). So in the university or laboratory, where the ability to perform complex calculations is paramount, a tool such as Matlab is indispensable while a tool like Visual Basic is much less so.

Hardware and software are, usually, the least costly components of a development project [McConnell 96], [Pressman 09] with the most costly being personnel *time*. Software tools that make it easier to do a particular job are going to be successful. Easier tools usually lead to faster development time and higher software quality. Software frameworks are the real-world expression of software architecture and also play a key role in reducing software complexity by abstracting complex subsystems in much the same way biological systems have. Software frameworks work hand-in-glove with tools to make programming much easier and both got a major assist with the development of Object-oriented programming.

Object-oriented programming (OOP), is an idea which has been around since the 1960s, but really took off with the introduction of C++. It was designed to allow the individual programmer to do reduce complexity at the functional level. Object-oriented

programming consists of three pillars: encapsulation, inheritance and polymorphism, each of which serves to reduce software development complexity. Combining OOP with productive tools such as Matlab and Visual Basic has resulted in greater functionality and diversity and overall reach of computer software.

In short, the most important aspect of software engineering is about putting processes in place to reduce complexity and thereby produce better software, more cheaply and quickly. This is more easily accomplished through the use of software frameworks and tools which abstract complex systems into more easily understood and useful components. This leads to more widespread adoption and use along with the corresponding benefits of such use.

Fuzzy logic has a special place in this paradigm because not only does it allow one to reduce a complex polynomial into a much simpler fuzzy representation, but it also allows one to abstract that process into simple linguistic terms which are easier to understand. Fuzzy contexts take that linguistic ease of fuzzy logic and uses it to turn entire problems spaces into linguistic terms, allowing for integration of disparate algorithms, or perhaps even tools. It allows an individual to look at an application at a higher-level, in terms of whole systems rather than individual algorithms, thereby freeing the individual from the underlying complexity.

#### *2.3.3.2 Software Architecture*

Software architecture, as the foundation upon which a software application is built, takes a very special role in the development process. Consider the Joint Strike Fighter (JSF) program. The JSF is intended to develop an aircraft design that is flexible enough to allow a single aircraft to replace a number of different aircraft. The intent is to make it easier to build and maintain the JSF as opposed to supporting multiple aircraft types. Beyond a flexible aircraft, the JSF is also supposed to be better than the aircraft it replaces. It is supposed to outperform, outmaneuver and outfight legacy aircraft and the competition.

Software architecture similarly must support multiple functions and/or application processes, whether it is a simple order entry application or an operating system. Like the frame of a building, it must support whatever array of diverse software tenants is required. Therein lies the difficulty. If diversity of software is low, as in a simple word processor, then

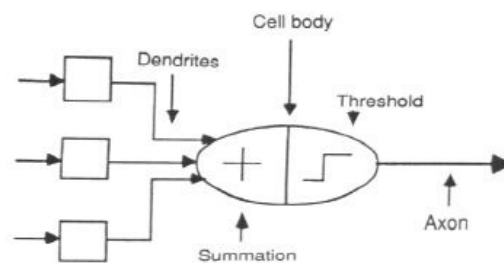
the architecture can be fairly simple. If diversity is high, as in a collaborative system like SharePoint, then the architecture must be able to support a wide range of requirements.

Like biological entities, computational entities must continue to evolve to solve more and more complex problems. Doing so will require sophisticated techniques beyond OOP. To get there requires software that understands and handles contexts much as we do, in order to solve problems as complex as those we face regularly.

Fuzzy Contexts is a software framework utilizing software engineering techniques, in particular OOP, as well as fuzzy techniques to hide the complexity of large systems as well as enable the creation of tools for diverse architectures. As such, it can bring machine even closer to the human experience. How this is done will be discussed in succeeding chapters.

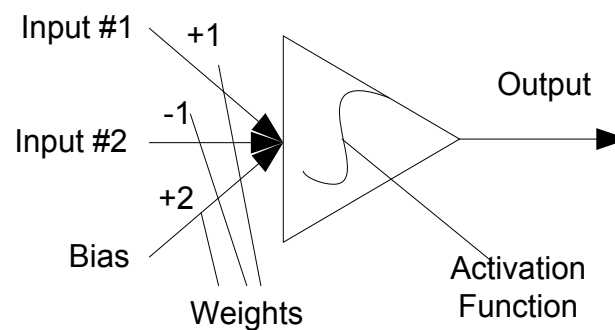
### 2.3.4: Neural Networks

Neural networks arose from a study of biological neural systems, although only superficial similarities exist [Schalkoff 97]. The concept of a neural network centers on a single unit that receives input from one of more sources. The unit is called a neuron and a collection of neurons action in concert is called a neural network. In the artificial version, inputs which are passed to an algorithm called an Activation Function that serves as the threshold as demonstrated in Figure 5.



**Figure 5 - A computer-based neuron**

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 6.



**Figure 6 - 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-



dimensional 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.

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 01] or classify objects. Neural networks come in a variety of configurations and display a variety of behaviors.

Typical uses for a neural network are classification, regression and prediction [Han 11]. 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 11].

For a more extensive treatment of Artificial Neural Networks see Appendix D at the back of this dissertation.

### 2.3.5: Radial Basis Functions

Radial Basis Function (RBF) networks are similar to Artificial Neural Networks (ANNs) in that they also accept one or more numeric values as input and generate one or more numeric values as output. RBF networks can be used to classify data as well as make predictions and have shown to be a useful machine learning technique [Li 13]. Consider a problem space  $D$  and a set of vectors  $v_1, \dots, v_n \in D$ . A Radial Basis Function is a function such that for any given  $v$ , its influence corresponds to its distance from a centroid  $v_0$ . Hence, given an RBF  $\phi$  each  $v$  influences  $\phi$  based upon the distance from  $v$  to  $v_0$  [Haykin 09].

$$\text{Each } v_i \in D \text{ influences } \phi(v) \text{ based upon } ||v_i - v_0|| \quad (2.7)$$

As a result  $\phi$  is radially symmetric, extending outward from the centroid. As a point gets nearer to the centroid, its influence, or “contribution” increases. It is very easy to normalize that contribution to a value between 0 and 1, which allows an RBF to emulate a

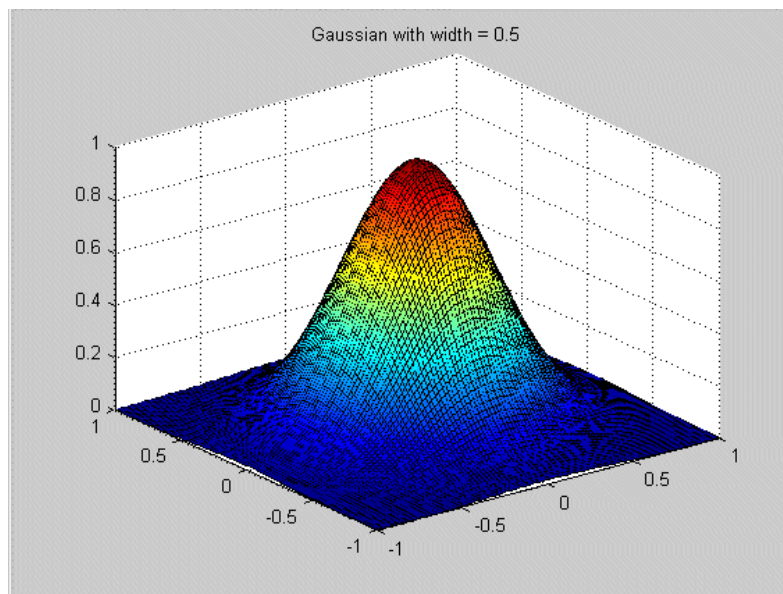
traditional fuzzy membership function. One typical approach is to use a Gaussian profile for the RBF. The profile function takes the form of:

$$\varphi(x) = e^{-r^2/\sigma^2} \quad (2.8)$$

Which produces the radial basis function

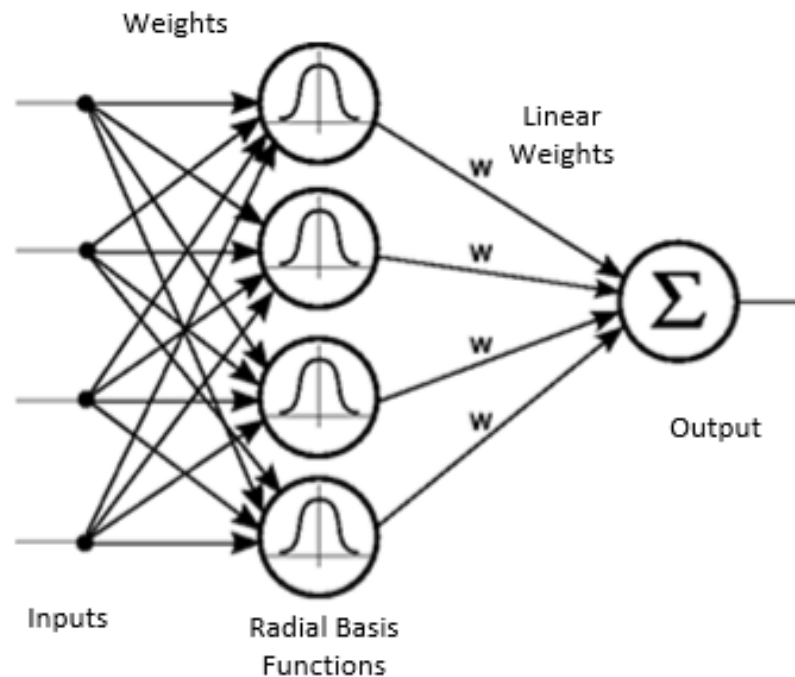
$$z(x) = \exp\left(\frac{\|x-x_0\|^2}{\sigma^2}\right) \quad (2.9)$$

Figure 7 shows what a Gaussian RBF function might look like.



**Figure 7 – A Radial Basis Function using a Gaussian**

Clearly, the closer  $x$  gets to  $x_0$  the greater the contribution. An RBF network, similar to an ANN consists of an input layer, an output layer and a hidden layer, demonstrated by figure 8.



**Figure 8 – Radial Basis Function Network**

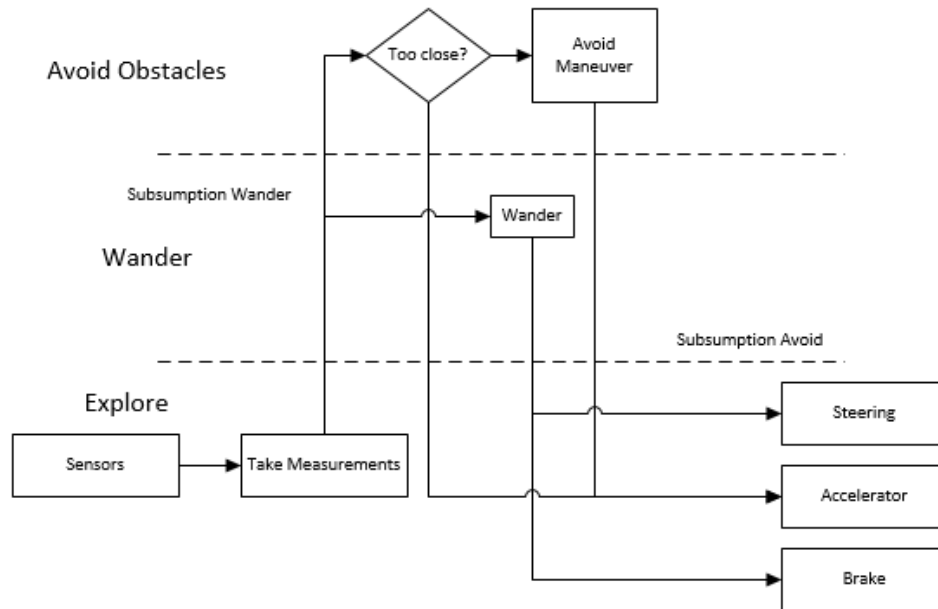
RBFs serve as activation functions within the hidden layer the RBF network. The idea behind RBFs is one emulated by Fuzzy Contexts.

### 2.3.6: Subsumption

Subsumption is a process whereby complex behavior is decomposed into sub-behaviors organized into a hierarchy. Lower levels encompass basic behaviors, such as obstacle avoidance. Higher levels encompass more complex behaviors, such as exploration. In a subsumptive process all the layers work in parallel, but lower level behavior will influence higher level behavior, or, put another way, higher level behaviors will “subsume” or incorporate lower level behaviors when trying to reach a goal [Liu 03].

A robot, for instance, tasked to explore its environment must move from place to place. In the process of exploring it must also subsume movement. This is useful because robots contain different components, such as sensors for exploring and actuators for movement and Subsumption allows for a complex interplay between the two subsystems. In addition, other basic behaviors such as avoiding obstacles, crucial for proper operation, are set at even lower levels [Rodrigues 08]. This allows the subsumptive process to suppress or

redirect operations in cases where normal methods are unable to handle the current problem space. A subsumptive controller must be constructed in a fashion similar to that shown in figure 9.



**Figure 9 – Subsumption in a Robot**

Fuzzy Contexts also draw inspiration from the application of Subsumption and can emulate Subsumption using a technique called “context inversion” which will be discussed in chapter 4.

### 2.3.7: Dynamic Programming

Dynamic programming (DP) is a method for solving complex problems by breaking them down into simpler sub-problems (Han, Park, & Kum, 2014), [Cormen 09]. The intent is to solve the simpler sub-problems first, then combine them into a total solution. DP is a bottom-up approach to decomposing a problem.

Dynamic programming is traditionally intended as an optimization technique, where the problem can have many solutions, but from among them there is an optimal solution. For example, take a sorting problem. Quicksort, heapsort, mergesort and bubblesort are all

possible solutions, but only one is considered optimal. DP attempts to discover the most optimal solution via one or more of the following steps:

### **Pseudo-code 2 – Steps for Dynamic Programming**

Characterize the structure of the optimal solution.  
Recursively define the value of an optimal solution.  
Compute the value of an optimal solution in a bottom-up fashion.  
Construct an optimal solution from computed information.

Suppose there is a factory with assembly lines, running at different speeds. While it might be advantageous to switch from a slower line to a faster one, there is also the cost of the switch to consider. Using a DP technique, a program can examine each line and weigh the costs individually, then combine them into an optimal path. Some classic DP problems are the Knapsack problem and the coin change problem. Dynamic Programming provides a methodology that can be extended to contexts, as will be demonstrated in Chapter 4.

### 2.3.8: Hybrid Systems

A hybrid system is considered a combination of dynamic systems each consisting of either continuous or discrete variables [Cheng 10] or whose state variables change over time [Du 10] or in response to significant events. As problem spaces become more complex the ability for any given traditional algorithm to adequately address the problem at hand decreases, hence the need to have combinations of more specific algorithms, each algorithm designed to handle a particular problem space. The intent is to combine the targeted algorithms into a single process to handle a complex problem, but to allow the individual advantages of the underlying algorithms to apply when appropriate.

This is analogous to a basketball team. Each team has players who excel at shooting, defending and passing and the coach must try to constantly mix and match his players in order to provide the best team he can at any point in the game. At times, each player will have an opportunity for the one-on-one interaction which is his specialty. At other times the same player will have little influence on the game. Similarly, problems with many facets can be

handled by multiple approaches, each of which comes with strengths to maximize and weaknesses to be avoided.

Hybrid algorithms, also called poly-algorithms consist of two or more distinct algorithms, each of which can solve the same problem, but solve it differently depending upon input values, constraints and resources. The goal is to maximize the performance of an algorithm strategically by selecting the appropriate process to handle a given system's state and goals. Algorithm selection can be *simple*, a choice from among many options or *baroque*, a choice dependent upon the input. Fuzzy Contexts seeks to combine *simple* and *baroque* options into a hierarchical framework. Chapter 4 covers this technique, while appendix B.5 take a deeper look at hybrid fuzzy systems.

## CHAPTER 3: FUZZY CONTEXTS

### 3.1: THE NEED FOR CONTEXT

As discussed in chapter 2, complex problems are becoming more and more common as computing machines develop ever greater functionality. Generalized solutions to complex problems often suffer from being overly complicated. One of the primary mechanisms for dealing with software complexity is through abstraction. Arguably, most of the major leaps in software were due at least partially to the introduction of an abstraction. Binary code led to machine code led to compilers led to high-level languages led to object-oriented programming and so on.

Why is abstraction so important? Recall the evolution of biological organisms and consider any complex living organism, like yourself. At any given moment thousands biological processes are going on inside you. There are larger ones involving many biological systems, such as breathing, digestion and temperature regulation. There are also many smaller ones such as cell regeneration, exchange of gasses, even simple thought. If we had to consciously attend to only a few of these processes, we would likely perish within minutes. There are just too many systems, large and small, to attend to. So why don't we perish in a heap of inattentiveness? Simply because our body, through the eons has abstracted more and more of our inner workings. If we want to go to the fridge and get a snack, we do it more or less without thinking. Our various internal systems handle the motions, related chemical and physical processes to enable us to get the snack. This allows us to focus our attention on the high-level task of acquiring a snack unencumbered by the corresponding low-level tasks required to simply get to the fridge.

Computing is evolving in much the same fashion. In creating machine code, suddenly programmers didn't need to know how to manipulate binary switches. Compilers freed us from the tedium of arranging machine code. And so it went. In each abstraction, one more layer of complexity was essentially hidden as the underlying technology was managed independently of the developer or user. The net result is was software (and hardware to some extent) grew more functional and comprehensive with each abstraction. Even more important was its impact on our lives grew ever more intimate, creating a new cycle of demand for more functionality and abstraction to achieve it. Thomas Watson, chairman of IBM is famously

believed to have stated the world market for computers was around 5. At the time, all computers did was compute ballistic firing tables. By the introduction of the first compiler, computers had moved into data processing. With the introduction of the operating system computers made their way into government, air traffic control, finance and industry. Since then we have created languages such as Ruby, Java, C# and C++ and it is no coincidence that the 5 computers of Thomas Watson's imagination have grown into over 100 computers in a single luxury car (Motavalli, 2010).

However, despite the dramatic growth in the past, competition has only gotten more intense, not less. Cars in the past took us from place to place, but now they regulate internal temperature, ensure our brakes don't lock and even check for blind spots. Some are even driving themselves. This means the problems car will face will be more complex than ever before.

Evolution shows us that the best way to address the problem of complexity is to add levels of abstraction. An architecture designed to do this successfully must allow for greater problem generalization without the traditional corresponding increase in complexity. One way to accomplish this is for the architecture to extend traditional fuzzy logic from the level of the algorithm to the level of the problem itself.

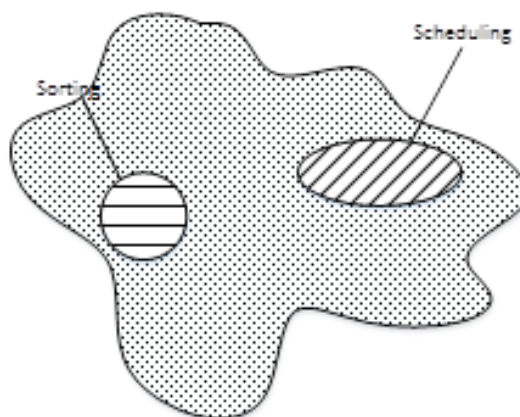
This dissertation proposes such an architecture, called Fuzzy Contexts or Fuzzy Logic Type-C. Fuzzy logic permits partial membership and values can belong to multiple fuzzy sets. By breaking down a problem space into smaller contexts and allowing algorithms themselves to have relaxed memberships in those contexts, a Type-C solution can support multiple solutions to complex problems. Using Type-C, problem spaces can be decomposed into smaller, more easily solvable components and fuzzified together under a Type-C hierarchy.

This dissertation will delve into a number of aspects of the Type-C framework. Because it intends to be a comprehensive solution for complex problems, it requires a number of components or "pillars". The first is the concept, definition, uses and implementation of the Fuzzy Contexts which will be introduced shortly. The second pillar is the "fuzzymorphic" behavior which allows transitions and hybridization when contexts overlap. The third pillar is the learning and optimization techniques.



Before going further it is important to lay the groundwork and understand algorithms and problem spaces in a general sense. An algorithm is defined as “any well-defined computational procedure that takes some value or set of values as input and produces some value or set of values as output” [Cormen 09]. In the decades since the introduction of the first digital computer, many kinds of algorithms were developed in order to solve specific classes of problems. For instance, when faced with a list of names, a developer may decide it necessary to sort them using a sorting algorithm such as a quicksort.

Recall that quicksort is very useful for sorting problems [Cormen 09] but would likely be a very poor application for a scheduling problem [Sipser 12]. Hence, other algorithms are needed as problems and requirements change. Take another look at a domain of problems in which some solutions are best served using a sorting algorithm and others using a scheduling algorithm as shown in figure 10.



**Figure 10 - Different Problems within a Problem Space**

It is not uncommon in a larger domain to see different problem spaces overlap. Such a case might be in a car driving in traffic. Speed and distance can differ based upon traffic conditions, road conditions and weather conditions. In many cases, traffic, road and weather can combine to create dramatically different responses to events.

### 3.11: The Situational Discontinuity Problem

Even within a given problem space, circumstances can arise which introduce “situational discontinuity”. Situational discontinuity occurs when the problem space, for example, a road, contains occurrences which change the resulting problem significantly, such as hitting a patch of ice. Because the subsequent behavior must be so different, it is effectively the same having a different problem space altogether. Living creatures are naturally well-equipped to adapt to Situational Discontinuity Problems (SDPs). A duck, for example, has to adapt for travel on land, under the water, on top of the water and in the air. Human beings deal with all kinds of different environments and issues from cold weather to traffic to dating.

In the artificial world, handling SDPs becomes a matter of using different algorithms or generalizing a single approach. In data mining, for example, there are many algorithms used to find interesting information from huge, often disparate data sets [Han 11]. An experienced data miner needs to be familiar with Decision Trees, Neural Networks, Linear Regression and a whole host of other algorithms, each of which has advantages depending upon the underlying patterns in the data [MacLennan 08]. A fuzzy thermostat may have some rules for temperature, other for humidity and still others for time of day in order to handle many different demands for climate control.

Intuitively, it seems obvious that different classes of problems require different approaches, but recall that the problem with SDPs is that they tend to be ambiguous, hence it can be difficult to determine when an SDP has occurred and what to do about it. A fuzzy controller trying to navigate a maze must already deal with a number of navigation problems without also having to negotiate obstacles such as ice and potholes that it may or may not encounter. Ideally there would be a generalized contextual approach capable of handling all the underlying SDPs encountered; one that was efficient, easy to understand and implement. Pseudo-code for such an approach might look something like the following:

### Pseudo-code 3 – Implementing a Context

**Algorithm:** TYPE-C\_EVALUATE (contexts, tuple)  
**Input:** *contexts*, a set of fuzzy contexts in which each context represents a problem scenario, such as ice, potholes, smooth, etc.  
           : *tuple*, set of values representing measurements or state of process  
**Output:** crisp result

**Begin**

- 1 Test each context to see if it is valid for this state
- 2 FOR EACH valid context:
- 3   Determine the corresponding weighting of this context
- 4   Determine the membership value for this context
- 5   Run the corresponding context algorithm against *tuple*
- 6   Combine algorithm results, weight and membership values to get final result.
- 7 NEXT context
- 8 return final result

**End**

This approach is useful to avoid the complexity problems of generalized algorithms [Mendis 10].

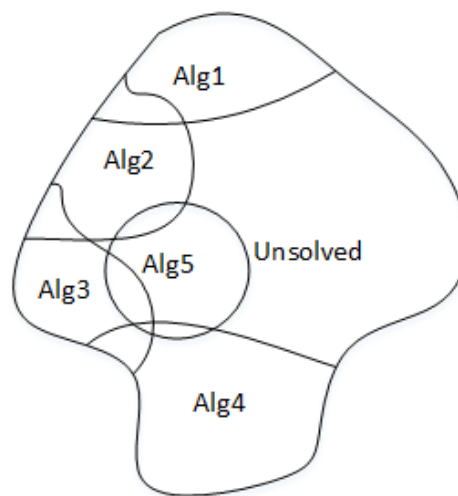
Among the many challenges for the software architect and developer is the ability to model behaviors that are very difficult to describe [Zadeh 08], [Surprise 13]. In the “crisp” world, this is a particularly difficult task since even a small number of inputs requires a complex equation in order to create a smooth, continuous result [Cox 94].

In particular, crisp solutions have difficulty properly describing behavior at boundaries. For example, a crisp thermostat trying to maintain a specific temperature might find itself frequently turning on and off as the temperature modulates around a desired level, an inefficient behavior. Discontinuity at the boundaries must be smoothed in order for the function to prevent the sort of hyper-oscillation around those values that leads to inefficiency or worse [Cox 95].

In more extreme cases, changes at these boundaries can be significant enough to require rapid and substantial changes in program behavior rather than a gradual modification. For example, consider an automated car driving along a smooth road that hits a patch of ice. These Situational Discontinuities (SDs) require a radically different behavior, much more so than a simple thermostat.

So while techniques such as fuzzy logic can address the thermostat problem, SDs require an even more dynamic approach [McCarty 14b]. Hybrid approaches [Cheng 10], [Liu 07], algorithmic extensions and subsumptive processes [Yongjie 06] have each been used to address SDs but each comes with correspondingly higher degrees of complexity and resource-intensive requirements as a trade-off [Mendis 10].

Fuzzy Contexts, or Fuzzy Logic Type-C attempts to reduce both complexity and resource requirements of SDs through a novel use of fuzzy techniques and linguistic methods combined with an object-oriented approach. This is accomplished through a highly dynamic and flexible architecture which attempts to decompose SDs into smaller, more easily solved subspaces, each with associated fuzzy terms, membership functions and algorithms as illustrated in figure 11. Then the framework recombines the parts into whole system of “contexts” which describe processes at a higher, more human-understandable level.



**Figure 11 - Solving a Problem Space Using Multiple Approaches**

### **3.2: THE FUZZY LOGIC FOUNDATION**

For a more detailed description of Fuzzy Logic, Type-1 and Type-2, please refer to Appendix A.

Fuzzy logic, also known as Type-1 Fuzzy Logic, introduced by Lofti Zadeh [Zadeh 65] uses “uncertain”, rather than precise, descriptions for terms and allows for polyvalent

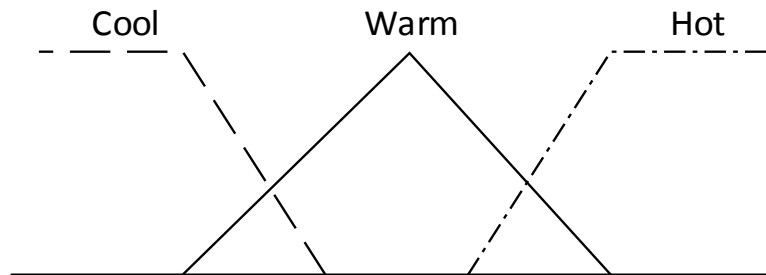
membership definitions. As compared to a “crisp” or traditional controller, a fuzzy controller allows for greater linguistic precision in describing a complex system behavior while at the same time relaxing precision around the boundary points and elsewhere. Recall this is done through the use of a membership function  $\mu$  whose output, instead of the traditional FALSE (0) and TRUE (1) allows for output of 0, 1 and all values in between. Thus, for a domain  $D$

$$\mu(x) \rightarrow [0, 1], x \in D \quad (3.1)$$

A fuzzy membership function defines a fuzzy set  $f_s$ , which can be described using a linguistic term such as WARM. A fuzzy set  $f_s$  is then a set of ordered pairs

$$f_s \equiv \{(x, \mu(x)) | x \in D\} \quad (3.2)$$

A fuzzy set can take any convex shape, with each fuzzy set depending upon its membership function. Triangles are one common shape. A fuzzy definition for the room temperature becomes a union of fuzzy sets as shown in figure 12.



**Figure 12 - Fuzzy Sets as Triangles**

Each fuzzy set contributes partially to the final result depending upon the resulting  $\mu(x)$ . Fuzzy algorithms are very good at approximating complex polynomials and provide stronger mechanisms for handling noise and uncertainty along with variations among “expert” definitions than their crisp cousins.

However, fuzzy logic also has limitations that pose new problems. Whereas the crisp algorithm has difficulty with the discontinuity at a boundary, likewise a fuzzy algorithm has trouble handling an SDP such as when an obstacle presents itself.

In the thermostat problem, a crisp solution could be improved by adding additional temperature tiers [Cox 94]. Likewise an SPD could be improved by the addition of fuzzy rules. However, adding tiers makes the temperature algorithm significantly more complex; likewise the addition of fuzzy rules adds significant additional complexity to a fuzzy solution [Mendis 10]. Just as Fuzzy Logic was necessary to solve the crisp boundary discontinuity problem, so there is a need for an approach to solve situational discontinuities within SDPs.

The underlying problem within fuzzy systems, and more generally, all approaches, is that problem domains are often more “solvable” using certain approaches than others. Within each of these specific problem areas often lies even more specific issues which require ever more specialized techniques.

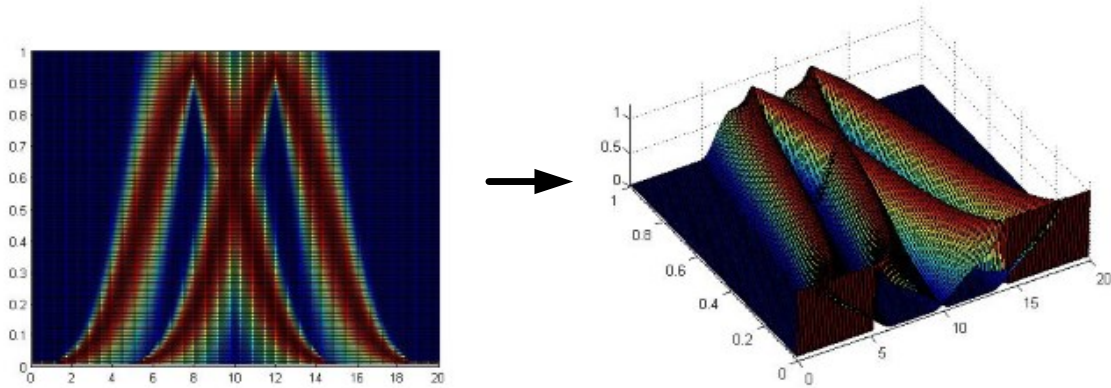
For instance [Linda 11b] demonstrates how a Fuzzy Type-1 controller was superior navigating around corners but inferior to a Fuzzy Type-2 controller navigating smoother surfaces. Even within a particular problem domain, one configuration of a Fuzzy Inference System (FIS) will be superior for handling a simple maze while another FIS is more appropriate elsewhere for obstacles.

A number of methods were introduced to extend fuzzy systems while also trying to limit the corresponding increase in complexity. Prior work in this area involves the use of Fuzzy Type-2 [Linda 11a], [Linda 11b], [Mendel 02], [Hagras 12] and Nonstationary Fuzzy [Garibaldi 08] sets and Polymorphic Fuzzy Signatures [Mendis 08], [Mendis 10].

Recall that Fuzzy Type-2 introduces uncertainty into the fuzzy sets themselves, in effect relaxing the boundaries of the membership function  $\mu_2$ , so in contrast to Equation 3.1:

$$\mu_2 = \{((x, \mu), \mu_2(x, \mu)) | \forall x \in D, \mu \in [0, 1]\} \quad (3.3)$$

Note also that output of  $\mu_2$  is also member of the set  $[0, 1]$ . Recall that whereas a Type-1 fuzzy set is a 2-dimensional object, Type-2 fuzzy sets are surfaces as demonstrated in figure 13.



**Figure 13 - Type-2 Fuzzy Sets**

A Type-2 fuzzy inference system is useful in dealing with problems such as extensive noise or smoothing out erratic behaviors that plague Type 1 controllers.

Recall that Nonstationary Fuzzy Sets (NFS) introduces the notion of variability of fuzzy sets over some dimension such as time, location, or even noise. The NFS  $nf_s$  is described as:

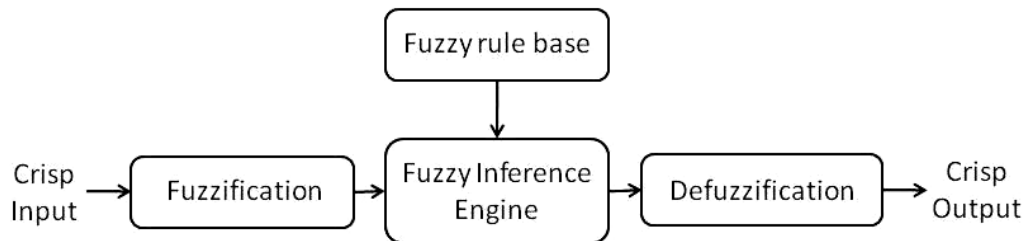
$$nf_s = \int_{d \in D} \int_{x \in X} \mu_{f_s}(d, x) / x / d \quad (3.4)$$

where  $d$  is some value along a dimension of the problem domain  $D$  and  $x$  is a tuple or point within the set of possible inputs  $X$ . Nonstationary Fuzzy Sets allow for a dynamic fuzzy membership function (and sets) able to accommodate significant changes to the problem space. A perturbation function adjusts the underlying membership functions as needs change. The NFS is then able to generate a variable FIS to handle changes in the problem space which otherwise might cause difficulties to a static Type-1 or Type-2 FIS.

The Fuzzy Logic Type-C, or Fuzzy Contexts is based upon precepts drawn from Object Oriented Programming (OOP), Fuzzy Logic and Radial Basis Functions (RBFs). Among the core goals of OOP is to make programming simpler and more reusable by abstracting common base functionality across disparate objects [Pressman 09]. This technique is called “polymorphism” and is accomplished via class inheritance as well as

through the use of interfaces. Fuzzy Contexts seek to extend polymorphism using fuzzy constructs.

One of the most common and arguably successful implementations of fuzzy logic is its combination with rule-based systems, generally referred to as a Fuzzy Inference System (FIS) [Cox 95]. The FIS performs a nonlinear mapping from an input data vector to a scalar output. It is typically composed of defined fuzzy sets combined with linguistic terms, a fuzzy rule base and a Defuzzifier as shown in figure 14. In a typical FIS, input is fuzzified from crisp numbers to fuzzy values. These values are then run through a fuzzy inference engine and test against the rule base to determine a corresponding rule “contribution”. Contributions are recombined in the defuzzification process to produce a crisp results.



**Figure 14 – A Typical Fuzzy Controller**

Work by [Kirillov 14], [Slavicek 13], [Octave 13] and others has produced a number of useful software frameworks for implementing a Type-1 FIS (T1-FIS). Others [Karnik 14] have also produced frameworks for implementing a Type-2 FIS. Fuzzy Contexts builds upon their work and others such as the [Moreno 12] introduction of a Fuzzy Definition Language called XFSML. The XFSML allows for a storage and retrieval of Type-1 fuzzy components in order to implement a dynamic T1-FIS. This paper takes the XFSML a bit further by introducing a novel Algorithm Definition Language (ADL) for more generalized algorithm definitions including Type-2 as well as a database implementation for both Type-1 and Type-2 ADLs.

Fuzzy Contexts borrows much of its technique from the methods described here.



### 3.3: OVERVIEW OF FUZZY CONTEXTS IN DETAIL

A Fuzzy Context,  $\hat{C}_i$  is a multidimensional object consisting of a crisp centroid core and a fuzzy shell [McCarty 14b]. The boundaries of the core and shell are expressed as a series of points and/or function defining the convex boundaries of the core,  $\rho$ , the inner fuzzy shell,  $\sigma$  and the outer fuzzy shell,  $\tau$ .

The core serves as an algorithmic classifier. Any and all inputs that lie within the core are considered to be completely addressable by the corresponding context algorithm. For example, an image of a face might be broken down into contexts for the eyes, nose and skin. A contextual core for the context EYE might consist of the pupil and iris only since they are easily distinguishable from other parts of the face as demonstrated in figure 15.

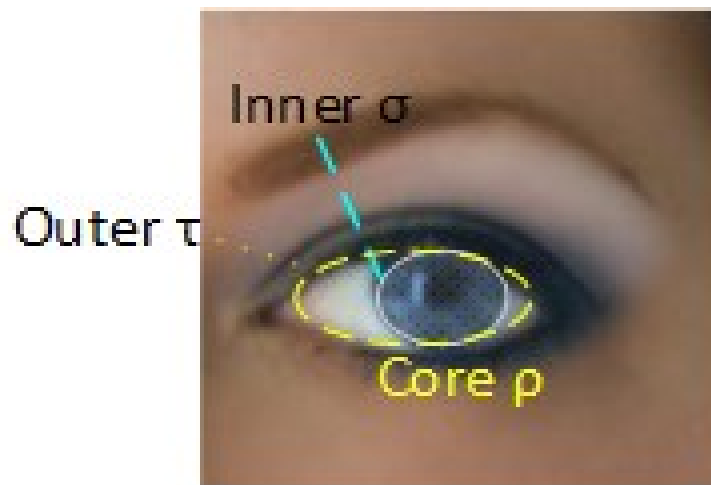


Figure 15 – Deconstructing an eye into  $\rho$ ,  $\sigma$ ,  $\tau$

Hence the core,  $\rho$ , is defined by a set of points and a function  $\rho_i(x)$  that defines the boundaries of the core.

$$\hat{C}_{i\rho} \equiv \rho_i(x), x = [x_1, \dots, x_l] \quad (3.5)$$

The  $\sigma$  defines the outermost boundary of the core. It is where the fuzziness begins to happen as (similar to the RBF) the contribution of the algorithm diminishes as a point moves

further from the core down to zero. In the case of the eye it is where the iris ends and the sclera begins as shown in figure 15. It is defined similarly to  $\rho$ .

$$\widehat{\mathcal{C}}_{i\sigma} \equiv \sigma_i(\mathbf{x}), \mathbf{x} = [x_1, \dots, x_m] \quad (3.6)$$

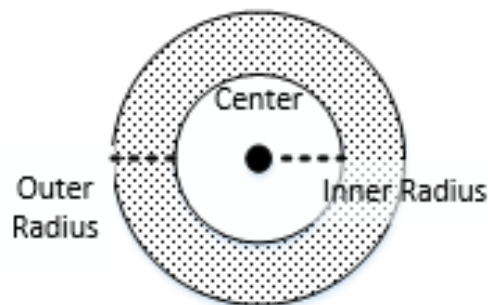
Finally the outermost boundary of the outer shell is where algorithm applicability falls to zero. In the case of the eye in figure 15, it is where the sclera meets the eyelid. Since the sclera and eyelids are different for each eye, it is advantageous to define a certain fuzziness where the two come together. The outer shell defined by  $\tau$  is constructed similarly to the others.

$$\widehat{\mathcal{C}}_{i\tau} \equiv \sigma_i(\mathbf{x}), \mathbf{x} = [x_1, \dots, x_n] \quad (3.7)$$

Hence any given contextual shape,  $\gamma$  for any context  $\widehat{\mathcal{C}}_i$  is a function of series of points  $x$ ,  $y$  and  $z$  and their associated constructs:

$$\widehat{\mathcal{C}}_{i\gamma} \equiv \rho_i(\mathbf{x})\sigma_i(\mathbf{y})\tau_i(\mathbf{z}), \mathbf{x} = [x_1, \dots, x_l], \mathbf{y} = [y_1, \dots, y_m], \mathbf{z} = [z_1, \dots, z_n] \quad (3.8)$$

In simplest form (aside from a singleton) the context consists of a centroid point, an inner radius defining an inner sphere and outer radius describing the distance between the boundaries of the inner and outer spheres. The resulting fuzzy context looks like a solid sphere within a fuzzy sphere such as shown in figure 16.



**Figure 16 – Simple Fuzzy Context**

There is no restriction to the boundaries of the shells except that along any line intersecting the context the Euclidean distance from the centroid to the inner shell is less than the distance to the outer shell.

$$\sqrt{\|\widehat{\mathbf{C}}_{iy}^c - \widehat{\mathbf{C}}_{iy}^{ir}\|^2} \leq \sqrt{\|\widehat{\mathbf{C}}_{iy}^c - \widehat{\mathbf{C}}_{iy}^{or}\|^2} \quad (3.9)$$

One other constraint that must be observed, similar to the intermediate value theorem for an ascending function, is that given two points  $a, b$  within a context and a point between them  $c$ , along some line through the centroid, the membership value of  $c$  must lie between the membership values of  $a$  and  $b$  and not decrease the closer you move to the center.

$$\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \mid f_d(\mathbf{a}) \leq f_d(\mathbf{c}) \leq f_d(\mathbf{b}) \Rightarrow \mu(\mathbf{a}) \leq \mu(\mathbf{c}) \leq \mu(\mathbf{b}) \quad (3.10)$$

In standard form the context takes as input a vector and returns a scalar output.

$$\widehat{\mathbf{C}}_{iy}(\mathbf{x}) = \mu_{iy}(\mathbf{x})\varphi_i(\mathbf{x}), \mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_k] \quad (3.11)$$

Where  $\mu_{iy}(\mathbf{x})$  is the corresponding fuzzy membership function for the context  $\widehat{\mathbf{C}}_{iy}$  and  $\varphi_i(\mathbf{x})$  is its assigned algorithm. Also similar to RBF and Artificial Neurons, the Type-C architecture can combine the individual contexts into a network, although it is not a requirement. Hence a Type-C implementation combines all the individual contexts over the universe of discourse.

$$\widehat{\mathbf{C}}(\mathbf{x}) = \sum_{i=1}^n \mu_{iy}(\mathbf{x})\varphi_i(\mathbf{x}), \mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_k] \quad (3.12)$$

Membership evaluation is similar to the signature method employed by PFS, for a given  $\widehat{\mathbf{C}}_{iy}$  with centroid  $\mathbf{x}_c$  is as follows:

$$\mu_{\widehat{\mathbf{C}}_{iy}} = \begin{cases} \mathbf{1}, & \|\mathbf{x} - \mathbf{x}_c\| \leq r_{inner} \\ \mu_{iy}, & r_{inner} < \|\mathbf{x} - \mathbf{x}_c\| < r_{outer} \\ \mathbf{0}, & \text{otherwise} \end{cases} \quad (3.13)$$

The Fuzzy Context also has the capability to “invert”, in which case membership gradually goes to 0 as a point moves closer to the centroid. This is useful in creating subsumptive behavior [Yongjie 06] where generic behaviors are overridden by specific ones.

$$\mu_{\hat{c}_{iy}} = \begin{cases} \mathbf{0}, & \|\mathbf{x} - \mathbf{x}_c\| \leq r_{inner} \\ \bar{\mu}_{iy}, & r_{inner} < \|\mathbf{x} - \mathbf{x}_c\| < r_{outer} \\ \mathbf{1}, & \textit{otherwise} \end{cases} \quad (3.14)$$

### 3.3.1: How Fuzzy Contexts extend Fuzzy Logic

Fuzzy Contexts, or Fuzzy Type-C extends the concepts of the Fuzzy Logic, Polymorphic Fuzzy Signatures (PFS) and Nonstationary Fuzzy Systems (NFS) into a rich framework supporting the use of multiple, distinct approaches to solve varied problems. Recall that Fuzzy Type-1 seeks to make it easier to describe and implement complex polynomials via approximation using a series of fuzzy sets and linguistic terms. Similarly, Fuzzy Contexts attempt to make it easier to describe and resolve complex problem spaces via a series of space decompositions, each combined with an algorithm and linguistic “context”. This particular construct serves to differentiate Type-C implementations from traditional NFS, PFS or even hybrid systems, for example, Neuro-Fuzzy. In a traditional hybrid or other approach, the resulting hybridization is in reality, still a single algorithm; albeit a more complex one. The resulting problem space expands, but is still finite, being bounded by the hybrid approach. Moreover, just as a Fuzzy Logic set provides a readily understandable description via linguistic terms, so also does Type-C provide a natural linguistic context to a problem space, rendering a more precise definition of the original problem context. Note also, that for a Fuzzy Context associated with an algorithm reduces to the algorithm (such as a Type-1 FLC) within the core,  $\rho$ .

Consider, for example, an application to do facial recognition. Features, such as the eyes and mouth, require a different approach than the hair or chin. A neuro-fuzzy hybrid approach attempts to recognize features and determine the appropriate algorithmic approach by creating a series of fuzzy rules to determine which features predominate and respond accordingly [Gomathi 10]. It may look for skin-tone, hair color, eye color, etc. and decide how to proceed based upon the resulting fuzzy rules. The limitation is that each feature requires adding an entirely new set of rules, yet each set of rules is functionally equivalent to every other set.

Suppose our facial recognition software is very simple has rules for skin-tone. There might be rules (consisting of fuzzy sets) such as DARK and LIGHT and TAN. The test of skin-tone requires testing 3 fuzzy sets. Easy enough, but what if the person is flushed from running to the subway on a hot afternoon. Adding rules to account for flushed faces means adding the FLUSHED fuzzy set and increasing the rules to (at least) FLUSHED AND DARK,

FLUSHED AND LIGHT, FLUSHED AND TAN, DARK, FLUSHED, TAN. Now suppose the face is sweaty. Now there has to be at least 12 rules: SWEATY AND FLUSHED AND DARK, FLUSHED AND DARK, SWEATY AND DARK, DARK and so on.

Each of these rules must be tested unless even more logic is applied to trim the sets of rules. The problem is that logic, say if face is SWEATY then only use rules that have SWEATY, still by limited a lack of context. SWEATY when hot is not the same as SWEATY when cold (CLAMMY). Adding temperature (HOT or COLD) just makes the problem that much more complicated, particularly when it comes to deciding what underlying fuzzy sets to use. Is there a definition for SWEATY that will work for both hot and cold situations? What of the underlying logic? Can the algorithm reliably trim rules without losing some of the intended precision?

Analogous to Zadeh showing how fuzzy sets can simplify a complex polynomial, fuzzy contexts can simplify the facial recognition through the use of contexts. In the example above DARK, LIGHT and TAN, FLUSHED, SWEATY comprise a core fuzzy inference system, with different contexts for temperature. Each contexts has its own FIS, adapted specifically for the context. Such a contextual system might read like this

Context – HOT DAY

DARK, LIGHT, TAN, FLUSHED, SWEATY (sets adapted for skin tone when temperature is hot)

Context – COLD DAY

DARK, LIGHT, TAN, FLUSHED, SWEATY (sets adapted for skin tone when temperature is cold)

Similar to Type-1, the resulting solution is determined by the corresponding contribution of each context. This is achieved by calculating the membership of a given tuple in the context. In short, a Fuzzy Context is an extensible, hierarchical framework that uses a fuzzy membership function to associate a specific algorithm to a context within a problem space. Fuzzy Logic Type-C combines Fuzzy Contexts and their corresponding problem spaces into a polymorphic algorithmic approach.

Type-C is also intended to also provide a relatively light-weight framework around any particular algorithm. Test results will show that a Type-C implementation add only a

small amount of overhead and can actually reduce the processing requirements of an implementation in some cases.

### 3.3.2: What is Fuzzymorphism?

The three pillars of object-oriented programming are encapsulation, inheritance and polymorphism [Purdum 12]. Of these, polymorphism is the least understood and most underutilized. Polymorphism is derived from the Greek, meaning “many shapes” or forms and is understood in computer science to be a technique whereby an object can assume different forms or attributes. These forms depend upon the type of descendent object that was instantiated but appear outwardly to the caller to be identical.

As a way of thinking about it, consider the science of Taxonomy [CLT 03], [Taxonomy 14]. Taxonomy is simply a methodology to group living organisms together based upon things they have in common. Similarly, polymorphism seeks to create objects which can be used interchangeably based upon a common interface. The difference is that whereas in Taxonomy, we consider physical traits, in Polymorphism we are more interested in function. For example consider an “animal”. It can be whatever you like but think about the functions the animal can perform that are common to all other animals.

One example of a common function is motion. All animals have the capability to move to and fro, even though they have very different ways to go about it. Another is “speaking”. Virtually all animals can make some sort of sound when necessary. An object-oriented animal therefore might have two functions: MOVE and SPEAK. The implementation of MOVE and SPEAK, however, differ based upon the actual animal instantiated.

For example, consider the DOG object and DUCK object, both derived from an ANIMAL class. We implement a MOVE function (run) and a SPEAK function (bark) for DOG. We implement a MOVE function (waddle) and a SPEAK function (quack) for DUCK. Later as part of a collection of ANIMAL objects (call it a ZOO), we have a DOG and a DUCK. We can then perform a loop:

***FOR EACH ANIMAL IN ZOO***

***ANIMAL.MOVE***

***ANIMAL.SPEAK***

***NEXT ANIMAL***

**(3.15)**

Here's where the power of polymorphism becomes apparent. When the calls to `ANIMAL.MOVE` and `ANIMAL.SPEAK` occur, the compiler checks for the *kind* of animal `ANIMAL` happens to be at the moment and then makes the appropriate underlying function call. The calling function, however, does not have to know anything about the `ANIMAL` other than it can speak and move, therefore it can remain blissfully ignorant of HOW it should speak or move. Thus polymorphism allows the caller to be able to work with animal objects without having to know anything about the specific types of animals. A simple example of polymorphism is illustrated in figure 17.



**Figure 17 – Polymorphic action SPEAK for DUCK vs. DOG vs. CAT**

Fuzzymorphism takes polymorphism into the realm of fuzzy logic. Just as a fuzzy value is to a bivalent crisp value, so a fuzzymorphic function is to a polymorphic one. Take another look at the `DUCK` object. Earlier, we suggested that `MOVE` was a waddle, but is that really true? How does a `DUCK` really move? A `DUCK` can paddle *on* the water, swim *in* the



water, waddle on land, and fly through the air; therefore a more complete DUCK is really a series of sub-objects:

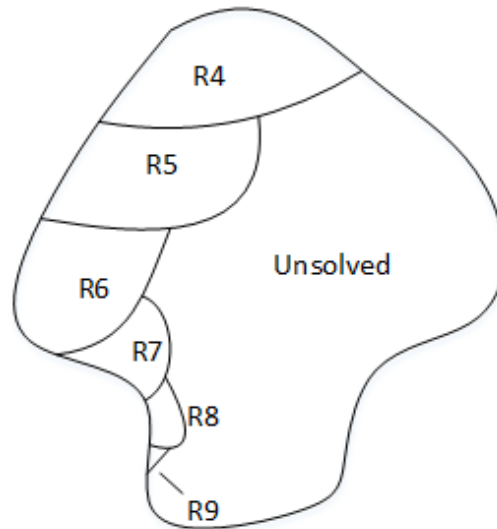
- a. DUCK\_UNDER\_WATER
- b. DUCK\_ON\_WATER,
- c. DUCK\_ON\_LAND,
- d. DUCK\_IN\_AIR.

The traditional polymorphic ANIMAL function MOVE for the DUCK object is now either DUCK\_UNDER\_WATER.MOVE, DUCK\_ON\_WATER.MOVE, etc. But what about when a DUCK is taking off or landing? In traditional polymorphism, a function is of a certain type (say, waddle) or it is not. A fuzzymorphic function, however, can consist of multiple algorithms, hence it can be part waddle *and* part fly for a context TAKING OFF FROM LAND or part fly *and* part paddle for a context LANDING ON THE WATER or anything else that makes contextual sense.

Fuzzymorphism is then able to handle transitions from one context to the next. It can also handle any space where membership within one or more contexts is ambiguous. It does this by applying algorithms fractionally, depending upon a current input's membership within corresponding contexts.

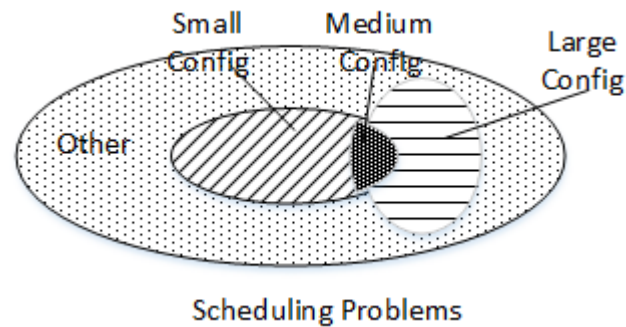
### 3.3.3: Advantages of Fuzzy Contexts and Fuzzymorphism

Traditional approaches take an existing algorithm and try to generalize over a larger problem domain. Typically this is accomplished by adding complexity to the algorithm. For instance, a neural network adds neurons while a fuzzy controller adds fuzzy rules, both at a cost of complexity to an algorithm with correspondingly diminishing returns [Mendis 10]. For example a simple fuzzy controller designed to solve a navigation problem can perform quite well with a small number of rules. Adding new rules gives the controller more capability, but each new rule expands the solution on a smaller and smaller scale. Conversely each new rule greatly increases the system's complexity [Mendis 08]. Even within a problem space suited for a particular approach, an algorithm can still fall victim to problems of complexity and diminishing returns as illustrated by figure 18.



**Figure 18 - Adding Fuzzy Rules Results in Diminishing Returns**

Furthermore, there are times when multiple approaches may be equally worthy at certain stages of a process. Consider the example of a scheduler - one with a small number of possible configurations may be best served with a global ranking system; while a larger number of configurations may require some sort of local search technique [Russell 09]. The effectiveness of these different approaches can overlap, creating an intersection of subdomains as demonstrated in figure 19.



**Figure 19 - Overlap of Techniques for a Scheduler**

In this situation, either approach is acceptable. More importantly, the union of the two spaces gives the combined algorithms a larger surface area with less overall complexity than trying to extend either approach separately. The problem lies in determining the situation, or

“context” in which to apply one algorithm or the other. For another example of how a context applies, albeit in a different way, consider an inventory control problem at a department store. Seasonal contexts dictate which items are most important to maintain inventory and how much. As before with the scheduler, an inventory control system needs to account for the season, or context, in order to be most efficient, this time at maintaining inventory levels.

Living things incorporate contexts quite well, we are naturally in tune with our external situation; but applying contexts to artificial processes requires a mechanism to both identify a given context as well as the best algorithmic behavior to apply, along with behavior at transition points where the “best” algorithm is ambiguous. Fuzzy logic provides a useful foundation for exploiting this imprecision and creating and using contexts [Zadeh 08]. With Fuzzy Type-C, diverse problem spaces can be combined without sacrificing the simplicity and power of individual problem solving techniques. Consider a problem space  $P$  over a domain  $D$ . It consists of a collection of states  $s$ , which is a tuple of  $s_i$  values, each  $s_i$  value belonging to  $D$ .

$$\mathbf{P} \equiv \{s = s_1, \dots, s_n, s_i \in D\} \quad (3.16)$$

An algorithm  $a$ , such as a Type-2 FIS operates on  $P$  taking as input an  $s_p$  and generates a result  $r_p$ .

$$\mathbf{a} \equiv f(s), f(s_p) = r_p \quad (3.17)$$

A fuzzy “signature” is a collection of problem states, upon which the algorithm works efficiently, hence:

$$\mathbf{f}_{sig} = \sum_{i=1}^n \mathbf{a}_{s_i}, s \in \mathbf{P}, s_i \in D \quad (3.18)$$

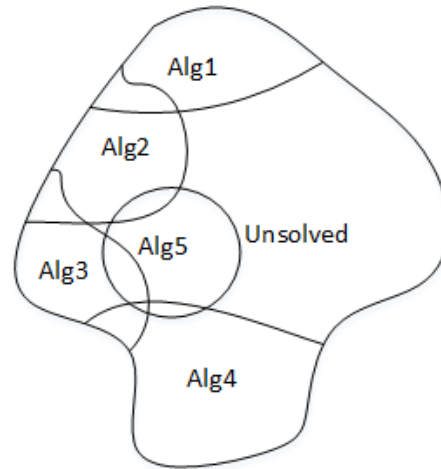
The “context” is the combination of the algorithm, the signature and all the associated states along with a membership function  $\mu_c$ .  $\mu_c$  determines membership within a given context of any particular state  $s_i$ .

$$C_i = \{f_{sig_i}, a_i, \sum s_i, \mu_{c_i}, s_i \in P, \mu_{c_i}(s_i) \in [0, 1]\} \quad (3.19)$$

The Type-C FIS contains all the contexts associated with  $P$ .

$$C = \sum_{i=1}^n C_i \quad (3.20)$$

Figure 20 demonstrates how this approach might handle a complex problem domain.



**Figure 20 - Solving a Problem Space Using Multiple Approaches**

Hence, Fuzzy Type-C encapsulates multiple problem solving approaches by associating a “signature” of an environment with a distinct Type-1, Type-2 FIS or other algorithm and all the potential states the algorithm was designed to address.

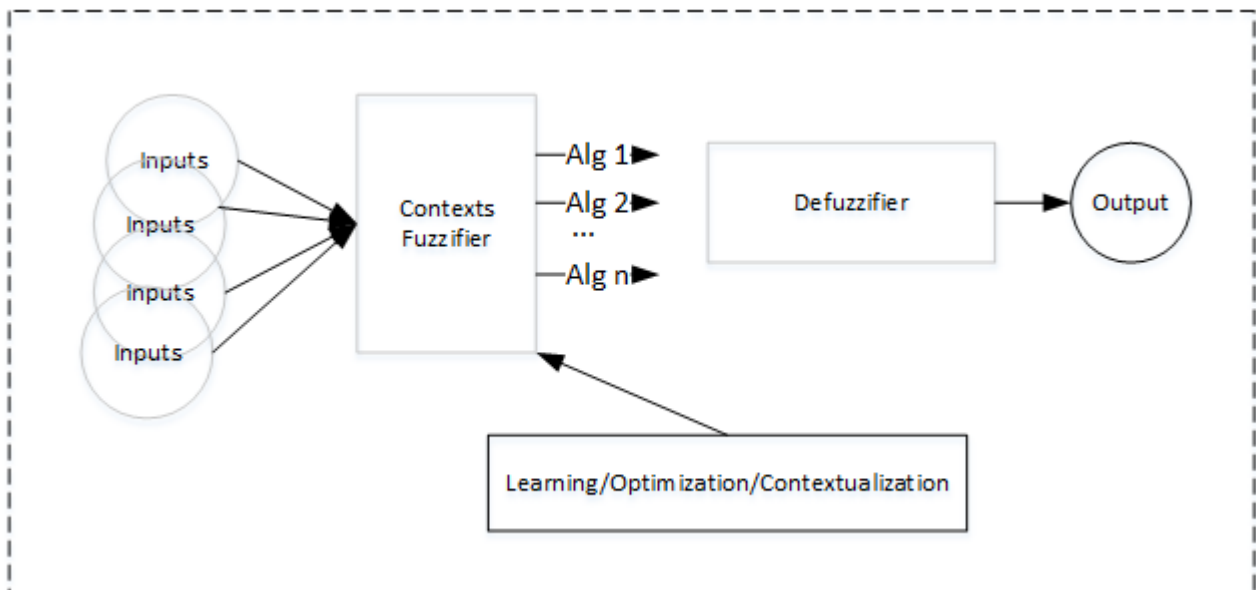
Because the new contexts can be added as a problem space expands, a Type-C FIS allows an expansion of a problem space into a larger domain without having to overly generalize.

The Type-C architecture consists of the following major components:

1. A set of inputs as a tuple
2. A series of contexts. Each context consists of:
  - a. Fuzzy signature
  - b. Membership function
  - c. Algorithm that receives the input tuple and returns a result.
3. Results fuzzifier

4. Optional optimizer/contextualizer for dynamic optimization and automated learning. It determines if the error rate is acceptable, otherwise will strive to optimized an existing context or generate a new context.
5. Defuzzifier that takes fuzzified output and generates a crisp result.

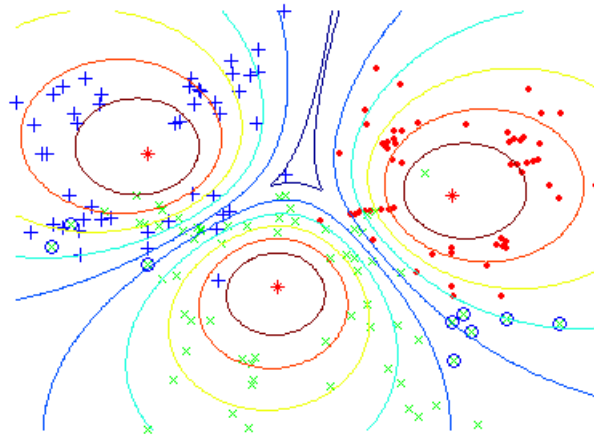
The architecture is illustrated by figure 21.



**Figure 21 - Fuzzy Context Architecture**

The Contexts Fuzzifier component uses a technique similar to that of fuzzy classification to determine membership of a context. Unlike in crisp sets where a data point is either in or not in a set, fuzzy classifications allow a point to have membership in multiple sets as shown in figure 22.

Fuzzy clusters are very useful in creating “transition” sets from one problem space to another within a domain. Transitions are those in-between areas where contexts overlap, where any single algorithm is not likely to generate the best result. Likewise fuzzy clusters can determine membership of a given tuple with a transition set for a given context while allowing for membership in multiple contexts. Cluster, and context, creation and membership is determined using techniques such as discussed in [Ming-Chuan 01].



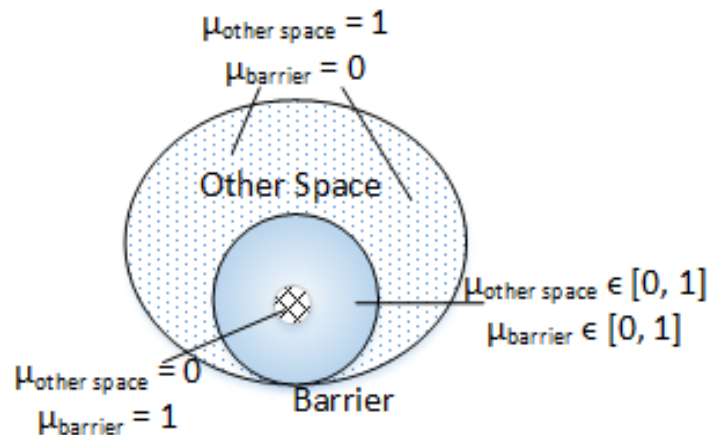
**Figure 22 - Points in Fuzzy Clusters**

At runtime a Type-C FIS determines the “contextualization” of each input tuple using each context’s corresponding membership function. Any context whose membership value is greater than zero will have its corresponding algorithm run and its output fuzzified. Defuzzification is achieved using traditional fuzzy methods

$$\mathfrak{R} = \frac{\sum_{i=0}^n w_i a_i(s_i) \mu_{c_i}(s_i)}{\sum_{i=0}^n \mu_{c_i}(s_i)}, s_i \in D, \mu_{c_i}(s_i) > 0 \quad (3.21)$$

Where  $\mathfrak{R}$  is crisp result,  $s_i$  is an input tuple in a domain  $D$ ,  $w_i$  is the weight,  $\mu_{c_i}$  is the context membership and  $a_i$  is the intrinsic function for any context whose  $\mu_{c_i}$  is greater than zero.

Fuzzymorphism occurs when contexts overlap as might occur in a problem space similar to that in figure 23. As a state moves away from the center of one context and closer to the center of another, the resulting defuzzification will take on more of the characteristics of the closest underlying context algorithm, hence a Type-1 FIS might slowly morph to a Type-2 FIS for example. Figure 23 illustrates the concept.

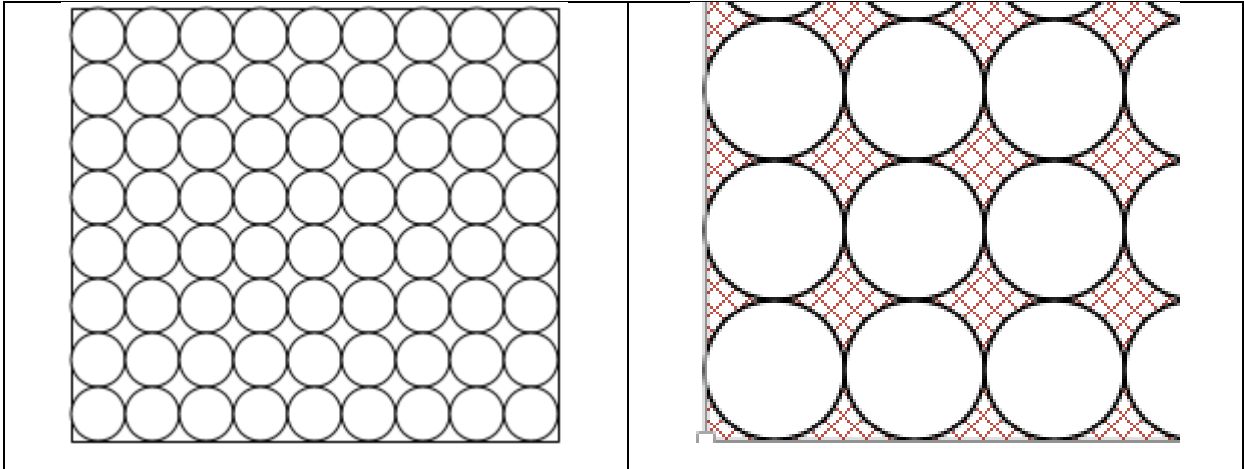


**Figure 23 - Type-C membership over a Problem Space**

Fuzzymorphism allows a Type-C based system containing multiple approaches to dynamically “morph” into the one most suitable for the problem at hand. In the case of multiple Type-1 FIS, Type-C performs similarly to a Nonstationary Fuzzy System. However, because Type-C is algorithm independent, the framework will support any algorithm capable of accepting the input tuple and producing a corresponding output, allowing for a much more diverse approach.

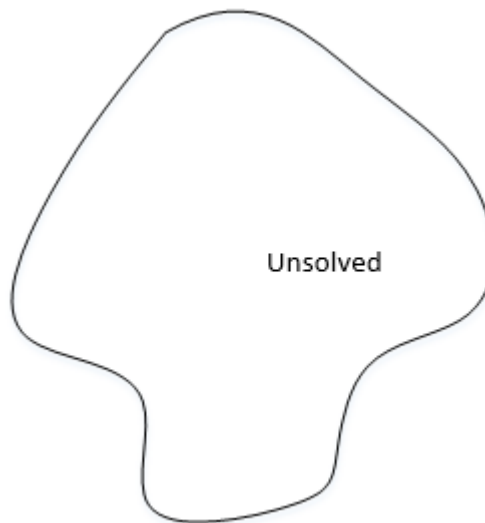
#### 3.3.4: Is there a Packing Problem?

The Packing Problem occurs when trying to fit a series of “objects” into a container such as one shown in figure 24. No matter how tightly packed the circles are, they will invariably leave some space (indicated by the hash marks in 24b).



**Figure 24 a, b – Circles in a rectangular container leads to “Packing Problem”**

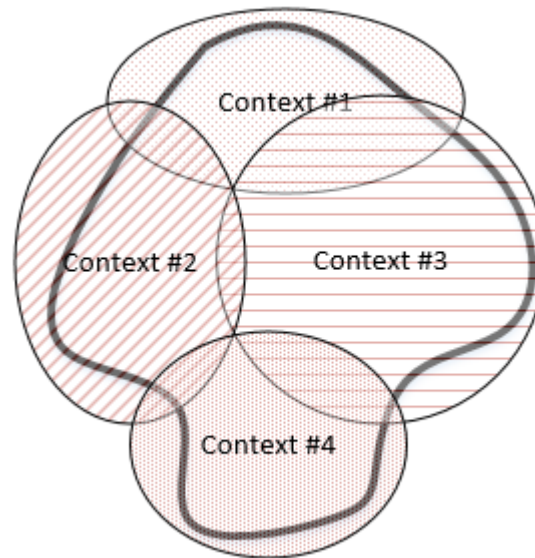
The aim of a packing solution is to pack the container with maximal density [Lopez 11]. When looking at Fuzzy Contexts, the question of whether the contexts can completely cover a problem space arises. It becomes a packing problem. Again look at the problem space example in figure 25.



**Figure 25 – Example Problem Space**

Do Fuzzy Contexts have to worry about the Packing Problem? The answer is no, for 2 separate reasons: 1) Fuzzy Contexts can overlap, hence it is relatively easy to overcome the limitations suffered by a series of regular objects as shown in figure 26.





**Figure 26 – Fuzzy Contexts overlap and cover a problem space**

Second, Chapter 4 will show how Fuzzy Context can take many different shapes and sizes, so long as the shape is convex, including a single point. So even if no overlap was allowed there exists a covering space (although it may not be finite) of convex shapes or points.

### **3.4: IMPLEMENTATION OF BASIC FUZZY FRAMEWORK AND ARCHITECTURE**

#### **3.4.1: Introduction**

Describing the behaviors of complex system presents many challenges for the software architect and developer. Foremost among them is the ability to model behaviors that are by their very nature imprecise [Zadeh 65] or dependent upon external factors not directly related to a specific process state [McCarty 08a]. Another vexing challenge is how to develop complex systems able to address a variety of differing environments, each with its own special requirements and methods for addressing them.

Consider the duck. It is a fowl with an uncommon ability to move on land, in the air, on top of the water and under the water. In each scenario, it is able to make use of its feet, wings, tail and other body parts as appropriate for locomotion; yet in each case these body parts may execute their functions quite differently. Moreover, the duck must also manage a

number of transition phases: 1) under/over water, 2) over water/air, 3) over water/land, 4) land/air. Each of these phases has an inverse for a total of 8 possible transitions. In the transition phase the duck must rapidly or gradually adjust the behavior of these body parts in order to make the transition from one form of locomotion to another.

Complex software systems face some of the same challenges as the duck. Because complex machines, such as robots, automated cars or even search engines often need to be able to adapt to a changing environment on the fly, they need to be able to modify their overall behavior by modifying the behavior of their underlying component systems. This includes not only being able to handle major changes of state, but also the transition phases between states where different component systems must exchange primary or secondary roles. As with the duck, hybrid behaviors and contributions must be smoothly adjusted in order that the transition move successfully from state to state. What is needed is an architecture that can do this but which allows complex descriptions and hybridization techniques to be encapsulated in a relatively easy human-understandable component.

Recall that Fuzzy logic is well suited for human systems interaction because they provides a natural way of implementing linguistic terms from human experts. This section introduces the basic fuzzy framework for building a Type-1 fuzzy controller. Fuzzy Logic presents a number of advantages in dealing with complex systems already described. Perhaps less appreciated is the way Fuzzy Logic encapsulates variables and rules as linguistic terms. This allows for descriptions which are human-readable and easier to relate. This mechanism works well in defining contexts, or problem spaces with particular characteristics and plays an important role in the framework.

Later enhancements will add the necessary components for a full-fledged Type-C implementation, but initially a traditional fuzzy framework was used to serve as the base and starting point for further development. The benefits of this framework are: 1) Reduced development time; 2) A standardized and portable codebase; 3) Support for Type-1 fuzzy sets and rules.

Every framework needs an application. Complex systems, such as robots, frequently incorporate a multitude of diverse inputs in order to perform a given task. Each input may or may not influence a system's behavior depending upon other inputs and/or system states. For example, a robot that senses a wall nearby will behave differently depending upon its speed,

direction of motion, goals, other walls, etc. Due to the potential wide variety of possible conditions and responses, fuzzy logic, in the form of a Fuzzy Inference System (FIS) provides an excellent implementation option and use for this framework. The FIS provides the foundation for Type-1 Fuzzy Logic Controllers (T1-FLCs) used to control a complex system and is desirable for the following reasons:

1. Fuzzy logic describes both input and output behaviors in human understandable terms.
2. Because real-world factors such as sensor noise, actuator variations and environmental factors add elements of uncertainty to the interpretation, T1-FLCs (and T2-FLCs) are often the most practical approaches available to describe and deal with the uncertainty.
3. Fuzzy logic often provides a much simpler way to describe and approximate complex behaviors than polynomial functions.

T1-FLCs operate on the principal that inputs and outputs can be “fuzzified”; that is, defined not as a specific number or boundary, but instead as a range of values defining varying degrees of membership between 0 and 1 [Cox 94]. Type-1 fuzzy logic does have its limitations, such as managing input noise and outliers.

Clearly, practical applications of T1-FLCs are everywhere. Despite this, there has been only limited progress in using FLCs for mainstream, commercial applications. Programmers often have little to no appreciation for, or even understanding of, fuzzy techniques and how they might be applied to solve common problems.

One way to address this problem is through the use of a software framework designed to support creation and maintenance of an FIS - a fuzzy framework. Software engineering methods rely on the use of frameworks to improve the quality of software while also reducing complexity and development costs [Limbu 11], [Pressman 09], [Busc 09]. Software frameworks, usually in the form of software class libraries, DLLs or other reusable software, allow developers to focus on solving a specific problem rather than spending time writing generic routines [Pressman 09].

Fuzzy frameworks do exist and are available as commercial offerings from sources such as Matlab and LabView. These offerings provide very sophisticated functionality as add-on toolkits to their core product line. Open source frameworks such as AForge.Net

[Kirillov 14], Sourceforge.net, [Octave 13] and CodeProject [Slavicek 13] provide another option for developers interested in creating their own custom solutions. Despite the availability of both commercial and non-commercial frameworks, limitations of both continue to inhibit more widespread use and adoption of FLCs in general. Among these limitations is the difficulty for novices in trying to understanding the various fuzzy objects and how they relate and how to implementing a working FIS in code [Wyne 12]. Applications wizards can help with this, along with a suitable modeling language for configuring FLCs dynamically [Moreno 12]. If the Type-C Framework (TCF) is to prove useful, it should also provide some tools to help developers understand how to implement it.

### 3.4.2: Need for a General Fuzzy Framework to Support Type-C

All the frameworks, including the starter TCF, described in section 3.4.1 above are unsuitable, however, for a Type-C based implementation. They do not support Fuzzymorphism of Fuzzy Contexts in any sense, nor provide the necessary software hooks for dynamic algorithm implementation or hybridizations. A new approach to a fuzzy architecture is needed. One that supports:

1. The ability to define and use Fuzzy Contexts
2. The ability to plug in algorithms on the fly as contexts merge and change
3. The ability to define and load algorithms via a definition language
4. The ability to add other enhancements such as optimization techniques
5. Integration with a relational database.

There are other issues as well to consider with existing frameworks. Commercial products, such as Matlab, have significant cost and limited portability to other languages. Their frameworks require purchase of the core product, often at an extra cost of thousands of dollars, and may involve a steep learning curve for use. Furthermore, portability issues make adding a Matlab Fuzzy Toolkit FIS to a generic web application based upon a language such as Visual Basic.NET a significant undertaking. Open source solutions, on the other hand, provide few, if any, useful tools for configuration and implementation which will need to be addressed, but at least they provide a good place to start.

The TCF proposed in this dissertation began life as an open source, Type-1 library from [Kirillov 14]. It is written in C#, a variant of C which has wide adoption in the computer science and business community. C# is also among the most popular programming languages for businesses using the Windows platform. The original framework was strictly designed as a general purpose library to implement a Type-1 FLC. The author then proceeded to overhaul and upgrade the software (and fix a few bugs) in order to support the many enhancements required to implement a Type-C based controller as described in the next section.

### 3.4.3: Framework Architecture

Object-oriented techniques have long been recognized as a way to reduce the complexity of software [Pressman 09], [McConnell 04]. As a starting point the framework architecture consists of a number of objects in support of a typical Fuzzy Inference System (FIS), most notably:

1. Membership Functions
2. Fuzzy Sets
3. Fuzzy Variables
4. Fuzzy Rules
5. A Rules Database
6. Defuzzifier

There are 8 steps necessary to implement the framework in order to create a functioning FIS:

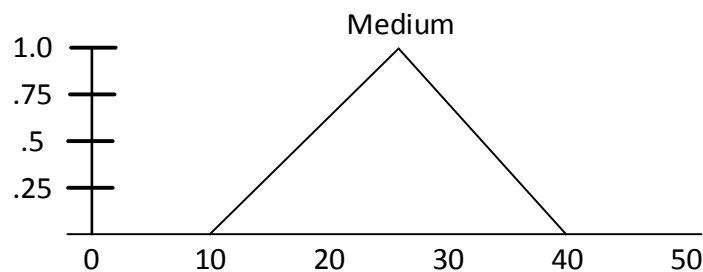
1. Define Membership Functions - Define each fuzzy function, or shape used to generate the fuzzy sets
2. Define Fuzzy Sets – Define each fuzzy sets using a linguistic term along with the corresponding membership function
3. Define Fuzzy Variables – Define each variable using a linguistic term, add the corresponding fuzzy set definitions
4. Assign Variable Inputs and Outputs – Assign variables to their corresponding input and output collections

5. Define Fuzzy Rules – Create the fuzzy rules by assigning fuzzy variables and antecedent/consequent terms
6. Define Defuzzifier – Specify type and number of intervals used for defuzzification
7. Assign Rules To Rules Collection – Add rules to the rules collection
8. Define FIS – Create FIS using Rules Collection and Defuzzification object

Step 1 implementation requires specifying a convex shape for a membership function  $\mu$ .  $\mu$  is a function over a domain  $D$  such that for each  $x$  in  $D$ ,  $\mu(x)$  is a number between 0 and 1.

$$\mu(x) \rightarrow [0, 1], x \in D \quad (3.22)$$

Zadeh's rules for fuzzy sets require the functions take the form of a convex shape such as a triangle, trapezoid, Gaussian or other convex curve. The resulting shape serves as the membership function  $\mu$  with the horizontal or x-axis supplying the range of values for which membership is defined with the vertical or y-axis providing the membership value as demonstrated in figure 27.



**Figure 27 – Triangle-shaped fuzzy membership function**

In the framework, each shape,  $f_{shape}$  is defined with one or more boundary points  $P_{bk}$  and one or more apex points  $P_{ak}$ .

$$\mu = f_{shape}(P_{b1}..P_{bn}, P_{a1}..P_{an}) \quad (3.23)$$

The boundary points specify where the membership function becomes a constant. All points to the left of the left boundary have the same membership value as the left boundary (normally 0 or 1, indicating no membership or full membership respectively), likewise all points to the right of the right boundary have the same value as the right boundary. The Apex point(s) specify the topmost point(s) of the shape along with the membership value (usually 1). For each point P in the shape, the X value specifies a specific value x in the domain while the Y value specifies the corresponding membership value between 0 and 1 at that value,  $\mu(x)$ .

$$f_{shape}(P_{b1}, \dots, P_{bn}, P_{a1}, \dots, P_{an}) = \mu(x), x \in D \quad (3.24)$$

In the framework, a fuzzy set  $f_s$  is defined by a membership function and a linguistic “term” which describes its purpose in a more easily understandable way. Giving the fuzzy set a “linguistic term” allows use of more intuitive language when describing subsequent rules in the FIS. For instance, a fuzzy set that determines *tallness* might be called “Tall”. Later fuzzy rules will reference that particular fuzzy set with the term *Tall*. This actually adds a certain level of precision to the fuzzy set that is very difficult to emulate using crisp sets or numbers and is one of the major advantages to using fuzzy logic [Zadeh 65], [Cox 95]. The framework will determine fuzzy set membership of any point using the membership function assigned to the corresponding fuzzy set.

Hence, step 2 simply requires applying the function defined in step 1 to a linguistic term to generate a fuzzy set  $f_s$ .

$$f_s = \mu + \textit{linguistic term} \quad (3.25)$$

Step 3 involves creation of at least two fuzzy variables, a minimum of 1 each for input and output variables. Each Fuzzy Variable is described by a “name”, another linguistic term to appropriately describe the variable’s purpose or function, along with a range of applicable inputs. Variable names should make sense within the FIS, describing in easily understood terms what that variable represents. The boundary of the domain of a particular variable should be large enough to encompass all of the fuzzy sets to be assigned. Once the domain is defined and the variable “termed”, the variable then is associated with one or more of the

fuzzy sets defined in step 2. These associations are used to determine which fuzzy sets are represented in the fuzzification/defuzzification process.

In step 4, the variable is assigned to its intended input or output collection. Input variables can then be used to construct the antecedent while output variables are used for the consequent. Antecedents and consequents are used to build fuzzy rules.

During the fuzzification process membership is within the variable depends upon the membership values of the underlying fuzzy sets as described in step 2.

Step 5 specifies the fuzzy rules that constitute the FIS. Each fuzzy rule consists of an antecedent which is a statement of the form:

$$\mathbf{IF \langle fuzzy\ variable \rangle IS \langle fuzzy\ variable\ or\ fuzzy\ set \rangle} \quad \mathbf{(3.26)}$$

The antecedent specifies a testable condition similar to a crisp IF statement, except instead of a true or false result, the fuzzy result consists of a value between 0 and 1 inclusive, dependent upon the input value and the various membership or fuzzy functions underlying the corresponding fuzzy sets shown as in Equation 3.25. Antecedents can be combined using AND/OR and parenthetical operators. For example, to test the distance of a barrier to the front of an obstacle the antecedent might take the form of:

$$\mathbf{IF\ FrontalDistance\ IS\ Far} \quad \mathbf{(3.27)}$$

“FrontalDistance” can consist of multiple fuzzy sets, for example: “Near”, “Medium” and “Far”. The antecedent “IF FrontalDistance IS Far” looks at the membership function of the fuzzy set “Far” assigned to the fuzzy variable “FrontalDistance”. Note again the use of linguistic terms that are easily understandable even to laypersons. A parser within the framework turns the text into its corresponding fuzzy sets and fuzzy variables.

The fuzzy rule also requires a consequent, which is constructed similarly to the antecedent but uses output variables and sets.

In step 6, the user defines the number of fuzzy intervals used for defuzzification. The framework currently supports traditional Zadeh rules for fuzzification of fuzzy Type-1 where



membership of a fuzzy variable is equal to the minimum membership of the corresponding fuzzy sets. This is also referred to as a fuzzy intersection of fuzzy sets.

$$\cap \mu_{C_i} = \mathbf{min}(\mu_{C_1}, \mu_{C_2}, \dots, \mu_{C_n}) \quad (3.28)$$

Defuzzification is achieved by then taking a fuzzy union across all intervals in the domain. This is accomplished by taking the maximum across the sets of intervals and their corresponding memberships.

$$\cup \mu_{C_i} = \mathbf{max}(\mu_{C_1}, \mu_{C_2}, \dots, \mu_{C_n}) \quad (3.29)$$

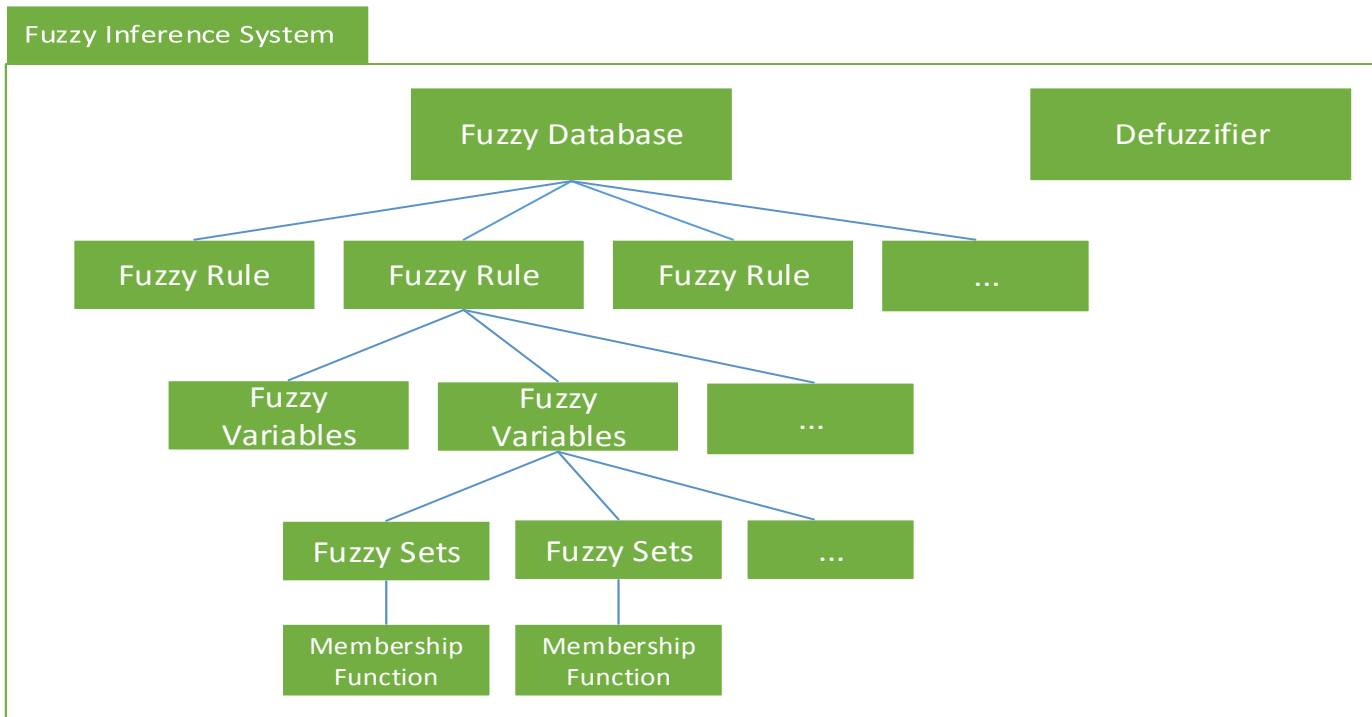
A centroid, or center of gravity, is then calculated by determining a weighted mean across the fuzzy region [Cox 94]. The fuzzy solution region  $\mathfrak{R}$  is calculated by the following:

$$\mathfrak{R} = \frac{\sum_{i=0}^n d_i \mu_A(d_i)}{\sum_{i=0}^n \mu_A(d_i)} \quad (3.30)$$

where  $d$  is the  $i^{\text{th}}$  domain value and  $\mu(d)$  is the membership value returned the corresponding fuzzy function defined in step 1.

In step 7, the user creates a database of the fuzzy rules defined to use in the FLC. Depending upon which rules are input and which are output, the FIS will attempt to evaluate all relevant rules during the fuzzification/defuzzification process.

Step 8 is performed by the framework. The resulting FIS then consists of the database repository of all the relevant fuzzy objects as well the domain space and fuzzy operators used. The overall architecture is described by figure 28.



**Figure 28 – Fuzzy Framework Architecture**

#### 3.4.4: Structural Changes to Support Type-C

The original framework had a number of shortcomings the needed to be initially addressed:

1. Limited extensibility – resolved by adding additional interfaces and abstract classes to provide generic functions, inputs and outputs. For example:
  - a. A generic Fuzzy Object class was added to provide the ability to accept generic inputs of any type, an ID and Parent value to allow for the creation of a bidirectional hierarchy of fuzzy objects and some basic database functionality.
  - b. Fuzzy sets and membership functions along with various related objects such as the fuzzified strength values and fuzzy operators were generalized.
  - c. All the affected higher level classes were rewritten to accommodate the new generalizations.

2. Demo software – major upgrades were needed to expose the underlying fuzzy values during operation along with a number of additions of various UI configuration inputs.
3. Configuration changes – required to implement the new software as previously implemented.

Other modifications, such as the introduction of a number of enumerated types, name changes and other refactors were made to improve the readability of the code.

Why all this extra work? The next chapter will describe the Fuzzy Logic Type-C (FLC) architecture. One of the advantages of FLC is its ability accept virtually any configuration. This is achieved via a high-level abstraction of the fuzzy framework itself to allow it to accept any “algorithm”. Recall that an algorithm according to [Cormen 09] is defined as “any well-defined computational procedure that takes as input as value or set of values and produces a value or set of values as output”. “Any” algorithm need only conform to that definition and nothing else to qualify. To achieve this, however, requires abstracting not only the algorithm, but also the input and output. Unlike typical reflection, which implies certain case operations against a particular type and requires a certain knowledge of the underlying object, the Context simply passes the input and collects the output.

More import to the Fuzzy Context is the applicability or membership of the current state of the input within the contextual problem space. In this way, algorithm integrity is preserved internally, but hybridization or Fuzzymorphism can still occur within the overall context of the current problem. In this fashion Type-C functions similarly to something like an Artificial Neural Network or Radial Basis Function Network but with added flexibility in the definition and handling of the context against the problem. So the work was necessary in order to support the much more comprehensive capabilities of the Type-C implementation.

### 3.5: GUIDELINES FOR CREATING FUZZY CONTEXTS

Creating a context depends upon the underlying nature of the problem, but in general a problem is decomposable into contexts by following a few basic guidelines using the following steps:

**Step 1** – Determine whether a default context is needed for unknowns. If so, define.

**Step 2** – Create known contexts.

- a. Define Center
- b. Define Inner Shape
- c. Define Outer Shape
- d. Determine the Fuzzy Membership Function
- e. Label the Context

#### Step 1: Determine Default Context if Necessary

In step 1, the developer must determine whether or not to define a *default* context. While it might be desirable to predefine fuzzy contexts if all contexts are known, many problems do not provide that luxury. In that case, it is advantageous to have a *default* context, which covers some aspect of a problem space, ideally, the largest *known* context. From there one can define other contexts in relation to the initial default context and other previously defined contexts. Imagine a robot car on a smooth road. The default context “smooth road” applies to the known road condition where it is just a plain road with no anomalies. This context is special in that it is the only context that does not require the typical contextual components such as the shape definition or membership function. The reason is that its shape and membership are actually dependent upon the results from non-default contexts. If no other contexts are defined then the underlying algorithm of the default context is simply *the* problem solution.

#### Step 2a: Define Center

**Step 2** involves the creation of other contexts. Now imagine the robot car suddenly encounters a patch of icy road such as might occur in figure 29.



**Figure 29 – A Patch of Ice on a Road**

Because the driving behavior is so dramatically different on an icy road, a new context definition is desired. Looking at the picture, it is tempting to want to create a context *topologically* by simply overlaying it on top of the icy figure.

$$\widehat{\mathcal{C}}_{i\rho} \equiv \rho_i(\mathbf{x}), \mathbf{x} = [x_1, \dots, x_l] \quad (3.31)$$

Where  $\widehat{\mathcal{C}}_{i\rho}$  is the new context and  $\rho_i(\mathbf{x})$  is the function defining the boundaries of the inner core over boundary points  $x_1 \dots x_n$ .

Because it consists of exclusively icy points, the context would be of contextual singleton form. The corresponding fuzzy membership function would then reduce to a crisp function:

$$\begin{cases} 1 \forall x \in \widehat{\mathcal{C}}_{i\rho} \\ 0 \text{ otherwise} \end{cases} \quad (3.32)$$

Because the above definition is of a contextual singleton, the center point is not really necessary in this case, as the fuzzy membership function is a constant. While this is a perfectly acceptable decomposition of this problem space, there may be other patches of ice ahead which would require additional contexts. In order to deal with them all, it is more advantageous to create a context *compositionally*, that is, reflecting the underlying composition of ice and road. This has the advantage of simplifying the context into the basic concentric spheres described earlier. In this case, the spheres are defined by a center, inner radius and outer radius. It is also preferable, though not required, to create a membership function that is smooth and continuous, but also reflective of the “real world” situation.

Hence, a more comprehensive solution requires a *center*. Recall the *center* is defined as the point of greatest applicability for an algorithm. In the case of the icy road, the easy answer is 100% icy AND 0% road. This is a good definition since the road can't get any more "icy" than 100%.

### Step 2b: Define Inner Shape

Because the inner shape is a sphere, step 2b only needs to define the inner radius. Any input within the inner radius always has membership of 1. Determine the inner radius by considering all possible situations where input is best served exclusively by underlying contextual algorithm. Experiential data might indicate that a road that is 50% or more "icy" is effectively all "icy", so, in this example, the inner radius is best defined as the distance from 100% to 50% or simply, radius is 50 units.

### Step 2c: Define Outer Shape

In this case, the outer shape is also a sphere which means the only requirement is to define the outer radius. Because a road with 0% ice is effectively "not icy" the outer radius is 100 units. Recall, however, that membership is calculated over the area of the outer radius that is not covered by the inner sphere, so the area of interest lies from length 50+ to the edge of the outer sphere.

### Step 2d: Define Fuzzy Membership Function

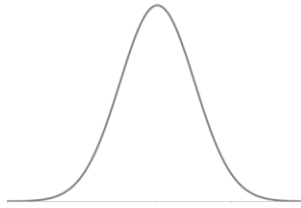
Defining the fuzzy membership function becomes a matter of determining a smooth, continuous function which allows for a gradual increase from 0 at the outer edge of the outer sphere to 1 at the outer edge of the inner sphere. In this case something like the following Gaussian creates a smooth gradient.

$$f(x) = ae^{\left(-\frac{(x-b)^2}{2c^2}\right)} + d \quad (3.33)$$

In the case of a fuzzy membership function, ranging from 0 to 1, the value of  $a = 1$  so we get a more generalized expression:

$$f(x) = e^{\left(-\frac{(x-b)^2}{2c^2}\right)} + d \quad (3.34)$$

Which looks like figure 30:



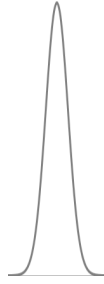
**Figure 30 – Simple Gaussian**

For an icy road, however, the presence of ice requires a more extreme sensitivity to icy behavior. In fuzzy parlance, the normal fuzzy Gaussian might require a hedge [Cox 95] such as “VERY” applied to the underlying equation. Decreasing the value of the standard deviation,  $c$ , applies the hedge and narrows the Gaussian to look more like the following figure:



**Figure 31 – Gaussian with Fuzzy Hedge**

Or even more extreme:



**Figure 32 – Gaussian with Extreme Fuzzy Hedge**

So given  $x$  as the percent icy composition of the road, and a standard deviation of .1 the resulting fuzzy membership function  $\mu_{icy}(x)$  would be as follows:

$$\mu_{icy}(x) = \begin{cases} e^{-\frac{(x-50)^2}{.02}}, & 50 < x \leq 100 \\ 1, & x \leq 50 \\ 0 & otherwise \end{cases} \quad (3.35)$$

#### Step 2e: Label the Context

Apply the term *Icy Road* for linguistic clarity and apply a fuzzy inference system designed for icy roads as the underlying algorithm and the context is complete.

Using the Type-C approach allows the new *Icy Road* context to work seamlessly with the default “Smooth Road” context because the robot car has a way to apply the two contexts as appropriate. Let  $f_{icy}$  represent the output from “Icy Road”. The contribution context *Icy Road* is a function of the output \* membership function of the corresponding context. Thus, *Icy Road*'s overall contribution could be defined as:

$$\mu_{icy}(x)f_{icy}(x) \quad (3.36)$$

And because “Smooth Road” is a default context its contribution would be:

$$1 - \mu_{icy}(x)f_{icy}(x) \quad (3.37)$$



In cases where there are more than 2 fuzzy contexts with membership for a particular input, contributions need to be normalized. Suitable methods for doing this are part of future work.

## CHAPTER 4: FUZZY CONTEXTS EXTENSIONS AND ENHANCEMENTS

### 4.1: ARCHITECTURAL ENHANCEMENTS

#### 4.1.1: Enhancement #1 – Adding Fuzzy Type-2 Support

Since a Type-C framework (TCF) is designed to manage multiple algorithms or algorithmic configuration, a natural first step in the implementation of a true TCF was extending the original Type-1 architecture to support Fuzzy Logic Type-2. However, despite outward similarities, Type-1 and Type-2 implementations are quite different. Whereas a Type-1 fuzzy set is a 2-dimensional object, a Type-2 fuzzy set represents more of a 3-dimensional surface. All of the various inputs and outputs for a Type-2 FIS are different as is the defuzzifier.

At a minimum, implementation of a fuzzy controller requires building a Fuzzy Inference System (FIS). The FIS consists of a number of software components:

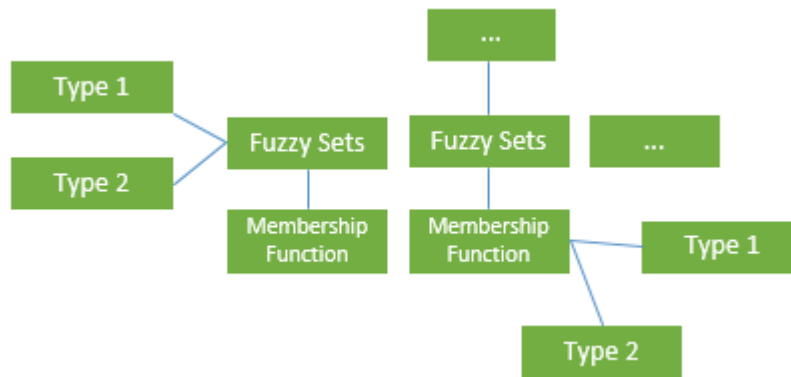
1. Fuzzy Sets (with Membership Functions)
2. Linguistic terms to define those sets
3. Collections of rules defining the behavior of a fuzzy system.
4. A Defuzzifier

The framework introduced in [McCarty 13] implemented basic Type-1 and Type-2 FIS. Type-2 functionality was added by initially creating a whole new set of classes for Type-2 to perform items 1-4. This was not a good long-term solution since the code was poorly integrated.

The goal was to find commonality between the two types and refactor accordingly. At a high level, drilling into an FIS looks somewhat like this:

FIS->Rules->Variables->Fuzzy Sets->Membership Functions

A Type-1/2 framework can find commonality from at least the Variables level on up. Hence the new Type-2 extensions were created by inheriting Type-1 objects at the Fuzzy Set level and below, and creating abstract classes to generalize the Fuzzy Set so that the Variables and above would work as before. Compare the Type-1 framework architecture and note the differences when implementing a combined framework.



**Figure 33 – Original Fuzzy Framework Architecture with Type-2 Extensions**

Not shown here are the additional changes to related objects, such as fuzzy operators, fuzzified values for firing strength and the defuzzifier. The new architecture allows for either fuzzy 1 or 2 implementations, but the high-level variables and rules, where “humans” interact, remain consistent with the original Type-1 construct.

#### 4.1.2: Enhancement #2 – Adding Fuzzy Type-C Support

Now that the groundwork was laid by adding various abstractions of inputs and outputs, adding Type-C support was a matter of adding the ability to define a Type-C object, assign an algorithm and membership function, and determine a way to add inputs and get outputs.

To address these issues the TCF architecture underwent a number of modifications. The primary goal of the architecture, also referred to as a software framework, is now to facilitate the creation of real-time software systems able to define and transition between states requiring significant changes in component behavior as well as describe behavior using linguistic fuzzy terms.

The new architecture consists of a number of discrete components; in particular a class library defining classes and interfaces for implementing a Type-C solution. Recall that complex software systems are presented with some interesting challenges. Among them:

1. How best to integrate disparate processes often requiring substantially different algorithmic approaches.

2. How best to describe a complex process both at the macro and operational level.
3. How to create a dynamic system where diverse algorithms can operate interchangeably or in a hybrid fashion.

The TCF architecture attempts to accomplish the above by a novel use of object-oriented techniques combined with fuzzy logic methods to create a “fuzzymorphic” inference system using Fuzzy Contexts.

Pseudo-code for the Type-C implementation which combines all the contexts, taking as input vector and returning a scalar value follows:

#### Pseudo-code 4 - Determine a Context Contribution

```
// Take vector input, apply all context contributions to
// generate scalar output

Algorithm: CombineContextContributions (x)
Input: a tuple x
Output: scalar result
Begin
1  FOR all  $C_i$  in C
2   set contribution =  $\mu_{i\gamma}(x)$ 
3   IF contribution > 0
4    set y += contribution * weighted result  $w_i\varphi_i(x)$ 
5  NEXT  $C_i$ 

6  return y
End
```

Similar to Nonstationary Fuzzy Systems (NFS), this allows for an input-based, dynamic operation, although fuzzy sets are not required as they are in NFS. The resulting operation enables one or more algorithm to hybridize into a solution. As opposed to being polymorphic, where algorithms operate on an either/or basis, the fuzzymorphic contextual algorithm has a fluid, fuzzy-like transition phase, dependent upon the membership values of the individual contexts.

The FLC was used to revisit and build a Type-C implementation of a robot vehicle able to traverse a simple maze shown in figure 34.



### 4.1.3: Enhancement #3 – Adding Relational Database Support

From the days of the abacus until the middle of the 1950s “computers” primarily served as computing aids [Augarten 84]. Little thought was given to the storage and persistence of application data simply because the technology for data storage was impractical [Zetta 14]. That began to change with the introduction of hard disks, tape drives and other kinds of storage devices. Suddenly computers became more than computing machines; they became data repositories. The traditional data-crunching mainframe/terminal client eventually gave way to distributed client-server architectures and finally to the cloud. Databases have gone from a specialized storage/retrieval option to a very common, almost indispensable component of many applications.

This is due to the growing power and flexibility of relational database systems and tools. Gone are the days when extensive knowledge of Standard Query Language (SQL) commands was required to query some department-level, multi-million dollar system. Today, low-end relational databases such as MySQL and SQL Server Express are available for free and the tools have sophisticated visual components such as SQL Server Management Studio, relieving developers of the tedium of command-line driven SQL [Davidson 12].

While not a necessary component of advanced processes such as a Fuzzy Inference System or Genetic algorithm, a relational database can nevertheless, add substantial power and capabilities to an existing application that might not normally use one. For instance, by adding database extensions, an application can now store configuration information in a database instead of a file. This has many advantages over using a file-based configuration such as XML. One major advantage is being able to quickly search for configurations with specific criteria. For instance, suppose we wanted to locate the XML configuration with fuzzy set  $FS$  and membership function  $MF$ . We would have to write a function to iterate through all of our XML files one at a time, looking at each fuzzy set element until we found  $FS$  with  $MF$ . How long this takes depends upon how many files to parse through before finding the one with the elements we want.

Not so with a database configuration. A simple SQL Statement will find the any and all configurations almost immediately (i.e. milliseconds). Unlike a slow parser, the following SQL might be all it takes:

***SELECT \* FROM Configuration  
WHERE FuzzySet = 'FS' and MembershipFunction = 'MF'*** (4.3)

Whereas finding *all* of the configurations with these characteristics would require an exhaustive search of all the XML configuration files, the database search will do this automatically. The database also handles all the management of record location and identification behind the scenes relieving the programmer of the burden to keep track of file names and locations. Looking at it another way, it is yet another abstraction, this one in the form of data storage that allows developers to focus on a more specific problem without having to delve too deeply into storing and retrieving associated data.

More than ease of storage, search and retrieval, databases systems also provide huge advantages in the actual manipulation of large datasets, such as that as might be generated by a complex FIS like the one in chapter 3 and this chapter. While a typical application uses a row-by-row sequential approach (in database parlance, it is called a cursor), a typical commercial relational database is designed to store, retrieve and manipulate data in sets using Standard Query Language (SQL). Relational databases are built for speed, using techniques natively that would be very difficult and time-consuming for a typical application to duplicate.

As an example, consider a simple application. All it does it scan through a list of objects stored in memory. It runs on a 24 Gb RAM computer with an installation of SQL Server 2012 Developer Edition. The following comparative analysis was performed on 4 different configurations:

1. SQL Server 2012 cursor-based operation
2. Application (running in debug mode) cursor-based operation
3. Application (running compiled) cursor-based operation
4. SQL Server 2012 set-based operation

Each configuration performed the following 4 operations:

1. Scan through the list and count each object to get a total count.
2. Scan through the list and count each object which has a specific value, or filter.
3. Scan through the list and update a particular value of each object to another value.
4. Scan through the list and update a particular value of each object to another value if the object has a specific value, or filter.

The application was tested against an in-memory (pre-instantiated and initialized) object with 3 fields with a total length of approximately 10 bytes, as might be generated from a simple FIS. The database was tested against a table of records, each record being approximately 10 bytes long and consisting of 3 columns.

Each of the 4 operations was performed on 10,000 rows/objects, 100,000 rows/objects, 1 million rows/objects, 10 million rows, objects and 100 million rows/objects.

**Table 1 – Application vs. Database Comparative Analysis – 10,000 rows/objects**

	DB Cursor	App Debug	App Compiled	DB Set
Count	.360s	.156s	.133s	.003s
Count w/Filter	.470s	.156s	.136s	.003s
Update	8.63s	.156s	.134s	.050s
Update w/Filter	8.64s	.156s	.130s	.050s

**Table 2 – Application vs. Database Comparative Analysis – 100,000 rows/objects**

	DB Cursor	App Debug	App Compiled	DB Set
Count	3.200s	1.481s	1.318s	.010s
Count w/Filter	3.203s	1.513s	1.353s	.010s
Update	1m 27.537s	1.500s	1.339s	.353s
Update w/Filter	1m 26.840s	1.534s	1.334s	.423s



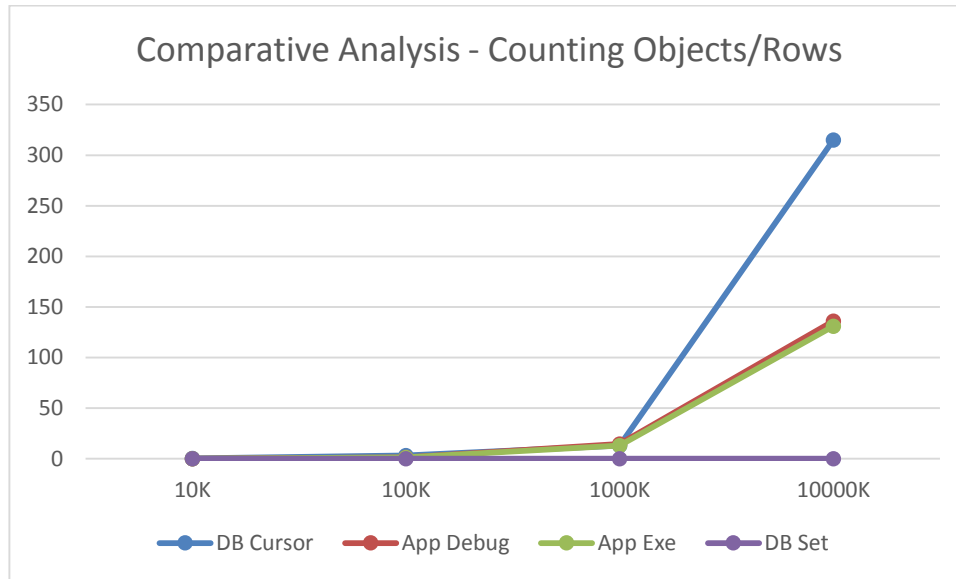
**Table 3 – Application vs. Database Comparative Analysis – 1,000,000 rows**

	DB Cursor	App Debug	App Compiled	DB Set
Count	31.440s	14.630s	13.176ss	.010s
Count w/Filter	32.807s	14.999s	13.816s	.010s
Update	14m 36.517s	14.906s	13.320s	.353s
Update w/Filter	15m 8.160s	16.507s	13.952s	.423s

**Table 4 – Application vs. Database Comparative Analysis – 10,000,000 rows**

	DB Cursor	App Debug	App Compiled	DB Set
Count	5m 15.416s	2m 16.069s	2m 11.711s	.153ss
Count w/Filter	5m 25.737s	2m 16.379s	2m 15.692s	.793s
Update	1hr 15m 59s	2m 19.240s	2m 12.384s	15.760s
Update w/Filter	1hr 14m 31s	2m 36.427s	2m 11.230s	19.477s

Attempts to test 100 million rows/objects failed because the application itself was unable to allocate enough memory to proceed, even with the SQL Server and all other applications shut down on the computer. The database, on the other hand, had no issues with 100 million rows or 1 billion rows or even 10 billion rows. Figure 35 shows how the set based counting operation has a very slight upward gradient compared with the other approaches, despite rowsets that vary by 4 orders of magnitude. This same relative performance was duplicated in the other 3 tests.



**Figure 35 – DB Set Operations Contrasted with Application and Database Cursor Operations**

Even for fairly small numbers of rows the database engine clearly outperformed the application. Moreover, the database engine was not nearly as constrained by available memory as the application and is able to, without special modification, handle datasets greater in size by at least 2 orders of magnitude. However, there is no “free lunch” which is demonstrated by the abysmal performance of the database engine when performing the same functions using a sequential or cursor-based approach. This is an important trade-off: if an operation can be done as a set-based query, the database engine is likely to be far more adept, otherwise it will much slower [Bolton 12].

By storing the data in an easily retrievable format on an RDBMS, an application can defer to the database engine to perform set-based operations on the entire dataset at once rather than record by record. It can also perform data mining and analysis, store results and make calculations that would otherwise be done by the client application [Davidson 12].

Adding database extensions to the Fuzzy Framework simplifies what would otherwise be a much more difficult operations, as will be demonstrated in later sections.

The SQL Server 2012 relational database engine used for the extension is based upon database concepts developed by E.F. Codd at IBM [Date 11], [Powell 05] to address the problem of custom data storage and retrieval. Codd developed a set of rules for relational

databases that became the standard implementation for the majority of databases in commercial use today [Oracle 10]. RDBMSs quickly became the preferred tool for backend data storage due to their flexibility and power. As such they add a new dimension to the power and intelligence of traditional applications.

For a more extensive treatment of relational databases see the Appendix F at the back of this dissertation.

## **4.2: FUNCTIONAL ENHANCEMENTS**

### **4.2.1: Enhancement #4 – Adding Memetic-Based Optimization**

With the framework enhancements in place, it was time to look at a number of functional enhancements. Optimized fuzzy processes are a particularly worthy, yet tricky goal. Recall that local search routines such as hill-climb and simulated annealing are used for finding optimal solutions when it is impractical to review every possible configuration. By picking a random starting point and following a gradient to a nearby maximum or minimum, a local search routine is very useful for fuzzy set optimization. Evolutionary algorithms, such as the memetic algorithm take local search a step further by combining a gradient search with natural selection [Eiben 07].

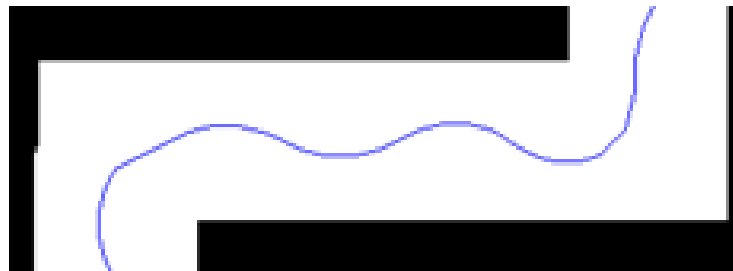
One drawback with earlier approaches is the limited use of relational databases for fuzzy optimization [McCarty 13]. By allowing the relational database to handle many of the processing tasks of the memetic algorithm, the FIS can dynamically create and test various fuzzy implementations automatically. It can evolve and determine which combinations are most optimal using batch operations instead of iterating measurement by measurement; which, as demonstrated in a previous section, is often slower by orders of magnitude. Batch operations are designed for simultaneous processing of large datasets and are generally more efficient than traditional value by value calculations over large sets.

The database was built to handle four major tasks of the memetic algorithm

1. The Fitness Function
2. Gene Sampling and comparison
3. Crossover Mutation

#### 4. Random Mutation

Looking back at the navigation problem (also see test examples 1-3), the reader might notice a lot of back-and-forth motion as the controller tries to maintain a “safe” distance from all of the barriers. Optimization for this FIS involves reducing the back and forth motion correction in the navigation as demonstrated in figure 36.



**Figure 36 – Sample Back and Forth Corrections Navigating Maze**

This means reducing the variations in the output values. The Fitness Function then must determine whether or not a particular decision is more optimal than another by comparing each FIS’s decision and how far it deviates from an optimal path. Determining the optimal path requires the following steps in pseudo-code:

#### **Pseudo-code 5 - Determining the Optimal Path**

**Algorithm:** DETERMINE\_OPTIMAL\_PATH (values)

**Input:** values, a set of inputs into FIS and final result

**Output:** a set of points along the optimal path

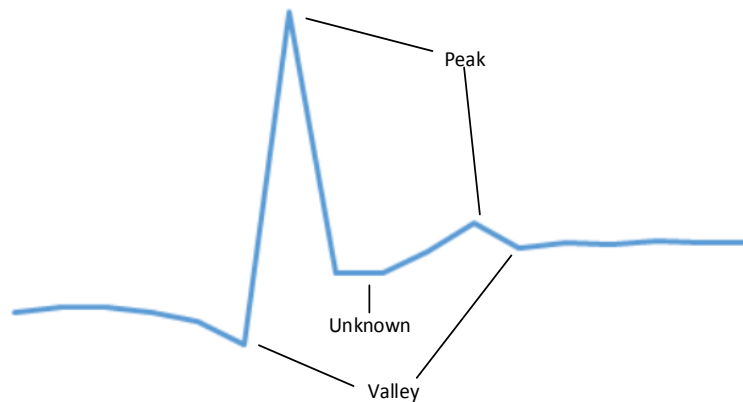
**Begin**

- 1 Record all changes as the FIS operates. In this case, as the “robot” tries to navigate the path.
- 2 Determine each “peak” and “valley” in the sequence.
- 3 Take the midpoint between each peak/valley pair.
- 4 return the set of midpoints

**End**

For Step 1, the program logs all inputs to the FIS and resulting outputs to the database. From there the application needs to get an appreciation of the amount “deviation” undertaken. This is most easily done by looking at the various highs and lows of the changes (1<sup>st</sup>

derivative). Now think of a map with standard terrain features: peak, valley, plateau and depression. Step 2 defines a “peak” as any output value surrounded on both sides by lesser values. In other words, a peak is a local maximum. One problem is the existence of plateaus, where the same maximal value exists in a sequence indicating a “flat” surface. In that case, take the median value and declare it the peak. Do likewise for valleys and depressions using local minimums as shown in figure 37.



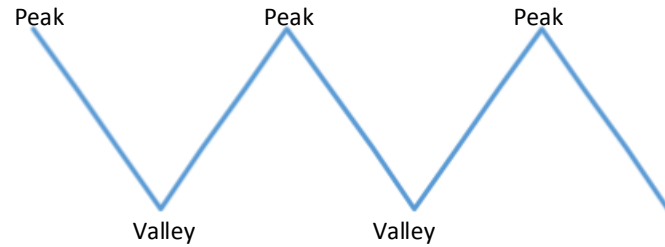
**Figure 37 - Determining Peaks and Valleys**

Note that each peak must be surrounded by valleys and vice versa so whenever peaks are adjacent to peaks such as the situation shown in figure 38 the greater peak will eliminate the lesser. Likewise for valleys.



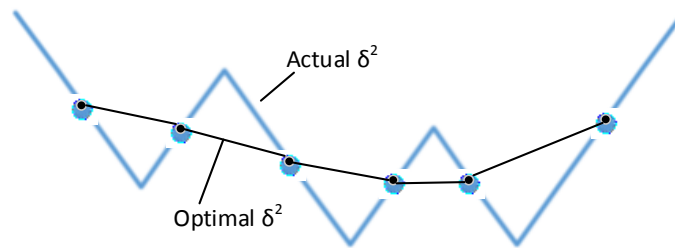
**Figure 38 - Peaks with Plateaus**

The final result is a set of alternating peaks and valleys like those shown in figure 39.



**Figure 39 - Ordered Peaks and Valleys after Processing**

Step 3 takes the midpoint between each peak and valley and connects them using a line. The path between the midpoints is the “optimal” path of the FIS shown in figure 40.



**Figure 40 - Creating an Optimal Gradient**

Calculate the “error” by taking the difference between the value of the optimal line returned from Step 4 at a given point and the actual value. This error is useful in the Fitness Function to determine the efficacy of a given FIS configuration.

This is where a relational database brings with it some formidable advantages. A typical algorithm might look at each point and then use neighboring points to determine peaks, valleys, plateaus and depressions, going back and forth in order to eliminate false peaks and valley as well as determine plateau and depression points. With some modifications, however, using SQL, that process can run over the entire set of results as a batch. The corresponding batch algorithm steps is as follows:

### Pseudo-code 6 - Set-based algorithm to find peaks and valleys

**Algorithm:** DETERMINE\_OPTIMAL\_PATH (values)

**Input:** values, a set of inputs into FIS and final result

**Output:** a set of points along the optimal path

**Begin** (Note this is done in the database)

- 1 Record all changes as the FIS operates.
- 2 Take the UNION of all known terrain features, unknowns, and the second point.
- 3 Rank the resulting set by value, epoch and type
- 4 Clump adjacent peaks and valleys using rank and epoch
- 5 Dense rank each clump
- 6 Get max and min rows for each clump
- 7 Assign unknowns to their corresponding clump
- 8 Order clumps to get local maximum or minimum
- 9 Take the midpoint between each peak/valley pair.
- 10 return the set of midpoints

**End**

Step 1 is as before. Step 2 creates the UNION  $F$  of the following sets of points from FIS results  $P$ :

$$F = K \cup V \cup X \cup p_2 \quad (4.4)$$

$K$  consists of true peaks, all left plateaus and right plateaus

$$K = \cup \left\{ \begin{array}{l} \{p_n | p_n > p_{n-1}, p_n > p_{n+1} \forall p \in P\} \\ \{p_n | p_n > p_{n-1}, p_n = p_{n+1} \forall p \in P\} \\ \{p_n | p_n = p_{n-1}, p_n > p_{n+1} \forall p \in P\} \end{array} \right. \quad (4.5)$$

$V$  consists of true valleys all left depressions and right depressions.

$$V = \cup \left\{ \begin{array}{l} \{p_n | p_n < p_{n-1}, p_n < p_{n+1} \forall p \in P\} \\ \{p_n | p_n < p_{n-1}, p_n = p_{n+1} \forall p \in P\} \\ \{p_n | p_n = p_{n-1}, p_n < p_{n+1} \forall p \in P\} \end{array} \right. \quad (4.6)$$

Another set of points exists where the middle point is equal to both the right and left adjacent points. There is no direct way to tell whether the point belongs to a plateau or depression. Call the set these unknown points  $X$ .

$$X = \{p_n | p_n = p_{n-1}, p_n = p_{n+1} \forall p \in P\} \quad (4.7)$$

Perform one final operation. The first epoch, or iteration, value has no previous entry, so it is excluded in the initial query. Label the second point by doing a compare with point 1 and assign peak or valley accordingly.

Step 3 ranks the results (number the rows 1, 2...) by value, epoch id and type (peak, valley or unknown). By adding the rank and original epoch ID, Step 4 is able to generate a new ordering. This causes the adjacent peaks and valleys to “clump” together.

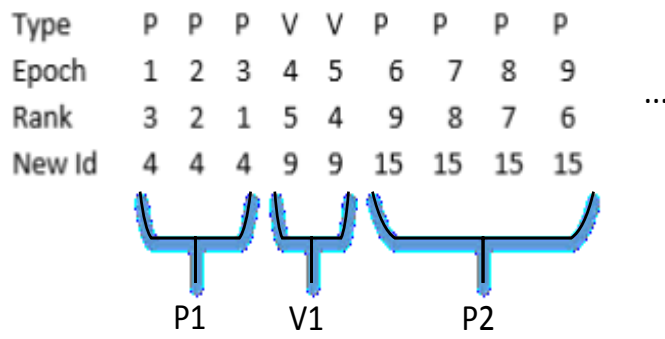
Step 5 creates a dense rank by type for each clump. This creates a unique identifier for each clump and also forces unknowns between their corresponding peaks and valleys.

Step 6 calculates the endpoints for each clump by taking their corresponding maximum and minimum epochs.

Step 7 assigns unknowns to their corresponding peaks or valleys. If an unknown lies between the endpoints of a “peak clump” they are peaks, otherwise they are valleys. Take the midpoint of each unknown and declare that the peak or valley. Now all points are classified as peaks or valleys.

Step 8 regroups again by type (peaks or valleys) as shown in figure 41 Type line, ordering the result by beginning endpoint of the peak or valley. This gives a sort of peaks and valleys by epoch order for each starting end point as demonstrated in figure 41, Epoch line. Identify clumps this time by ranking the types in reverse order – figure 41 Rank line. Because they are sequential, each type row number plus ordering will amount to a unique number allowing a batch process to identify each clump uniquely as shown in figure 41 New Id line. At this point each clump is either a peak or valley with no like adjacent type, i.e. a local maximum or minimum. The maximum or minimum point for each clump is the true local maximum or minimum and the process can continue as before.



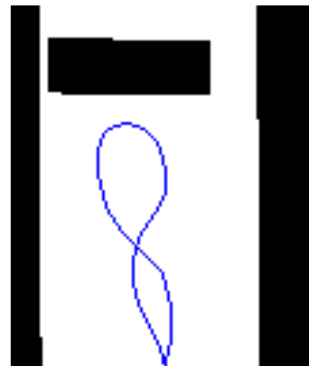


**Figure 41 - Set Operation to Isolate True Peaks and Valleys**

Another benefit of using a relational database is that it can handle multiple requests simultaneously from different FIS's. This would allow disparate, lightweight fuzzy controllers the ability to engage a high performance computer to handle much of the computational and storage load necessary for sophisticated problem solving. Finally the database is persistent storage which allows for an additional Tabu-like search option and data mining.

4.2.2: Enhancement #5 – Adding Unsupervised Learning Algorithm to Discover New Fuzzy Contexts

Consider the situation where the robot car encounters a barrier. Under the current fuzzy controller, it simply turns around and proceeds back the way it came without trying to explore for an opening as shown in figure 42.



**Figure 42 – Robot Car Unable to Navigate a Barrier**

Recall the Fitness Function described in above section. In addition to optimizing the fuzzy controller, it can also serve to measure error rates at runtime. Let  $\epsilon$  be the max error measured by the fitness function. By using the same epoch-to-epoch metric, the rate of change  $\delta$  over time can be measured against  $\epsilon$ . If  $\delta > \epsilon$ , that is the point at which a “candidate” context is available. The epoch where this first occurs sets of the outer limit of the outer radius of the context. The fuzzy controller will continue until one of two situations occur:

1. The controller will make a (presumably correct) decision and the error will once again drop below the  $\epsilon$  threshold.
2. The controller will fail, i.e. the robot crash into the barrier.

The key to creating a new context is finding 3 key components relevant to the car’s recent performance, assuming the concentric spheres representation:

- a. The length of the outer radius
- b. The length of the inner radius
- c. The center point of the new context

For (a), the outermost limit of the outer radius can start at the point where the  $\delta$  first exceeds  $\epsilon$ . Generalized, this means that any time during operation, whenever  $\delta > \epsilon$  is an indication that a new context is recommended. Call the point at that epoch  $p_o$ . The next step is to try to determine the location of the outermost limit of the inner radius. In the event of (1) the outermost limit occurs at the point of greatest error. In this case the center point becomes the straight line distance to the nearest point in the barrier. Call the point at that epoch  $p_i$  and the barrier point  $p_c$ . For (2) it is the point where the controller fails, which also becomes the center point. Hence the point of failure becomes  $p_i = p_c$ .

Taking the 3 points  $p_o$ ,  $p_i$  and  $p_c$ , we can now calculate the context.

As a result, the context of (1) is the hollow-sphere shape, which (2) is a solid sphere. An example of the process using (1) is demonstrated in figure 43.

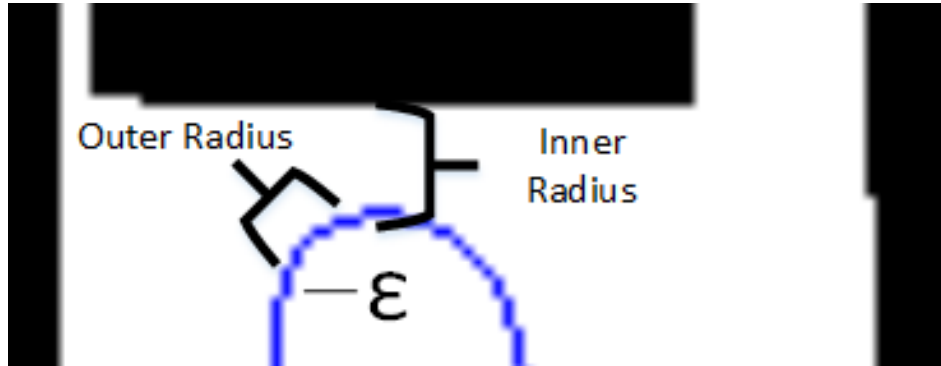


Figure 43 – An Unsupervised Technique to create a new Context

### 4.3: STRUCTURAL ENHANCEMENTS

#### 4.3.1: Enhancement #6 - Algorithm Definition Language

Recall the steps to implement a Fuzzy Inference System:

1. Define Membership Functions - Define each fuzzy function, or shape used to generate the fuzzy sets.
2. Define Fuzzy Sets – Define each fuzzy sets using a linguistic term along with the corresponding membership function.
3. Define Fuzzy Variables – Define each variable using a linguistic term, add the corresponding fuzzy set definitions.
4. Assign Variable Inputs and Outputs – Assign variables to their corresponding input and output collections.
5. Define Fuzzy Rules – Create the fuzzy rules by assigning fuzzy variables and antecedent/consequent terms.
6. Define Defuzzifier – Specify type and number of intervals used for defuzzification.
7. Assign Rules To Rules Collection – Add rules to the rules collection.
8. Define FIS – Create FIS using Rules Collection and Defuzzification object.

Implementing steps 1-8 can be done strictly in code as calls to the framework, but it is not a trivial process. Human systems interactions benefit when the interaction is as simple as

possible. To help a user define a fuzzy inference system, with its corresponding fuzzy objects, the framework implements a Fuzzy Modeling language. Previous work [Moreno 12] demonstrates how useful an XML-based modeling language is in supporting dynamic configuration of FISs and fuzzy objects in general.

Step 1 implementation requires specifying a convex shape for a membership function  $\mu$ .  $\mu$  is a function over a domain  $D$  such that for each  $x$  in  $D$ ,  $\mu(x)$  is a number between 0 and 1.

$$\mu(x) \rightarrow [0, 1], x \in D \quad (4.8)$$

Zadeh's rules for fuzzy sets require the functions take the form of a convex shape such as a triangle, trapezoid, Gaussian or other convex curve. The resulting shape serves as the membership function  $\mu$ . In the framework, each shape,  $f_{shape}$  is defined with one or more boundary points  $P_{bk}$  and one or more apex points  $P_{ak}$ .

$$\mu = f_{shape}(P_{b1}..P_{bn}, P_{a1}..P_{an}) \quad (4.9)$$

For each P, the X value specifies a specific value  $x$  in the domain while the Y value specifies the corresponding membership value between 0 and 1 at that value,  $\mu(x)$ .

$$f_{shape}(P_{b1}, \dots, P_{bn}, P_{a1}, \dots, P_{an}) = \mu(x), x \in D \quad (4.10)$$

The modeling language, provides textual representations of the Fuzzy Function and all other fuzzy objects. The resulting FSXML is shown in figure 44 for a Type-1 membership function.

```

<MembershipFunction Name="MediumF" FuzzySetType="Full Trapezoid">
- <Points>
    <Point Id="1" X="15" Y="0"/>
    <Point Id="2" X="50" Y="1"/>
    <Point Id="3" X="60" Y="1"/>
    <Point Id="4" X="100" Y="0"/>
</Points>
</MembershipFunction>

```

**Figure 44 - A Fuzzy Function Definition in XML**

In the TCF, a fuzzy set  $f_s$  is defined by a membership function and a linguistic “term” which describes its purpose in a more easily understandable way. Step 2 simply requires applying the function defined in step 1 to a linguistic term to generate a fuzzy set  $f_s$ .

$$f_s = \mu + \text{linguistic term} \quad (4.11)$$

A sample XML entry for a fuzzy set is shown in fig 45.

```

<FuzzySets>
    <FuzzySet Name="Near" MembershipFunction="NearF"/>
    <FuzzySet Name="Medium" MembershipFunction="MediumF"/>
    <FuzzySet Name="Far" MembershipFunction="FarF"/>

```

**Figure 45 - Combining a Term and Fuzzy Function to Create a Fuzzy Set**

In step 3, users create one or more fuzzy variables. Each fuzzy variable must have a “name”, a linguistic term to appropriately describe the variable’s purpose or function. Variable names should make sense within the FIS, describing in easily understood terms what that variable represents. Next, the user defines the boundary of the domain of a particular variable. The domain should be large enough to encompass all of the fuzzy sets to be assigned. Finally, the user designates whether the variable represents input (the antecedent) or output (consequent) when used to build fuzzy rules.

In step 4, once the domain is defined and the variable “termed”, the variable then is associated with one or more of the fuzzy sets defined in step 2. These associations are used to determine which fuzzy sets are represented in the fuzzification/defuzzification process. The corresponding XML is shown in figure 46.

During the fuzzification process, the fuzzy sets assigned to the fuzzy variable are evaluated using the underlying fuzzy function assigned to the corresponding fuzzy set

```
<FuzzyVariables>
- <FuzzyVariable Name="Angle" UpperBound="50" LowerBound="-50" Type="Output">
  <FuzzySet Name="VeryNegative"/>
  <FuzzySet Name="Negative"/>
  <FuzzySet Name="LittleNegative"/>
  <FuzzySet Name="Zero"/>
  <FuzzySet Name="LittlePositive"/>
  <FuzzySet Name="Positive"/>
  <FuzzySet Name="VeryPositive"/>
</FuzzyVariable>
```

**Figure 46 - Defining a Fuzzy Variable in XML**

Step 5 specifies the fuzzy rules that constitute the FIS. Each fuzzy rule consists of an antecedent which is a statement of the form:

$$IF \langle \text{fuzzy variable} \rangle IS \langle \text{fuzzy variable or fuzzy set} \rangle \quad (4.12)$$

The antecedent specifies a testable condition similar to a crisp IF statement, except instead of a true or false result, the fuzzy result consists of a value between 0 and 1 inclusive, dependent upon the input value and the various membership or fuzzy functions underlying the corresponding fuzzy sets shown in equation 4.12. Antecedents can be combined using AND/OR and parenthetical operators. For example, to test the distance of a barrier to the front of an obstacle the antecedent might take the form of:

$$IF \text{FrontalDistance} IS \text{Far} \quad (4.13)$$

“FrontalDistance” can consist of multiple fuzzy sets, for example: “Near”, “Medium” and “Far”. The antecedent “IF FrontalDistance IS Far” looks at the membership function of the fuzzy set “Far” assigned to the fuzzy variable “FrontalDistance”. Note again the use of linguistic terms that are easily understandable even to laypersons. A parser within the framework turns the text into its corresponding fuzzy sets and fuzzy variables.

The fuzzy rule also requires a consequent, which is constructed similarly to the antecedent but uses output variables and sets. The Wizard application described in the next

section contains a simple text builder a user can employ to construct the both the antecedent and consequent.

In step 6, the user defines the number of fuzzy intervals used for defuzzification. The framework currently supports traditional Zadeh rules for fuzzification of fuzzy Type-1 where membership of a fuzzy variable is equal to the minimum membership of the corresponding fuzzy sets. Recall this is also referred to as a fuzzy intersection of fuzzy sets.

$$\cap \mu_{C_i} = \mathbf{min}(\mu_{C_1}, \mu_{C_2}, \dots, \mu_{C_n}) \quad (4.14)$$

Defuzzification is achieved by then taking a fuzzy union across all intervals in the domain. This is accomplished by taking the maximum across the sets of intervals and their corresponding memberships.

$$\cup \mu_{C_i} = \mathbf{max}(\mu_{C_1}, \mu_{C_2}, \dots, \mu_{C_n}) \quad (4.15)$$

A centroid, of center of gravity is then calculated by determining a weighted mean across the fuzzy region [Cox 95]. Recall the fuzzy solution region  $\mathfrak{R}$  is calculated by the following:

$$\mathfrak{R} = \frac{\sum_{i=0}^n d_i \mu_A(d_i)}{\sum_{i=0}^n \mu_A(d_i)} \quad (4.16)$$

where  $d$  is the  $i^{th}$  domain value and  $\mu(d)$  is the membership value returned the corresponding fuzzy function defined in step 1. Fuzzy Type-2 defuzzification uses the Karnik-Mendel Interval Technique described in [Karnik 01] which is a variation on the centroid technique involving a calculation of multiple centroids over the footprint of uncertainty.

In step 7, the user creates a database of the fuzzy rules defined to use in the FLC. Depending upon which rules are input and which are output, the FIS will attempt to evaluate all relevant rules during the fuzzification/defuzzification process.

Step 8 is performed by the framework. The resulting FIS then consists of the database repository of all the relevant fuzzy objects as well the domain space and fuzzy operators used.

The XML defining the final FIS is listed in figure 47.

```
<FuzzyInferenceSystem Name="MySystem" OutputToDB="false">  
  <FuzzyDB Name="FuzzyDB"/>  
  <Defuzzifier Name="Defuzzifier" Intervals="1000"/>  
  - <Rules>  
    <FuzzyRule Name="Rule 1"/>  
    <FuzzyRule Name="Rule 2"/>  
    <FuzzyRule Name="Rule 3"/>  
    <FuzzyRule Name="Rule 4"/>  
    <FuzzyRule Name="Rule 5"/>  
    <FuzzyRule Name="Rule 6"/>  
    <FuzzyRule Name="Rule 7"/>  
  </Rules>  
  <SQLCn Name="SQLCn"/>  
</FuzzyInferenceSystem>
```

**Figure 47 - Defining an FIS Using the Fuzzy Modeling Language**

Pseudo-code for constructing the FIS from the model language is as follows:



### Pseudo-code 7 – Construct a Fuzzy Inference System from Modeling Language

**Algorithm:** Init(XMLFile filename)

Input: the path/filename of an XML definition file

Output: a fuzzy inference system

**Begin**

```

1  set fis = GetInfModel(filename)
2  set fis properties as defined in XML
3  create empty fuzzy database
4  set fuzzy variables array to variables as defined in XML
5  FOR EACH fuzzy variable in fuzzy variables array
6    set fuzzy sets array to sets as defined in XML for variable
7    FOR EACH set in fuzzySets
8      set fuzzy function to function as defined in XML for fuzzy set
9      assign function to set
10   assign set to variable
10  NEXT set
12  assign variable to database
13  NEXT variable
14  add database to fuzzy inference system

15  set fuzzy rules array to rules as defined in XML
16  FOR EACH rule in rules
17    IF rule is used in this fuzzy inference system, as defined in XML THEN
18      add rule to fuzzy inference system
19    END IF
20  NEXT rule

21  return fuzzy inference system

```

**End**

Of great importance to a Type-C implementation is this ability to separate the algorithm from the application. This allows for the process of Fuzzymorphism to occur dynamically. Polymorphism allows objects to be used interchangeably based upon a common interface. In particular, polymorphism allows an application to apply generic process call that transforms into a specific process call dependent upon the existence of objects at runtime [Pressman 09]. Contextual Fuzzymorphism takes Polymorphism a step further: it allows a dynamic hybridization of disparate algorithms for states that satisfy multiple contexts as demonstrated by the pseudo-code above.

Doing this is easier via an abstraction of the algorithm template such that it can be maintained external to the program. Such a template, called an Algorithm Definition Language (ADL) is based upon the FSXML described in [Moreno 12] but is generalized and allows a template to be loaded at runtime and an algorithm constructed on the fly.

Pseudo-code for the process follows:

#### **Pseudo-code 8 - Create an Algorithm from the ADL**

**Algorithm:** CreateAlgorithm( name) returns Algorithm  
**Input:** a name used to identify the algorithm  
**Output:** an algorithm that takes input(s) and produces output(s)  
**Begin**  
1 lookup algorithm ADL based upon name  
2 build algorithm based upon ADL  
3 return algorithm  
**End**

The **BuildAlgorithm** operation depends upon the particular algorithm selected. For a Fuzzy Logic Type-1 Inference System (T1-FIS) the build steps take as input the ADL and output the code for the T1-FIS. Recall the steps to generate the T1-FIS are as follows:

### **Pseudo-code 9 – Steps for Building a Type-1 Fuzzy Inference System**

Define Membership Functions - Define each fuzzy function, or shape used to generate the fuzzy sets

Define Fuzzy Sets – Define each fuzzy sets using a linguistic term along with the corresponding membership function

Define Fuzzy Variables – Define each variable using a linguistic term, add the corresponding fuzzy set definitions

Assign Variable Inputs and Outputs – Assign variables to their corresponding input and output collections

Define Fuzzy Rules – Create the fuzzy rules by assigning fuzzy variables and antecedent/consequent terms

Define Defuzzifier – Specify type and number of intervals used for defuzzification

Assign Rules To Rules Collection – Add rules to the rules collection

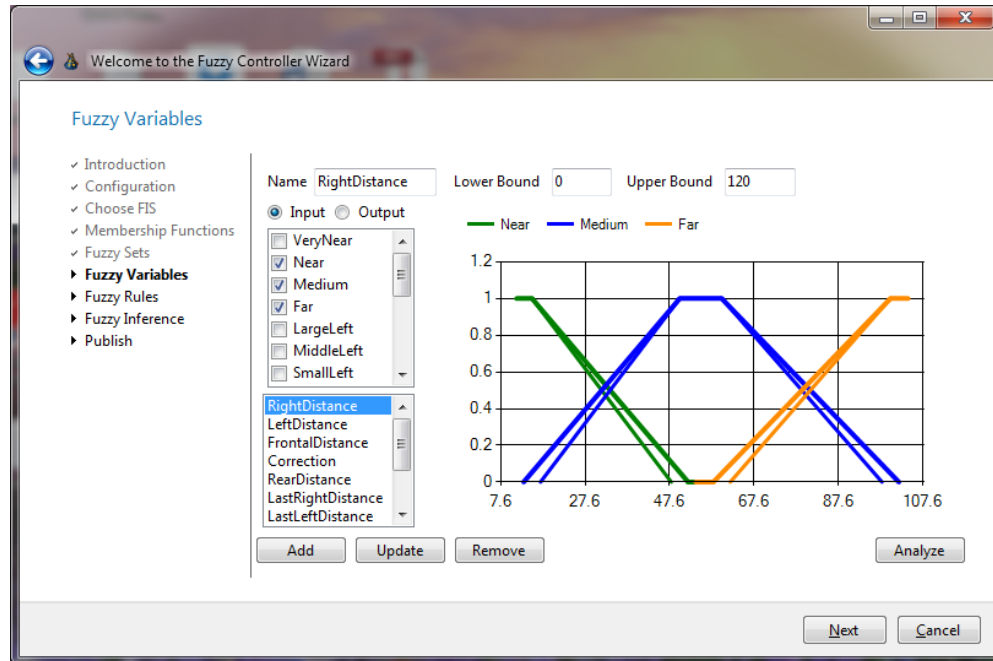
Define FIS – Create FIS using Rules Collection and Defuzzification object

#### 4.3.2: Enhancement #7 - The Wizard Tool

Because implementing algorithms like the T1-FIS is not a trivial process, the TCF implements a Fuzzy Modeling language as part of its ADL. However, XML is difficult to create by hand so the framework also provides a Wizard tool to allow a user to generate and maintain the underlying XML-based model for the T1-FIS. It is a step-by-step guide to building and/or maintaining a fuzzy definition. The framework also introduces the capability to create and maintain a General Type-2 Fuzzy Logic Inference System (GT2-FIS). T1 and GT2 Fuzzy Systems are similar: they both employ similar fuzzy rule bases, and they operate in a similar fashion. However, the underlying fuzzy sets, membership functions, operators and defuzzification are quite different. A GT2 ADL incorporates these differences into a definition which allows for the construction and operation of a functioning GT2-FIS according to Karnik-Mendel [Karnik 01].

The framework has also added a database-based repository and database objects capable of handling both T1-FIS and GT2-FIS implementations.

The Wizard tool presents a user interface that allows a user to “step” through each necessary configuration element in the proper sequence. It is also a way to break down a complex overall process into a series of simpler steps [Anderson 10]. As a result, the Wizard application can serve both as a developer’s tool and a training tool for the novice learning to use fuzzy logic. A step of the Wizard application, is shown in figure 48.



**Figure 48 – Fuzzy Wizard Variables Page**

The ADL produced by the Wizard provides textual representations of the fuzzy objects such as the Fuzzy Membership Function and corresponding Fuzzy Sets. For example, the Wizard application handles step 1 of the **BuildAlgorithm** process as a Membership Functions page shown in figure 49. The resulting ADL is shown in figure 50 for a Type-1 membership function.

## Membership Functions

Define the membership functions used to generate the fuzzy sets


Type of Fuzzy Set Desired		X	Y
Full Trapezoid			
Name	Left Boundary	15	0
MediumF	Left Apex	50	1
	Right Apex	60	1
	Right Boundary	100	0

Figure 49 – Defining a Membership Function using the Wizard

```
<MembershipFunction Name="MediumF" FuzzySetType="Full Trapezoid">
- <Points>
  <Point Id="1" X="15" Y="0"/>
  <Point Id="2" X="50" Y="1"/>
  <Point Id="3" X="60" Y="1"/>
  <Point Id="4" X="100" Y="0"/>
</Points>
</MembershipFunction>
```

Figure 50 - A Fuzzy Function Definition in XML

Pseudo-code for constructing the FIS from the model language is as follows:

### Pseudo-code 10 – Build Fuzzy Inference System from ADL

```

Algorithm: BuildFISFromXML(algName)
Input: name of a fuzzy inference system
Output: fuzzy inference system

Begin
1  get ADL based upon input name
2  create new fuzzy inference system fis
3  set fis properties based upon ADL
4  create new fuzzy database
5  create array of fuzzy variables based upon ADL
6  FOR EACH variable in fuzzy variables array
7    create array of fuzzy sets based upon ADL
8    FOR EACH set in fuzzy sets array
9      create membership function based upon ADL
10     assign membership function to fuzzy set
11     add fuzzy set to fuzzy variable
12  NEXT set
13  add fuzzy variable to fuzzy database
14  NEXT variable
15  add fuzzy database to fuzzy inference system

16  create array of fuzzy rules based upon ADL
17  FOR EACH rule in fuzzy rules array
18    IF rule used in fis THEN
19      add rule to fis
20    END IF
21  NEXT rule

22  return fis

End

```

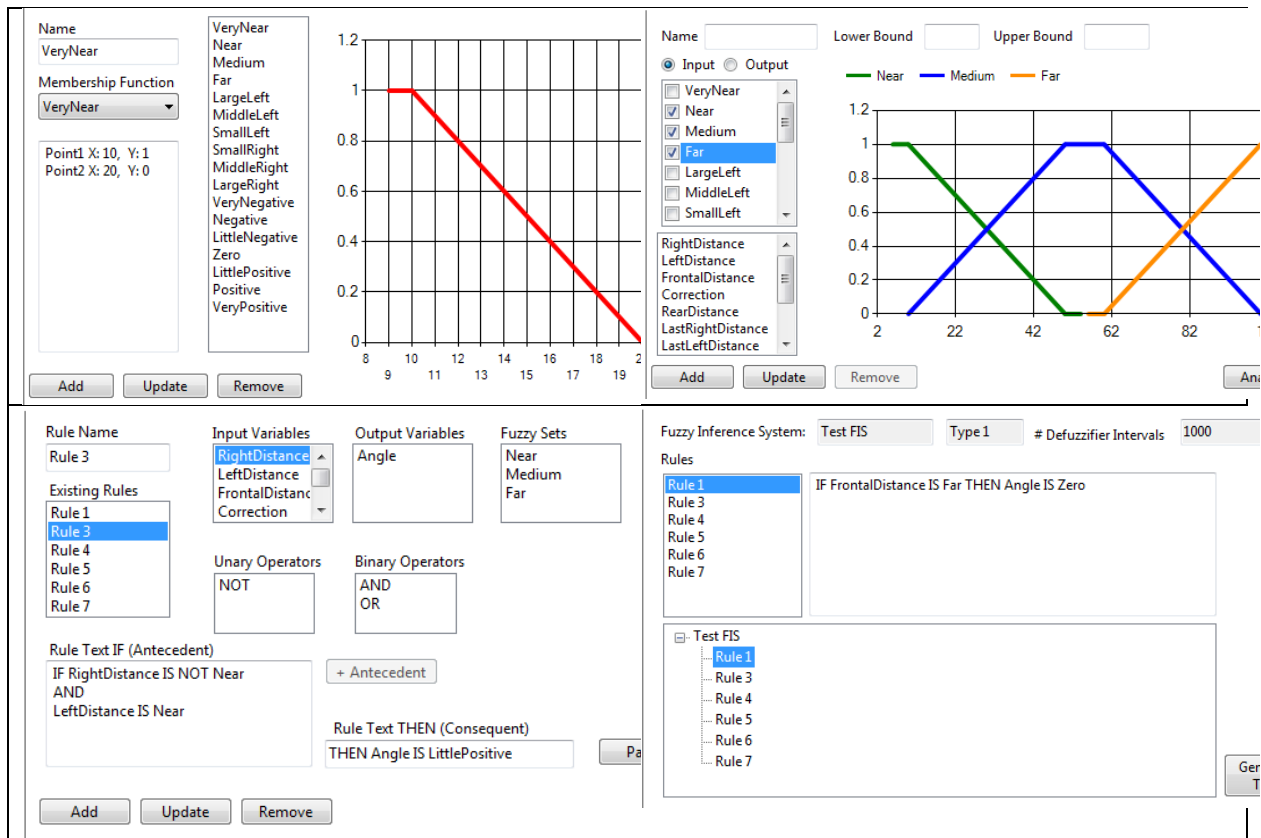
Being able to generate an FIS or other algorithm from the ADL presents several advantages over static construction:

1. Changes in fuzzy sets, rules or other components do not require recompiling the application. This allows a user to make changes to program behavior without requiring access to source code.
2. Algorithms become dynamically “swappable”. This allows for greater flexibility for SDPs and other problems which require behavior modifications on the fly.

3. Multiple versions of the same algorithm can be stored and used/tested as appropriate.
4. The ADL can be “contextualized” or labeled in human-readable terms in order to describe a particular ADL-defined behavior.

Similarly the FIS is constructible from an ADL residing in a SQL Server database.

Overall, the Wizard application walks the user through the configuration process via a series of configuration steps as demonstrated in figure 51.



**Figure 51 – Windows for building a Type-1 Fuzzy Inference System**

### 4.3.3: Enhancement #8 - Hierarchical Technique for Diverse Contextual Dynamic Programming and Optimization

Recall that an algorithm, by definition, is any well-defined computational procedure that takes as input as value or set of values and produces a value or set of values as output [Cormen 09]. Algorithms do many things, from sorting a list of names to guiding spacecraft around the solar system. Many problems, such as sorting a list, are simple and very straightforward, as are the algorithms used to solve them. Other problems, such as navigating a spacecraft, are far more difficult, involving a series of independent and interdependent processes, each of which might utilize multiple algorithms.

Just as biological organisms evolved from simple proteins to complex multi-system creatures so are computing machines evolving. A typical must have gadget from the 1970's was a simple calculator or digital watch. Today, it is a smart phone or tablet PC. Whereas the calculators of old could do little more than manipulate numbers, today's smart phone serves as a personal planner, communicator, child monitor, GPS, restaurant locator, camera, video recorder, stock advisor and perhaps a hundred other functions. On occasion, it can even make a phone call.

As computer hardware takes on an increasingly more important, comprehensive and also more difficult role so must computer software. Competition forced biological organisms to develop specialized, interdependent systems, like eyes, feet and skin. These biological organisms developed motion, pattern-recognition techniques, defense against bacteria and other traits, using these new specializations to better compete against other biological organisms. Likewise economic competition is forcing computer components to evolve and incorporate a myriad of ever more sophisticated capabilities [Camara 14]. The cell phone, once little more than a large dialer with a radio transmitter, now might include a camera, blue-tooth device, motion-detection sensor and high-res touch-screen. Like the subsystems of their biological forebears, these new devices are more complex and interconnected than ever before. This calls for better and more flexible algorithmic processes [Vidal 13] to enable each new generation of cell-phone or other device to stay competitive.

Two particular problems arise from this conundrum: 1) algorithm selection and 2) algorithm optimization. The first requires finding an algorithm appropriate to the task, like a quicksort for a list of names. It appears straight-forward, but what about those situations



where multiple algorithms might be required? Suppose instead of a simple list, it was a list of possible places to eat. New factors come into play, such as location, type of restaurant, average price and customer ratings, maybe even time of day and who you are with. Often, such as when a robot must navigate a flight of stairs, multiple very different algorithms come into play in order to meet ever more complex goals.

The second issue also poses problems as the increasing complexity of the underlying problem makes it more difficult to determine an optimal solution [Tang 11].

To address these issues this dissertation proposes a novel database-driven hierarchical architecture for algorithm determination and optimization. The primary goal of this architecture is to find suitable algorithms for problems, then combine those algorithms with known optimization techniques to form an optimal solution.

The proposed architecture consists of a number of components:

1. A class library defining classes and interfaces for implementing a hierarchical solution.
2. An algorithm definition language (ADL) used to define algorithms for testing, implementation and optimization purposes.
3. A database backend for storage of algorithm definitions, inputs and outputs.
4. Utilities for creating and maintaining the ADL
5. A mechanism for combining the architecture into the Fuzzy Logic Type-C framework.

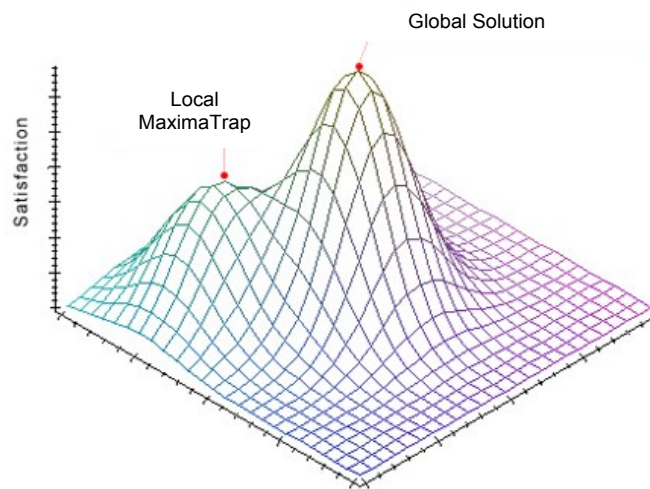
#### *4.3.3.1: Prior Contributions*

Program optimization takes on a number of different forms, but for purposes of this paper, consider “optimization” to be the process to determine the most efficient operation of a technique specified by criteria such as speed, resource usage or accuracy. The optimal algorithm is therefore able to achieve its goals better by some metric than the alternatives.

Getting to that optimal solution can be a significant challenge for complex systems, however, because the optimal behavior may not be easily determined or understood. Similar to a doctor trying to diagnose an ailing patient, the developer or architect will often have to rely on an extensive and vague list of “symptoms”, many of which may not be relevant.

As an example, consider a traditional Fuzzy Inference System (FIS). The FIS fits the definition of an algorithm, having a series of inputs and producing an output. Creating a FIS requires a series of steps from defining the underlying fuzzy sets and membership functions and rules to determining how the defuzzification process will work. Any misstep can undermine the entire FIS without giving much indication exactly where the problem lies.

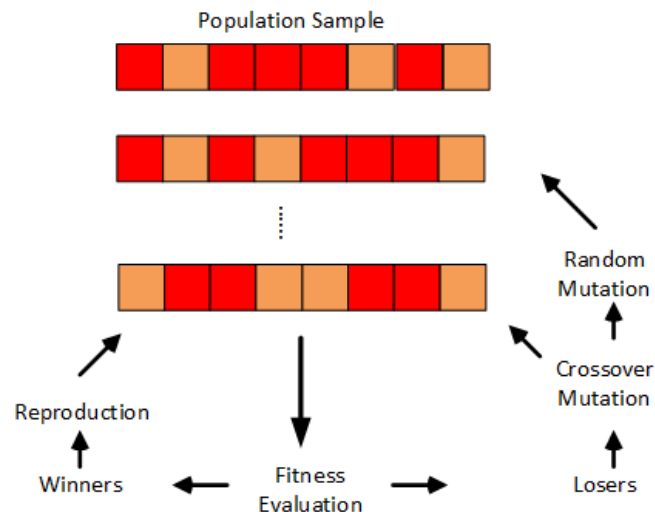
Typical optimization of a FIS usually involves finding and testing alternative configurations and choosing which configuration most closely produces the desired result. In many cases, testing every alternative configuration is impractical due to the huge number of potential configurations. One of the approaches to dealing with an impractical number of possibilities is using a Local Search technique. Local search, or gradient search, techniques take a random starting point and conduct a series of tests, following the gradient of a result in order to find a local maximum (or minimum). Think of it as a blind man trying to climb a mountain. While he may not be able to see *the* peak, he knows in which direction a peak lies by following the gradient or slope of the ground. A problem occurs, however, when our blind traveler finds himself on a maxima trap where every direction is down but he is not on the true peak as shown in figure 52.



**Figure 52 - Local Peak Prevents Discovery of Global Solution**

In order to address the maxima trap problem, many different Local Search techniques were created with probabilistic heuristics to “shake” the climber from a maxima trap to a more suitable peak; but they come at a greatly increased complexity cost. One such approach

is simulated annealing, modeled on the annealing process used to reduce metallurgic defects, in which the climber is shaken out of a local maximum through the annealing process. Evolutionary computation techniques provide another form of local search in which configurations “evolve” in a fashion similar to natural processes. In one type of genetic algorithm, a set of “genes” or configurations is built demonstrated by figure 53. A pair of genes is selected at random and compared using a fitness function. Winners and losers in the sample can then be crossed and/or mutated and compared with the winning configuration. Through the evolutionary process, new genes come into existence which have the potential to be superior to even the best gene in the original sample. A memetic algorithm takes the evolutionary process a step further by incorporating a local search technique post-evolution. This allows the gene to evolve and immediately search for a solution from the new starting point without having to resample.



**Figure 53. Genetic Mutation**

Nature provides other techniques such as Swarm and Ant Colony Optimization and computational constructs such as the Artificial Neural Network and many others [Eiben 07], [Juang 08]. All of these are designed to allow the machine to “learn” how best to solve a problem. Prior work has been done showing the effectiveness of these techniques when applied to many optimization problems [Pasiadis 04], [Guimaraes 07].

All of these efforts are designed to allow machines, through heuristics and trial and error, to automate the learning process. Prior work [Carrasco 03], [Mantawy 99] and others show the great variability and applicability of these techniques, whether it is a memetic algorithm trying to discover better configurations to navigate a maze [McCarty 14a] or an inventory problem using simulated annealing to optimize its layout [McCarty 08b], or a neural network discovers patterns in data [Kawady 14].

However, while this allows for optimization of *an* approach, it doesn't necessarily give any indication whether the approach itself is an optimal choice. The proliferation of algorithms is due primarily to human developers recognizing the need for different approaches to different kinds of problems [Skiena 10]. As problems grow more complex and diverse and the capabilities of computer hardware grow more sophisticated it becomes imperative to move the responsibility for determining *how* to solve a particular problem from human to machine.

#### 4.3.3.2: Foundations of FC-OAP

Recall that the Fuzzy Logic Type-C architectural framework is a three-pronged approach to solving complex problems. The first is the context, which defines the problem space [McCarty 14b]. The second is the algorithm definition language and tools [Moreno 12], [McCarty 13] and the third is algorithm selection and optimization. Fuzzy Context Optimization Abstractions for Processes (FC-OAP) is an approach to achieving this third component. It requires a hierarchical architecture able to take a specific problem and apply different algorithms in order to determine which algorithm is best suited for the desired solution. The architecture accomplishes this in the following steps:

1. Determine which algorithms from a repository of definitions can solve the problem.
2. Apply heuristics to determine which algorithms from among the applicable algorithms is a good candidate for further optimization.
3. Optimize and determine best algorithm/configuration option.

### Step 1 – Algorithm Determination

The first step in the determination process is to whittle down from a master list of algorithms to a list of applicable algorithms. The algorithms are stored as definitions, either as XML [Moreno 12] or in a database [McCarty 13]. Let  $\mathfrak{R}$  be a repository of algorithms. The goal is to determine each  $r_i \in \mathfrak{R}$  that is applicable. This is done using a applicability function  $\alpha$  such that:

$$\alpha(r_i) = \begin{cases} \text{true if } r_i \text{ suitable} \\ \text{false otherwise} \end{cases} \quad (4.17)$$

The Algorithm Definition Language (ADL) contains the necessary blueprint to build an algorithm on the fly and allows for testing by the applicability function whether it will accept input from the underlying problem process and return output in the form of a solution. At this point the algorithm becomes a candidate and is added to the candidate array.

Pseudo-code for the algorithm determine process is as follows:

#### Pseudo-code 11 - Finding which algorithms solve a particular problem

**Algorithm:** DetermineAvailableAlgorithms(x)  
**Input:** a problem definition x  
**Output:** an algorithm definition array darr  
**Begin**  
 FOR EACH algorithm in algorithm repository  
   Determine if algorithm can be applied to problem  
   IF algorithm is applicable THEN add to array darr  
 NEXT algorithm  
  
 return darr  
**End**

### Step 2 – Choose the best candidate

The second step involves choosing from among the available algorithms the one best suited for solving the problem. This is accomplished via a suitability function. The suitability function contains heuristics which are used to rank each of the available algorithm candidates according to how each best fits the suitability metrics. Suitability is dependent upon

heuristics, such as a simple greedy algorithm that picks the best performer based upon some specified metric.

$$\sigma(\mathbb{C}) = c_i | \sigma(c_i) = \mathbf{max}(\sigma(c_i)) \forall c_i \in \mathbb{C} \quad (4.18)$$

Pseudo-code for the suitability function is as follows:

### Pseudo-code 12 - Finding most suitable algorithm for a particular problem

**Algorithm:** DetermineBestSuitedAlgorithm(arr)  
**Input:** an array of algorithm definitions arr  
**Output:** most suited algorithm definition ai  
**Begin**  
 FOR all algorithm in arr  
   Rank algorithm using SuitabilityRank function  
   IF algorithm rank > best rank THEN  
     Save algorithm as best algorithm, algorithm rank as best rank  
   END IF  
 NEXT algorithm  
  
 return best algorithm  
**End**

### Step 3 – Algorithm Optimization

Applicable algorithms may undergo a final optimization step, if applicable, using traditional optimization techniques such as those described in [Juang 08], [Martinez 10]. This final step seeks to improve the algorithm’s performance according to a fitness function:

$$f_{r_i}(r_i(o)) = \mathbf{max}(r_i(o)), o = o_1, \dots, o_n \quad (4.19)$$

This completes the optimization hierarchy:

1. Determine all algorithms suitable to solve a given problem.
2. Determine the “best” solution.
3. Optimize the best solution.

#### 4.3.3.3: The Relational Database

For the FC-OAP, the database provides the following functions:

1. Repository for the ADL.
2. Temporary or permanent storage of results for optimization.
3. Engine for large-scale database manipulation and optimization as described in [McCarty 14a].

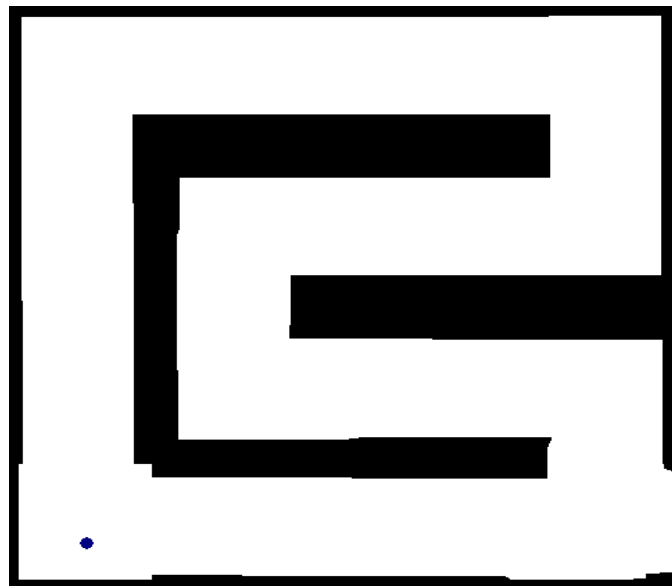
The relational database first serves as the data-store for the ADL. It can serve to store both algorithm definitions as well as problem “properties” and heuristics for problem solving and optimization. Advanced querying capabilities and speed enable the FC-OAP to perform lookups and comparisons against large repositories of algorithms and data. Because of the database’s almost unlimited storage capacity, it can function as a Tabu-search, storing results so that searches can avoid repeating previous mistakes. The database is also able to support multiple independent clients so multiple applications can “collaborate” on a problem. Another potential advantage not addressed here is the ability to data mine. This mining can take several forms:

1. Look for patterns in problem-algorithm solutions
2. Look for optimization parameters that are most effective in optimization
3. Attempt to predict algorithm/optimization testing, in effect providing a Tabu-like capability that is forward-looking.

## 4.4: TEST EXAMPLES

### 4.4.1: Basic Framework Tests

Consider a robot car whose goal is trying to navigate a simple maze as illustrated in figure 54.



**Figure 54 - Robot in a Maze**

The car must figure out how to move through the maze while also maintaining an efficient path and avoiding walls and obstacles. Doing this requires continual adjustment of its direction in order to avoid coming too close.

Ideally, the robot should try to position itself as far away from each barrier as possible while maintaining forward motion. In order to do this, it must constantly reevaluate its position as it moves around the maze and adjust its angle of motion in order to maintain maximum distance from each wall. Thus, the following factors have to be accounted for:

1. Frontal Distance
2. Distance to the Right Wall
3. Distance to the Left Wall
4. Angle of forward movement



The first 3 factors constitute the inputs and the last one is the resulting output. Both inputs and outputs are necessary for the fuzzy inference system (FIS) solutions used in the subsequent test examples.

*4.4.1.1: Test Example #1 – A Simple Navigation Problem – Type-1*

The Type-C framework created a Fuzzy Controller based upon a Type-1 Fuzzy Inference System. This is the simplest and most basic implementation of the framework. The T1-FLC was used to help a software robot successfully navigate the maze without hitting any of the walls or barriers. Pseudo code to describe the final process is as follows:

**Pseudo-code 13 – Navigate Through a Maze**

**Algorithm: NavigateMaze**

**Begin**

- 1 configure FIS
- 2 WHILE(true)
- 3   navigate()
- 4 END WHILE

**End**

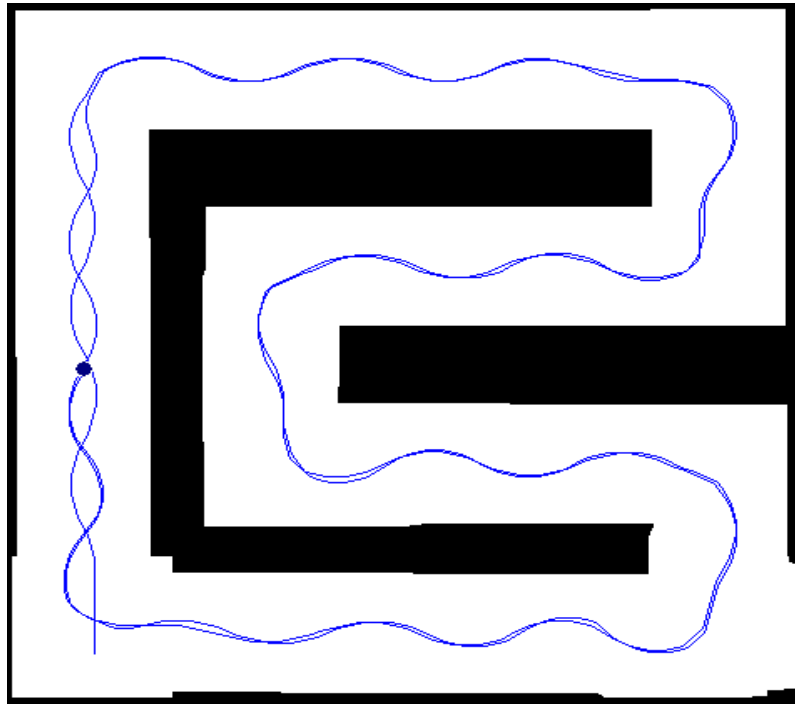
**Algorithm: Navigate**

**Begin**

- 1 input distance to front
- 2 input distance to left
- 3 input distance to right
- 4 fuzzify inputs
- 5 defuzzify inputs to angle change
- 6 apply angle change to current angle
- 7 move robot

**End**

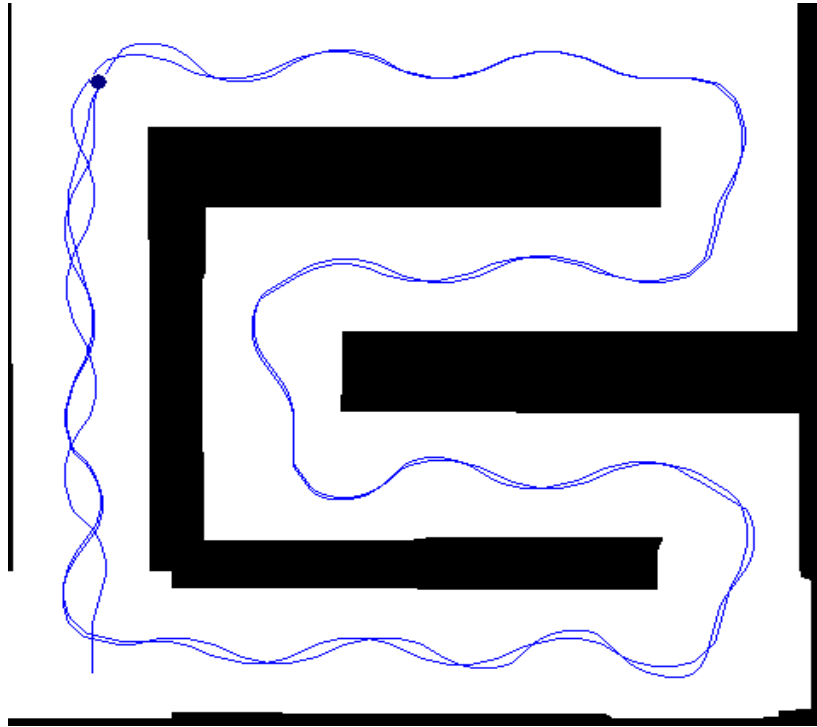
Its path was tracked and shown in figure 55.



**Figure 55 - Robot Navigating Maze Using Framework-Based T1 FLC**

*4.4.1.2: Test Example #2 – A Simple Navigation Problem – Type-2*

The Type-C framework created a Fuzzy Controller based upon a General Type-2 Fuzzy Inference System. The robot was able to successfully navigate the maze without hitting any of the walls or barriers. Its path was tracked and shown in figure 56.

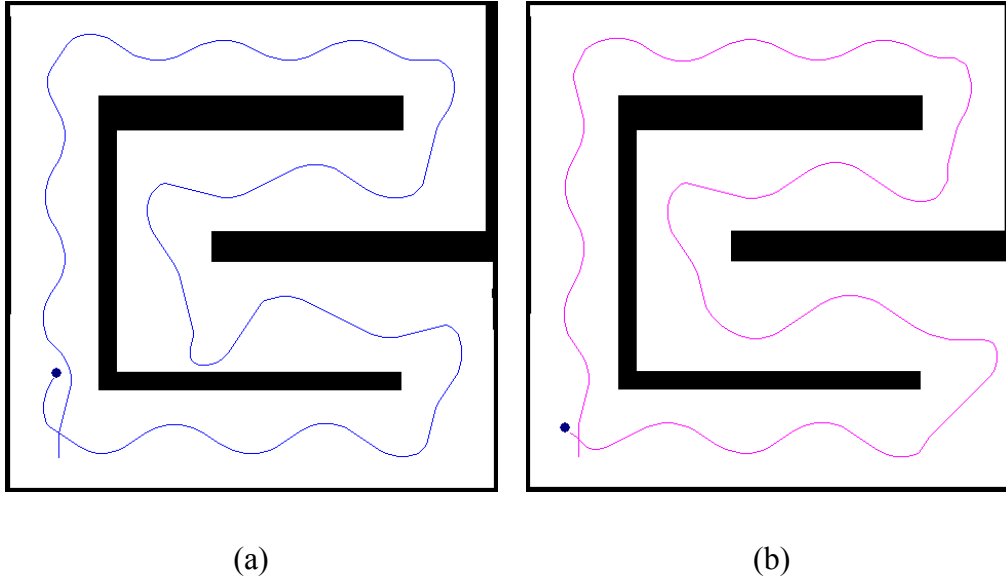


**Figure 56 - Robot Navigating Maze Using Framework-Based T2 FLC**

*4.4.1.3: Test Example #3 – Simple one algorithm implementation and comparison.*

In this test example a robot vehicle is tasked to move around a maze. There are no obstacles or special terrain features. Implementations are a traditional T1-FIS, GT2-FIS.

The FLC uses a database-based ADL to construct both the T1-FIS and GT2-FIS, both with the same 6 fuzzy rules, and successfully navigate the maze as shown in figures 57a and 57b.



**Figure 57 (a) - T1-FIS and 57 (b) – GT2-FIS**

The T1-FIS managed to navigate the maze in 554 iterations, with an average over 100 runs of 1.247 ms/iteration. The GT2-FIS managed to navigate the maze in 538 iteration with an average over 100 runs of 11.113 ms/iteration.

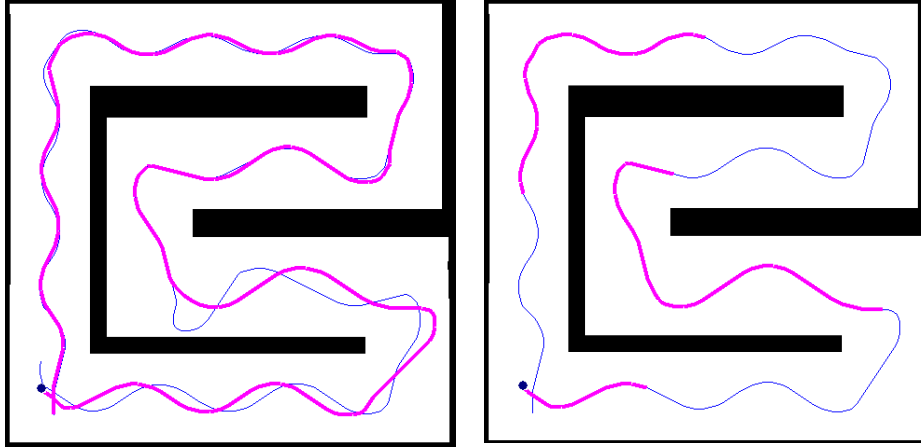
**Table 5 – Simple Comparison T1-FIS/GT2-FIS**

Process	# iterations	Time (ms/iteration)
T1-FIS	554	1.247
GT2-FIS	538	11.113

*4.4.1.4: Test Example #4– Dynamic swapping of algorithms.*

A necessary feature of the TCF is the ability to cleanly separate the algorithm from the application, important for dynamic hybridization of algorithms. In this example the TCF versions of the T1-FIS and GT2-FIS are built from the ADL and swapped multiple times in order to demonstrate the TCF's ability to swap techniques created by the ADL on the fly. In figure 58 the T1-FIS (thin blue line) and GT2-FIS (thick magenta line) are run over the same

surface one after the other for comparison. In figure 59 the two techniques are swapped back and forth over the same run.



(58)

(59)

**Figure 58 – T1-FIS overlaid with GT2-FIS**

**Figure 59 – T1-FIS interlaced with GT2-FIS**

#### 4.4.2: Context Tests

##### *4.4.2.1: Test Example #5 – Comparative analysis Traditional T1, GT2 against TCF T1, GT2.*

Under normal circumstances the TCF does not perform a membership test when only a single context is defined, however, for purposes of comparison, the test was forced in order determine how much overhead the context imposed upon a standalone algorithm. This forces the running of the membership tests and other contextual overhead likely to be encountered in a typical TCF implementation. The results are as follows (averaged over 100 tests):

**Table 6 - Comparing Traditional and Contextual Performance**

Process	Time (ms/iteration)	% C Overhead
T1-FIS	1.247	N/A
GT2-FIS	11.113	N/A
T1-FIS TCF	1.268	1.7%
GT2-FIS TCF	11.135	.02%

*4.4.2.2: Test Example #6 – Traditional Algorithm vs. Contextual Extensions*

In this case both the T1-FIS and GT2-FIS were extended with to support some “contextual” information regarding the addition of a second surface (one with a barrier to navigate around) and performance was compared against a pair of contexts representing the same two surfaces, with the first being the original FIS and the second being a new context. The results are as follows (averaged over 100 tests):

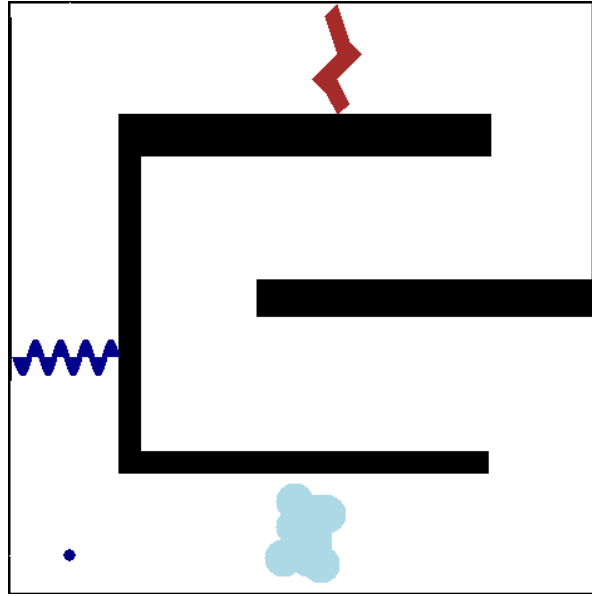
**Table 7 - Extended Traditional vs Contextual Performance**

Process	Time (ms/iteration)
T1-FIS ext	1.313
2 Contexts – T1	1.102
GT2-FIS ext	11.562
2 Contexts – GT2	8.7

*4.4.2.3: Test Example #7 – Dynamic Handling of Contexts with Fuzzymorphism*

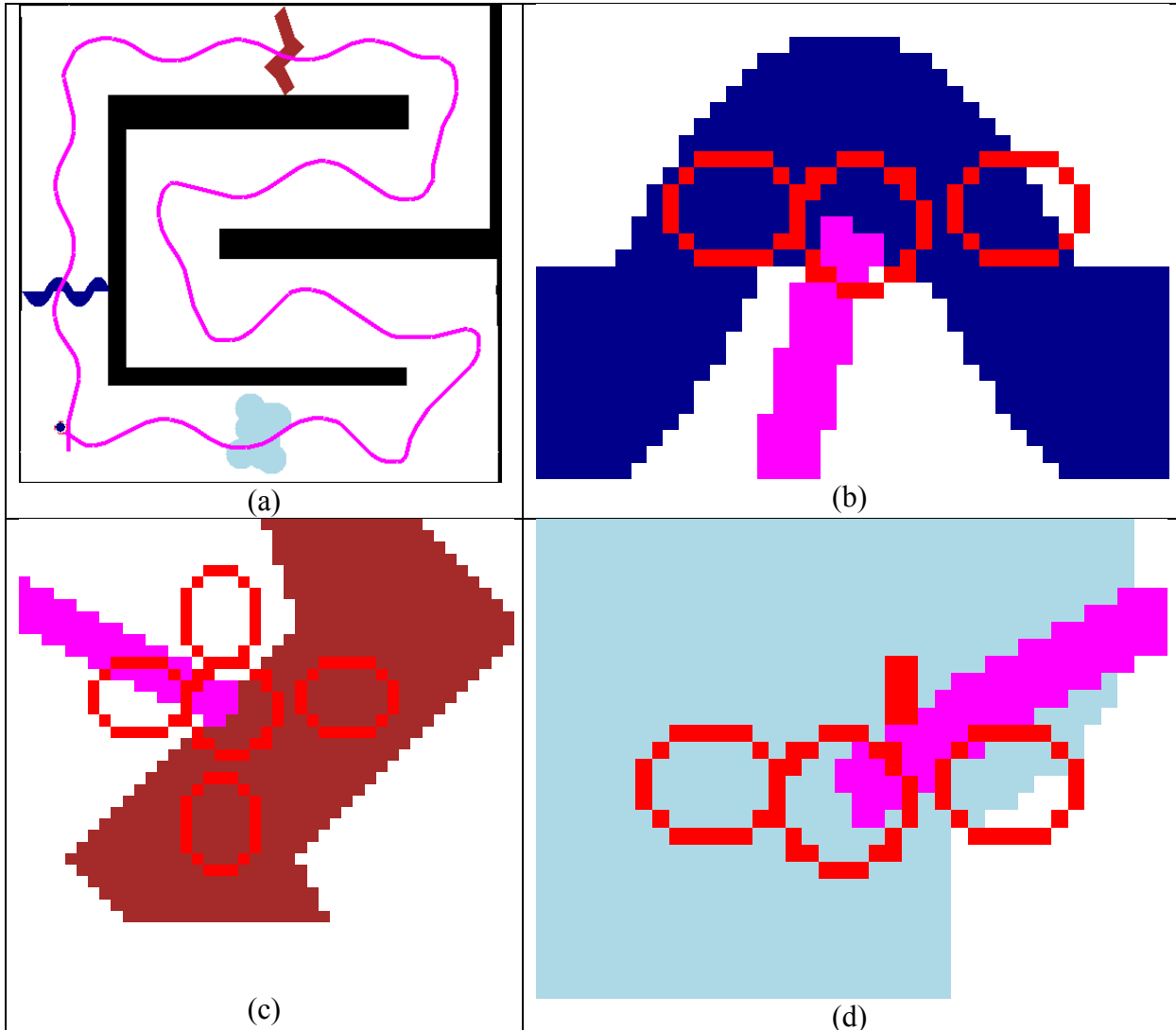
With a nod to the duck, the FLC was used to build an implementation of a robot vehicle able to traverse the following four terrain features:

1. A ROAD running through a maze
2. A RIVER obstacle
3. A CANYON obstacle
4. A LAKE obstacle



**Figure 60 - New maze with terrain features**

In navigating the track the vehicle tests ahead for the type of terrain. When spotting a different terrain feature, or context, it quickly morphs into the new vehicle, combining behaviors as it nears before becoming fully adapted to the new behaviors when the context has completely changed. Figure 61 a shows the vehicle navigating the maze, primarily as a car (blue dot), transitioning to a canoe (3 circles) at a river, a helicopter (5 circles) at a canyon and a submarine (3 circles and a periscope) at a lake. Each context uses a different T1-FIS, but could just as easily apply a GT2-FIS or other algorithm from the ADL.



**Figure 61 – Navigating Terrain Features (a) Transitioning to Canoe (b), Helicopter (c), Submarine Vehicle (d)**

This separation ultimately allows FLC implementation to maintain a separate algorithmic identity and lookup for each subspace and serves to reduce the complexity associated with extending an existing algorithm or implementing a hybrid solution.

Consider a subspace consisting of 2 contexts, one for a road and one for a river. The automated vehicle attempting to navigate both must incorporate distinct behaviors: in one case that of a car, in the other, a canoe, as well as two hybrid behaviors, the car-canoe when entering a river and canoe-car when leaving.



A T1-FIS attempting to do this will consist of a set of rules which can be formulated as:

$$\mathbf{Rule } R_k: \mathbf{IF } x_1 \mathbf{ is } A_1^k \mathbf{ AND } \dots \mathbf{ AND } x_n \mathbf{ is } A_n^k \mathbf{ THEN } y \mathbf{ is } B^k \quad (4.20)$$

The value of  $R_k$  can be computed by applying a t-norm operator to the rule antecedents as well as the rule consequents [Mendel 01]:

$$\mu_{R_k}(\vec{x}, y) = \mu_{A_1^k}(x_1) \prod \dots \prod \mu_{A_n^k}(x_n) \prod B_k(y) \quad (4.21)$$

which reduces to:

$$\mu_{R_k}(\vec{x}, y) = \left[ \prod_{i=1}^n \mu_{A_i^k}(x_i) \right] \prod B_k(y) \quad (4.22)$$

So given equation 4.22, the overall computational complexity of the T1-FIS can be said to be a function of the number of fuzzy rules. Combining the rules for two different T1-FIS, however, means applying the environmental “context” in which those rules operate. For example a simple rule for a car might be:

$$\mathbf{IF } \mathbf{FrontalDistance } \mathbf{ IS } \mathbf{Far } \mathbf{ THEN } \mathbf{Angle } \mathbf{ IS } \mathbf{Zero} \quad (4.23)$$

Whereas for the canoe it might be:

$$\mathbf{IF } \mathbf{Current } \mathbf{ IS } \mathbf{None } \mathbf{ THEN } \mathbf{Angle } \mathbf{ IS } \mathbf{Zero} \quad (4.24)$$

This would amount to evaluating two rules and four (FontalDistance, Far, Current, None) underlying fuzzy sets.

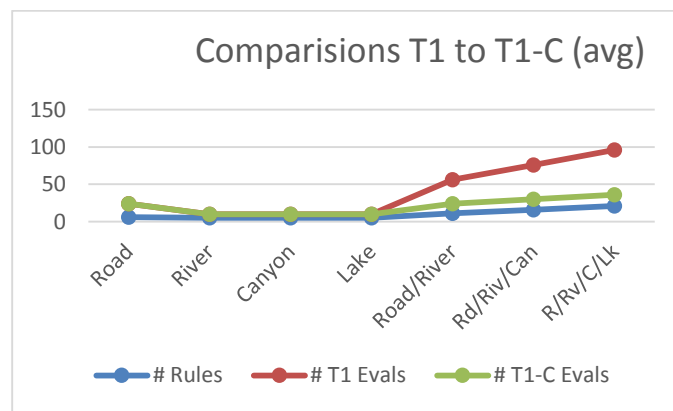
A combined equivalent system however would look like this:

$$\begin{aligned} \mathbf{IF } \mathbf{Surface } \mathbf{ IS } \mathbf{Road } \mathbf{ AND } \mathbf{FrontalDistance } \mathbf{ IS } \mathbf{Far } \mathbf{ THEN } \mathbf{Angle } \mathbf{ IS } \mathbf{Zero} \\ \mathbf{IF } \mathbf{Surface } \mathbf{ IS } \mathbf{River } \mathbf{ AND } \mathbf{Current } \mathbf{ is } \mathbf{None } \mathbf{ THEN } \mathbf{Angle } \mathbf{ IS } \mathbf{Zero} \end{aligned} \quad (4.25)$$

Hence, the combination, which forces the introduction of the “surface”, has more fuzzy sets to evaluate (8) as does the sum of the two individual approaches separately (4). [Mendis 10] shows that complexity increases are disproportionately greater as more rules are added. Type-C, in contrast, tests whether the input is “in context” before attempting to evaluate the corresponding  $\varphi$ . Only those contexts for which membership is nonzero need be evaluated. Hence the algorithm separation means it is not necessary to embed contextual information in the fuzzy rules as shown above, nor is it necessary to evaluate rules which are “out of context”. Table 8 compares a simple, extended T1-FIS to a corresponding Type-C for a quartet of possible scenarios for evaluating just the antecedent.

**Table 8 - Comparing Standalone and Contextual T1-FIS**

Configuration	# Rules	# T1 Evals	# T1-C Evals
Road	6	24	24
River	5	10	10
Canyon	5	10	10
Lake	5	10	10
Road/River	11	56	$10 \cdot 34 + 2\mu$
Rd/Riv/Can	16	76	$10 \cdot 44 + 3\mu$
R/Rv/C/Lk	21	96	$10 \cdot 54 + 4\mu$



**Figure 62 – Comparison Growth T1-FIS vs Contexts**

What about a hybrid algorithm? Or even a simple conditional? An argument can be made that the extended T1-FIS shown earlier is a case of a hybrid algorithm. Other algorithms might be more efficient than the T1-FIS, or even the TCF-FIS, but the hybrid

approach doesn't have a built-in mechanism to easily describe its intended purpose. Zadeh [Zadeh 65], [Zadeh 08] showed that one benefit of fuzzy sets was their ability to define themselves in simple, easily understandable terms such as TALL and SHORT. The Fuzzy Context, in this case, describes a surface, such as ROAD and RIVER; hence the associated contextual algorithm clearly describes its intended purpose – a navigating a ROAD in one case and navigating a RIVER in another.

Another disadvantage of the hybrid approach is it lacks a generic capacity to easily extend itself. In some cases, such as an Artificial Neural Network or Fuzzy Inference System, there is the ability to extend the algorithm through the addition of native components, such as neurons or fuzzy rules respectively, but the integration of a totally new technique, such as an Ant Colony Optimization, becomes a lot trickier. Under the TCF, it is simply a matter of generating an ADL and assigning it to a context.

As for the conditional approach, it is fairly straightforward to add a statement to a process such as:

***IF ROAD THEN***

***NavigateCar***

***ELSE***

***NavigateCanoe***

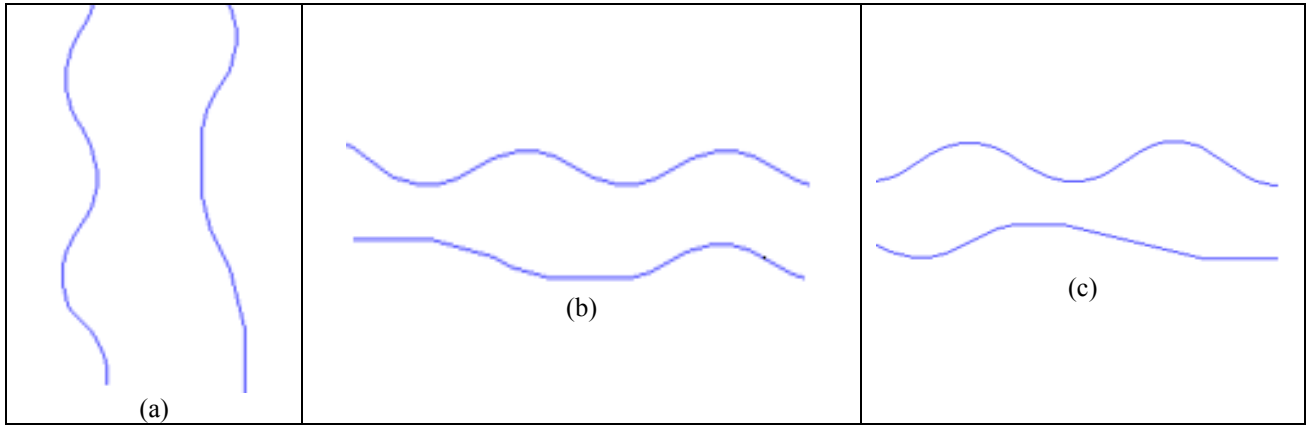
**(4.26)**

But the conditional suffers both from the lack of clarity of description for the problem above as well as a native ability to easily extend the algorithm with new approaches. Moreover a series of conditionals can be difficult to maintain as the series gets large [Pressman 09].

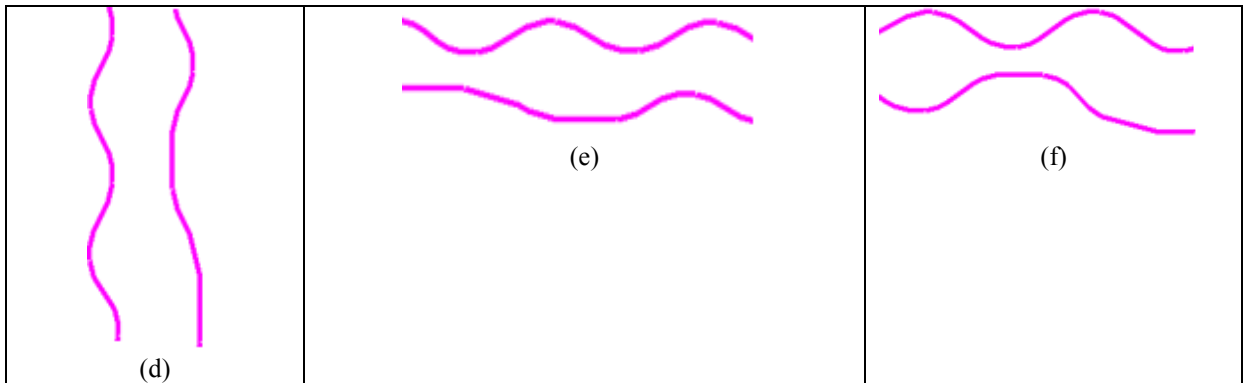
However, there is no free lunch either. The downside of the TCF compared to either approach is the need to test for membership in systems where more than one context is present. This adds some overhead but test results show this to be often less than the corresponding improvement in the overall process.







**Figure 65 – Comparing T1-FIS (left and top) and T1-FIS (right and bottom) with Contextual Subsumption for (a) left leg of maze, (b) top leg of maze, (c) bottom leg of maze**



**Figure 66 – Comparing GT2-FIS (left and top) and GT2-FIS (right and bottom) with Contextual Subsumption for (a) left leg of maze, (b) top leg of maze, (c) bottom leg of maze**

**Table 9 - Comparison of Traditional vs. Subsumptive (Inverted Context) Approaches**

	T1-FIS	GT2-FIS	T1-FIS-S	GT2-FIS-S
# Iterations	551	536	522	524
ms/Iteration	1.316	13.122	1.302	13.06

Differences in performance varied by roughly 1% between the traditional and subsumptive variations. The contextual/subsumptive version however showed a significant improvement in the number of iterations required to complete the maze. The T1-FIS required

5.3% fewer iterations while the GT2-FIS required 2.2% fewer. This more than overcomes any contextual overhead. Note the smoother paths in figures 65, and 66. This smoother path resulted in a shorter distance required to navigate the entire maze.

#### 4.4.3: Other Enhancement Tests

Combining the framework and tools allows for test examples which involve applying the FC-OAP to a diverse set of simple problems:

1. The traditional 8-Queens problem, trying to arrange a random set of queens on a chessboard so that none can attack any others.
2. A navigation problem: trying to navigate a software robot around a simple maze
3. A sorting problem: trying to alphabetically sort a list of names.

The algorithm repository contains a total of 16 ADLs representing a number of different approaches to problem solving:

1. 10 different local search techniques (LST)
2. Fuzzy Type-1 and Type-2 Inference System
3. 4 sorting algorithms (Quicksort, Bubblesort, Mergesort, Heapsort)

The local search techniques are based upon work done in [McCarty 08b], [Martinjak 07], and [Russell 09]. The Fuzzy Inference Systems are based upon work done in [McCarty 13]. The Quicksort and Bubblesort come from [Cormen 09].

Testing an algorithm requires the following steps:

1. Generate a list of available ADLs.
2. For each ADL look up and generate the corresponding algorithm
3. Test the algorithm for suitability
4. If suitable, add inputs and execute
5. Examine results and determine rank using greedy algorithm based upon ranking criteria

Pseudo-code for the test process is as follows:

**Pseudo-code 14 – Testing Algorithms against a problem**

```
Algorithm: TestAlgorithms(x)
Input: a problem definition x
Output: an array of results rarr
Begin
  Determine Available Algorithms
  FOR EACH algorithm
    Add Inputs
    Execute algorithm
    Rank results
    Add ranking to rarr
  NEXT algorithm

  return rarr
End
```

*4.4.3.1: Test Example #10 - The 8-Queens problem*

The 8-Queens problem is a classical problem in computer science in which an algorithm starts with 8 queens arranged randomly on a chess board. The algorithm must then determine how to arrange the queens in such a way so that no one queen can attack another as shown in figure 67.



				Q			
							Q
			Q				
Q							
						Q	
	Q						
					Q		
		Q					

**Figure 67 – A solution for the 8-Queens problem**

To see how the framework responds to different suitability functions, each algorithm was evaluated using three different suitability criteria: 1) Speed for successful resolution, 2) Accuracy of tries and 3) Speed and Accuracy.

The program determined the FISs and sorting algorithms were unsuitable for this problem, but that each of the 10 LSTs could be used. It ran each LST through 100,000 iterations of the problem, taking note of how long each iteration took, and how many of the initial configurations were actually solved. Results are presented in table 10.

**Table 10 - Comparing Algorithms for the 8-Queens Problem**

Algorithm	Successes	Failures	Time
Hill Climb	3343	21196	56.659
Stochastic Hill Climb	2596	16429	56.191
DD-Stochastic Hill Climb	3343	21115	58.94
Random Restart Hill Climb	1265	162	21.962
Simulated Annealing	85	70	20.165
Genetic Mutation	45	154	53.318
Minimum Conflicts	3202	334	25.893
Tabu Search	3408	1506	2.642
DD-Simulated Annealing	228	1	18.93
Memetic Mutation	86	51	3.398

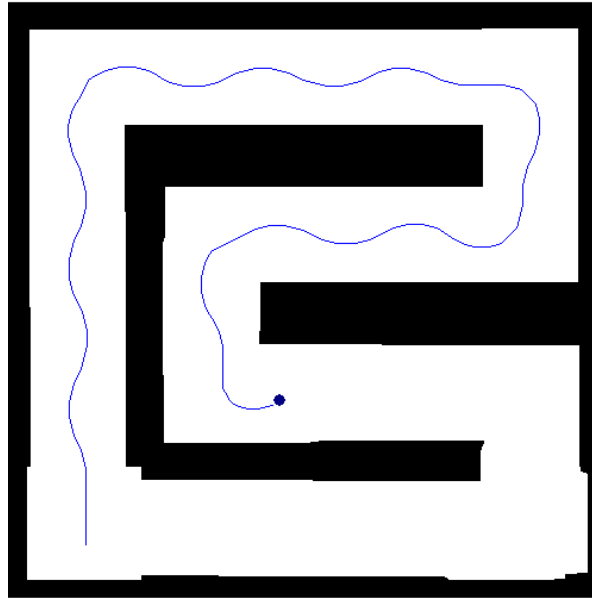
The framework then generated and presented the LST rankings per the specified criteria. Ties were broken by giving slightly more weight to speed of resolution. The results are shown in table 11.

**Table 11 - Ranking Solutions to the 8-Queens Problem**

Algorithm	Speed	Accuracy	Speed + Accuracy
Hill Climb	3	10	6
Stochastic Hill Climb	6	9	8
DD-Stochastic Hill Climb	5	8	7
Random Restart Hill Climb	4	3	3
Simulated Annealing	9	6	9
Genetic Mutation	10	7	10
Minimum Conflicts	2	2	1
Tabu Search	1	4	2
DD-Simulated Annealing	8	1	4
Memetic Mutation	7	5	5

#### 4.4.3.2 Test Example #11 - Navigation problem

Back to the navigation problem. Consider a robot car whose goal is trying to navigate a simple maze as illustrated in figure 68.



**Figure 68 - A Robot Car Navigating a Maze**

The car must figure out how to move through the maze while also maintaining an efficient path and avoiding walls and obstacles. Doing this requires continual adjustment of its direction in order to avoid coming too close.

Among the algorithms, the framework chose the Type-1 and Type-2 FIS and applied them to the navigation problem. Suitability was determined as a function of speed and accuracy defined as minimal number of iterations to run a complete circuit of the maze. Results of the initial suitability determination are shown in table 12.

**Table 12 - Comparative Analysis T1-FIS vs. T2-FIS**

	Time	# Iterations
T1-FIS	1.313	551
T2-FIS	11.562	522

In this case, as before, the tiebreaker is speed. Hence the ranking as shown in table 13:

**Table 13 - Ranking T1-FIS vs. T2-FIS**

Algorithm	Speed	Accuracy	Spd + Acc
T1-FIS	1	2	1
T2-FIS	2	1	2

Taking this example a step further, the algorithms in this case are also coded as optimizable in the ADL. As a result the framework added an optimization step using the memetic algorithm as described in [McCarty 14a]. Fuzzy sets were randomized to generate a population sample of 30 T1-FIS which were tested and genetically altered. The resulting optimization reduced the number of iterations required to complete the maze from 551 to 524, allowing the optimized T1-FIS to outperform the original T2-FIS in this instance.

#### *4.4.3.3: Test Example #12 - Sorting problem*

Next the framework was tasked with sorting a simple list. A list of 20,000 last names was queried from a customer database in first name order to prevent any accidental ordering of last names. The framework searched the algorithm repository and determined the 4 sorting algorithms were suitable and ran each using a suitability metric of speed only since accuracy of 100% is assumed. Table 14 shows the results:

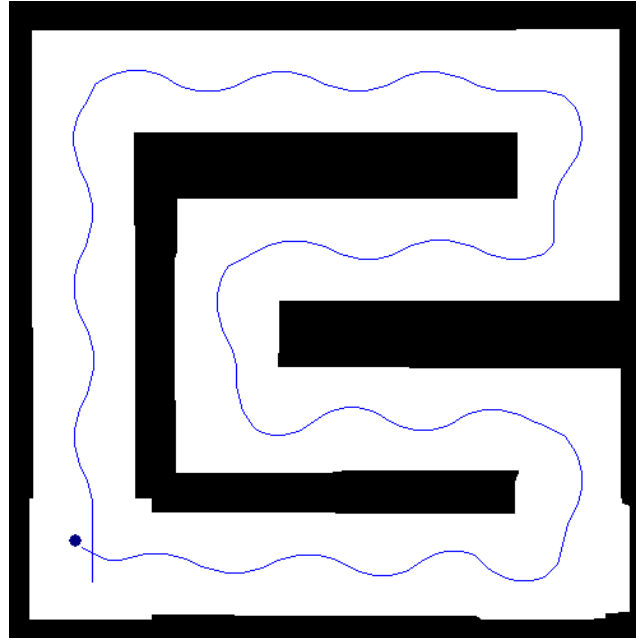
**Table 14 - Comparative Analysis and Ranking of Sorting Algorithms**

Algorithm	Speed	Rank
Quick Sort	0.516	3
Bubble Sort	56.45	4
Merge Sort	0.509	2
Heap Sort	0.149	1

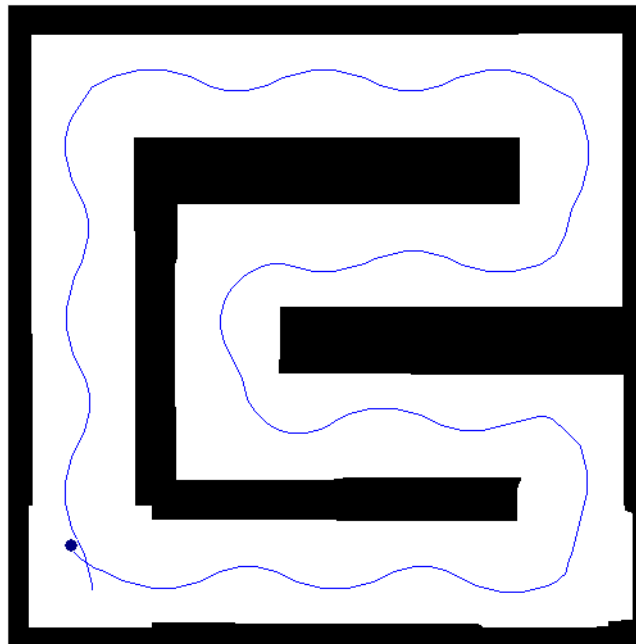
*4.4.3.4: Test Example #13 Memetic Optimization of working configuration*

In this example, a working FIS is run through the optimizer. 500 iterations was chosen as the number of steps to evaluate. Results were saved to a SQL Server 2012 database. Deviation from the “optimal” path by the original FIS was 2011 pixels.

The memetic algorithm is a combination of stored procedures and uses a sample of 30 randomly generated variations of a given fuzzy set within an FIS. It adjusts the endpoints of losers via crossover mutation with the best in the sample. Finally it performs a random mutation before reevaluating the results with the sample and seeing how the closely the new path tracked the optimal one. The best sample deviated by 1824 pixels from optimal. The resulting track showed sharper turns and fewer back and forth adjustments as shown in fig 69 and 70.



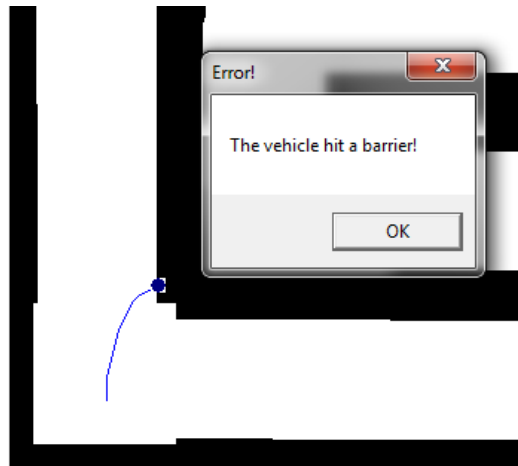
**Figure 69 - A Robot Car Navigating a Maze, Typical FIS – Type 1 FLC**



**Figure 70 - A Robot Car Navigating a Maze, Optimized T1-FIS**

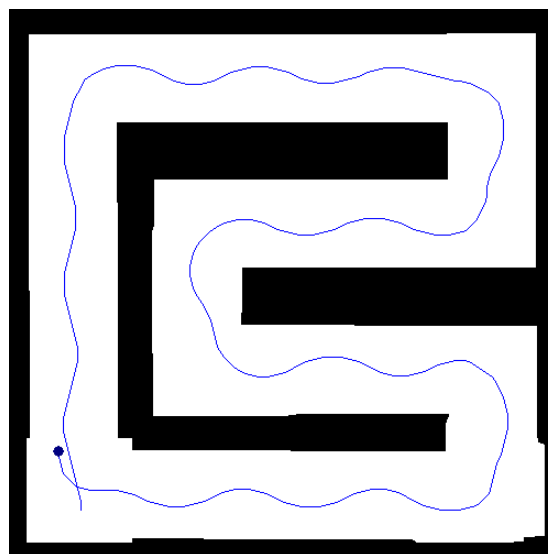
4.4.3.5: Test Example #14 *Memetic Optimization of a non-working configuration*

A fuzzy set is adjusted to force the car to crash. Then the FIS was run through the optimizer which was able to find and adjust the bad fuzzy set, creating a working FIS as demonstrated in figures 71 and 72.



**Figure 71 - Bad FIS cause robot car to hit barrier**

The FIS is run through the optimizer for 30 iterations. After completion, it choose from among the population sample a working configuration.

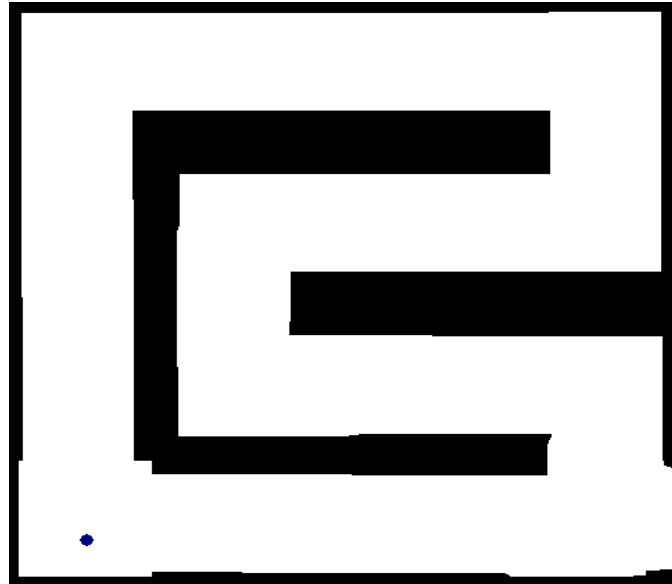


**Figure 72 - Bad FIS repaired**



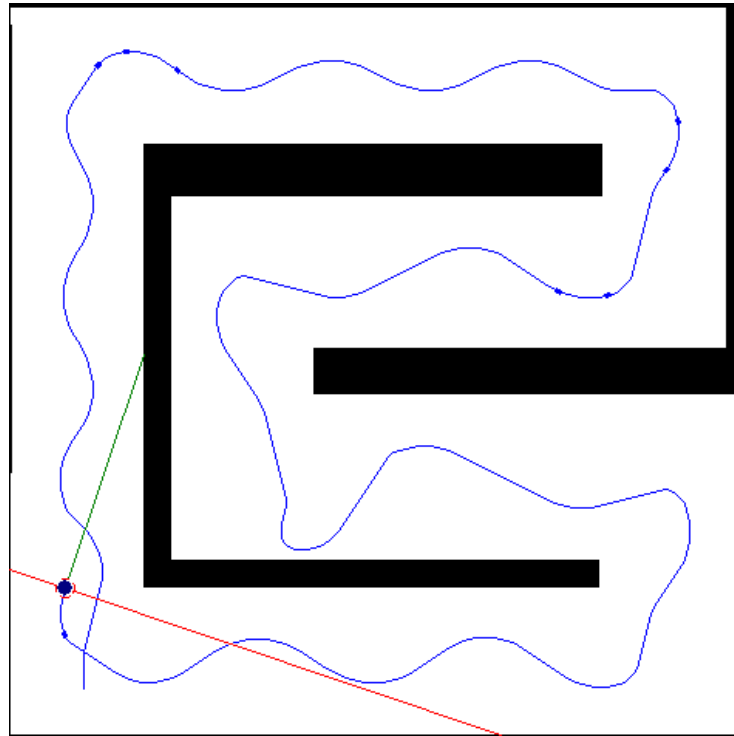
4.4.3.6: Test Example #15 *Unsupervised learning of simple context*

To test a simple example of the unsupervised learning technique, take another look at the robot car and the maze:



**Figure 73 - Robot in a Maze**

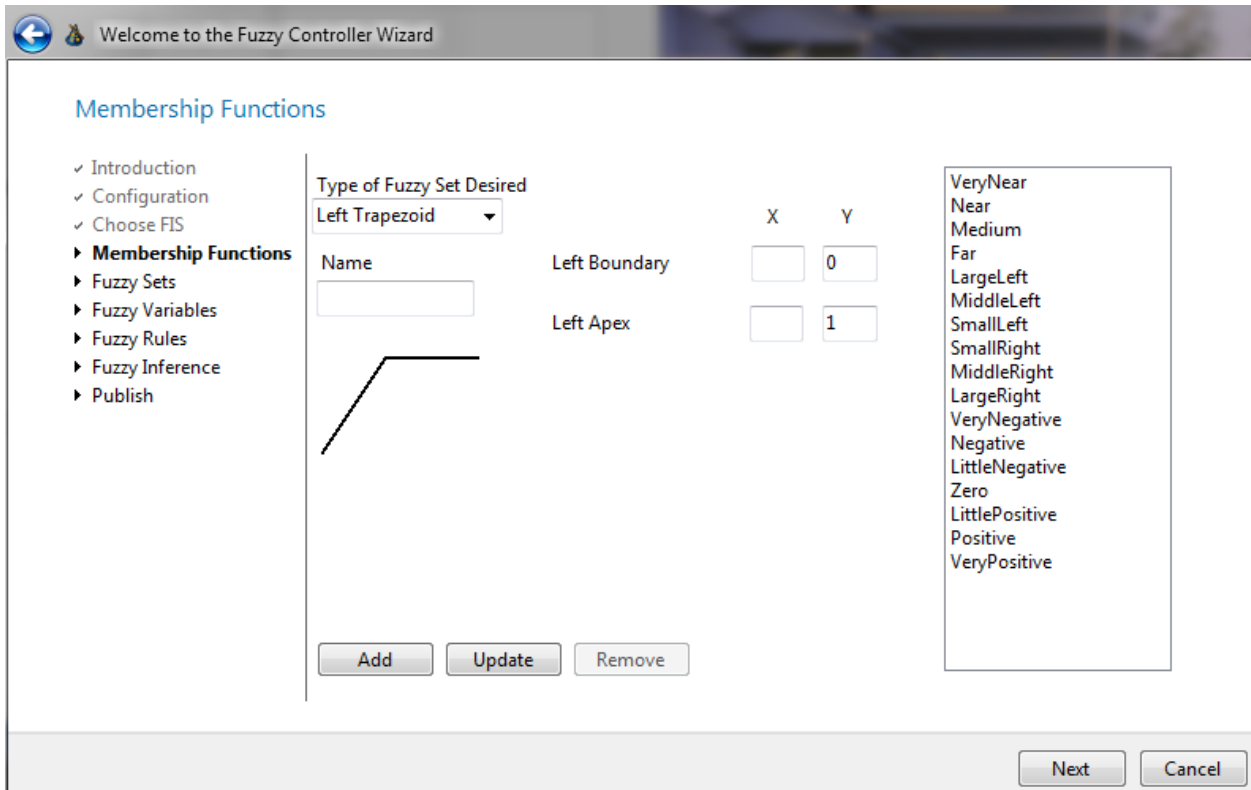
Note that there are straight areas and places to turn. Suppose the robot car only knew about a straight road. In this case, any turn would be significantly different and require a new context. Could it learn how to pick the next context? For this test, the framework was modified to add a training component used to generate the fitness function. By restricting training to the straight portions, the robot car now has to learn about the curves. While the initial FIS is capable of navigating the curves, the difference between the curve and straight portion is enough to flag each curve as a new context. Figure 74 shows that the algorithm “discovered” potential contexts, marking them with blue dots.



**Figure 74 – Algorithm Learning Contexts**

*4.4.3.7: Test Example #16 – Building Workable XML Using Wizard Tool*

Recall the Fuzzy Configuration Wizard Tool, a sample page of which is shown in figure 75:



**Figure 75 – The Fuzzy Wizard Tool**

The resulting XML was used by the framework to build a working FIS that was able to successfully navigate the maze. The XML generated is listed below:

```
<?xml version="1.0" encoding="UTF-8"?>
<FuzzyWizardConfig>
<FuzzyInferenceSystem Name="MySystem" OutputToDB="true">
<FuzzyDB Name="FuzzyDB"/>
<Defuzzifier Name="Defuzzifier" Intervals="1000"/>
<Rules>
<FuzzyRule Name="Rule 1"/>
<FuzzyRule Name="Rule 2"/>
<FuzzyRule Name="Rule 3"/>
<FuzzyRule Name="Rule 4"/>
<FuzzyRule Name="Rule 5"/>
<FuzzyRule Name="Rule 6"/>
<FuzzyRule Name="Rule 7"/>
</Rules>
<SQLCn Name="Server = KMCCARTY-L01;Database = FuzzyTests;
Trusted_ConnectionTrue;"/>
```

```

</FuzzyInferenceSystem>
<MembershipFunctions>
<MembershipFunction Name="VeryNegativeF" FuzzySetType="Right Trapezoid">
<Points>
<Point Id="1" X="-40" Y="1"/>
<Point Id="2" X="-35" Y="0"/>
</Points>
</MembershipFunction>
<MembershipFunction Name="NegativeF" FuzzySetType="Full Trapezoid">
<Points>
<Point Id="1" X="-40" Y="0"/>
<Point Id="2" X="-35" Y="1"/>
<Point Id="3" X="-25" Y="1"/>
<Point Id="4" X="-20" Y="0"/>
</Points>
</MembershipFunction>
<MembershipFunction Name="LittleNegativeF" FuzzySetType="Full Trapezoid">
<Points>
<Point Id="1" X="-25" Y="0"/>
<Point Id="2" X="-20" Y="1"/>
<Point Id="3" X="-10" Y="1"/>
<Point Id="4" X="-5" Y="0"/>
</Points>
</MembershipFunction>
<MembershipFunction Name="ZeroF" FuzzySetType="Full Trapezoid">
<Points>
<Point Id="1" X="-10" Y="0"/>
<Point Id="2" X="-5" Y="1"/>
<Point Id="3" X="5" Y="1"/>
<Point Id="4" X="10" Y="0"/>
</Points>
</MembershipFunction>
<MembershipFunction Name="LittlePositiveF" FuzzySetType="Full Trapezoid">
<Points>
<Point Id="1" X="5" Y="0"/>
<Point Id="2" X="10" Y="1"/>
<Point Id="3" X="20" Y="1"/>
<Point Id="4" X="25" Y="0"/>
</Points>
</MembershipFunction>
<MembershipFunction Name="PositiveF" FuzzySetType="Full Trapezoid">
<Points>
<Point Id="1" X="20" Y="0"/>
<Point Id="2" X="25" Y="1"/>
<Point Id="3" X="35" Y="1"/>
<Point Id="4" X="40" Y="0"/>

```

```

</Points>
</MembershipFunction>
<MembershipFunction Name="VeryPositiveF" FuzzySetType="Left Trapezoid">
<Points>
<Point Id="1" X="35" Y="0"/>
<Point Id="2" X="40" Y="1"/>
</Points>
</MembershipFunction>
<MembershipFunction Name="NearF" FuzzySetType="Right Trapezoid">
<Points>
<Point Id="1" X="15" Y="1"/>
<Point Id="2" X="50" Y="0"/>
</Points>
</MembershipFunction>
<MembershipFunction Name="MediumF" FuzzySetType="Full Trapezoid">
<Points>
<Point Id="1" X="15" Y="0"/>
<Point Id="2" X="50" Y="1"/>
<Point Id="3" X="60" Y="1"/>
<Point Id="4" X="100" Y="0"/>
</Points>
</MembershipFunction>
<MembershipFunction Name="FarF" FuzzySetType="Left Trapezoid">
<Points>
<Point Id="1" X="60" Y="0"/>
<Point Id="2" X="100" Y="1"/>
</Points>
</MembershipFunction>
</MembershipFunctions>
<FuzzySets>
<FuzzySet Name="VeryNegative" MembershipFunction="VeryNegativeF"/>
<FuzzySet Name="Negative" MembershipFunction="NegativeF"/>
<FuzzySet Name="LittleNegative" MembershipFunction="LittleNegativeF"/>
<FuzzySet Name="Zero" MembershipFunction="ZeroF"/>
<FuzzySet Name="LittlePositive" MembershipFunction="LittlePositiveF"/>
<FuzzySet Name="Positive" MembershipFunction="PositiveF"/>
<FuzzySet Name="VeryPositive" MembershipFunction="VeryPositiveF"/>
<FuzzySet Name="Near" MembershipFunction="NearF"/>
<FuzzySet Name="Medium" MembershipFunction="MediumF"/>
<FuzzySet Name="Far" MembershipFunction="FarF"/>
</FuzzySets>
<FuzzyVariables>
<FuzzyVariable Name="Angle" UpperBound="50" LowerBound="-50" Type="Output">
<FuzzySet Name="VeryNegative"/>
<FuzzySet Name="Negative"/>
<FuzzySet Name="LittleNegative"/>

```

```

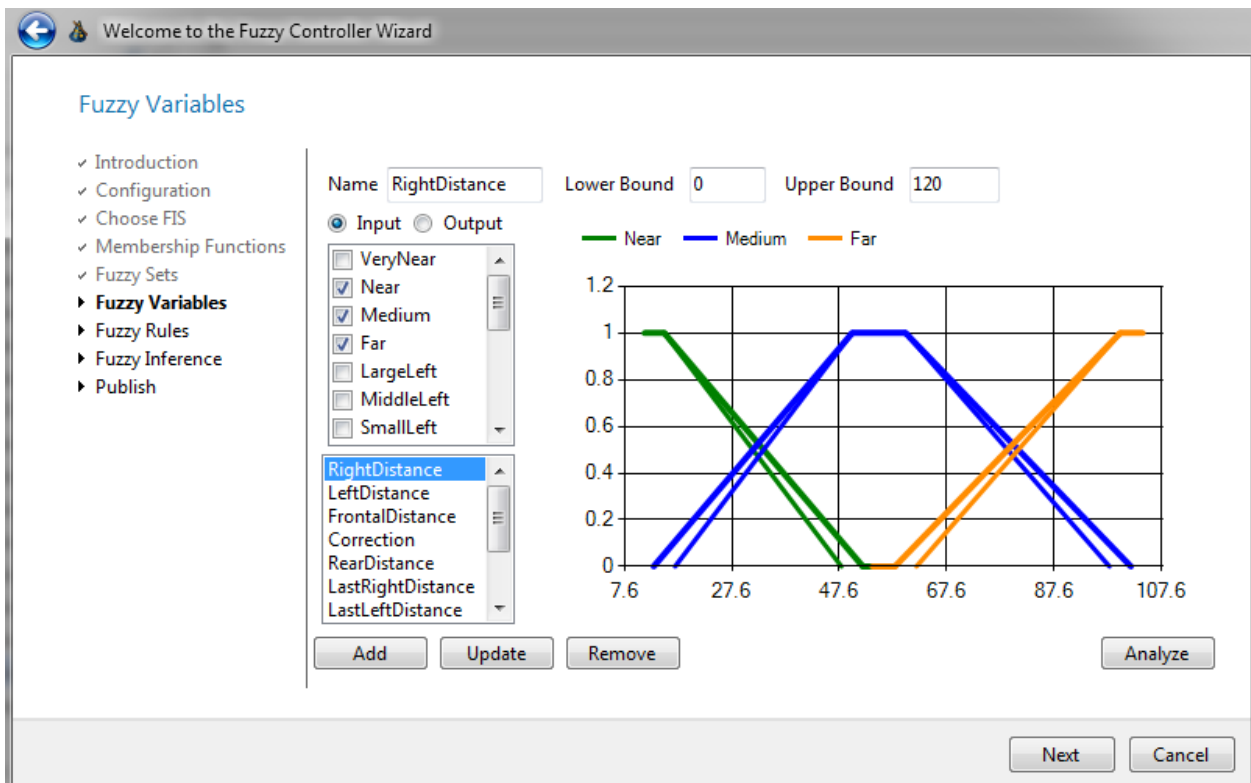
<FuzzySet Name="Zero"/>
<FuzzySet Name="LittlePositive"/>
<FuzzySet Name="Positive"/>
<FuzzySet Name="VeryPositive"/>
</FuzzyVariable>
<FuzzyVariable Name="RightDistance" UpperBound="120" LowerBound="0"
Type="Input">
<FuzzySet Name="Near"/>
<FuzzySet Name="Medium"/>
<FuzzySet Name="Far"/>
</FuzzyVariable>
<FuzzyVariable Name="LeftDistance" UpperBound="120" LowerBound="0"
Type="Input">
<FuzzySet Name="Near"/>
<FuzzySet Name="Medium"/>
<FuzzySet Name="Far"/>
</FuzzyVariable>
<FuzzyVariable Name="FrontalDistance" UpperBound="120" LowerBound="0"
Type="Input">
<FuzzySet Name="Near"/>
<FuzzySet Name="Medium"/>
<FuzzySet Name="Far"/>
</FuzzyVariable>
</FuzzyVariables>
<FuzzyRules>
<FuzzyRule Name="Rule 1" Text="IF FrontalDistance IS Far THEN Angle IS Zero"/>
<FuzzyRule Name="Rule 2" Text="IF FrontalDistance IS Far AND RightDistance IS Far
AND LeftDistance IS Far THEN Angle IS Zero"/>
<FuzzyRule Name="Rule 3" Text="IF RightDistance IS Near AND LeftDistance IS Not
Near THEN Angle IS LittleNegative"/>
<FuzzyRule Name="Rule 4" Text="IF RightDistance IS Not Near AND LeftDistance IS
Near THEN Angle IS LittlePositive"/>
<FuzzyRule Name="Rule 5" Text="IF RightDistance IS Far AND FrontalDistance IS
Near THEN Angle IS Positive"/>
<FuzzyRule Name="Rule 6" Text="IF LeftDistance IS Far AND FrontalDistance IS Near
THEN Angle IS Negative"/>
<FuzzyRule Name="Rule 7" Text="IF RightDistance IS Far AND LeftDistance IS Far
AND FrontalDistance IS Near THEN Angle IS Positive"/>
</FuzzyRules>
</FuzzyWizardConfig>

```

Figure 76 – XML for a Type-1 FIS

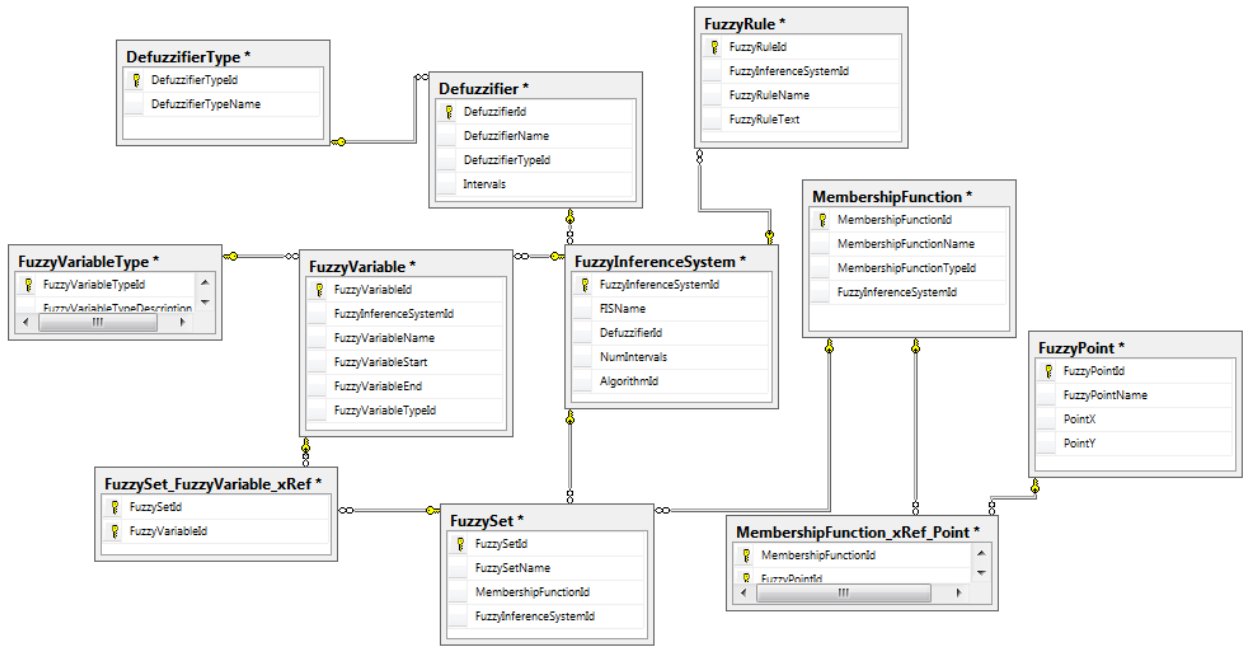
#### 4.4.3.8 Test Example #17 – Building Workable Database Configuration Using Wizard Tool

The test example used the Wizard Tool to construct a General Type-2 FIS configuration stored in a SQL Server 2012 database. A sample page is shown in figure 77:



**Figure 77 – General Type-2 FIS in Wizard**

The resulting database records were used by the framework to build a working FIS that was able to successfully navigate the maze. The database schema is shown below:



**Figure 78 – Database Schema for a Fuzzy Configuration**



## 4.5: CHAPTER SUMMARY AND COMPARATIVE ANALYSIS

This chapter demonstrates some of the specific advantages of the Type-C Framework (TCF) using test examples. In general, the concepts and software presented in this dissertation compare favorably to other approaches in these areas:

1. Compared to other fuzzy frameworks.
2. Compared to traditional Fuzzy Type-1 and Type-2 and hybrid approaches for solving SDPs.

### 4.5.1: TCF vs. Other Frameworks

Compared to the frameworks offered from [Karnik 14], [Kirillov 14], [Octave 13], [Slavicek] and [Zamani 08], only the TCF provides an integrated Type-1/Type-2 capability for a generalized FIS (at this time it is the only known framework to do so). Hence, only the TCF can provide a truly objective and polymorphic FIS supporting both Type-1 and Type-2 characteristics. [Zamani 08] also provides a wizard tool for constructing an IT2-FIS; however, it only generates Matlab m-files. These files, unlike XML or a database record, are not portable to other languages, hence their usage is limited only to Matlab without some form of intermediary interpretation.

### 4.5.2: TCF vs. Traditional Fuzzy Type-1, Type-2 and other approaches

Consider a very simple FIS with two disparate dimensions  $d_1, d_2$ . Each dimension has an input fuzzy set  $f_i$ , and a fuzzy rule  $f_r$ . Let there also be a pair of output fuzzy sets  $f_{o1}, f_{o2}$  shared by the dimensions. The rule base for the two dimensions is as follows:

$$\begin{aligned} f_{r1} &\equiv \text{IF } d_1 \text{ IS } f_{i1} \text{ THEN } f_{o1} \text{ IS } f_{o2} \\ f_{r2} &\equiv \text{IF } d_2 \text{ IS } f_{i2} \text{ THEN } f_{o1} \text{ IS } f_{o2} \end{aligned} \tag{4.27}$$

Four input fuzzy set evaluations are required for each input in order to fuzzify the two rules. However, because the two dimensions are disparate, as is the case in an SDP, combining them in a traditional FIS requires introducing additional input fuzzy sets to link

them together. A minimum of one fuzzy set is required to generalize the dimensions  $d_1, d_2$ , call it  $f_d$ , in addition to two more fuzzy sets to test for the membership of  $d_1, d_2$ , call them  $f_{d1}, f_{d2}$ . So the new combined FIS would require modifications to the two original rules:

$$\begin{aligned} f_{r1} &\equiv \text{IF } f_d \text{ IS } f_{d1} \text{ AND } d_1 \text{ IS } f_{i1} \text{ THEN } f_{o1} \text{ is } f_{o2} \\ f_{r2} &\equiv \text{IF } f_d \text{ IS } f_{d2} \text{ AND } d_2 \text{ IS } f_{i2} \text{ THEN } f_{o1} \text{ is } f_{o2} \end{aligned} \quad (4.28)$$

Now eight fuzzy set evaluations are required to fuzzify the two rules. Consider that same system using Fuzzy Contexts. Each context is evaluated separately, hence there is no need to add the additional fuzzy sets of the extra dimensions. The equivalent Type-C rule would look like this:

$$\begin{aligned} f_{r1} &\equiv \text{IF } d_1 \text{ IS } f_{i1} \text{ THEN } f_{o1} \text{ is } f_{o2} \quad \times \text{ Contextual Membership} \\ f_{r2} &\equiv \text{IF } d_2 \text{ IS } f_{i2} \text{ THEN } f_{o1} \text{ is } f_{o2} \quad \times \text{ Contextual Membership} \end{aligned} \quad (4.29)$$

Which only requires evaluation and fuzzification of the original four sets.

Computational complexity for a fuzzy inference system is based upon the number of rules and the underlying number of fuzzy set evaluations [Mendis 10]. Assuming for simplicity's sake that the dimensions share the output fuzzy sets, this leads to the relative number of fuzzy set evaluations dependent upon the following:

1. Input Fuzzy Sets,  $f_i$
2. Test Dimension Fuzzy Sets (for multiple dimensions),  $f_t$
3. Generalization fuzzy sets (to bridge or generalize multiple dimensions),  $f_g$

Hence the relative processing requirement,  $P_r$  is a product of the three items listed above:

$$P_r = \sum f_i \sum f_t \sum f_g \quad (4.30)$$

Using Fuzzy Contexts, however, replaces  $f_t$  and  $f_g$  with a contextual contribution applied to  $f_i$ . Now  $P_r$  is a product of just the original  $f_i$  applied to  $C_j$  where  $C_j$  is the corresponding context contribution.

$$P_r = \sum C_j f_i \quad (4.31)$$

For simple problems  $f_i$  and  $f_g$  can reduce down to 1 at best. Hence the number of fuzzy set evaluations using a contextual FIS will always be equal or less than the equivalent combined FIS. Contextual calculations will add some overhead but test results show cases where that is less than the overhead due to the extra fuzzification.

What about a hybrid or generalized algorithm? Or even a simple conditional? An argument can be made that the complex FIS is simply a case of a hybrid algorithm and the same rules apply. In addition, Zadeh [Zadeh 65], [Zadeh 08] showed that one benefit fuzzy sets was their ability to define themselves in simple, easily understandable terms such as TALL and SHORT which is not a traditional characteristic of a hybrid or generalized approach.

[Linda 11b] describes an implementation to determine the relative performance differences between a Type-1 FIS and IT2-FIS. In it the author remarks how there are situations where a T1-FIS is preferred over the IT2-FIS and vice versa. Using the TCF over the implementation described has two distinct advantages:

1. A combined T1/T2 FIS under the TCF requires only one set of shared fuzzy rules instead of two separate sets of rules.
2. The relative advantages of each system can be contextualized, allowing for a single unified system taking advantage of the relative strengths of each approach.

#### 4.5.3: How TCF reduces the problem of over-fitting in ANNs

Finally, although not directly addressed via test examples, Fuzzy Contexts also have the ability to reduce the problem of over-fitting that affects some traditional Artificial Neural Networks (ANNs). [Zurada 92] and [Haykin 09] speak to this common problem which is primarily the result of the following two issues:

1. The inclusion of outliers in the data
2. Too many data points leading to an over-generalization of the classifier  
- and correspondingly overly large classifier construction.

[Haykin 09] contends that a good generalization for an ANN requires the size of the training sample  $N$  to satisfy the condition:

$$N = O\left(\frac{W}{\epsilon}\right) \quad (4.32)$$

where  $W$  is the total number of parameters (i.e. weights and biases) in the network and  $\epsilon$  represents the fraction of classification errors permitted on the test data and  $O(\cdot)$  denotes the order of quantity enclosed. Now suppose the data comprises two distinct states  $S_A$  and  $S_B$  with different behaviors for each but consistent within each subset.

Now apply a contextual classification for each distinct state,  $C_A, C_B$  and let  $P$  be original (untrained) sample. Hence  $C_A, C_B$  are disjoint sets that combine to equal  $P$ .

$$P = C_A \cup C_B. \quad (4.33)$$

Let the size of the training sample be  $N$  as defined for sample  $P$

By definition  $C_A, C_B$  are internally consistent, being classified contextually. We know that worst case, by simply combining  $C_A, C_B$  to get  $P$  the original generalization rule still applies, hence

$$N = T_{C_A \cup C_B} \quad (4.33)$$

Because  $C_A, C_B$  are disjoint, it follows that their training samples are disjoint as well, hence:

$$T_A + T_B \subseteq N \quad (4.34)$$

Where  $T_A$  is the training sample for  $C_A$  and  $T_B$  is the training sample for  $C_B$ . This implies the size of the training sample for  $C_A, C_B$ , will be at most as large as  $N$ .

$$Size_{T_A} + Size_{T_B} \leq Size_N \quad (4.35)$$

However, as  $C_A, C_B$ , are internally consistent, by definition, it also follows that they are not consistent with each other. The original training sample, in order to account for the inconsistency between the two subsets must therefore, in order to maintain the original  $\varepsilon$ , have at least one additional sample to account for the inconsistency and distinguish between the two contexts. This implies that the contextualized ANNs will actually require fewer samples to maintain the same  $\varepsilon$ .

[Han 11] and [Cox 05] also discuss over-fitting involving fuzzy sets, rules and other classifiers with the same general issues as affect the ANN. As such, similar benefits from a Type-C approach applied to other classification techniques are expected.

## CHAPTER 5: WORK IN PROGRESS

Future work is ongoing in order to broaden both the functionality and applicability of the framework. Among the planned extensions are a Wizard for configuring other algorithms, additional unary and binary fuzzy operators, discrete fuzzy sets and fuzzy numbers, and complex rules as well as improvements to the Wizard interface in general and the demo software. Also in progress are the addition of more sophisticated genetic, memetic and local search algorithms and data mining techniques into the framework. The intent is to enhance the creation of a framework that is “trainable” and self-optimizing.

Future work needs to be done to generalize and demonstrate how Fuzzy Contexts and weights can be generated using unsupervised classification methods as well as differing shape definitions using complex functions such as Radial Basis Functions and point-wise definitions for more radical contexts. Many of the individual components are still not tightly integrated or particularly robust so the framework and demo software still require a lot of tweaking to highlight individual functionality. Because the overall software suite is fairly extensive it could use an equally sophisticated suite of support tools for testing and debugging.

Future work is underway to further expand the scope of the framework by adding additional algorithms and optimization techniques. Additional heuristics are planned to both more quickly and more accurately determine which algorithms from among the algorithm repository make the best candidates for determination testing along with properties which better link an algorithm to the underlying problem. Tighter integration with existing Type-C class libraries is also needed to provide superior capabilities for application development and testing. In addition, more work needs to be done to improve the Wizard application to build and manage algorithm definitions, along with the related class libraries. Further abstractions are required to integrate generalized fitness functions and their corresponding specifications. Lastly, conversion to another language such as C++ or Java needs to be undertaken to allow for a larger audience of potential users.

This dissertation has as one of its goals the extension of work done by the University of Idaho Modern Heuristics Research Group along with Mendel (University of Southern California, John (University of Nottingham) and others to educate and explain the advantages of fuzzy solutions, particularly Fuzzy Logic Type-2 (for Mendel’s efforts see

<http://sipi.usc.edu/~mendel/software/>). The fuzzy contextual framework is a necessary first step in the creation of an easy-to-use, flexible, standalone utility able to create usable, sophisticated T1-FLCs, T2-FLCs and TC-FLCs. The Wizard application provides a useful assist both in the creation of Fuzzy definitions and as a teaching tool. For implementation, a simple software robot, relying upon the framework, provides a nice visual to cleanly and correctly navigate a maze under both a Type-1 and Type-2 controller embedded into a Type-C object.

## CHAPTER 6: CONCLUSION

### SUMMARY OF CONTRIBUTIONS

This dissertation began with a series of specific goals. Each goal was addressed and resulted in one or more contributions. A summary of the goals and contributions of this dissertation is as follows:

1. What Fuzzy Logic Type-C is and how it can be used.
  - a. Chapter 3, sections 1-5 presented an overview of Fuzzy Logic Type-C as an n-dimensional, extensible, hierarchical abstraction that uses fuzzy membership within “contexts” and linguistic terms to associate a specific algorithm to a context within a problem space. Fuzzy Logic Type-C combines Fuzzy Contexts and their corresponding problem spaces into a fuzzymorphic algorithmic approach. Chapter 3 presents pseudo-code and definitions on how to build and use Fuzzy Contexts within an application.
2. How Fuzzy Logic Type-C fits within general algorithm techniques and advantages for Type-C implementations as compared to various alternatives.
  - a. Chapter 3 and 4 establish a software framework for Type-C development along with a number of tools and enhancements for defining and optimizing the algorithm. Chapter 4 presents a number of examples of Type-C applications and advantages they have over other approaches.
3. How Fuzzy Logic Type-C is distinguished from Fuzzy Logic Type 1 or Type 2, or other traditional hybrid and non-hybrid algorithms.
  - a. Chapter 3 describes how the Type-C framework is distinguished from other approaches. Chapters 3 and 4 demonstrate how a Type-C implementation uses Fuzzymorphism to both emulate and supersede a variety of other algorithmic approaches.
  - b. Chapter 4 describes how contexts can be “learned” in an unsupervised process.
4. How Fuzzy Logic Type-C can be used as a software framework for implementations.



- a. Chapter 4 describes the various components of the Type-C framework in detail and provides many examples of implementation.
5. What Fuzzymorphism is and how Fuzzy Contexts achieve Fuzzymorphism.
  - a. Chapter 3 describes Fuzzymorphism and Chapter 4 shows how Fuzzymorphism in an implementation can improve the performance and simplicity of an underlying technique.
6. The usefulness of relational databases to a Type-C framework
  - a. Chapter 4 does a comparative analysis of the relative speed advantages of using a relational database and provides examples of how a database can assist in a Type-C or non-Type-C implementation.
  - b. Chapter 4 describes an optimization technique using a memetic algorithm against a relational database.
7. How to implement dynamic algorithms and algorithm definitions in support of a Type-C application.
  - a. Chapter 3 describes a Fuzzy Definition Language as the basis for a more generalized Algorithm Definition Language.
  - b. Chapter 4 introduces a novel extension to the Fuzzy Definition Language to incorporate Fuzzy Type-2 definitions and expands the original Fuzzy Definition Language to the more general Algorithm Definition Language.
  - c. Chapter 3 and 4 describe the Wizard tool, used as a visual representation of Fuzzy Type-1 and Type-2 definitions.
  - d. Chapter 4 presents numerous test examples of algorithm switching achieved using the Algorithm Definition Language and algorithmic polymorphism achieved using Fuzzymorphism.
8. Applications of Fuzzy Contexts in solving a diverse spectrum of problems.
  - a. Chapter 4 presents a number of variations on navigation problems, an 8-Queens solver and sorting algorithm within contexts.
9. Additional contributions: Minor Contributions #1 and #2 demonstrate how Fuzzy Contexts can be applied to improve advanced data mining techniques. Minor Contribution #1 describes a top-down technique called CoFuH-DT which is a mechanism for contextually pruning fuzzy decision trees. Minor Contribution #2

describes a bottom-up technique called the CoT-DT algorithm. The technique combines a fuzzy-neural approach to contextually group related or interesting decision tree nodes, providing contextual definitions for the CoFuH-DT algorithm. Minor Contribution #3 compares a number of local search techniques and shows a novel modification of the traditional techniques. The modification, applied to Simulated Annealing, Genetic Mutation and the Stochastic Hill Climb improvements of 12-99% over the original algorithms when used to solve the 8-Queens problem.

Test examples demonstrate all of these techniques and show some of the speed, flexibility and simplicity advantages of a contextual implementation via comparative analysis with alternative methods. Comparative analysis with traditional Fuzzy Type-1 and Type-2 methods show Type-C can reduce the number of fuzzy sets evaluations for a complex problem as well as reduce the problem over-fitting in Artificial Neural Networks and other classification techniques. Ultimately, the author asserts that the concepts and examples presented in this dissertation show that Fuzzy Contexts or Fuzzy Logic Type-C is a viable and useful technique for solving complex problem spaces.

### **FINAL NOTE**

The author wishes to again recognize the time and efforts of the committee members, and additional UI staff: Cheri Cole, Kirsty Pinchuk and Melinda Deyasi whose advice and patience were instrumental during this process. The author is very grateful for their assistance to insure the totality and quality of this dissertation.

## REFERENCES

[Adomavicius 01]	Adomavicius, G., & Tuzhilin, A. (2001). Using Data Mining Methods to Build Customer Profiles. <i>Computer</i> , 74-82.
[Anderson 10]	Anderson, J., McRee, J., & Wilson, R. (2010). <i>Effective UI: The Art of Building Great User Experience in Software</i> . Sebastopol, California, United States of America: O'Reilly.
[Augarten 84]	Augarten, S. (1984). <i>Bit by Bit: An Illustrated History of Computers</i> . New York, New York, United States of America: Houghton Mifflin.
[Ben-Gan 06]	Ben-Gan, I., Sarka, D., & Wolter, R. (2006). <i>Inside Microsoft SQL Server: T-SQL Programming</i> . Microsoft Press.
[Bolton 12]	Bolton, C., Langford, J., Berry, G., Payne, G., Banerjee, R., & Farley, R. (2012). <i>Professional SQL Server 2012 Internals and Troubleshooting</i> . Hoboken, New Jersey, United States of America: Wrox Press.
[Busc 09]	Buschmann, F. (2009). Introducing the Pragmatic Architect. <i>Software, IEEE</i> , 26(5), 10-11.
[CaGov 14]	California Department of Fish and Wildlife. (2014, January 06). <i>California Grunion Facts and Runs</i> . Retrieved from CA.Gov: <a href="http://www.dfg.ca.gov/marine/grunionschedule.asp">http://www.dfg.ca.gov/marine/grunionschedule.asp</a>
[Camara 14]	Camara, J., de Lemos, R., Laranjeiro, N., Ventura, R., & Vieira, M. (2014). Testing the robustness of controllers for self-adaptive systems. <i>Journal of the Brazilian Computer Society</i> , 20(1), 1-14.
[Carrasco 03]	Carrasco, R. A. (2003). Queen Bee Genetic Optimization of an Heuristic Based Fuzzy Control Scheme for a Mobile Robot. <i>First IEEE Latin American Conference on Robotics and Automation</i> (pp. 61-66). Santiago: IEEE.
[Castillo 08]	Castillo, O., & Melin, P. (2008). <i>Type-2 Fuzzy Logic: Theory and Applications</i> . Berlin, Germany: Springer.
[Chase 06]	Chase, R. B., Jacobs, F. R., & Aquilano, N. J. (2006). <i>Operations Management for Competitive Advantage</i> . Irwin: McGraw-Hill.
[Cheng 10]	Cheng, S., Dong, R., & Pedrycz, W. (2010). A Framework of Fuzzy Hybrid Systems for Modeling and Control. <i>International Journal of General Systems</i> , 165-176.
[CLT 03]	<i>Classification of Living Things</i> . (2003, May 27). Retrieved from Study of Northern Virginia Ecology: <a href="http://www.fcps.edu/islandcreekes/ecology/classification.htm">http://www.fcps.edu/islandcreekes/ecology/classification.htm</a>
[Cormen 09]	Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). <i>Introduction to Algorithms</i> (3 ed.). Cambridge, Massachusetts, United States of America: MIT Press.
[Cox 94]	Cox, E. (1994). <i>The Fuzzy Systems Handbook</i> . Boston, Massachusetts, United States of America: Academic Press.
[Cox 95]	Cox, E. D. (1995). <i>Fuzzy Logic for Business and Industry</i> . Rockland, Massachusetts, United States of America: Charles River Media, Inc.

[Cox 05]	Cox, E. (2005). <i>Fuzzy Modeling and Genetic Algorithms for Data Mining and Exploration</i> . Boston, Massachusetts, United States of America: Morgan Kaufmann.
[Dai 07]	Dai, M., & Huang, Y.-L. (2007). Data Mining Used in Rule Design for Active Database Systems. <i>International Conference on Fuzzy Systems and Knowledge Discovery</i> (pp. 588-592). IEEE.
[Date 11]	Date, C. J. (2011). <i>SQL and Relational Theory</i> (2nd ed.). Sebastopol, California, United States of America: O'Reilly Media.
[Davidson 12]	Davidson, L., & Moss, J. M. (2012). <i>Pro SQL Server 2012 Relational Database Design and Implementation</i> . New York City, New York, United States of America: Apress.
[Dawkins 06]	Dawkins, R. (2006). <i>The Selfish Gene</i> . New York, New York, United States of America: Oxford University Press.
[de Berg 10]	de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2010). <i>Computational Geometry Algorithms and Applications</i> (3 ed.). Berlin, Germany: Springer Verlag.
[Du 10]	Du, X., Ying, H., & Lin, F. (2010). Fuzzy Hybrid Systems Modeling. <i>North American Fuzzy Information Processing Society</i> (pp. 1,6,12-14). IEEE.
[Eiben 07]	Eiben, A. E., & Smith, J. E. (2007). <i>Introduction to Evolutionary Computing</i> . Berlin, Heidelberg, Germany: Springer-Verlag.
[EQP 14]	<i>Eight queens puzzle</i> . (2014, March 30). Retrieved April 28, 2014, from Wikipedia: <a href="http://en.wikipedia.org/wiki/Eight_queens_puzzle">http://en.wikipedia.org/wiki/Eight_queens_puzzle</a>
[Fa-Chao 03]	Fa-Chao, L., Su, J., & Xi-Zhao, W. (2003). Analysis on the Fuzzy Filter in Fuzzy Decision Trees. <i>International Conference on Machine Learning and Cybernetics</i> (pp. 1457-1462). IEEE.
[Gao 07]	Gao, M., & Tian, J. (2007). Path Planning for Mobile Robot Based on Improved Simulated Annealing Artificial Neural Network. <i>Third International Conference on Natural Computation</i> (pp. 8, 12, 24-27). IEEE.
[Garcia 08]	Garcia, M. E., Valero, S., Argente, E., Giret, A., & Julian, V. (2008). A FAST Method to Achieve Flexible Production Programming Systems. <i>IEEE Transactions on Systems, Man, and Cybernetics</i> , 242-252.
[Garibaldi 08]	Garibaldi, J. M., Jaroszewski, M., & Musikasawan, S. (2008, August). Nonstationary Fuzzy Sets. <i>IEEE Transactions on Fuzzy Systems</i> , 1072-1086.
[Gauge 11]	Gauge, C. (2011, March 9). <i>Computer Vision Applications with C# - Fuzzy C-means Clustering</i> . Retrieved from Code Project: <a href="http://www.codeproject.com/Articles/91675/Computer-Vision-Applications-with-C-Fuzzy-C-means">http://www.codeproject.com/Articles/91675/Computer-Vision-Applications-with-C-Fuzzy-C-means</a>
[Gomathi 10]	Gomathi, V., Ramar, K., & Santhiyaku Jeevakumar, A. (2010). A Neuro Fuzzy approach for Facial Expression Recognition using LBP Histograms. <i>International Journal of Computer Theory and Engineering</i> , 245-49.

[Guimaraes 07]	Guimaraes, F. G., Campelo, F., & Igarashi, H. (2007). Optimization of Cost Functions Using Evolutionary Algorithms With Local Learning and Local Search. <i>IEEE Transactions on Magnetics</i> , 1641-1644.
[Haberkorn 13]	Haberkorn, J., & Cheney, K. (2013, May 30). <i>\$474M for 4 failed Obamacare exchanges</i> . Retrieved from Politico: <a href="http://www.politico.com/story/2014/05/obamacare-cost-failed-exchanges-106535.html">http://www.politico.com/story/2014/05/obamacare-cost-failed-exchanges-106535.html</a>
[Hagras 04]	Hagras, H. A. (2004, August). A Hierarchical Type-2 Fuzzy Logic Control Architecture for Autonomous Mobile Robots. <i>IEEE Transactions on Fuzzy Systems</i> , 524-539.
[Hagras 12]	Hagras, H., & Wagner, C. (2012, August). Towards the Widespread Use of Type-2 Fuzzy Logic Systems in Real World Applications. <i>IEEE Computational Intelligence Magazine</i> , 14-24.
[Han 11]	Han, J., & Kamber, M. (2011). <i>Data Mining Concepts and Techniques</i> (3 ed.). Boston, Massachusetts, United States of America: Morgan Kaufmann.
[Hanss 05]	Hanss, M. (2005). <i>Applied Fuzzy Arithmetic</i> . Berlin, Germany: Springer-Verlag.
[Haruechaiyasak 05]	Haruechaiyasak, C., Tipnoe, C., Kongyoung, S., Damrongrat, C., & Angkawattanawit, N. (2005). A Dynamic Framework for Maintaining Customer Profiles in E-Commerce Recommender Systems. <i>International Conference on e-Technology, e-Commerce and e-Service</i> (pp. 768-771). IEEE.
[Haykin 09]	Haykin, S. (2009). <i>Neural Networks and Learning Machines</i> (3rd ed.). Upper Saddle River, New Jersey, United States of America: Pearson Education.
[Hidalgo 10]	Hidalgo, D., Melin, P., Castillo, O., & Licea, G. (2010). Comparative Study of Type-2 Fuzzy Inference System Optimization Based on the Uncertainty of Membership Functions. In P. Melin, J. Kacprzyk, & W. Pedrycz, <i>Soft Computing for Recognition Based on Biometrics</i> (pp. 103-120). Berlin Heidelberg: Springer.
[Hung 11]	Hung, C.-C., Kulkarni, S., & Kuo, B.-C. (2011, June). A New Weighted Fuzzy C-Means Clustering Algorithm for Remotely Sensed Image Classification. <i>IEEE Journal of Selected Topics in Signal Processing</i> , 543-553.
[Ishibuchi 99]	Ishibuchi, H., & Murata, T. (1999). Local Search Procedures in a Multi-Objective Genetic Local Search Algorithm for Scheduling Problems. <i>International Conference on Systems, Man, and Cybernetics</i> (pp. 665-670). IEEE.
[Tang 11]	J.F. Tang, L. M. (2011). Optimization of software components selection for component-based software system development. <i>European Journal of Operational Research</i> , 301-311.
[Johnson 13]	Johnson, K., & Wolf, R. (2013, December 16). <i>Federal judge rules against NSA spying</i> . Retrieved April 28, 2014, from USA Today: <a href="http://www.usatoday.com/story/news/nation/2013/12/16/judge-nsa-">http://www.usatoday.com/story/news/nation/2013/12/16/judge-nsa-</a>

	surveillance-fourth-amendment/4041995/
[Juang 08]	Juang, C.-F., & Lo, C. (2008). Zero-order TSK-type fuzzy system learning using a two-phase swarm intelligence algorithm. <i>Fuzzy Sets and Systems</i> , 159, 2910-2926.
[Karnik 01]	Karnik, N. N., & Mendel, J. M. (2001). Centroid of a Type-2 Fuzzy Set. <i>Information Sciences</i> , 195-220.
[Karnik 14]	Karnik, N. N., Liang, Q., Liu, F., Wu, D., Joo, J., & Mendel, J. M. (n.d.). <i>Type-2 Fuzzy Logic Software</i> . (University of Southern California) Retrieved May 2, 2014, from <a href="http://sipi.usc.edu/~mendel/software/">http://sipi.usc.edu/~mendel/software/</a>
[Kawady 14]	Kawady, T. A., Elkalashy, N. I., Ibrahim, A. E., & Taalab, A.-M. I. (2014). Arcing fault identification using combined Gabor Transform-neural network for transmission lines. <i>International Journal of Electrical Power &amp; Energy Systems</i> , 61, 248-258.
[Ke-jun 07]	Ke-jun, F., & Dian-ming, G. (2007). Using the Data Mining Approach to Determine the Product Preferences of Target Customers. <i>International Conference on Service Systems and Service Management</i> (pp. 1-5). IEEE.
[Kirillov 14]	Kirillov, A. (2014, Jan 23). <i>AForge.Net Framework</i> . Retrieved from AForge.NET: <a href="http://www.aforget.net/news/2013.07.17.releasing_framework_2.2.5.html">http://www.aforget.net/news/2013.07.17.releasing_framework_2.2.5.html</a>
[Kitayama 02]	Kitayama, M., Matsubara, R., & Izui, Y. (2002). Application of Data Mining to Customer Profile Analysis in the Power Electric Industry. <i>Power Engineering Society Winter Meeting</i> (pp. 632-634). IEEE.
[Kliwer 00]	Kliwer, G., & Tschoke, S. (2000). A General Parallel Simulated Annealing Library and Its Application in Airline Industry. <i>International Parallel and Distributed Processing Symposium</i> , (pp. 55-61). 2000.
[Kurbel 98]	Kurbel, K., Schneider, B., & Singh, K. (1998). Solving optimization problems by parallel recombinative simulated annealing on a parallel computer-an application to standard cell placement in VLSI design. <i>IEEE Transactions on Systems, Man, and Cybernetics</i> , 454-61.
[Kwan 02]	Kwan, I. (2002). A Mental Cognitive Model of Web Semantic for E-Customer Profile. <i>International Workshop on Database and Expert Systems Applications</i> (pp. 116-120). IEEE.
[Lee 03]	Lee, J., Sun, J., & Yang, L.-Z. (2003). A Fuzzy Matching Method of Fuzzy Decision Trees. <i>International Conference on Maching Learning and Cybernetics</i> (pp. 1569-1573). IEEE.
[Li 13]	Li, M., Huang, X., Liu, H., Liu, B., & Wu, Y. (2013). Prediction of the gas solubility in polymers by a radial basis function neural network based on chaotic self-adaptive particle swarm optimization and a clustering method. <i>Journal of Applied Polymer Science</i> , 3825-3832.
[Li 06]	Li, S., Li, Y., & Liu, Y. (2006). Effects of Process Planning Upon

	Production Scheduling Under Concurrent Environment. <i>Sixth World Congress on Intelligent Control and Automation</i> (pp. 7282-7286). IEEE.
[Limbu 11]	Limbu, D. K., Yeow-Kee, T., Ridong, J., & Tran, A. (2011). A Software Architecture Framework for Service Robots. <i>Robotics and Biomimetics</i> (pp. 1736-41). IEEE.
[Linda 11a]	Linda, O., & Manic, M. (2011, August). Fuzzy Force-Feedback Augmentation for Manual Control of Multi-Robot System. <i>IEEE Transaction on Industrial Electronics</i> , 3213-3220.
[Linda 11b]	Linda, O., & Manic, M. (2011, November). Uncertainty-Robust Design of Interval Type-2 Fuzzy Logic Controller for Delta Parallel Robot. <i>IEEE Transaction on Industrial Information</i> , 661-671.
[Liu 07]	Liu, D.-R., Lai, C.-H., & Lee, W.-J. (2007). A Hybrid of Sequential Rules and Collaborative Filtering for Product Recommendation. <i>International Conference on Enterprise Computing, E-Commerce, and E-Services</i> (pp. 211-220). IEEE.
[Liu 03]	Liu, H., & Iba, H. (2003). Multi-agent learning by Evolutionary Subsumption. <i>Congress on Evolutionary Computation</i> (pp. 1115-1122). IEEE.
[Lopez 11]	Lopez, C. O., & Beasley, J. E. (2011). A heuristic for the circle packing problem with a variety of containers. <i>European Journal of Operational Research</i> , 512-525.
[MacLennan 08]	MacLennan, J., Tang, Z., & Crivat, B. (2008). <i>Data Mining with Microsoft SQL Server 2008</i> . Hoboken, New Jersey, United States of America: Wiley.
[Mamdani 75]	Mamdani, E. H. (1975). Advances in the linguistic synthesis of fuzzy controllers. <i>International Journal of Man-Machine Studies</i> , 7, 1-13.
[Mantawy 99]	Mantawy, A. H., Abdel-Magid, Y. L., & Selim, S. Z. (1999). Integrating Genetic Algorithms, Tabu Search, and Simulated Annealing for the Unit Commitment Problem. <i>IEEE Transactions on Power Systems</i> , 829-836.
[Marshall 11]	Marshall, A. D. (2011). <i>Heuristic Search</i> . Retrieved from Cardiff University, School of Computer Science and Informatics: <a href="http://www.cs.cf.ac.uk/Dave/AI2/node23.html">http://www.cs.cf.ac.uk/Dave/AI2/node23.html</a>
[Martinez 10]	Martinez-Soto, R., Castillo, O., Aguilar, L. T., & Melin, P. (2010). Fuzzy Logic Controllers Optimization Using Genetic Algorithms and Particle Swarm Optimization. In <i>Advances in Soft Computing</i> (pp. 475-486). Pachuca, Mexico: Springer.
[Martinjak 07]	Martinjak, I., & Golub, M. (2007). Comparison of Heuristic Algorithms for the N-Queens Problem. (pp. 759-764). IEEE.
[Matteucci 12]	Matteucci, M. (2012, October 24). <i>A Tutorial on Clustering Algorithms Fuzzy C-Means Clustering</i> . Retrieved April 21, 2014, from Politecnico Di Milano: <a href="http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html">http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html</a>

[McCarthy 95]	McCarthy, J. (1995). <i>Dynamics of Software Development</i> . Redmond: Microsoft Press.
[McCarty 08a]	McCarty, K., & Manic, M. (2008). Contextual Fuzzy Type-2 Hierarchies for Decision Trees (CoFuH-DT) - An Accelerated Data Mining Technique. <i>Conference on Human System Interactions</i> , (pp. 699-704). Krakow, Poland.
[McCarty 08b]	McCarty, K., & Manic, M. (2008). Descending Deviation Optimization Techniques for Scheduling Problems. <i>IEEE International Conference on Emerging Technologies and Factory Automation</i> , (pp. 257-260).
[McCarty 08c]	McCarty, K., & Manic, M. (2008). Line-of-sight tracking based upon modern heuristics approach. <i>3rd IEEE Conference on Industrial Electronics and Applications</i> , (pp. 40-45).
[McCarty 09a]	McCarty, K., & Manic, M. (2009). Adaptive Behavioral Control of Collaborative Robots in Hazardous Environments. <i>2nd Conference on Human Systems Interactions</i> , (pp. 10-15).
[McCarty 12]	McCarty, K., & Manic, M. (2012). A Proposed Data Fusion Architecture for Micro-Zone Analysis and Data Mining. <i>5th International Symposium on Resilient Control Systems</i> , (pp. 72-76).
[McCarty 14a]	McCarty, K., & Manic, M. (2014). A Database Driven Memetic Algorithm for Fuzzy Set Optimization. <i>7th International Conference on Human System Interaction</i> . Lisbon, Portugal: IEEE.
[McCarty 14b]	McCarty, K., & Manic, M. (2014). Fuzzy Contexts (Type C) and Fuzzymorphism to Solve Situational Discontinuity Problems. <i>International World Congress on Computational Intelligence</i> . Beijing, China: IEEE.
[McCarty 13]	McCarty, K., Manic, M., & Gagnon, A. (2013). A Fuzzy Framework with Modeling Language for Type 1 and Type 2 Application Development. <i>The 6th International Conference on Human Systems Interaction</i> (pp. 334-341). Gdansk, Poland: IEEE.
[McCarty 09b]	McCarty, K., Manic, M., & Stan, D. (2009). Contextual Data Rule Generation for Autonomous Vehical Control. In T. Sobh, & T. Sobh (Ed.), <i>Innovations and Advances in Computer Sciences and Engineering</i> (Vol. 1, pp. 123-128). Bridgeport, Connecticut, USA: Springer-Verlag.
[McCarty 10]	McCarty, K., Manic, M., Cherry, S., & McQueen, M. (2010). A Temporal-Spatial Data Fusion Architecture for Monitoring Complex Systems. <i>3rd Conference on Human Systems Interactions</i> , (pp. 101-106).
[McCarty 09c]	McCarty, K., Manic, M., Goodwin, P., & Piasecki, M. (2009). Submission and Querying Tools for a Hydrologic Information Systems Database. <i>8th International Conference on Hydroinformatics</i> . Concepcion, Chile.
[McConnell 96]	McConnell, S. (1996). <i>Rapid Development</i> . Redmond: Microsoft Press.



[McConnell 04]	McConnell, S. (2004). <i>Code Complete</i> . Redmond: Microsoft Press.
[Mendel 01]	Mendel, J. M. (2001). <i>Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions</i> . Upper Saddle River, NJ, United States of America: Prentice-Hall.
[Mendel 02]	Mendel, J. M., & John, R. I. (2002, April). Type-2 Fuzzy Sets Made Simple. <i>IEEE Transactions on Fuzzy Systems</i> , 117-127.
[Mendel 06]	Mendel, J. M., & John, R. I. (2006, December). Interval Type-2 Fuzzy Logic Systems Made Simple. <i>IEEE Transactions on Fuzzy Systems</i> , 808-821.
[Mendel 10]	Mendel, J. M., & Wu, D. (2010). <i>Perceptual Computing: Aiding People in Making Subjective Judgements</i> . Hoboken, New Jersey, United States of American: John Wiley & Sons, Inc.
[Mendis 08]	Mendis, B. (2008, March 1). Fuzzy Signatures: Hierarchical Fuzzy Systems and Applications. <i>Fuzzy Signatures: Hierarchical Fuzzy Systems and Applications</i> . Acton, Acton, Australia: Australian National University.
[Mendis 10]	Mendis, B. S., & Gedeon, T. D. (2010). Polymorphic Fuzzy Signatures. <i>IEEE International Conference on Fuzzy Systems</i> (pp. 18-23). Barcelona, Spain: IEEE.
[Mendonca 97]	Mendonca, P., & Caloba, L. (1997). New Simulated Annealing Algorithms. <i>International Symposium on Circuits and Systems</i> . Hong Kong: IEEE.
[Ming-Chuan 01]	Ming-Chuan, H., & Don-Lin, Y. (2001). An Efficient Fuzzy C-Means Clustering Algorithm. <i>IEEE International Conference on Data Mining</i> (pp. 225-232). San Jose, CA USA: IEEE.
[Mizumoto 76]	Mizumoto, M., & Tanaka, K. (1976). Some Properties of Fuzzy Sets of Type 2. <i>Science Direct, Information and Control</i> , 312-340.
[Moler 04]	Moler, C. (2004). <i>The Origins of Matlab</i> . Retrieved from MathWorks: <a href="http://www.mathworks.com/company/newsletters/articles/the-origins-of-matlab.html">http://www.mathworks.com/company/newsletters/articles/the-origins-of-matlab.html</a>
[Moreno 12]	Moreno-Velo, F. J., Barriga, A., Sanchez-Solano, S., & Baturone, I. (2012). XFSML: An XML-based Modeling Language for Fuzzy Systems. <i>International Conference on Fuzzy Systems</i> (pp. 1-8). IEEE.
[Niu 02]	Niu, L., Yan, X.-W., Zhang, C.-Q., & Zhang, S.-C. (2002). Product Hierarchy-Based Customer Profiles for Electronic Commerce Recommendation. <i>International Conference on Machine Learning and Cybernetics</i> (pp. 1075-1080). IEEE.
[Octave 13]	<i>Octave Fuzzy Logic Toolkit</i> . (2013, Jan 12). Retrieved from SourceForge.NET: <a href="http://sourceforge.net/projects/octave-fuzzy/?source=directory">http://sourceforge.net/projects/octave-fuzzy/?source=directory</a>
[Omizegba 09]	Omizegba, E. E., & Adebayo, G. E. (2009). Optimizing Fuzzy Membership Functions Using Particle Swarm Algorithm. <i>International Conference on Systems, Man and Cybernetics</i> (pp. 3866-3870). IEEE.
[Oracle 10]	Oracle. (2010). <i>Codd's Rules</i> . Retrieved from Oracle:

	<a href="http://www.oracle-dba-online.com/sql/Codd_rules.htm">http://www.oracle-dba-online.com/sql/Codd_rules.htm</a>
[Palm 98]	Palm, R., & Driankov, D. (1998). Fuzzy Switched Hybrid Systems-Modeling and Identification. <i>IEEE International Symposium on Computational Intelligence in Robotics and Automation</i> (pp. 130-135). Gaithersburg, MD: IEEE.
[Palias 04]	Palias, V., Karras, D. A., & Papademetriou, R. C. (2004). Traffic Engineering in Multi-Service Networks Comparing Genetic and Simulated Annealing Optimization Techniques. <i>International Joint Conference on Neural Networks</i> (pp. 2325-2330). IEEE.
[Powell 05]	Powell, G. (2005). <i>Beginning Database Design</i> . Hoboken, New Jersey, United States of America: Wrox.
[Pressman 09]	Pressman, R. S. (2009). <i>Software Engineering A Practitioner's Approach</i> (7 ed.). New York, New York, United States of America: McGraw-Hill.
[Purdum 12]	Purdum, J. (2012). <i>Beginning Object-oriented Programming with C#</i> . Hoboken, New Jersey, United States of America: John Wiley & Sons.
[Qiming 99]	Qiming, C., Dayal, U., & Hsu, M. (1999). A Distributed OLAP Infrastructure for E-Commerce. <i>International Conference on Cooperative Information Systems</i> (pp. 209-220). IEEE.
[Random 13]	Random House. (2013, January 06). <i>Context</i> . (Random House) Retrieved January 06, 2014, from Dictionary.com: <a href="http://dictionary.reference.com/browse/context">http://dictionary.reference.com/browse/context</a>
[Rodrigues 08]	Rodrigues Neto, A. C., Lima De Campos, G. A., De Souza, J. T., Riosenberg, M., & Marques, V. X. (2008). Autonomous Agents and Subsumption as Models for Simulations of Population Dynamics. <i>Seventh International Conference on Machine Learning and Cybernetics</i> (pp. 2440-45). Kunming: IEEE.
[Roychowdhury 98]	Roychowdhury, S. (1998). Fuzzy Curve Fitting Using Least Square Principles. <i>International Conference on Systems, Man, and Cybernetics</i> (pp. 4022-4027). IEEE.
[Russell 09]	Russell, S., & Norvig, P. (2009). <i>Artificial Intelligence A Modern Approach</i> (3 ed.). Upper Saddle River, New Jersey, United States of America: Prentice Hall.
[Schalkoff 97]	Schalkoff, R. J. (1997). <i>Artificial Neural Networks</i> . (E. M. Munson, Ed.) New York, New York, United States of America: McGraw-Hill.
[Sipser 12]	Sipser, M. (2012). <i>Introduction to the Theory of Computation</i> (3 ed.). Boston, Massachusetts, United States of America: Thompson Course Technology.
[Skiena 10]	Skiena, S. S. (2010). <i>The Algorithm Design Manual</i> . London: Springer.
[Slavicek 13]	Slavicek, V. (2013, January 12). <i>Fuzzy Framework</i> . Retrieved from Code Project: <a href="http://www.codeproject.com/Articles/151161/Fuzzy-Framework">http://www.codeproject.com/Articles/151161/Fuzzy-Framework</a>
[Sun 05]	Sun, J., & Wang, X.-Z. (2005). An Initial Comparison on Noise Resisting Between Crisp and Fuzzy Decision Trees. <i>International</i>

	<i>Conference on Machine Learning and Cybernetics</i> (pp. 2545-2550). IEEE.
[Surprise 13]	<i>Suprise 96, Fuzzy Logic and Its Uses</i> . (2013, December 29). Retrieved from Imperial College London: <a href="http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol2/jp6/article2.html">http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol2/jp6/article2.html</a>
[Taheri 06]	Taheri, J., & Zomaya, A. Y. (2006). Fuzzy Logic. In A. Y. Zomaya, <i>Handbook of Nature-Inspired and Innovative Computing</i> (pp. 221-252). New York: Springer.
[Taxonomy 14]	<i>Taxonomy</i> . (2014, March 27). Retrieved from Wikipedia: <a href="http://en.wikipedia.org/wiki/Taxonomy">http://en.wikipedia.org/wiki/Taxonomy</a>
[Tsai 11]	Tsai, D.-M., & Lin, C.-C. (2011, August). Fuzzy C-means Based Clustering for Linearly and Nonlinearly Separable Data. <i>Elsevier Pattern Recognition Journal</i> , 1750-1760.
[van de Vlag 07]	van de Vlag, D., & Stein, A. (2007). Incorporating Uncertainty via Hierarchical Classification Using Fuzzy Decision Trees. <i>IEEE Transactions on Geosciences and Remote Sensing</i> , 237-245.
[Vidal 13]	Vidal, T., Crainic, T., Gendreau, M., & Prins, C. (2013). A unified solution framework for multi-attribute vehicle routing problems. <i>European Journal of Operational Research</i> , 231(1), 658-673.
[Wang 01]	Wang, X., Yeung, D., & Tsang, E. (2001). A Comparative Study on Heuristic Algorithms for Generating Fuzzy Decision Trees. <i>IEEE Transactions on Systems, Man, and Cybernetics</i> , 215-226.
[Wang 06]	Wang, Z. P., Ge, S. S., Lee, T. H., & Lai, X. C. (2006). Adaptive Smart Neural Network Tracking Control of Wheeled Mobile Robots. <i>Control, Automation, Robotics and Vision</i> (pp. 1-6). IEEE.
[Wiki 14]	Wikipedia. (2014, March 31). <i>History of Computing Hardware</i> . Retrieved from Wikipedia: <a href="http://en.wikipedia.org/wiki/History_of_computing_hardware">http://en.wikipedia.org/wiki/History_of_computing_hardware</a>
[Witten 11]	Witten, I. H., & Frank, E. (2011). <i>Data Mining Practical Machine Learning Tools and Techniques</i> (3 ed.). Boston, Massachusetts, United States of America: Morgan Kaufmann.
[Wyne 12]	Wyne, M. F. (2012). SOOD: A Simulation Tool for OODB. <i>Global Engineering Education Conference</i> (pp. 1-9). IEEE.
[XE 12]	<i>XE Currency Converter</i> . (2012, 03 26). Retrieved from XE: <a href="http://www.xe.com/currencyconverter/convert/?Amount=1&amp;From=ZWD&amp;To=USD">http://www.xe.com/currencyconverter/convert/?Amount=1&amp;From=ZWD&amp;To=USD</a>
[Yongjie 06]	Yongjie, Y., Qidan, Z., & Chengtao, C. (2006). Hybrid Control Architecture of Mobile Robot Based on Subsumption Architecture. <i>International Conference on Mechatronics and Automation</i> , (pp. 2168-2172). Luoyang.
[Yu 06]	Yu, L., Wang, S., & Lai, K. (2006). An Integrated Data Preparation Scheme for Neural Network Data Analysis. <i>IEEE Transactions on Knowledge and Data Engineering</i> , 217-230.
[Zadeh 65]	Zadeh, L. A. (1965). Fuzzy Sets. <i>Information Control</i> , 338-353.

[Zadeh 08]	Zadeh, L. A. (2008). Is there a need for Fuzzy Logic? <i>International Journal of Information Sciences</i> , 178(13), 2751-2779.
[Zamani 08]	M. Zamani, H. Nejati, A. T. Johromi, A. R. Partovi, S. H. Nobari and G. N. Shirazi, "Toolbox for Interval Type-2 Fuzzy Logic Systems," in <i>11th Joint International Conference on Information Sciences</i> , Shenzhen, China, 2008
[Zal 13]	Zal, F., Chen, T.-S., Chi, S.-W., & Kuo, C.-H. (2013). Fuzzy Controller Based Subsumption Behavior Architecture for Autonomous Robotic Wheelchair. <i>International Conference on Advanced Robotics and Intelligent Systems</i> (pp. 158-163). Tainan: IEEE.
[Zetta 14]	Zetta.net. (2014, April 12). <i>The History of Computer Storage</i> . Retrieved from Zetta.net Corporate Web Site: <a href="http://www.zetta.net/history-of-computer-storage/">http://www.zetta.net/history-of-computer-storage/</a>
[Zhao 06]	Zhao, J., & Chang, Z. (2006). Neuro-Fuzzy Decision Tree by Fuzzy ID3 Algorithm and Its Application to Anti-Dumping Early-Warning System. <i>International Conference on Information Acquisition</i> (pp. 1300-1304). IEEE.
[Zheng 06]	Zheng, S., Shu, W., & Gao, L. (2006). Task Scheduling using Parallel Genetic Simulated Annealing Algorithm. <i>Service Operations and Logistics, and Informatics</i> (pp. 46-50). IEEE.
[Zurada 92]	Zurada, J. M. (1992). <i>Artificial Neural Systems</i> . St. Paul, Minnesota, United States of America: West Publishing Company.

## APPENDICES

### APPENDIX A: MINOR CONTRIBUTIONS

#### Minor Contribution #1: Contextual Fuzzy Hierarchies for Decision Trees (CoFuH-DT) – An Accelerated Data Mining Technique

##### *M1.1: Introduction*

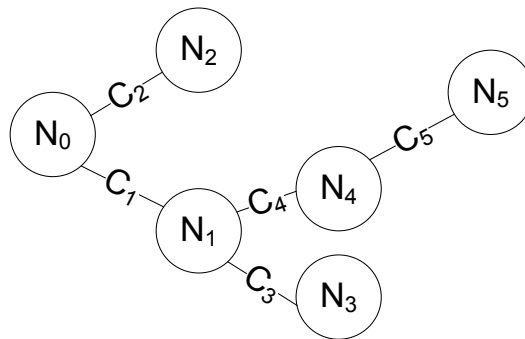
This section presents a technique for modifying a Decision Tree using Fuzzy Type 1 & Type-C operations. The resulting contextual tree is substantially smaller and semantically more meaningful.

Consider a robot vehicle attempting to cross difficult terrain. It has goals and instructions 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], [Kitayama 02], [Ke-jun 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], [Fa-Chao 03], [Niu 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 79 - A basic, horizontal Decision Tree**

Hence, in Figure. 79, 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  $C_{N_j}$  to its parent:

$$N_j = C_k(N_{jParent}) \quad j=1,..,j_{max}, k=1,..,k_{max} \quad (M1.1)$$

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 [Adomavicius 01], [Kwan 02], [Dai 07].

***DEFINE RULE rule\_name***

***ON event***

***IF condition***

***DO action***

**(M1.2)**

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%***

**(M1.3)**

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 computers 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-C 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 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.

*M1.2: 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:

$$\textit{Total number of nodes in Decision Tree} = \prod_{i=1}^n d_i \quad (\text{M1.4})$$

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*** **(M1.5)**

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 [Kitayama 02], [Haruechaiyasak 05], Online Analytical Mining (OLAM) [Qiming 99], [Adomavicius 01], [Kwan 02] and multi-level association rules [Niu 02], [van de 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 (M1.5) 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 (M1.6)
```

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,

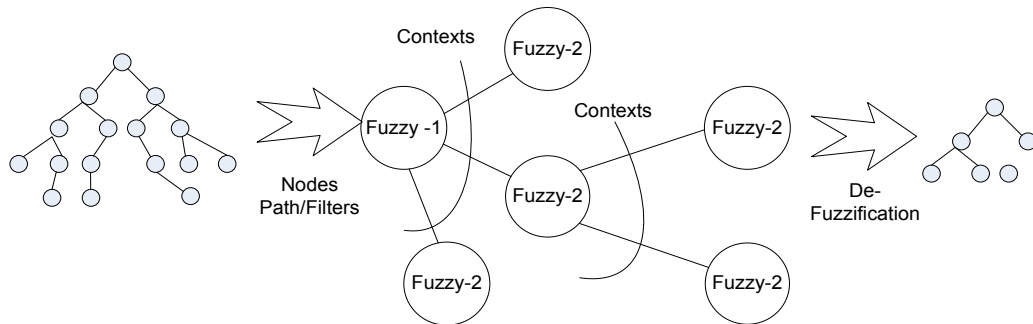
what sorts of other products they might be interested in and so on. His initial Decision Tree might consist of the following, Customer Type (CT), Product Type (PT), Relative Product Price (RPP), Day of Week (DW), Time of Year (TY), Customer Age (CA), Geographic Location (GL) as shown in Table 15.

**Table 15 - Dimensions for a virtual store manager**

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

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.

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-C 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 80.



**Figure 80 - CoFuH-DT reduction of Decision Tree**

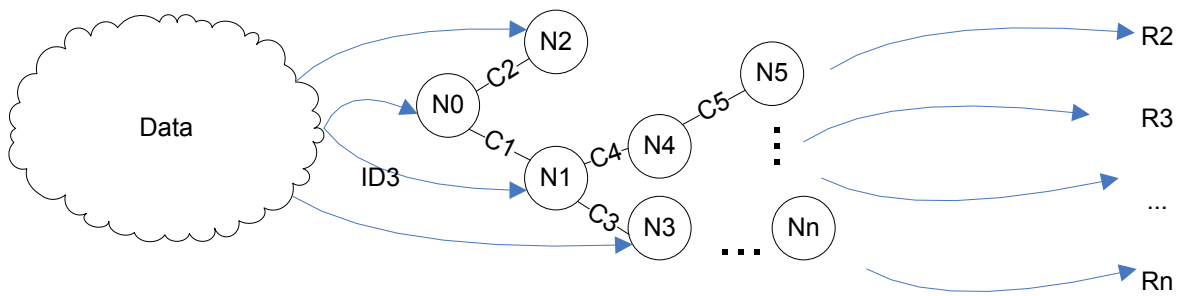
The steps of the CoFuH-DT algorithm is as follows:

### Step 1. Condition creation

Let  $N_1..N_n$  be the set of nodes generated through data mining techniques such as ID3 [Haruechaiyasak 05], creating a Decision Tree for the original data set  $D$ .

$$N = \{N_1, N_2, \dots, N_n\} \quad (\text{M1.7})$$

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 81.

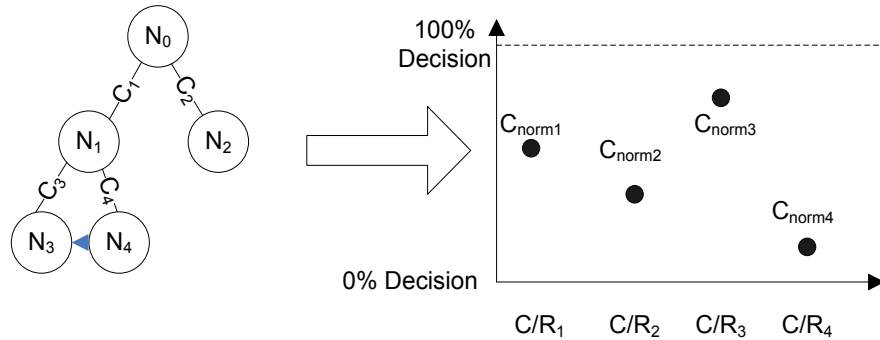


**Figure 81 - Rule Creation using Decision Tree**

**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  $C_{norm}$ :

$$C_{norm} = \{f(C_i), C_i \in C_R, f(C_i) \in [0, 1]\} \tag{M1.8}$$

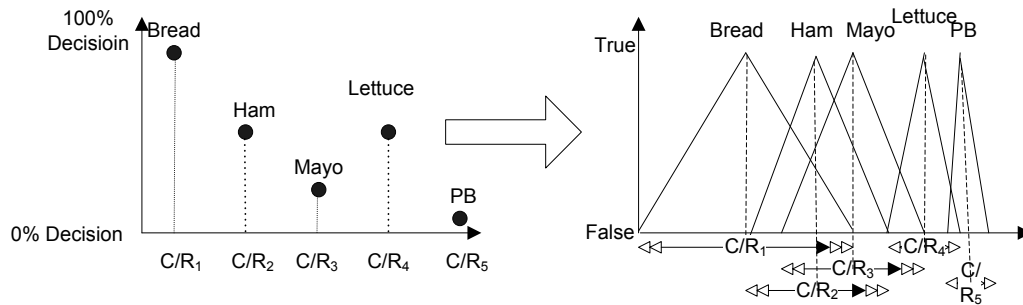


**Figure 82 - Normalization of a Decision Tree**

**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  $\mu_{C_{norm}}$ .

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 83 with the original crisp conditions represented as a series of ranges at the base of each triangle.



**Figure 83 - Fuzzifying customer's Decision Tree**

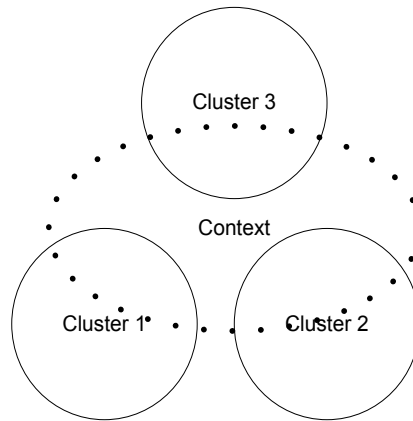
In cases where there are multiple Boolean conditions for a node we can apply Zadeh's operators AND and OR for fuzzy unions and intersections for conditions  $C_1 \dots C_n$

$$\begin{aligned} \cap \mu_{C_i} &= \max(\mu_{C_1}, \mu_{C_2}, \dots, \mu_{C_n}) \\ \cup \mu_{C_i} &= \min(\mu_{C_1}, \mu_{C_2}, \dots, \mu_{C_n}) \end{aligned} \quad (\text{M1.9})$$

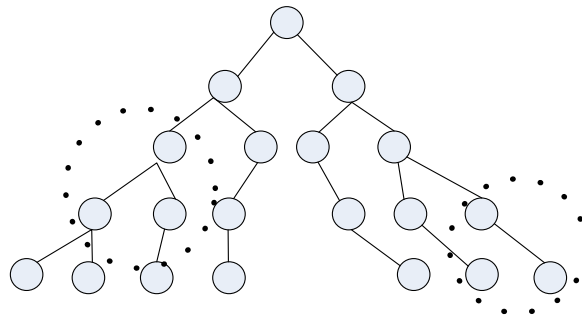
Further, more extreme examples can make use of mean and weighted mean or other general algebraic operators [Cox 05].

#### Step 4. Context creation

Create fuzzy sets using a method such as that demonstrated in Chapters 3 and 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 84. For the Decision Tree, this has the effect of "pruning" all those nodes which fall out of context as demonstrated in Figure 85.

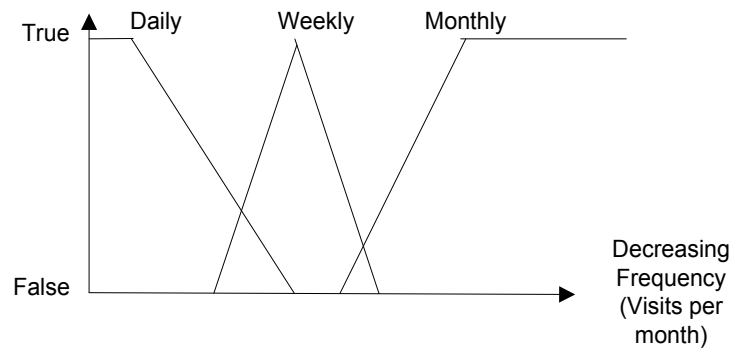


**Figure 84 - Context unifying 3 clusters**



**Figure 85 - Nodes pruned by context**

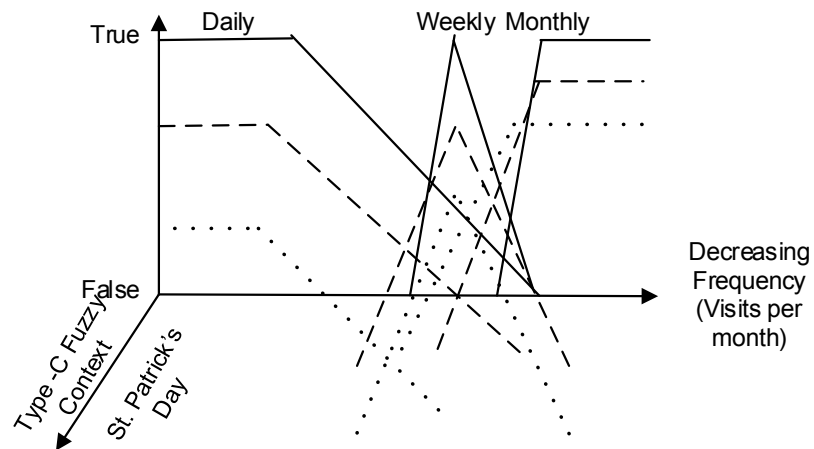
Using fuzzy, new dimensions of uncertainty are added, allowing new specifications to exist and altering existing ones. In the example of the woman doing her monthly shopping, the context and new dimension of uncertainty “monthly shopping” alters the notion of both “food” and “cleaning supplies” by increasing 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 Context, “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 86 - Context of shopping type.**

Here the Fuzzy 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 87 - Fuzzy deformation under a context.**

### **Step 5. Fuzzy Type-C application of contexts to fuzzified conditions**

Fuzzy Type-C contexts extend the newly created Fuzzy Type 1 set by adding an additional dimension similar to a Nonstationary Fuzzy System (NFS). The context creates an NFS set  $\hat{C}$  [Garibaldi 08], whose members are the combination of the context functions over



the original Fuzzy Type 1 membership functions over the original conditions shown in Equation (M1.9). 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  $R_C$ :

$$R_C = \cap \hat{C} \quad (\text{M1.10})$$

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 M1.6 can now be generalized to:

***DEFINE RULE ShoppingType***  
***ON Customer PURCHASE***  
***IF PURCHASE IS MonthlyContextItem THEN***  
***DO Recommend Other MonthlyContextItems*** (M1.11)

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”.

*M1.4: 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 effect 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 M1.10.

**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 15. 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:

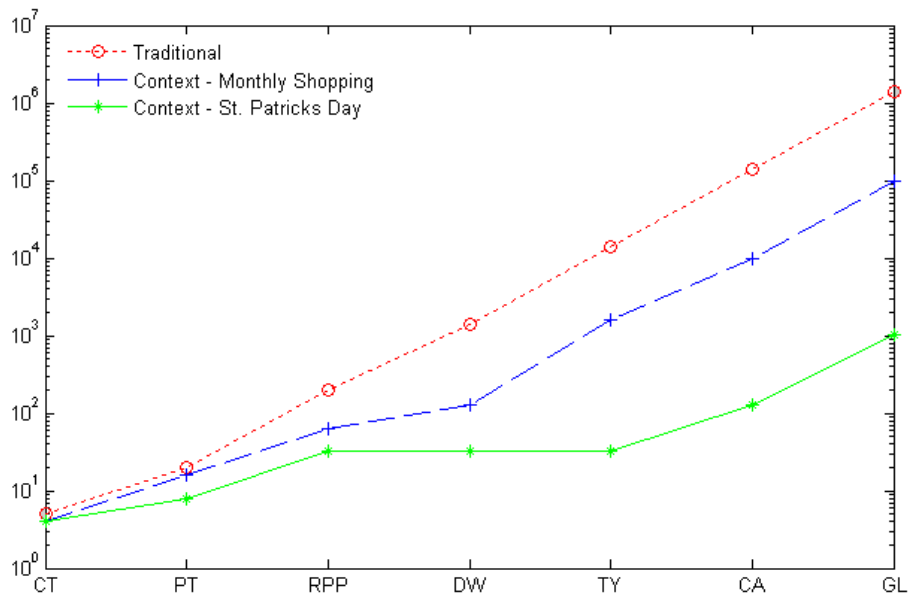
1. The “impulsive” customer type (CT) falls out of context as MS is considered planned, thus reducing the size of CT from 5 to 4.
2. Many of the product types (PT) that are considered impulse buys (e.g books, candy) or quickly perishable items (e.g. bread, lettuce) or irregular purchases (e.g nails), 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 16 and figure 88.

**Table 16 - Example 2: Node Reduction Under Contexts**

	Traditional DT	Ctx - Mnthly Shopping	Ctx - St. Patrick’s Day
Dimensions In Context	7	7	7
Elements In Context	51	42	24
Potential Nodes	$1.4 \times 10^6$	$9.6 \times 10^4$	1024



**Figure 88 - 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.

**Table 17 - Node Reduction Under Contexts**

Traditional DT    Ctx Pref. Customer    Ctx Holiday Sched.

Dimensions In Context	7	7	7
Elements In Context	70	27	33
Potential Nodes	$1 \times 10^7$	4400	$3.6 \times 10^4$

#### *M1.5: 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  $1.4 \times 10^6$  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.

As described in [Han 11], decision trees are constructed using a technique whereby “information gain” and “entropy” determines how nodes are split and what attributes are passed onto descendent nodes (C4.5 being extensions of original ID3). The total possible number of nodes in a decision tree with  $n$  dimensions is calculated by taking the cross product of the number of elements  $e$  of each dimension  $d_i$  used to branch.

$$\text{Potential nodes in Decision Tree} = \prod_{i=1}^n d_i \quad (\text{M1.12})$$

Pruning a decision tree requires recursively drilling down and evaluating each node for “inappropriateness” until one is found, at which point it and its children are eliminated. It is an iterative process with a worst case of having to evaluate each node. CoFuH-DT uses fuzzy evaluation to generate smaller, more directed decision trees under fuzzy contexts where  $c_i$  is the number of dimensions in context.

$$\text{Potential nodes in Contextual Decision Tree} = \prod_{i=1}^n c_i \quad (\text{M1.13})$$

As a result a “contextual” tree,  $DT_c$  will be at worst no larger than the original tree,  $DT$  and smaller if any dimensions are “out of context”.

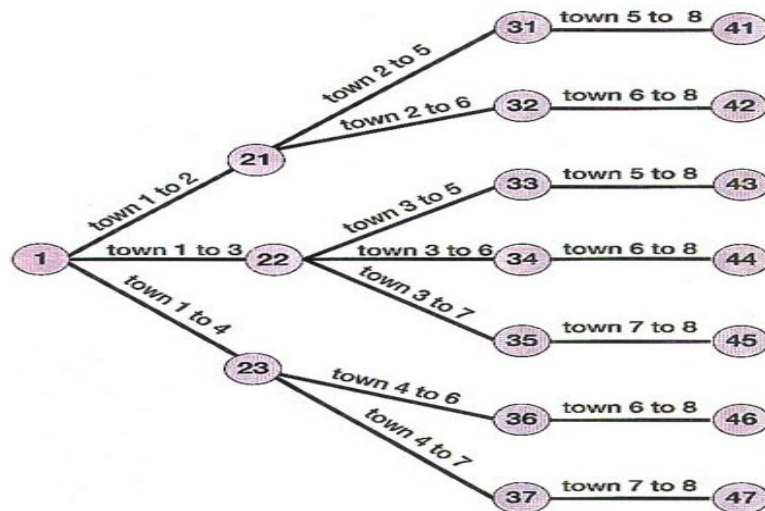
$$DT_c \leq DT \quad (\text{M1.14})$$

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.

## Minor Contribution #2: Contextual Derivation From Decision Trees (CoT-DT) Based on Advanced Data Mining Techniques and Intelligent Control

### M2.1: Introduction

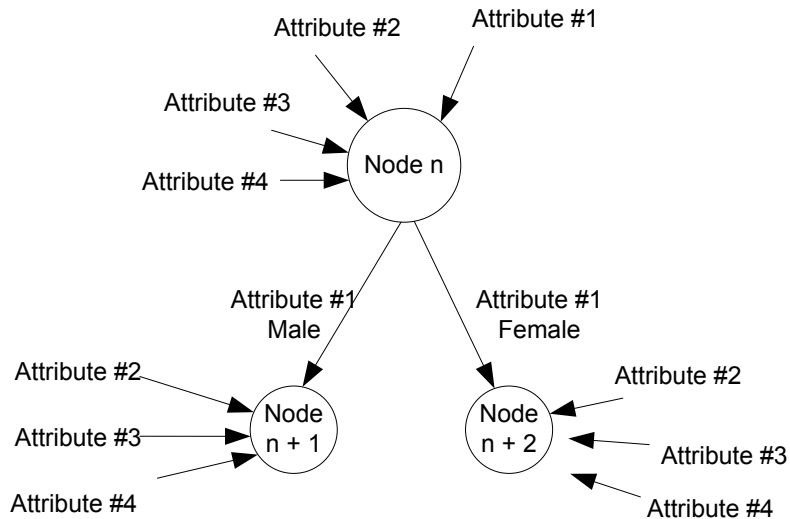
Effective data mining requires the ability to quickly sift through mountains of data and extract meaningful kernels of knowledge [Han 11]. This new knowledge manifests in new rules for intelligent systems from e-commerce to intelligent controllers. There are a number of Advanced Data Mining Techniques 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 [Witten 11], [Adomavicius 01]. One of the more popular techniques is the Decision Tree.



**Figure 89 - A typical Decision Tree used for data mining**

Decision Trees are built using techniques such as ID3 and C4.5 [Han 11], [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 90. ID3 uses a simpler notion of “information

content” while C4.5 attempts to overcome the bias of uneven sampling by normalizing across attributes.



**Figure 90 - 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 90 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 90 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 11] 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

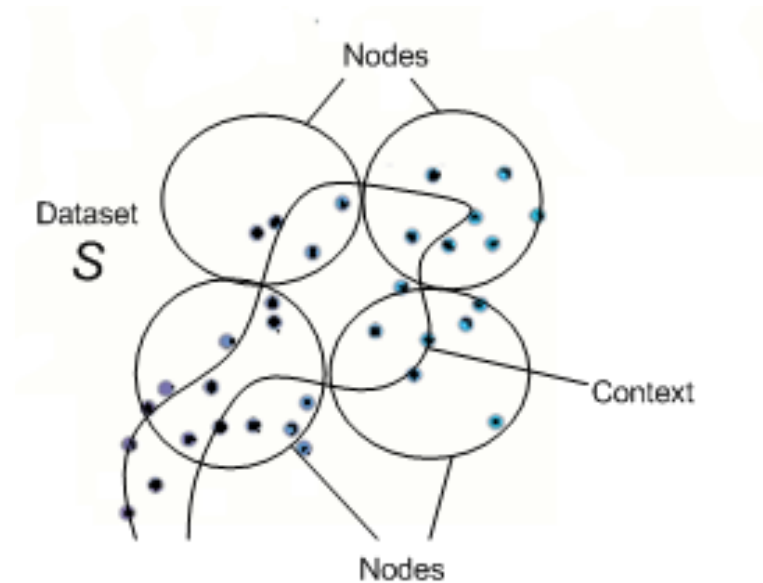
Large tree searches for rule generation, using methods such as depth-first or breadth-first, are very expensive and time-consuming [Russell 09],[Cormen 09].

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-C Hierarchies for Decision Trees (CoFuH-DT) method. The CoFuH-DT is fuzzification of the Decision Tree followed by application a Fuzzy Type-C 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 11]. As such ADMTs are an effective tool for generating a range of candidates for a Fuzzy Type-C context.



**Figure 91 - Context spanning several nodes**

This contribution 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.

*M2.2: 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 18.

**Table 18 - Attributes of a typical customer**

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

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.

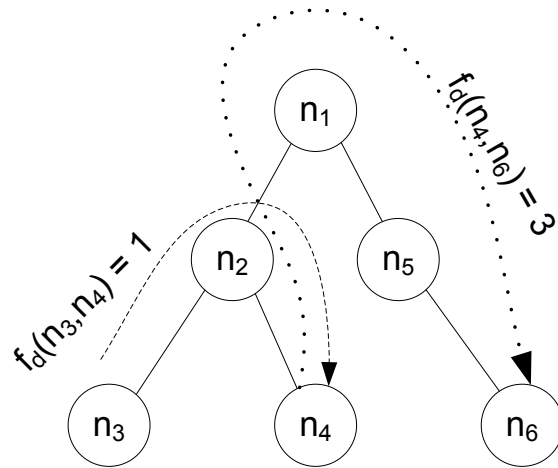
CoFuH-DT presents an alternative by combining the efficiency of the Decision Tree with the power of Fuzzy Type-C contexts. Generating the contexts can be a difficult task, but is made easier through the use of ADMTs such as an Artificial Neural Network (ANN). This is accomplished by applying the ANN 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.

### *M2.3: 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  $S$ .

$$\forall n_i \in N, f(n_i) = \{s_i \subset S\}, \cup s_i = S, i = 1, \dots, N \quad (\text{M2.1})$$

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 92.



**Figure 92 - Calculating distance between nodes**

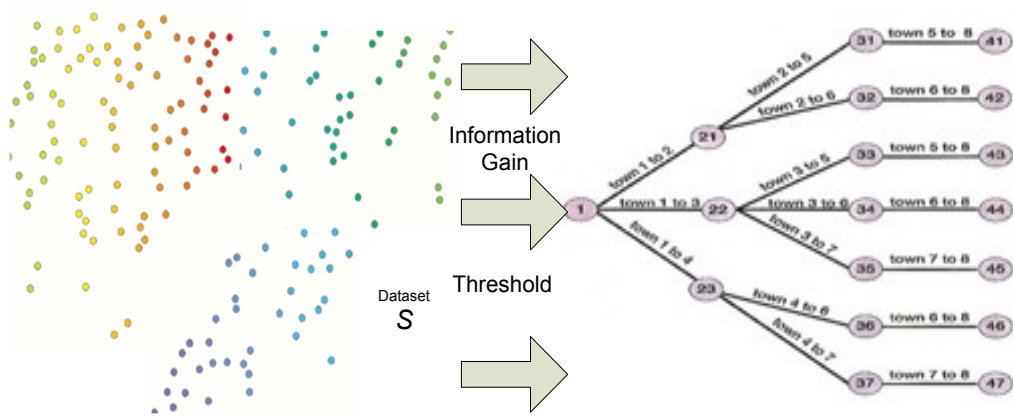
Unlike traditional classification using ANNs or other ADMT, which seeks to create clusters of data based upon some measure of “closeness”, context generation seeks to discover relationships that exist between sets of data within a given set of nodes. This is accomplished by examining the intersection 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 (M2.1) to determine the information threshold and generate the Decision Tree from the dataset  $S$  shown in Figure 93.

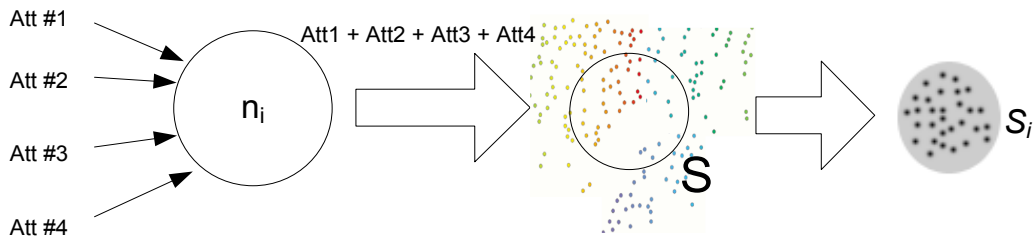


**Figure 93 - CoT-DT Step 1 - Creation of Decision Tree**

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 94.



**Figure 94 - Nodes of Decision Tree produce subset  $s_i$  of original set  $S$ .**

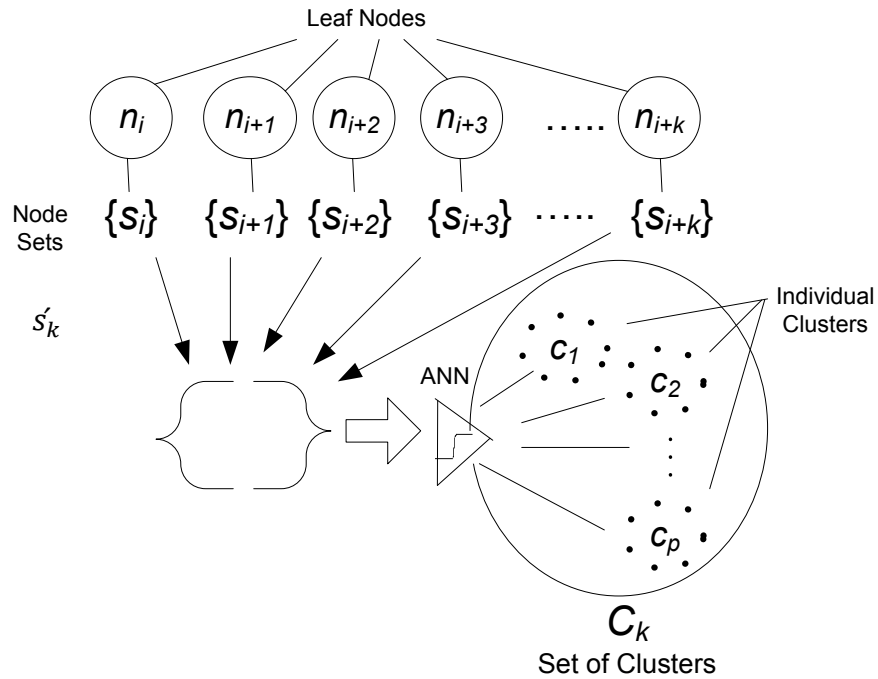
Figure 94 shows how the collection of attributes  $A$  of the leaf combine to create a filter that when applied to  $S$ , produces the data set  $s_i$  of the leaf.

$$\forall n_i \in N, A_{n_i}(S) = s_i, i = 1, \dots, N \tag{M2.2}$$

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 95.



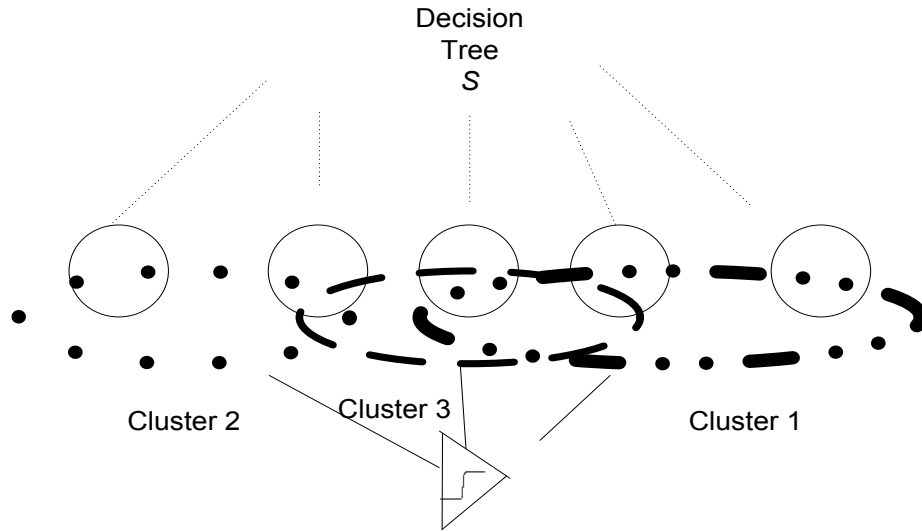
**Figure 95 - ANN classifier applied to leaf node sets produces clusters.**

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  $S$ .

$$s'_k \subset S, g(s'_k) = \{c_p \mid \cup c_p = C_k \ p = 1, \dots, k\} \quad (\text{M2.3})$$

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



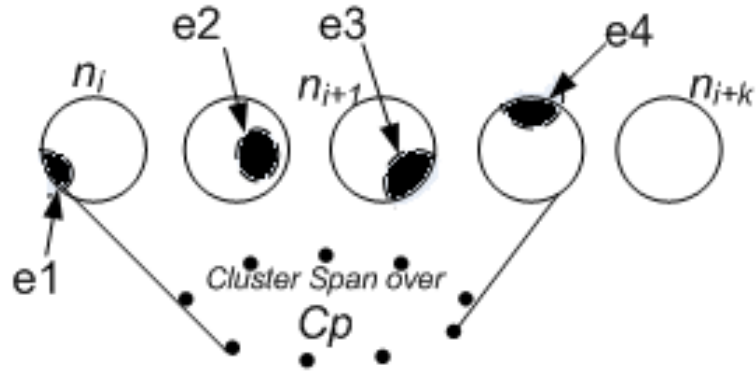
**Figure 96 - ANN cluster generation**

#### 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  $e_j$ .

$$e_j = s_i \cap c_p, \quad e_j \neq \emptyset \quad (\text{M2.4})$$

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 97.



**Figure 97 - 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  $s_i$ .

$$f_{cvg}(e_j) = \frac{f_{dp}(e_j)}{f_{dp}(s_i)} \quad (\text{M2.5})$$

Let  $f_d(e_i, e_j)$  be the distance between the corresponding nodes for  $e_i$  and  $e_j$  as illustrated in Figure 97. 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.

$$f_{dm}(e_i) = \max(f_d(e_i, e_j), \forall e_j \in C_p, i = 1, \dots, n, j = 1, \dots, n, p = 1, \dots, k) \quad (\text{M2.6})$$

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

$$f_{int}(e_j) = f_{cvg}(e_j) * f_{dm}(e_j) \quad (\text{M2.7})$$

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.



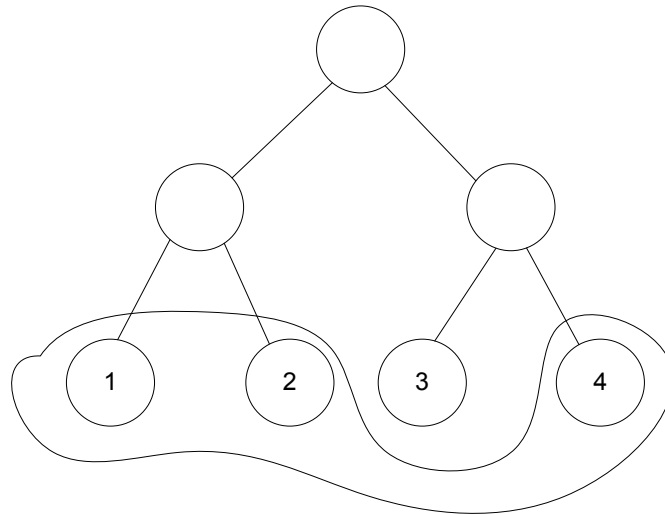
$$CT_i = \cup e_j \quad (M2.8)$$

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 an entire context,  $F_{int}$  is the weighted sum of the interestingness of its corresponding elements.

$$F_{int}(CT_i) = \sum_j w_j f_{int}(e_j), e_j \subset c_p \in C_k, j = 1, \dots, p, i=1, \dots, k \quad (M2.9)$$

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 98. Each leaf node contains exactly 100 elements.



**Figure 98 - 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 M2.9, its corresponding context “interestingness” is calculated as follows:

$$\begin{aligned}
 f_{int}(e_1) &= 3 \times .5 = 1.5, \\
 f_{int}(e_2) &= 3 \times .5 = 1.5, \\
 f_{int}(e_3) &= 3 \times .25 = .75 \\
 F_{int}(CT_i) &= 1.5 + 1.5 + .75 = 3.75 \qquad \qquad \qquad (M2.10)
 \end{aligned}$$

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

#### *M2.4: 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  $2 \times 10^{10}$  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 [Wang 06], [McCarty 08a].

***DEFINE RULE rule\_name***  
***ON event***  
***IF condition1***  
***AND condition2***  
 ...  
***AND condition***  
***DO action*** **(M2.11)**

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*** **(M2.12)**

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 [McCarty 08c]. Among them is the use of rules derived from Decision Trees.

Consider a robotic land rover as it attempts to navigate a landscape with a myriad of environmental and physical obstacles and hazards [McCarty 09a]. 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 19. 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.

**Table 19 - Node comparisons using various contexts**

	Nodes	Avg. # Purch	Avg. \$ Purch
Org DT	2x10 <sup>10</sup>	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

### *M2.5: Conclusion*

This chapter demonstrates two 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%.

## Minor Contribution #3: Applications of Heuristics to Local Search Algorithms

### *M3.1: Introduction*

This section presents heuristics used to modify traditional local search algorithms. Test results show significant increases in effectiveness were obtained. Some results were presented at ETFA08 Conference, September 2008 and this dissertation expands upon that original work and work in chapter 4.

Local Search Algorithms (LSAs) are designed to find solutions where comprehensive searches are impractical. Some of these techniques, however, suffer from limitations of their own; in particular, high failure rates. This chapter looks at ways heuristics can be used to address some of these limitations through the use of a heuristic to modify certain local search techniques called Descending Deviation Optimizations.

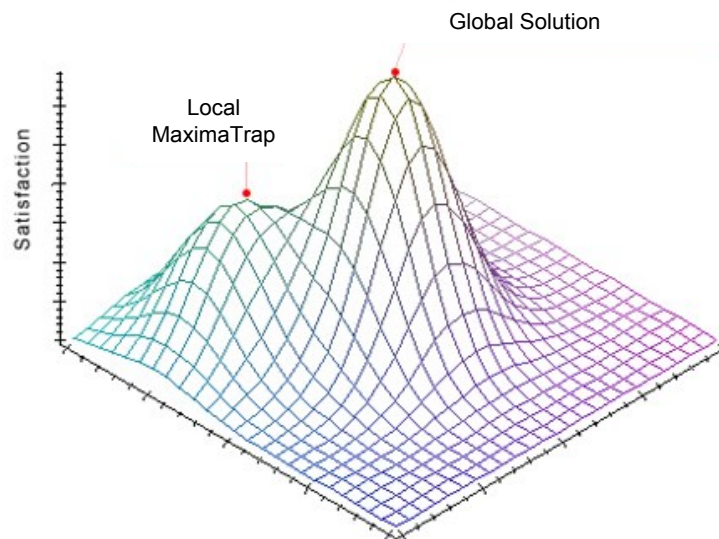
The 8-Queens problem is a classical problem in computer science whereby 8 queens are placed on a chessboard and an algorithm must determine how to arrange the queens in order that no queen can attack another. For an 8x8 standard chessboard there are 4,426,165,368 possible arrangements of 8 queens but only 92 solutions. A random process would need to try roughly 48 million combinations before finding one. Unfortunately, combinations of variables and constraints in similar problems can quickly result in the factorial growth of the possible permutations. This puts a comprehensive search beyond the practical ability of modern computer systems to perform. Problems like the 8-Queens problem 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 [Martinjak 07], [Sipser 12].

Local Search Algorithms (LSAs) have proven very useful for finding solutions to CSPs [Guimaraes 07] when comprehensive techniques are impractical. 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 09]. 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. In cases like the 8-Queens problem, where the solutions are fairly evenly distributed, any one random starting point is usually not too far from a solution. This makes LSAs the preferred method for solving CSPs [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 more controlled fashion so as to have a higher likelihood of finding global maxima.

*M3.2: Descending Deviation Optimization Technique For the 8-Queens Problem*

The 8-Queens problem, introduced in 1848 by Max Bezzel, is a good representative CSP because it contains a great many local maxima in addition to a small number of global solutions. In testing the various LSAs a random state generator was used to create 1000 random starting states to see how well the 8-Queens problem could be resolved to a goal state.

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 [Martinjak 07], [Russell 09], [Mantawy 99]; the so-called “greedy” approach. The problem is that problems populated with localized maxima can lure an unsuspecting algorithm into following a local gradient to a localized top, “trapping” an algorithm into thinking it has reached *the* peak, when in fact it has only reached a local peak. Such a “trap” is demonstrated by figure 99.

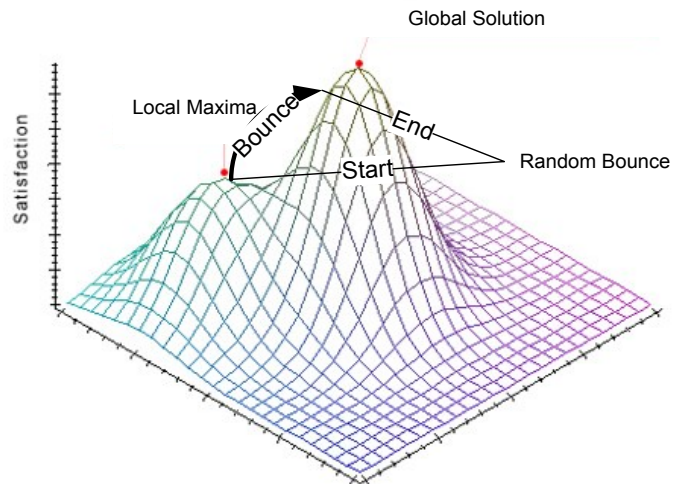


**Figure 99 - Comparison of Local/Global Maxima.**

Some LSAs attempt to escape out of these local maxima through some sort of random “bounce” [Kurbel 98], [Pasiadis 04], [Kliwer 00], [Mendonca 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 100.



**Figure 100 - Bounce out of a Local Maxima Trap.**

These random “bounces” are often not successful however; leading an algorithm away, rather than towards, a solution and expending time and computing resources in a fruitless search. Descending Deviation Optimization (DDO) tries to improve upon an LSAs ability to escape local maxima and find a goal state by restricting its movements somewhat in order to prevent it from moving too far away in any random direction from a potential goal state. The assumption is that for many CSPs the local maxima is not too far from a global solution so it might be advantageous to restrict the bounce in some way. This process works well for the 8-Queens problem, and potentially similar CSPs, because the state space contains a number of goal states spread uniformly throughout.

Local Search algorithms tried as described in [Martinjak 07] and [Russell 09]:

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 20.

**Table 20 - Initial Results of LSA Testing**

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

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, number iterations max set to 1000
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.

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

**Table 21 – Steps for Descending Deviation Optimization Technique**

<p><b>Step 1.</b> DDO-LSA generates a potential random choice. If the choice leads to a goal state then declare success.</p> <p><b>Step 2.</b> 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 is found or all choices are tested. If the choice involves adding a configuration that is below an acceptable threshold, the choice is skipped.</p> <p><b>Step 3.</b> If choice is accepted, the optimal threshold reduced by a predetermined amount and the algorithm moves to Step 1.</p>
---

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], [Mendonca 97]. SA’s pseudo-random selection method measures a random pick against a slowly descending de-optimization threshold.

Consider the following:

Let  $j$  be state of a CSP.

Let  $S$  be a set consisting of all the states of a CSP.  $S = \sum j \text{ states}$

Let there be a cost function  $C(j)$  for  $S$ .

Let another set  $\hat{S}$  be the set of global maxima (or minima) of all  $C(j)$ .

Hence  $\hat{S}$  is a proper subset of  $S$ :  $\hat{S} \subset S$ .

Let  $S(k)$  be the neighbors of  $S(j)$  (not including  $S$  itself) for some  $j$ .

Also, for every  $k$  neighbor, there exists a collection of positive coefficients,  $q_{jk}$ ,  $j \in S(k)$  such that  $\sum_{j \in S(k)} q_{jk} = 1$ .

Finally, let  $S(0)$  be the initial state

For Simulated Annealing, we also need a cooling schedule which is a non-increasing function

$$T: N \rightarrow (0, \infty)$$

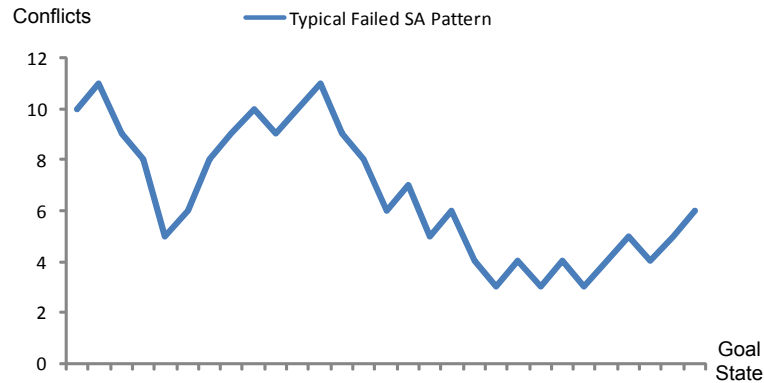
Where  $N$  is the set of positive integers and  $T(t)$  is the “temperature” at time  $t$ .

The SA algorithm is a discrete-time inhomogeneous Markov chain  $x(t)$  such that:

For  $x(t) = j$  choose a  $k$  neighbor at random using the probability  $q_{jk}$ . Choose each subsequent  $t + 1$  state (assuming you haven't reached the goal state) using the following equation:

$$P[x(t + 1) = k | x(t) = j] = q_{jk} e^{-\frac{1}{T(t)} \max\{0, J(j) - J(k)\}}, j \neq k, j \in S(k) \quad (\mathbf{M3.1})$$

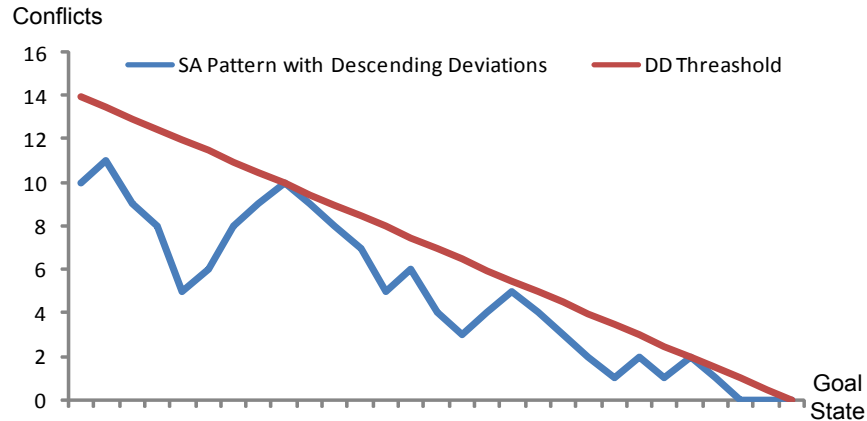
The algorithm allows a large range (nearly random) set of choices early on, getting progressively more restrictive as  $t$  increases in favor of better choices with each iteration. Since the range of options is greater in the beginning, it will have a tendency to explore more maxima and is correspondingly 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 “straight-up” local search techniques. However SA is comparatively computationally expensive. In addition, the algorithm can have a tendency to be led hopelessly astray by a succession of less than optimal choices as demonstrated in Figure 101.



**Figure 101 - 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 102. 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 102 - Simulated Annealing with Descending Deviations**

*M3.4: Test Examples*

In order to see how effective DDOs are, three LSAs using a random component were chosen for implementation; Simulated Annealing (SA), Genetic Mutation and Stochastic Hill Climbing (SHC).

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 09]. 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 version of the Genetic Mutation algorithm works by discarding mutations which exceed the descending threshold, in essence, only allowing genes with a minimal fitness level to “reproduce” and “killing” off the others.

The DDO-SHC, DDO-GM 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 22.

**Table 22 - Results of Modified Local Search Algorithm Testing**

Algorithm	Tries	Success	Failure	% Success
DDO-Stochastic Hill Climbing <sup>1</sup>	1000	253	747	25.3
DDO-Genetic Mutation <sup>2</sup>	1000	360	640	36.0
DDO-Simulated Annealing <sup>3</sup>	1000	993	7	99.3

1. DDO-Hill rescued 121 failures, deviations set at 5
2. DDO-Genetic Mutation number iterations max set to 1000, deviations set to 30
3. DDO-Simulated Annealing alpha set at .99, number iterations max set to 1000, temp set to 1000

#### *M3.5: Conclusion*

The Descending Derivation Optimization (DDO) heuristic was applied to a Stochastic Hill Climb Algorithm and Simulated Annealing Algorithm as described in [Russell 09] and a Genetic Mutation algorithm described in [Eiben 07]. Failure reduction for 8-Queens problem was as follows:

Stochastic Hill Climb – failures reduced from 854 to 747 or 12.5%

Genetic Mutation – failures reduced from 771 to 640 or 17%

Simulated Annealing – failures reduced from 729 to 7 or 99%

In all 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.



## APPENDIX B: FUZZY LOGIC PRIMER

This Appendix contains background material on Fuzzy Logic and its accepted variants: Type-1 and Type-2 along with extensions Nonstationary Fuzzy Logic, Polymorphic Signatures and Fuzzy Clustering. It also discusses techniques for building Hybrid Fuzzy Systems, or systems with a fuzzy and non-fuzzy algorithmic component. Each of these topics is used, referenced and/or built upon in the dissertation to develop the concept of Fuzzy Contexts and Fuzzymorphism in discussions, test examples and software.

### B.1: Fuzzy Logic Type 1

This Section contains background material on Type-1 Fuzzy Logic. The concept of Type-1 Fuzzy Sets is explained together with the elementary set-based operations.

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*

**(B.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], [Taheri 06]. 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*

**(B.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  $\leq 60 - 10$  mph then*  
     ***HARD\_ACCELERATE***  
*ElseIf SPEED  $\leq 60 - 5$  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  $\leq 60 + 5$  then*  
     ***SOFT\_BRAKE***  
*ElseIf SPEED  $> 60 + 5$  AND SPEED  $\leq 60 + 10$  then*  
     ***MEDIUM\_BRAKE***  
*ElseIf SPEED  $> 60 + 10$  then*  
     ***HARD\_BRAKE*** **(B.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  $-10 \text{ ft/s}^2$  provides no description for neighboring values such as deceleration of  $-9.9 \text{ ft/s}^2$ . 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 [Zadeh 65], 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”,  $\tilde{F}_1$  which consists of the set of all  $\mu(x)$  where  $\mu$  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  $X$ .

$$\tilde{F}_1 = \{x, \mu(x) \mid \forall x \in X, \mu(x) \subseteq [0, 1]\} \quad (\text{B.4})$$

The fuzzy set,  $\tilde{F}_1$ , as well as the membership function,  $\mu$ , 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  $\mu$ , 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:

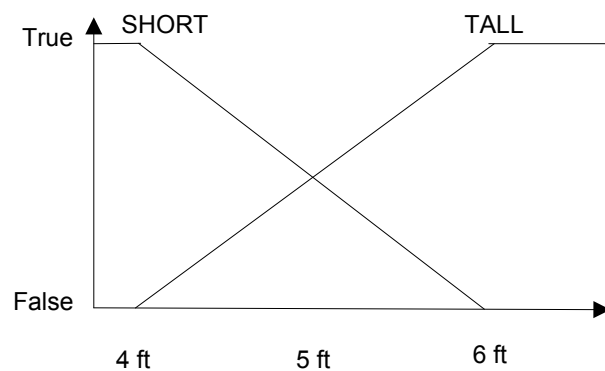
$$\begin{aligned} &6 \text{ feet IS TALL (Tall = True)} \\ &6 \text{ feet IS NOT SHORT (Short = False)} \end{aligned} \quad (\text{B.5})$$

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:

$$\begin{aligned}\mu_{Tall}(5'11.5'') &= 0.95 \\ \mu_{Short}(5'11.5'') &= 0.05\end{aligned}\tag{B.6}$$

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 103.



**Figure 103 - A basic, fuzzy description of a person's height**

The shapes in Figure 103 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  $\mu$  must produce exactly the same value as its Boolean counterpart. In the case described above, 6 feet = TALL:

$$\begin{aligned}\mu_{Tall}(6') &= 1 \Rightarrow 6 \text{ feet IS TALL} \\ \mu_{Short}(6') &= 0 \Rightarrow 6 \text{ feet IS NOT SHORT}\end{aligned}\tag{B.7}$$

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

To further extend Boolean logic, Zadeh introduced fuzzy set operators [Cox 94], taking the traditional operators intersection, union and complement operators and creating fuzzy equivalents. For example the fuzzy version of the intersection operator is the t-norm function  $\mu_t$  such that:

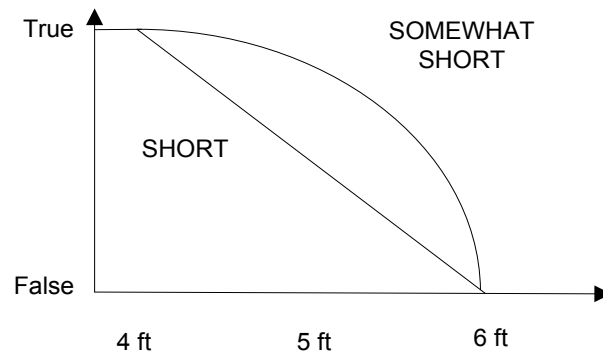
$$\mu_t(x, y) = \min(\mu(x), \mu(y))\tag{B.8}$$

These fuzzy operators reduce to Boolean equivalents at the endpoints 0 and 1, as demonstrated by the following truth table comparing the Boolean intersection (AND) with the fuzzy operator  $\mu_t$ .

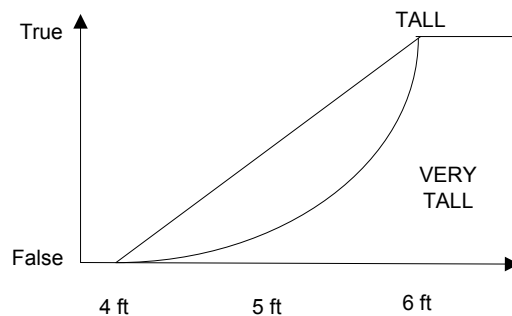
**Table 23 - Fuzzy and Boolean AND Truth Table**

Value A	Value B	A AND B	$\mu_t(A, B)$
1	1	1	$\min(1, 1) = 1$
1	0	0	$\min(1, 0) = 0$
0	1	0	$\min(0, 1) = 0$
0	0	0	$\min(0, 0) = 0$

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 104 and figure 105.



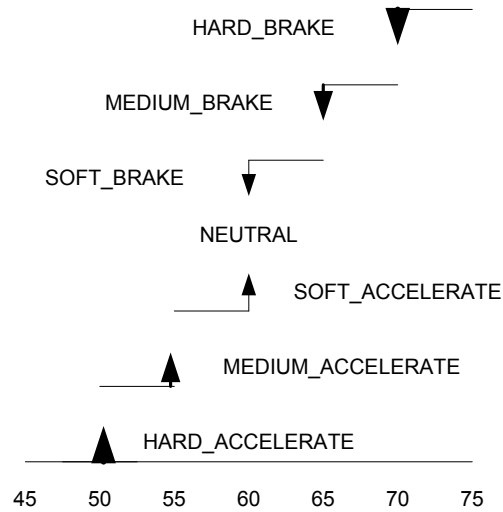
**Figure 104 - Fuzzy description of a person’s height using hedge SOMEWHAT.**



**Figure 105 - A basic, fuzzy description of a person’s height using hedge VERY.**

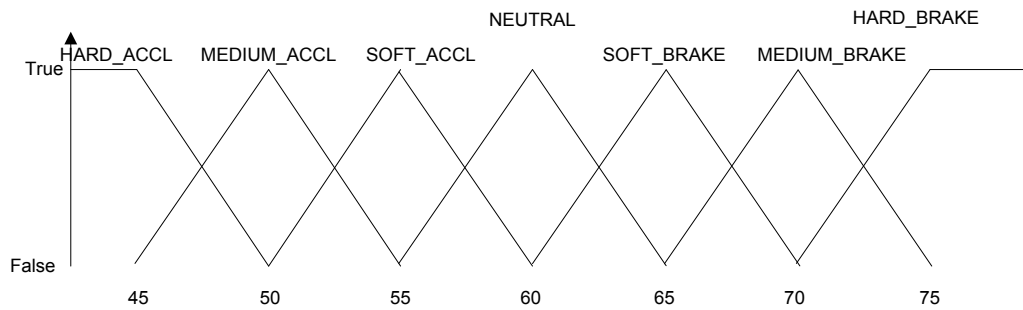
As would be expected, a person who is to some degree SHORT is to a greater degree SOMEWHAT SHORT, while a person who is to some degree TALL is to a lesser degree; VERY TALL.

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 106 A stair-step, crisp implementation.**

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



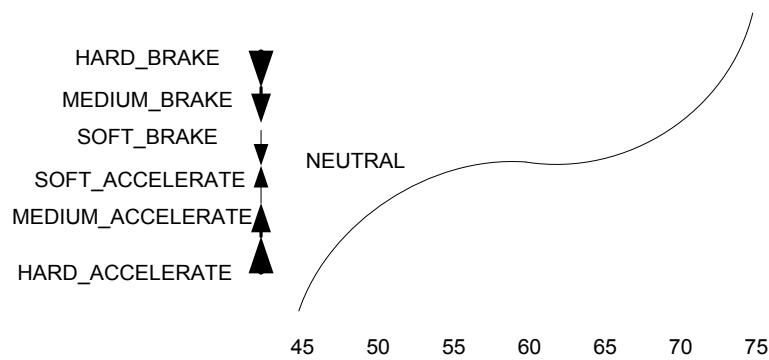
**Figure 107 - Constructing fuzzy sets from speed ranges.**



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

$$\begin{aligned} \mu_{Action}(SPEED) = & \mu_{Hard\_Accl}(SPEED) + \mu_{Medium\_Accl}(SPEED) + \mu_{Soft\_Accl}(SPEED) + \\ & \mu_{Neutral}(SPEED) + \mu_{Soft\_Brake}(SPEED) + \mu_{Medium\_Brake}(SPEED) + \mu_{Hard\_Brake}(SPEED) \end{aligned} \quad (B.9)$$

The resulting implementation, instead of a fragmented stair-step, would look more like a smoother sigmoid, or S-Curve as shown in Figure 108.



**Figure 108 - A smooth fuzzy implementation**

In many cases Fuzzy logic provides a number of advantages over traditional, crisp implementation [Hanss 05], [Cox 94], [Zadeh 08]:

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.

#### *B.1.1: Operations on Type-1 Fuzzy Sets*

Recall that in classical Boolean logic, a crisp set  $A$  in a domain  $X$  can be described using various methods, such as listing all of its members, providing a conditional description, or filter, of all members of  $A$ , or by specifying a binary function  $\mu_A \in \{0, 1\}$  where elements of  $A$  have a value of 1 (or true) and elements of the complement of  $A$  have a value of 0 (or false). Hence an element  $x$  either completely belongs to set  $A$  or it does not. As a result the crisp set  $A$  can be described as

$$\mu_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases} \quad \forall x \in X \quad (\text{B.10})$$

Fuzzy set theory, as opposed to the conventional Boolean logic, determines the “degree of belonging” of a particular element to a desired set. Hence, fuzzy set theory is a generalization of crisp set theory, since the degree of belonging of element  $x$  to set  $A$  is determined by the membership grade  $\mu_A(x)$  taking on value from the unit interval  $[0, 1]$ . Recall that, at the boundaries, the fuzzy set reduces to the crisp equivalent.

More formally, the fuzzy set  $A$  in the domain of  $X$  is defined as a set of ordered pairs of element  $x$  and its degree of membership  $\mu_A(x)$ :

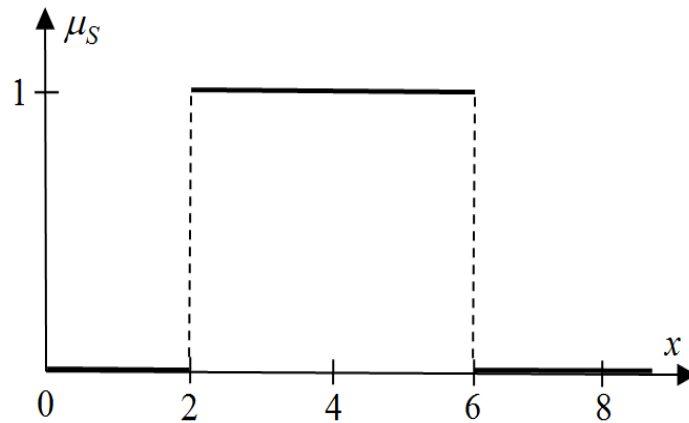
$$A = \{(x, \mu_A(x)) | x \in X\} \quad (\text{B.11})$$

In the special case when the domain  $X$  is a continuous space of real numbers the membership function  $\mu_A$  maps each number to values between 0 and 1,  $\Re \rightarrow [0 \dots 1]$ . Hence, the fuzzy set  $A$  can be described as:

$$A = \int_{x \in X} \mu_A(x) / x \quad (\text{B.12})$$

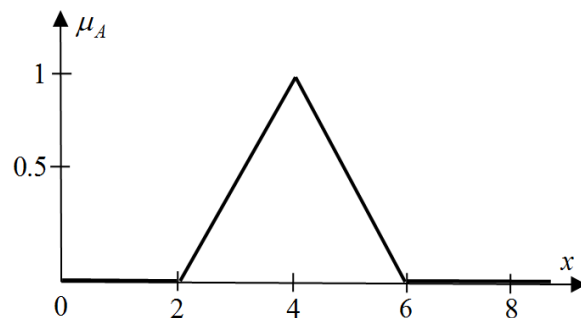
Note that in this particular case, the integral symbol does not denote integration, rather it symbolizes a collection of all the points in domain  $X$ . This is a common notation to use to describe the behaviors of fuzzy functions.

As an example, consider a membership function  $\mu_S(x)$  defining a crisp set  $S$  of numbers from the interval  $[2 \dots 6]$ . The function might look something like that shown in figure 109.



**Figure 109 – A crisp membership function**

Extending the above function to a fuzzy equivalent requires redefining the membership function. One possibility would be to introduce the notion of numbers CLOSE to number 4 as any number of distance 2 or less. Moreover, the closer a number is to 4, the greater its *close*-ness. This results in a membership function  $\mu_A$ , which defines the fuzzy set  $A$ . Membership function  $\mu_A(x)$  is shown in figure 110. The mathematically vague linguistic concept of CLOSE numbers cannot be accurately described by Boolean logic due to the inherent limitations of bivalent logic. Fuzzy set theory, in contrast, not only allows for such, but it also provides an intuitive way for describing such concepts through the use of terms which have their basis in the use of natural human language.



**Figure 110 – A Fuzzy Membership Function**

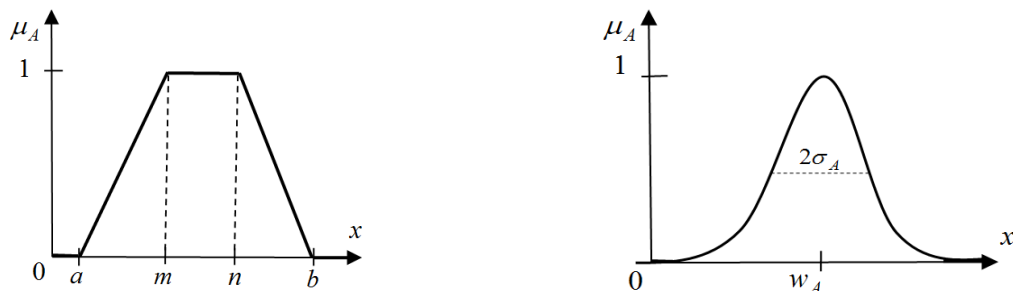
Fuzzy set  $A$  is defined by its membership function  $\mu_A(x)$  for all  $x$  in  $\mathfrak{R}$ . Typically, membership functions are expressed as parameterized mathematical functions. The equation for the  $\mu_A(x)$  shown above is a triangle and represented by the following:

$$\mu_A(x) = \begin{cases} 0, & \text{if } x < 2 \\ \frac{x-2}{2}, & \text{if } x \in [2, 4] \\ \frac{6-x}{2}, & \text{if } x \in [4, 6] \\ 0, & \text{if } x > 6 \end{cases} \quad (\text{B.13})$$

A generalized version of the above is:

$$\mu_A(x) = \begin{cases} 0, & \text{if } x < a \\ \frac{x-a}{c-a}, & \text{if } x \in [a, c] \\ \frac{b-x}{b-c}, & \text{if } x \in [c, b] \\ 0, & \text{if } x > b \end{cases} \quad (\text{B.14})$$

The functions do not have to be continuous but they have to be convex. The triangle shape is among the most commonly used fuzzy set. It is easy to calculate and does a good job of representing a fuzzy concept. Other commonly used membership functions are the trapezoid, and Gaussian (bell-shaped) membership functions, shown in figure 111.



**Figure 111 – Trapezoid and Gaussian shaped membership functions**

The trapezoid has certain advantages over other shapes in that it allows a range of values to have membership 1, through the use of a “plateau”. Trapezoids can also be either

“right” or left”, where the corresponding left or right leg is removed and the plateau is allowed to extend into infinity. An example of a left trapezoid is shown in figure 112.

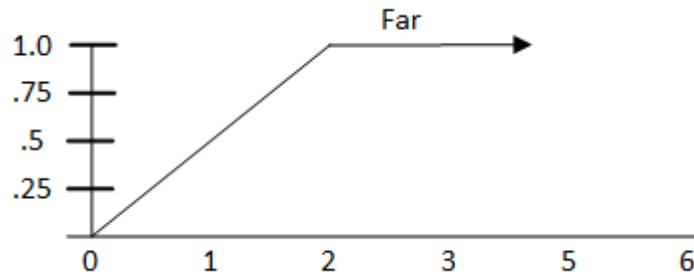


Figure 112 – A left trapezoid

A generalized mathematical representation of the trapezoid membership function is:

$$\mu_A(x) = \begin{cases} 0, & \text{if } x < a \\ \frac{x-a}{m-a}, & \text{if } x \in [a, m] \\ 1, & \text{if } x \in [m, n] \\ \frac{b-x}{b-n}, & \text{if } x \in [n, b] \\ 0, & \text{if } x > b \end{cases} \quad (\text{B.15})$$

The Gaussian provides a smooth non-linear transformation. A generalized mathematical representation of the Gaussian membership function is as follows:

$$\mu_A(x) = \exp\left(-\left(\frac{x-w_A}{\sigma_A}\right)^2\right) \quad (\text{B.16})$$

Fuzzy set theoretic operations extend classical set arithmetic to fuzzy sets, which is then utilized by a fuzzy inference system to both fuzzify and defuzzify results [Cox 94]. Recall the basic fuzzy set operations of union, intersection and complement. Consider two Type-1 fuzzy sets  $A$  and  $B$  described by their membership functions  $\mu_A(x)$  and  $\mu_B(x)$ . The operation of fuzzy intersection is defined as follows:

$$A \cap B = \mu_{A \cap B}(x) = \mu_A(x) \prod \mu_B(x) \quad (\text{B.17})$$

In this case, the symbol  $\prod$  denotes a fuzzy t-norm operation. The fuzzy t-norm operation is a binary operation on a unit interval that satisfies the following four axioms for all  $a, b, c \in [0,1]$  [Hanss 05]:

$$\text{Axiom 1 (Boundary condition): } a \prod 1 = a \quad (\text{B.18})$$

$$\text{Axiom 2 (Monotonicity): } b \leq c \Rightarrow (a \prod b) \leq (a \prod c) \quad (\text{B.19})$$

$$\text{Axiom 3 (Commutativity): } a \prod b = b \prod a \quad (\text{B.20})$$

$$\text{Axiom 4 (Associativity): } a \prod (b \prod c) = (a \prod b) \prod c \quad (\text{B.21})$$

As an example, consider a pair of fuzzy sets  $\mu_A(x)$  and  $\mu_B(x)$ , shown in figures 113 and 114.

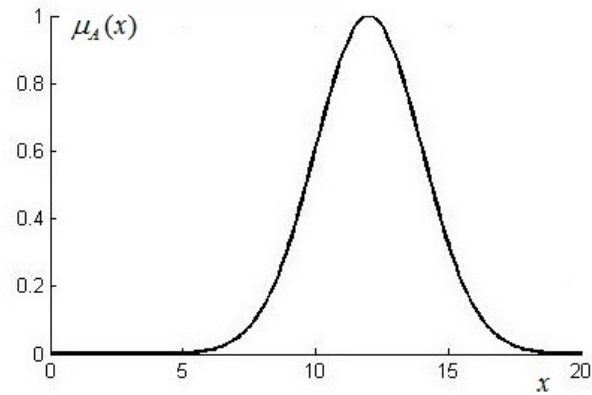


Figure 113 -  $\mu_A(x)$

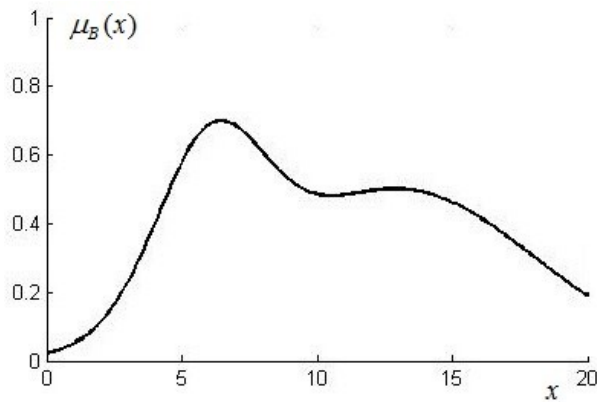


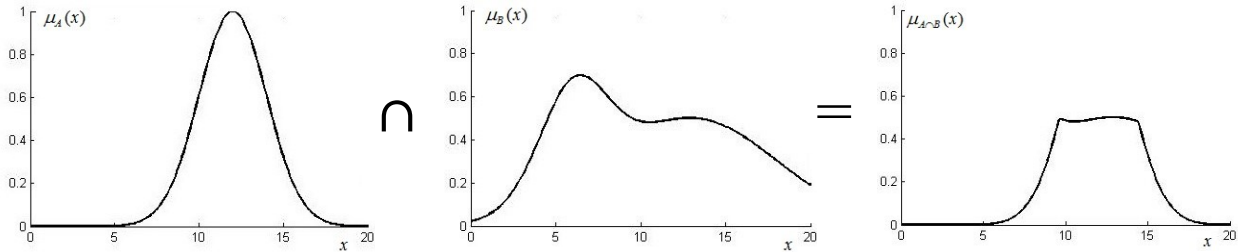
Figure 114 -  $\mu_B(x)$

Recall the  $\mu_{min}$  operator introduced earlier in this section. Suitable t-norm operations are the minimum and the product operations. Using the minimum t-norm the fuzzy intersection can be written as:

$$A \cap B = \mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)) \quad (\text{B.22})$$



The result of the intersection operation on two example Type-1 fuzzy sets using the minimum t-norm is demonstrated in figure 115.



**Figure 115 – Operations on T1 Fuzzy Sets Using Minimum T-Norm Operator**

Similarly, for the  $\mu_{max}$  operation, there is the fuzzy union. The operation of fuzzy union is defined as:

$$A \cup B = \mu_{A \cup B}(x) = \mu_A(x) \amalg \mu_B(x) \quad (\text{B.23})$$

In this case, the symbol  $\amalg$  denotes a fuzzy t-conorm operation. A fuzzy t-conorm operation is a binary operation on fuzzy sets that satisfies the following four axioms for all  $a, b, c \in [0,1]$  [Hanss 05]:

$$\text{Axiom 1 (Boundary condition): } a \amalg 0 = a \quad (\text{B.24})$$

$$\text{Axiom 2 (Monotonicity): } b \leq c \Rightarrow (a \amalg b) \leq (a \amalg c) \quad (\text{B.25})$$

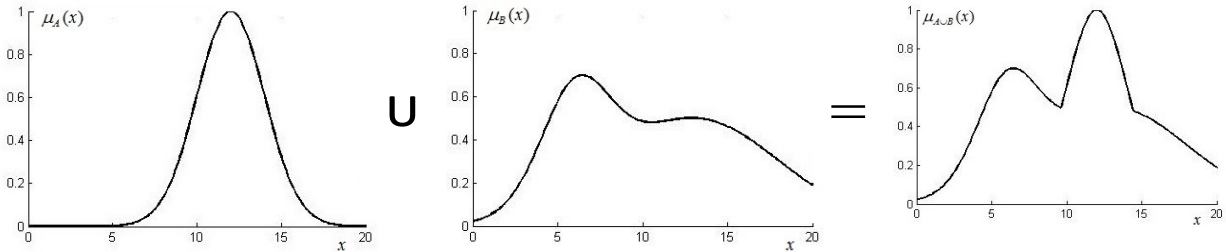
$$\text{Axiom 3 (Commutativity): } a \amalg b = b \amalg a \quad (\text{B.26})$$

$$\text{Axiom 4 (Associativity): } a \amalg (b \amalg c) = (a \amalg b) \amalg c \quad (\text{B.27})$$

The maximum operation now becomes a t-conorm operation. Using the maximum t-conorm, the fuzzy union can be written as:

$$A \cup B = \mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)) \quad (\text{B.28})$$

Applying the union operation on the two sample Type-1 fuzzy sets using the maximum t-conorm is shown in figure 116.



**Figure 116 – Operations on T1 Fuzzy Sets Using the Maximum T-Conorm Operator**

The minimum t-norm and maximum t-conorm operations are widely used in fuzzy inference systems due to their ease of implementation and computational simplicity. Analogous to the classical Boolean *AND* and *OR* logic operations, they serve to bridge the gap between crisp input and output and the internal fuzzy implementation.

Lastly, the fuzzy complement  $\bar{A}$  of fuzzy set  $A$  has the following representation:

$$\bar{A} = \mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad (\text{B.29})$$

#### B.1.2: Type-1 Fuzzy Inference Systems

Since the introduction of fuzzy logic in 1965 by Lofti Zadeh, fuzzy logic has been applied in various forms in a wide range of applications [Zadeh 65], [Cox 94], [Hanss 05]. One of the most common and arguably successful implementations of fuzzy logic is its combination with rule-based systems, generally referred to as a Fuzzy Inference System (FIS).

FISs can be seen as a nonlinear mapping of an input data vector into a scalar output [Cox 95]. The core of the FIS is composed of a fuzzy inference engine that processes linguistic fuzzy terms and rules. The simplicity and easily-termed fuzzy rules allows for

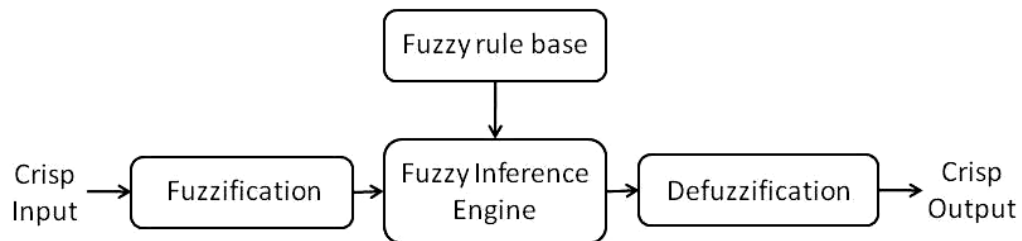
relatively easy programmatic implementation; although conceptually fuzzy logic has made only small inroads into mainstream business applications. A proliferation of fuzzy tools from companies such as Matlab, or cheap visual wizards, like the one described in chapter 4 of this dissertation should help bring FISs into more widespread use.

The advantage of the FIS is that human knowledge can be easily encoded into and extracted from a set of given fuzzy rules. Despite their relatively simple structure, FISs can approximate sophisticated non-linear controls in complex systems.

The most commonly used FIS is composed of four major components:

1. An input fuzzification mechanism to convert a crisp input vector into a fuzzy number.
2. A fuzzy rule base storing a series of fuzzy rules and terms in the form of IF..THEN statements.
3. A fuzzy inference engine to determine the contributions of each rule to the total output.
4. An output defuzzification operation to convert the fuzzy output to a crisp, scalar result.

This structure of the typical FIS is depicted in figure 117.



**Figure 117 – Typical Fuzzy Inference System**

The FIS operates in the following manner:

**Step 1:** The FIS accepts as input a real valued input vector.

**Step 2:** From the rule base, fuzzy rules are tested with against the input to determining the contribution of each rule.

The fuzzy rule base contains a set of linguistic rules written in an implicative form combining fuzzy terms. Consider an FIS with  $n$  inputs  $x_1 \in X_1, \dots, x_n \in X_n$  and a single output  $y$ . The  $k^{th}$  linguistic fuzzy rule can be formulated as:

$$\text{Rule } R_k: \text{IF } x_1 \text{ is } A_1^k \text{ AND } \dots \text{ AND } x_n \text{ is } A_n^k \text{ THEN } y \text{ is } B^k \quad (\text{B.30})$$

Where  $A_i^k$  denotes the input fuzzy set in the  $i^{th}$  input dimension and  $B^k$  is the output fuzzy set for the  $k^{th}$  rule. This type of FIS using fuzzy rules in the form of (eq above) was originally defined in by Mamdani [Mamdani 75]. The FIS uses the Mamdani type implication to compute the output of rule  $R_k$ . The membership function of rule  $R_k$  can be denoted as  $\mu_{R_k}(\vec{x}, y)$ , where  $\vec{x} = (x_1, \dots, x_n)$ . Its value can be computed by applying a t-norm operator to the rule antecedents as well as the rule consequents:

$$\mu_{R_k}(\vec{x}, y) = \mu_{A_1^k}(x_1) \prod \dots \prod \mu_{A_n^k}(x_n) \prod B_k(y) \quad (\text{B.31})$$

Which reduces to:

$$\mu_{R_k}(\vec{x}, y) = \left[ \prod_{i=1}^n \mu_{A_i^k}(x_i) \right] \prod B_k(y) \quad (\text{B.32})$$

**Step 3:** For each rule, the input is tested against each of the rule's fuzzy terms, then a t-norm operation, such as the one described in equation B.22, is applied to all the terms (and their representative fuzzy sets) to determine the strength, or contribution, of the rule.

**Step 4:** A t-conorm operator, such as the one described in equation B.28, combines all the rules to determine the overall fuzzy result.

Suppose that there are  $K$  rules in the fuzzy rule base, the result of applying the Mamdani implication to all rules will be a set of  $K$  output fuzzy sets defined by their respective membership function  $\mu_{R_i}(\vec{x}, y), i = 1, \dots, K$  which denotes the contribution, or strength, of each corresponding fuzzy set. The final output fuzzy set,  $B(y)$  for the input vector  $\vec{x}$  is computed by aggregating the membership functions of all rules via the t-conorm operator:

$$B(y) = \coprod_{i=1}^K \mu_{R_i}(\vec{x}, y) \quad (\text{B.33})$$

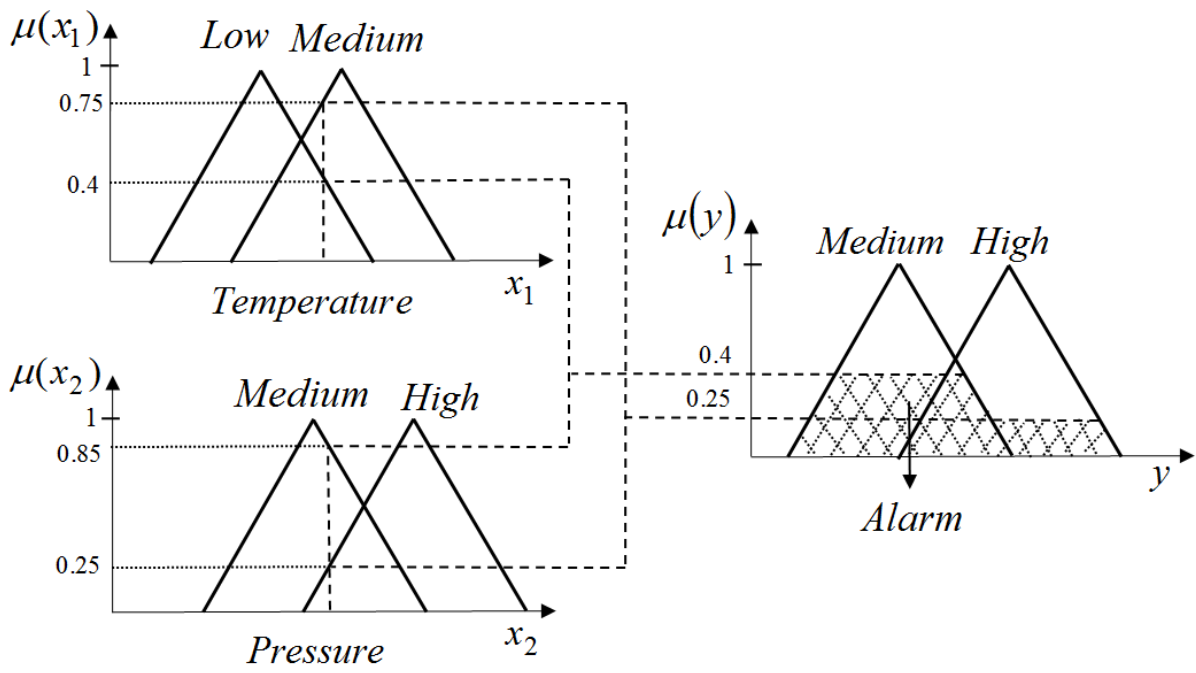
**Step 5:** A defuzzification process, such as the one described in equation B.33, converts the fuzzy result into a crisp, scalar value which is passed back to the caller.

The purpose of the output defuzzification stage is to compute a scalar output based on the output fuzzy sets. Many methods for output defuzzification are available in the literature, e.g. centroid defuzzifier, center-of-sums defuzzifier, height defuzzifier or center-of-sets defuzzifier [Mendel 01]. In this case, consider the centroid defuzzifier. The centroid defuzzifier calculates the centroid of the output fuzzy set,  $B(y)$ , which is described by using its membership function  $\mu_B(y)$ . The final output value  $y(\vec{x})$  produced by the FIS is then computed by the following equation:

$$y(\vec{x}) = \frac{\sum_{i=1}^N y_i \mu_B(y_i)}{\sum_{i=1}^N \mu_B(y_i)} \quad (\text{B.34})$$

In this case,  $N$  denotes the number of discretized samples in the output domain of variable  $y$  and  $y_i$  is the discretized sample.

To illustrate the workings of the fuzzy inference mechanism, consider an application to an alarm system such as the one shown in figure 118. The amplitude of the alarm signal depends upon the combined measures of temperature and pressure. The input values are encoded into human-readable linguistic terms, such as LOW, MEDIUM, and HIGH. Using the fuzzy inference system, the consequents of each rule is fired according to the fuzzified input vector. The resulting aggregated output fuzzy sets is defuzzified into a crisp, scalar output value.



**Figure 118 – Fuzzification/Defuzzification Process**

## B.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], [Castillo 08]:

1. Experts can disagree on meaning of linguistics terms.

“Cold”, “Warm” and “Hot” can have different meanings to experts living in different regions of the world. 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.

Trying to ascertain a number “near” a fuzzy boundary becomes even more difficult when the number itself is uncertain. All of these “uncertainties” can influence the ability of a Type-1 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 [Mendel 10]. 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  $\mu$ :

$$\widetilde{F}_1 = \{x, \mu(x) \mid \forall x \in X, \mu(x) \subseteq [0, 1]\} \quad (\text{B.35})$$

Fuzzy Type-2 creates a new fuzzy set  $\widetilde{F}_2$  which is the union of a new membership function  $\mu_2$  applied to members of  $\widetilde{F}_1$ :

$$\widetilde{F}_2 = \{((x, \mu(x)), \mu_2(x, \mu(x))) \mid \forall x \in X, \forall \mu(x) \subseteq [0, 1]\} \quad (\text{B.36})$$

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.

Mendel states, “When we cannot determine the membership of an element in a set as 0 or 1, we use fuzzy sets of type-1. Similarly, when the circumstances are so fuzzy that we have trouble determining the membership grade even as a crisp number in  $[0, 1]$ , we use fuzzy sets of type-2.” [Mendel 02]

Put another way, a Type-2 Fuzzy System  $\tilde{A}$  can be expressed using a Type-2 fuzzy membership function  $\mu_{\tilde{A}}(x, u)$ , where  $x \in X$  and  $u \in J_x$ .

$$\tilde{A} = \int_{x \in X} \int_{u \in J_x} \mu_{\tilde{A}}(x, u) / (x, u), J_x \subseteq [0, 1] \quad (\text{B.37})$$

The operator  $\int \int$  symbolizes a union over all possible values of  $x$  and  $u$ , and the membership result  $\mu_{\tilde{A}}$ ,  $0 \leq \mu_{\tilde{A}} \leq 1$ . Variables  $x$  and  $u$  represent the primary and the secondary variables and  $J_x$  denotes the primary membership of  $x$ . Two different representations of Type-2 fuzzy sets are commonly used. The first is the *vertical-slice* representation and the second is the *wavy-slice* representations.

Recall that the Footprint of Uncertainty (FOU) is a bounded region that consists of the uncertainty of primary memberships. The *vertical-slice* representation is where each of the primary memberships consists of a vertical slice, the union of which is:

$$FOU(\tilde{A}) = \cup_{x \in X} J_x \quad (\text{B.38})$$



In determining a specific value for  $x = \acute{x}$  a vertical slice  $\mu_{\tilde{A}}(x, \acute{u})$  of the Type-2 fuzzy membership function  $\mu_{\tilde{A}}(x, u)$  is obtainable. This vertical slice defines a secondary membership function:

$$\mu_{\tilde{A}}(\mathbf{x} = \acute{x}, \mathbf{u}) \text{ for } \acute{x} \in X \text{ and } \forall \mathbf{u} \in J_{\acute{x}} \subseteq [0, 1] \quad (\text{B.39})$$

Such that

$$\mu_{\tilde{A}}(\mathbf{x} = \acute{x}, \mathbf{u}) \equiv \mu_{\tilde{A}}(\acute{x}) = \int_{u \in J_{\acute{x}}} \frac{f_{\acute{x}}(u)}{u} J_{\acute{x}} \subseteq [0, 1] \quad (\text{B.40})$$

In this case  $f_{\acute{x}}(u)$  represents the secondary grade or the amplitude of the secondary membership function, hence  $f_{\acute{x}}(u) \in [0, 1]$ . The primary membership  $J_{\acute{x}}$  can be also understood as the support of the secondary membership function. It is this particular nature of the secondary membership function that defines the type of the underlying Type-2 fuzzy set. If, for example, all the secondary membership functions are intervals, the Type-2 fuzzy set is called an Interval Type-2 (IT2) fuzzy set. In case of an arbitrary secondary membership functions such as a Gaussian, the Type-2 fuzzy set is considered a General Type-2 (GT2) fuzzy set. As an example, assume that the primary domain  $X$  is discretized using  $N$  samples. The corresponding GT2 fuzzy set  $\tilde{A}$  can be represented as a composition of all its vertical slices:

$$\tilde{A} = \sum_{i=1}^N \left[ \int_{u \in J_{x_i}} f_{x_i}(u)/u \right] / x_i \quad (\text{B.41})$$

Similar to the vertical-slice, the *wavy-slice* representation of a GT2 fuzzy set  $\tilde{A}$  can be also constructed as a composition of all of its embedded fuzzy sets:  $\tilde{A}_e$ . Consider the case of a continuous universe of discourse  $X$  and  $U$ , an embedded GT2 fuzzy set  $\tilde{A}_e$  can be represented using the following equation:

$$\tilde{A}_e = \int_{x \in X} \frac{\lceil \frac{f_x(\theta)}{\theta} \rceil}{x}, \theta \in J_x \subseteq U \in [0, 1] \quad (\text{B.42})$$

Hence at each value of the primary variable  $x$  the embedded fuzzy set has the single value of the primary membership  $\theta$  together with its corresponding secondary value  $f_x(\theta)$ . In the case of a discrete universe of discourse with  $N$  elements, the embedded fuzzy set  $\tilde{A}_e$  becomes the following:

$$\tilde{A}_e = \sum_{i=1}^N [f_{x_i}(\theta_i)/\theta_i]/x_i \quad \theta_i \in J_x \subseteq U \in [0, 1] \quad (\text{B.43})$$

However there is a problem. In the case of a continuous universe of discourse the number of existing embedded fuzzy sets turns out to be uncountable. For the discrete universe of discourse the number of possible embedded fuzzy sets is:

$$n = \prod_{i=1}^N M_i \quad (\text{B.44})$$

where  $M_i$  is the number of discretized samples in the primary membership  $J_{x_i}$ . Because of this large number of existing embedded fuzzy sets, the *wavy-slice* representation is generally considered unsuitable for practical applications. Regardless, it has proven to be a useful technique for deriving some of the fundamental ideas of GT2 fuzzy sets and translating from T1 fuzzy sets to GT2 fuzzy sets [Hidalgo 10].

Per the Mendel and John representation theorem, the GT2 fuzzy set  $\tilde{A}$  can be described as a union of all of its  $n$  embedded fuzzy sets [Mendel 06]:

$$\tilde{A} = \cup_{j=1}^n \tilde{A}_e^j \quad (\text{B.45})$$

In order to properly assess the amount of uncertainty modeled by the GT2 fuzzy sets we return again to the concept of the Footprint of Uncertainty (FOU). The FOU of a GT2 fuzzy set  $\tilde{A}$  can be defined as the bounded region created by taking the union of all the primary memberships:

$$FOU(\tilde{A}) = \cup_{x \in X} J_x \quad (\text{B.46})$$

In general, the larger the FOU of a GT2 fuzzy set, the more corresponding uncertainty there is about the respective membership grades. Fortunately, the bounded FOU region can be also conveniently described using the upper fuzzy membership function:  $\bar{\mu}_{\tilde{A}}(x)$ , and the lower membership function:  $\underline{\mu}_{\tilde{A}}(x)$  which happen to be Type-1 FISs:

$$\bar{\mu}_{\tilde{A}}(x) = \overline{FOU(\tilde{A})} \quad (\text{B.47})$$

$$\underline{\mu}_{\tilde{A}}(x) = \underline{FOU(\tilde{A})} \quad (\text{B.48})$$

### B.2.1: Operations on General Type-2 Fuzzy Sets

Zadeh [Zadeh 65] showed how Type-1 Fuzzy Sets supported classical Boolean set operations. Similarly, the elementary set theoretic operations of union, intersection and complement can also be applied to General Type-2 Fuzzy Sets. Recall that Type-1 fuzzy set operations produced Type-1 fuzzy sets. Similarly, the product of applying the union intersection and complement operations to General Type-2 fuzzy sets is another GT2 fuzzy set. Consider two GT2 fuzzy sets. Let the first set  $\tilde{A}$  be defined as follows:

$$\tilde{A} = \int_{x \in X} \frac{\mu_{\tilde{A}}(x)}{x} = \int_X \frac{\left( \int_{J_x^u} \frac{f_x(u)}{u} \right)}{x} \quad J_x^u \subseteq [\mathbf{0}, \mathbf{1}] \quad (\text{B.49})$$

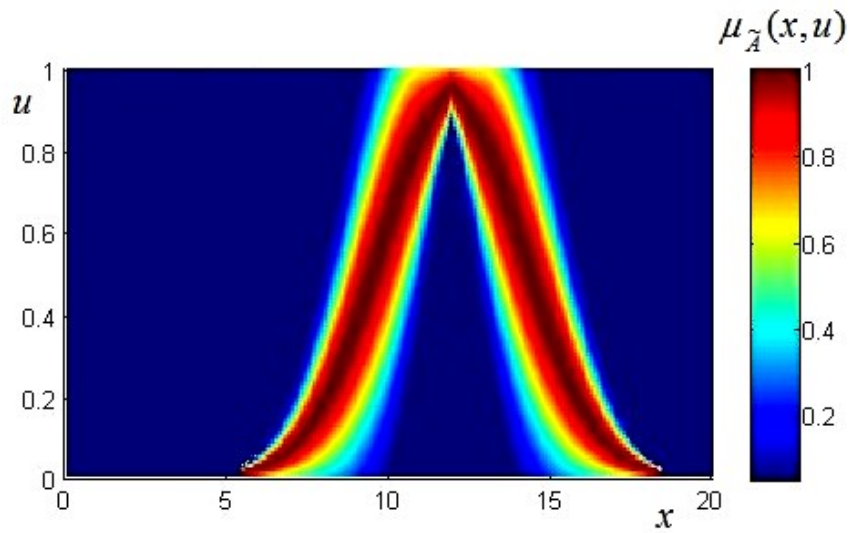


Figure 119 - GT2 Fuzzy Set  $\tilde{A}$

Let the second GT2 set  $\tilde{B}$  be defined as follows:

$$\tilde{B} = \int_{x \in X} \frac{\mu_{\tilde{B}}(x)}{x} = \int_X \frac{\left( \int_{J_x^w} \frac{g_x(w)}{w} \right)}{x} \quad J_x^w \subseteq [0, 1] \quad (\text{B.50})$$

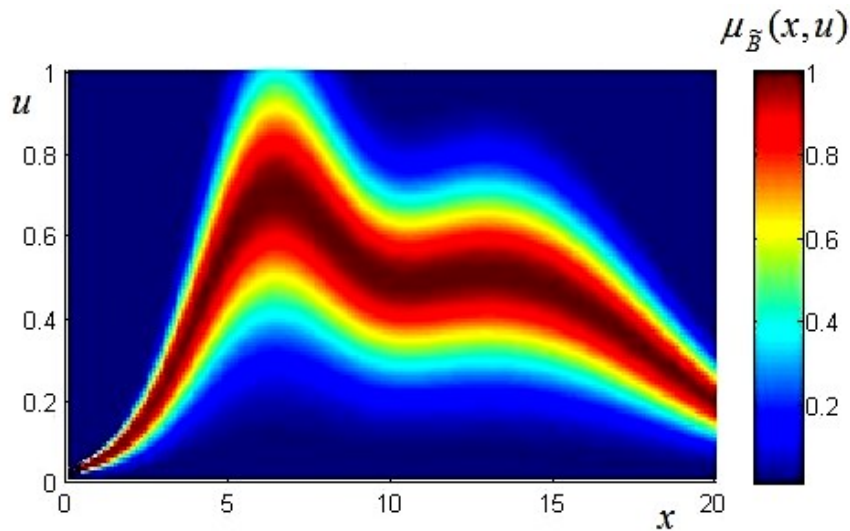
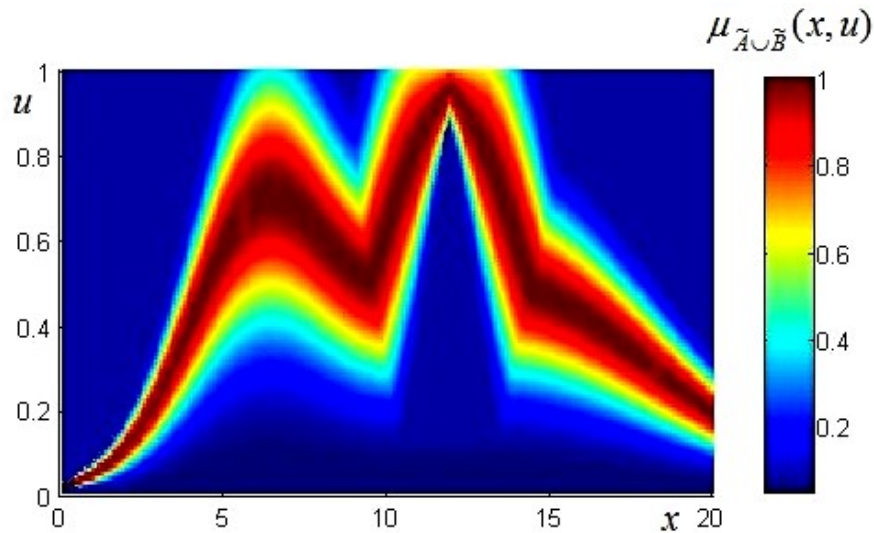


Figure 120 – GT2 Fuzzy Set  $\tilde{B}$

Functions  $f_x(u)$  and  $g_x(w)$  serve as the secondary membership functions of GT2 fuzzy sets  $\tilde{A}$  and  $\tilde{B}$ , respectively. Hence, the union of the two GT2 fuzzy sets can be defined by the extension principle represented by the equation [Mendel 02]:

$$\tilde{A} \cup \tilde{B} = \mu_{\tilde{A} \cup \tilde{B}}(x) = \int_{\mu \in J_x^u} \int_{w \in J_x^w} \frac{f_x(u) \Pi g_x(w)}{(u \Pi w)} \quad x \in X \quad (\text{B.51})$$

The above expression calculates the union of two GT2 fuzzy sets by computing the t-conorm operations between every possible pair of primary membership values  $u$  and  $w$  and by calculating the t-norm operation between their respective membership function results  $f_x(u)$  and  $g_x(w)$ . This process is repeated across the domain  $X$ . The t-norm and t-conorm operations are the same operations as discussed in Section B.1. The resulting union operation applied to the two GT2 fuzzy sets  $\tilde{A}$  and  $\tilde{B}$  are shown in figure 121.



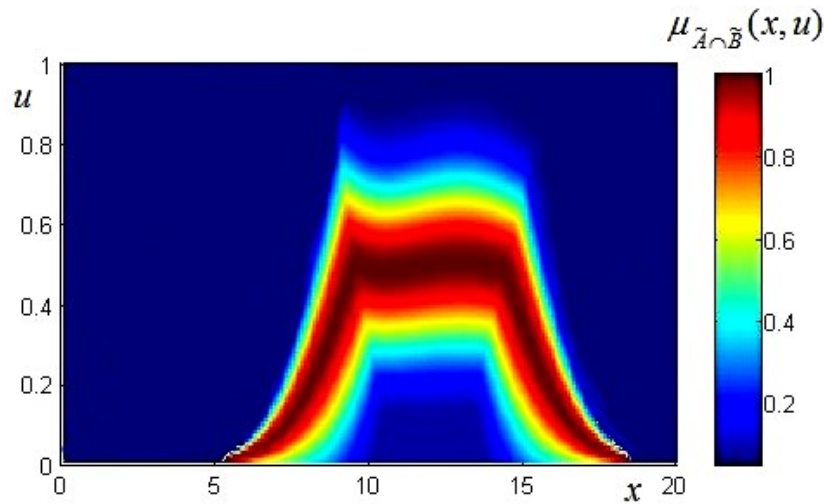
**Figure 121 – The resulting Union operation of GT2 Fuzzy Sets  $\tilde{A}$  and  $\tilde{B}$**

The intersection operation, on the other hand, applied to the above GT2 fuzzy sets is represented as:

$$\tilde{A} \cap \tilde{B} = \mu_{\tilde{A} \cap \tilde{B}}(x) = \int_{u \in J_x^u} \int_{w \in J_x^w} \frac{f_x(u) \Pi g_x(w)}{(u \Pi w)} \quad x \in X \quad (\text{B.52})$$

Contrast this with the union operation shown in equation B.51. The operations are very similar except in the case of the intersection, the application of the t-norm operation is applied to every possible pair of the primary membership values  $u$  and  $w$ .

The resulting intersection is demonstrated by figure 122.



**Figure 122 – The resulting Intersection operation of GT2 Fuzzy Sets  $\tilde{A}$  and  $\tilde{B}$**

The complement of a GT2 FS can be computed by calculating the negation of the secondary membership function. Hence, the complement of the GT2 fuzzy set  $\tilde{A}$ ,  $\tilde{\bar{A}}$  can be expressed as:

$$\tilde{\bar{A}} = \mu_{\tilde{\bar{A}}}(x) = \int_{u \in J_x^u} \frac{f_x(u)}{1-u} \quad x \in X \quad (\text{B.53})$$

#### B.2.2: Type-2 Fuzzy Inference System

A Type-2 Fuzzy Inference System follows much the same high-level construct as Type-1. It receives an input vector  $\vec{x}$ , and returns a scalar, crisp value  $y(\vec{x})$ . Type-2 sets are combined into linguistic terms and the terms into fuzzy rules. At a lower level, Type-2 operations extend their Type-1 equivalents in order to accommodate the Type-2 constructs. First is the need to calculate the centroid of a Type-2 consequent in order to obtain a crisp

value from a Type-2 FIS. The first step in this process is to type-reduce the Type-2 results into a Type-1 equivalent [Karnik 01] after which the new centroid is defuzzified using a standard defuzzification method for Type-1.

Recall that the centroid  $C_A$  of Type-1 fuzzy set  $A$  with the domain  $X$  discretized into  $N$  samples can be computed as the weighted average of the sampled domain values, where the particular membership strength is used as a weighting factor:

$$C_A = \frac{\sum_{i=1}^N x_i \mu_A(x_i)}{\sum_{i=1}^N \mu_A(x_i)} \quad (\text{B.54})$$

The Type-2 equivalent centroid  $C_{\tilde{A}}$  has as its fuzzy set  $\tilde{A}$ , represented by the equation:

$$\tilde{A} = \sum_{i=1}^N \left[ \int_{u \in J_x} f_{x_i}(u) / u \right] / x_i \quad (\text{B.55})$$

As before assume the domain  $X$  is discretized into  $N$  samples and can be defined based on the *Extension Principle* represented by the equation [Mendel 01]:

$$C_{\tilde{A}} = \int_{\theta_1 \in J_{x_1}} \dots \int_{\theta_N \in J_{x_N}} [f_{x_1}(\theta_1) \wedge \dots \wedge f_{x_N}(\theta_N)] / \frac{\sum_{i=1}^N x_i \theta_i}{\sum_{i=1}^N \theta_i} \quad (\text{B.56})$$

In this case, every possible combination of variables  $\theta_1, \dots, \theta_N$  comprise an embedded fuzzy set, which has a secondary contribution of  $f_{x_1}(\theta_1) \wedge \dots \wedge f_{x_N}(\theta_N)$ . The wedge operator  $\wedge$  is the specific t-norm used - in this case, the minimum operator. The elements of the centroid can be computed by defuzzifying the embedded fuzzy set.

This process is possible through type reduction of the Type-2 set. There are a number of proposed methods of type reduction. Two discussed here are 1) Exhaustive type reduction and 2) Center of sets type reduction.

B.2.2.1: Exhaustive type-reduction

Recall the definition of the type-2 centroid:

$$\int_{\theta_1 \in J_{x_1}} \cdots \int_{\theta_N \in J_{x_N}} [f_{x_1}(\theta_1) \wedge \cdots \wedge f_{x_N}(\theta_N)] / \frac{\sum_{i=1}^N x_i \theta_i}{\sum_{i=1}^N \theta_i} \quad (\text{B.57})$$

Under this definition all the possible combination of variables  $\theta_1, \dots, \theta_N$  must be computed in order to produce a result. This requires the exhaustive enumeration of all possible embedded fuzzy sets as defined in equations B.55 and B.56. This process is the initial proposed method for type-reduction of General Type-2 fuzzy sets [Karnik 01].

In the case of a domain  $X$  that has been discretized into  $N$  samples the number of possible embedded fuzzy sets is equal to  $n = \prod_{i=1}^N M_i$ , where  $M_i$  is the number of discretized samples in the primary membership  $J_{x_i}$ . Clearly, even for a small number of discretized samples, the number of embedded fuzzy sets grows prohibitively large. For a continuous domain it uncountable. As a result, this method is generally not practical, nevertheless, this exhaustive type-reduction method does prove to be useful theoretically in that it enables the deriving of other theoretical concepts of GT2 fuzzy sets and fuzzy inference systems. The exhaustive type-reduction algorithm has the following steps:

**Step 1:** Discretize the domain  $X$  into  $N$  points.

**Step 2:** Discretize each primary membership  $J_{x_i}$  into  $M_i$  points.

**Step 3:** Enumerate over all possible embedded fuzzy sets. There are a total of  $n$  of them where  $n = \prod_{i=1}^N M_i$ .

**Step 4:** For each embedded fuzzy set find the primary contribution value of the centroid of the embedded fuzzy set  $a(\theta_1, \dots, \theta_N)$  as:

$$a(\theta_1, \dots, \theta_N) = \frac{\sum_{i=1}^N x_i \theta_i}{\sum_{i=1}^N \theta_i} \quad (\text{B.58})$$



**Step 5:** For each embedded FS calculate the minimum secondary contribution  $b(\theta_1, \dots, \theta_N)$  as:

$$b(\theta_1, \dots, \theta_N) = f_{x_1}(\theta_1) \wedge \dots \wedge f_{x_N}(\theta_N) \quad (\text{B.59})$$

**Step 6:** Pair the computed domain value from **Step 4** with the secondary contribution computed in **Step 5**.

**Step 7:** For each unique domain value, select the maximum between primary and secondary contribution.

#### B.2.2.2: Center of Sets Type-Reduction

As mentioned previously, the exhaustive type-reduction is not really practical due to the prohibitively large number of embedded fuzzy sets that needs to be evaluated. In order to avoid this issue, this dissertation uses the center of sets type reduction outlined in [Hagras 04] as a way to approximate the exhaustive type-reduction method while also being able to provide for real-time operation. Here, only a subset of embedded bounded interval fuzzy sets with specified cardinality is chosen from the set of all possible embedded fuzzy sets. The resulting centroid is then computed by averaging the leftmost and rightmost centroids for each fuzzy set.

Let  $Y_{cos}(x)_k$  be the  $k^{\text{th}}$  interval set in domain  $X$  bounded on its left by  $y_{lk}$  and on its right by  $y_{rk}$  and let  $M$  be the number of rules. Discretize each fuzzy set in  $Z$  points,  $y_1, \dots, y_n$  where  $z = (1 \dots Z)$ . Let  $J_{y_z} \equiv [L_z, R_z]$ , the combined left and right membership values. Let  $y_k^i$  correspond to the centroid of the Type-2 interval consequent set  $\tilde{G}_k^i$  of the  $i^{\text{th}}$  rule for the  $k^{\text{th}}$  output. Furthermore,  $y_k^i$  is a Type-1 interval fuzzy set, bounded by its leftmost point  $y_{lk}^i$  and rightmost point  $y_{rk}^i$  which is calculated as follows:

$$y_k^t = [y_{lk}^t, y_{rk}^t] = \int_{\theta_1 \in J_{y_1}} \dots \int_{\theta_z \in J_{y_z}} \mathbf{1} / \frac{\sum_{z=1}^Z y_z \theta_z}{\sum_{z=1}^Z \theta_z} \quad (\text{B.60})$$

Let  $f^i$  be the contribution, or firing strength of the  $i^{\text{th}}$  rule which consists of its leftmost firing strength  $\underline{f}^i$  and rightmost firing strength  $\bar{f}^i$ .

The center of sets type reduction,  $Y_{cos}$  is now expressed by the following iterative equation:

$$Y_{cos}(x)_k = [y_{lk}, y_{rk}] = \int_{y_k^1 \in [y_{lk}^1, y_{rk}^1]} \dots \int_{y_k^M \in [y_{lk}^M, y_{rk}^M]} \int_{f^1 \in [\underline{f}^1, \bar{f}^1]} \dots \int_{f^M \in [\underline{f}^M, \bar{f}^M]} \mathbf{1} / \frac{\sum_{i=1}^M f^i y_k^i}{\sum_{i=1}^M f^i} \quad (\text{B.61})$$

Defuzzification occurs by taking the average of the right and left centroid.

$$\frac{y_{lk} + y_{rk}}{2} \quad (\text{B.62})$$

### B.3: Nonstationary Fuzzy Sets

Nonstationary Fuzzy Sets (Fuzzy NFS) introduces the notion of variability of fuzzy sets over some dimension such as time, location, or even noise [Garibaldi 08]. Take for example an airplane taking passengers from to various points in the United States. It has a fuzzy thermostat. However, the airline 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, making heavy clothing the norm, an expected fuzzy representation of “Hot”, “Warm”, and “Cold” might look like Figure 123.

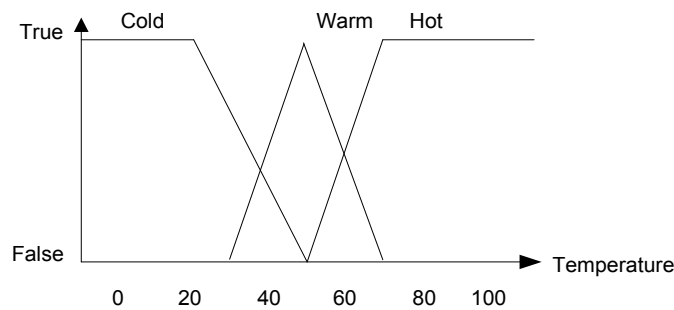
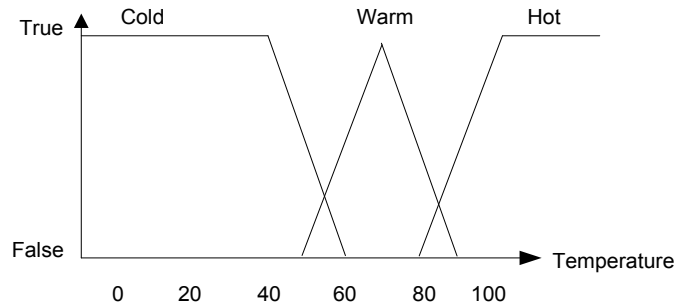


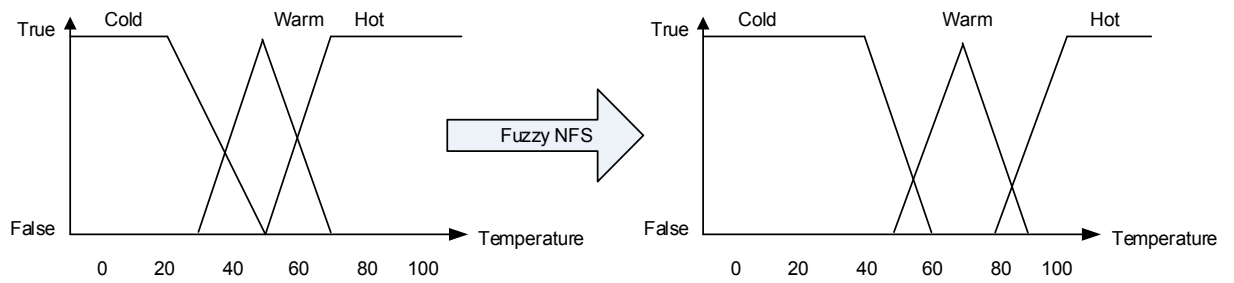
Figure 123 - Fuzzy sets for a thermostat in Barrow, Alaska

However, an expert who happens to live in Phoenix, Arizona, where temperatures tend to remain above freezing for long periods, making shorts, tank-tops and flip-flops preferred attire, might decide that a fuzzy temperature gage should use a representation like Figure 124.



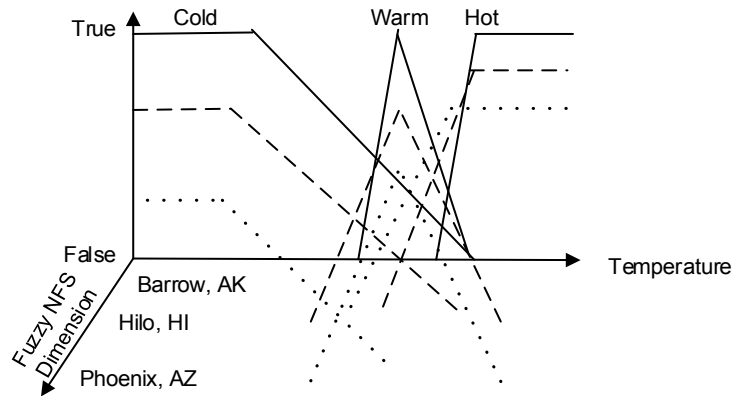
**Figure 124 - Fuzzy sets for a thermostat in Phoenix**

So how does the airline accommodate the differing definitions of the fuzzy terms to make everybody comfortable? 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. Since the airplane will be in the air a long time flying smoothly between destinations, there should be an equally smooth transformation of fuzzy sets. Nonstationary Fuzzy Sets (Fuzzy NFS) allows the resulting Fuzzy Type 1 definitions the flexibility to change according to requirements, but remain internally consistent at a higher level within the overall Fuzzy Inference System (FIS). In other words, the fuzzy sets change depending upon the value of the external dimension, but the fuzzy variables and rules remain unaffected. The goal then, is to figure out an appropriate transformation function along the dimension; an example of which is shown in figure 125.



**Figure 125 - A Fuzzy NFS transition of Fuzzy Type 1 sets**

Example external dimensions are space (location) and time. Figure 126 demonstrates how such a dimension, the location between cities Barrow, Alaska and Phoenix, Arizona with a quick stopover in Hilo, Hawaii (all in the name of science of course), would look. The transitions provided by the Fuzzy NFS would take effect as the plane moved along its course, transforming the fuzzy sets over time. The transformations would appear something like those demonstrated in figure 126.



**Figure 126 - Fuzzy NFS dimension of uncertainty added to Fuzzy 1**

The Fuzzy NFS  $\hat{A}$  is described as

$$\hat{A} = \int_{t \in T} \int_{x \in X} \mu_{\hat{A}}(t, x) / x / t \tag{B.63}$$

where  $t$  is some value along a dimension of the problem domain  $D$  and  $x$  is a tuple or point within the set of possible inputs  $X$ . In order to maintain a consistent relationship between tradition and nonstationary fuzzy sets, an additional constraint is required for  $\mu_{\hat{A}}$ . First consider that  $\mu_{\hat{A}}(x)$  is expressible as

$$\mu_{\hat{A}}(x) = \mu_A(x, p_1(t), \dots, p_m(t)) \quad (\text{B.64})$$

where  $p_i(t) = p_i + k_i f_i(t)$  and  $i = 1, \dots, m$ . This ensures that each parameter is varied along the dimension by a perturbation function. Hence, for a given traditional fuzzy set  $A$  and a set of dimension points  $T$ , the nonstationary fuzzy set  $\hat{A}$  is really a set of duplicates of  $A$  varied over the dimension (e.g. time).

Nonstationary Fuzzy Sets provides the dynamic fuzzy membership function transform, or perturbation function (and resulting sets) able to accommodate significant changes to the problem space over that dimension. The perturbation function is simply responsible for adjusting the underlying membership functions as needs change. The Fuzzy NFS is then able to generate a variable FIS to handle changes in the problem space which otherwise might cause difficulties to a static Type-1 or Type-2 FIS.

In some sense, Fuzzy NFS and Type-2 look fairly similar. Both generate three-dimensional fuzzy surfaces over the domain but whereas at any point, Fuzzy NFS reduces to a traditional Type-1 FIS, Type-2 implementations still maintain the extra fuzziness within the footprint of uncertainty.

#### B.4: Polymorphic Fuzzy Signatures

Polymorphic Fuzzy Signatures (PFS) describe a multidimensional fuzzy tree of fuzzy sets where each leaf contains a specific fuzzy membership function. Fuzzification occurs by traversing the branches whose meta-fuzzy signature indicates membership and combining the applicable membership functions at the leaves using traditional fuzzy functions such as *max* and *min*. The polymorphic fuzzy signature is described as:

$$\mu_{sig}: X \rightarrow [c_i]_{i=1}^k (\equiv \prod_{i=1}^k c_i) \quad (\text{B.65})$$

$$\text{where } c_i = \begin{cases} [c_{ij}]_{j=1}^k; & \text{if branch } (k_i > 1) \\ [0,1]; & \text{if leaf} \end{cases} \quad (\text{B.66})$$

Polymorphic fuzzy signatures allow one to break down an SDP into smaller, easier to describe, components. Each component is then associated with a particular FIS and signature. Each FIS output is fuzzified, with the resulting defuzzification using a traditional methods.

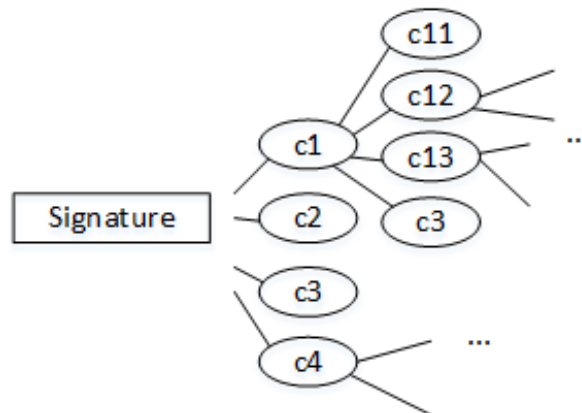


Figure 127 - Polymorphic Fuzzy Signature Tree

## B.5: Hybrid Fuzzy Systems

Hybrid Algorithms, discussed in chapter 2 are algorithms such that “there is a choice at a high-level between at least two distinct algorithms, each of which could solve the same problem” (Dat Cung, et al., 2006). Algorithms can be oblivious or independent of inputs or state properties, or they can be responsive to them either by self-tuning or engineered according to the parameters of the systems.

An example of a hybrid algorithm is the Memetic Mutation algorithm discussed in chapter 4, combining a Genetic Mutation algorithm with a Hill Climb. Another is the DDO-Simulated Annealing introduced in Minor Contribution #3, which combines a modified Simulated Annealing with a brute-force specialized search.

Hybrid Systems are “a class of complex dynamic systems composed of continuous variables and discrete events that mix and interact with each other” [Cheng 10] Many automated systems fall into this category and are well-served by hybrid algorithms. Because of the complexity of such systems, Fuzzy Logic has proven useful as a technique to incorporate all the various inputs and deliver a stable and predictable output without having to resort to a complex polynomial equation.

Hybrid fuzzy systems often combine fuzzy logic with non-fuzzy processes, such as neural networks [Gomathi 10] or optimization techniques like the genetic algorithm described in [Cox 05] or the memetic algorithm demonstrated in chapter 4 of this dissertation. Hybrid fuzzy systems can also combine disparate fuzzy inference systems [Mendis 10] into a weighted composite to address Situational Discontinuity problems whose sub-components overlap.

## B.6: Fuzzy Clustering

Clustering is a data mining classification technique used to group sets of objects which are similar [Han 11]. The goal for a clustering algorithm is to define attributes to describe a grouping or “cluster”. It does this by first analyzing a set of points and then separating them into specific groups. Each group of points is related by the similarities among their respective attributes. For instance, suppose you had a group of points that represented sales records. Each record had certain demographic information:

1. Buyer’s location (city, state)
2. Age
3. Product

In this simple example, a *point* is simply the tuple (location, age, product). Some possible attributes might be:

- Age in range of 20-30 years
- Location within city of Seattle, WA
- Product is box of Cheerios

A cluster with these three attributes will have a certain number of points. Giving the cluster a name such as *Young adult Cheerio lovers in Seattle* distinguishes it from other clusters. Any point with these three attributes will belong to *Young adult Cheerio lovers in Seattle* while any points with one or more attributes that differ will belong to another cluster.

There are a number of clustering techniques and among them is Fuzzy Clustering, also known as Fuzzy C-means or Fuzzy K-means. The advantage of Fuzzy Clustering is the ability to assign to a given point membership in multiple clusters. For example, suppose one of the sale records above had the following attributes:

Location: Bellevue (just east of Seattle)

Age: 19 years

Product: Cheerios

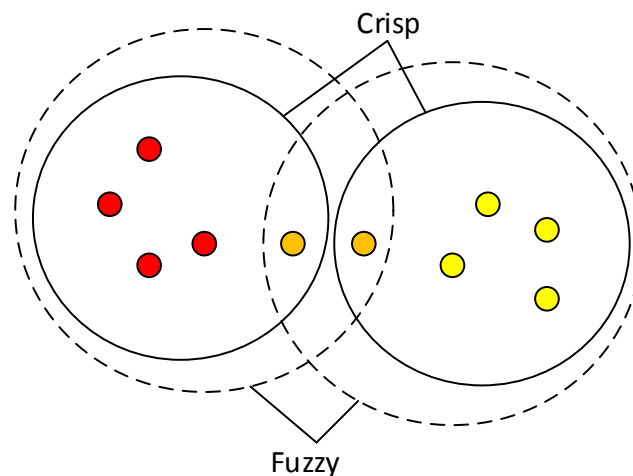
In a traditional K-means algorithms, such a point would not be classified as belonging to *Young adult Cheerio lovers in Seattle* despite the relative closeness of the three attributes.



In order to make the association, the cluster would have to be “expanded” to include the external point, possibly diluting the original definition. This is not necessarily desirable either since *Young adult Cheerio lovers in Seattle* doesn’t for instance, intend to include the nearby city of Bellevue.

So what to do? It would be useful to be able to relax the cluster boundaries in order to, in some fashion, include nearby points while still retaining the integrity of the cluster. By the same token, it would be useful for a point that doesn’t quite fit within a particular cluster to be able to associate to some degree with nearby clusters. Fuzzy clustering allows this by creating fuzzy cluster boundaries.

In general, clustering techniques seek to create clusters with attributes such that they can assign every object or point to an individual cluster. In some clustering techniques, such as K-means, clusters have a centroid and a radius and points that belong to the cluster. These points lie within the radius “distance” of the centroid. Similar to crisp logic, any individual point is either a member or NOT a member of any particular cluster. Fuzzy C-means clustering, however, using fuzzy boundaries, is able to relax the crisp rules of K-means to allow points to belong to multiple clusters [Hung 11]. The difference between the two approaches is illustrated by figure 128.



**Figure 128 – Contrasting K-means and Fuzzy C-means clusters**

Note the 2 points in orange (or light grey) between the two other clusters of points, 1 on the right and one on the left. The orange points themselves seem to *some* have

characteristics of both clusters without really belonging to either. They could be new data or possibly noisy data. A K-means classifier could extend the cluster circles to encapsulate the orange points but that would mean altering the original integrity of the classification for data which might simply be noise. Another alternative could be to create a whole new cluster in order to maintain the original classification but that might be equally undesirable since processing would have to include the new clusters with dubious characteristics.

Fuzzy C-means takes a different approach. While it is similar to K-means in that they both create clusters with centroids and a radius and the sum of their memberships adds up to 1; points in fuzzy C-means have membership in the range of 0 to 1 rather than just 0 or 1 [Gauge 11]. This gives a points in fuzzy C-means more flexibility in their associations as well as better handling of outliers and noise [Tsai 11].

In the case of figure 128, each of the orange points has a certain association with both clusters to the right and left, but not so much that the original clusters need change. We can still distinguish between the points, however, in that the right orange point has *more* in common with the cluster on the right, similarly for the orange point on the left. In the case of the Bellevue purchase, while it is not completely within *Young adult Cheerio lovers in Seattle*, it is “close” enough that it might be wise to include it in certain situations (such as a Cheerios discount mailer to the Seattle “area”).

Fuzzy C-means techniques have been used in hybrid algorithms to handle the switching and contributions of the appropriate processes dynamically [Palm 98]. This is because fuzzy metrics provide a useful way to determine how “relevant” an algorithm is at a particular point. This idea is also the basis for Fuzzy Contexts, particularly when problem spaces overlap.

Key to creating fuzzy c-means clusters is the “fuzziness” factor, a constant, greater than 1 (1 being the hard K-means number), which determines the fuzziness of the clusters. The higher the number, the greater the number of clusters any individual point may belong. Fuzzy C-means seeks to create clusters similar to K-means by minimizing an objective function:

$$J_m = \sum_{i=1}^N \sum_{j=1}^C u_{ij}^m \|x_i - c_j\|^2, 1 \leq m < \infty \quad (\text{B.67})$$

where  $m$  is any real number greater than 1. This is called the fuzziness factor and determines how “relaxed” the cluster boundaries become. In other words, the greater the  $m$ , the more clusters to which any individual point can belong. When  $m = 1$ , fuzziness disappears and the algorithm matches K-means. The membership function  $u_{ij}^m$  is an element from a membership table and indicates the degree of membership of a particular point  $x_i$  in given cluster  $j$  with a centroid  $c_j$ .

One element of the objective function is the distance equation  $\|x_i - c_j\|^2$  which is the Euclidean distance between the point  $x_i$  and the centroid  $c_j$  defined by the equation

$$\|x_i - c_j\|^2 = \sqrt{\sum_{i=1}^n (x_i - c_j)^2} \quad (\text{B.68})$$

The centroid  $c_j$  of the  $j^{\text{th}}$  cluster itself is calculated by the equation

$$c_j = \frac{\sum_{i=1}^n \mu_{ij}^m x_i}{\sum_{i=1}^n \mu_{ik}^m} \quad (\text{B.69})$$

Finally the fuzzy membership table is calculated by applying the following equation

$$\mu_{ij} = \frac{1}{\sum_{k=1}^n \left( \frac{|x_i - c_j|}{|x_i - c_l|} \right)^{\frac{2}{m-1}}} \quad (\text{B.70})$$

Pseudo-code for the Fuzzy C-means follows

### Pseudo-code 15 – Fuzzy C-means algorithm

**Algorithm:** Calculate Fuzzy C-means(points[] p, clusters[] c, numClusters c, maxIterations)

**Input:** *p* an array of points to cluster

*c* an array of clusters

*numClusters* how many clusters to create

*maxIterations* maximum number of iterations before quitting

**Output:**

**Begin**

```

1  IF NOT Initialized
2    set c = an array of random clusters
3    FOR EACH cluster in c
4      set a random centroid for cluster
5    NEXT cluster
6  END IF

7  DO
8    Calculate objectiveValue to get the initial value
9    move centroid centers if necessary
10   calculate membership matrix, new centers using CalcMatrixandNewCenters
11   IF Abs(objectiveValue – previousObjectiveValue) <= min error) THEN exit LOOP
12  ELSE
13    set previousObjectValue = objectiveValue
14  END IF
15  increment iterations
16  WHILE iterations <= maxIterations

```

**End**

**Algorithm:** CalcMatrixandNewCenters (points[] p, clusters[] c)

**Inputs:** *p* an array of points

*c* an array of clusters

**Outputs:** a new membership matrix and array of clusters

**Begin**

```

1  FOR i = 0 to count of c
2    FOR j = 0 to count of p
3      MembershipMatrix[i, j] = CalculateDistanceToAllClusters(p[j])
4    NEXT j
5  NEXT i

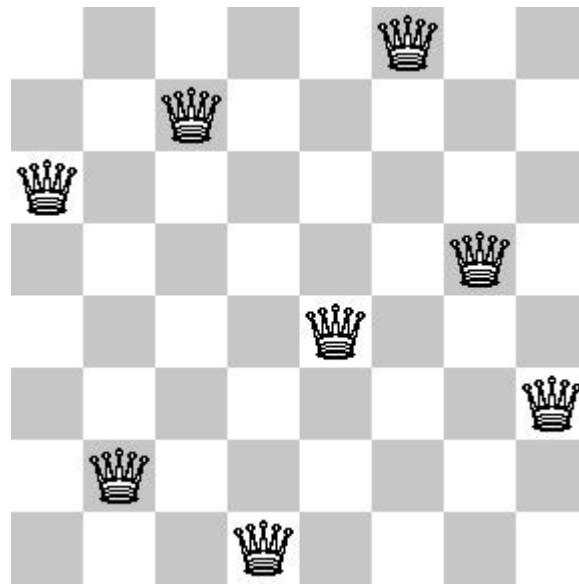
6  normalize cluster membership values to ensure total membership for any point is equal to
1

```

**End**

## APPENDIX C: LOCAL SEARCH TECHNIQUES

Consider a simple 8x8 chessboard with 8 queens placed randomly on the board. A program is tasked to arrange the queens on the board so that no queen is able to attack any other queen. This is the classical N-Queens Problem. For a very small chessboard, a program could easily test all possible configurations until it found one where there were no conflicts.



**Figure 129 – Queens Arranged on Chessboard with no Conflicts**

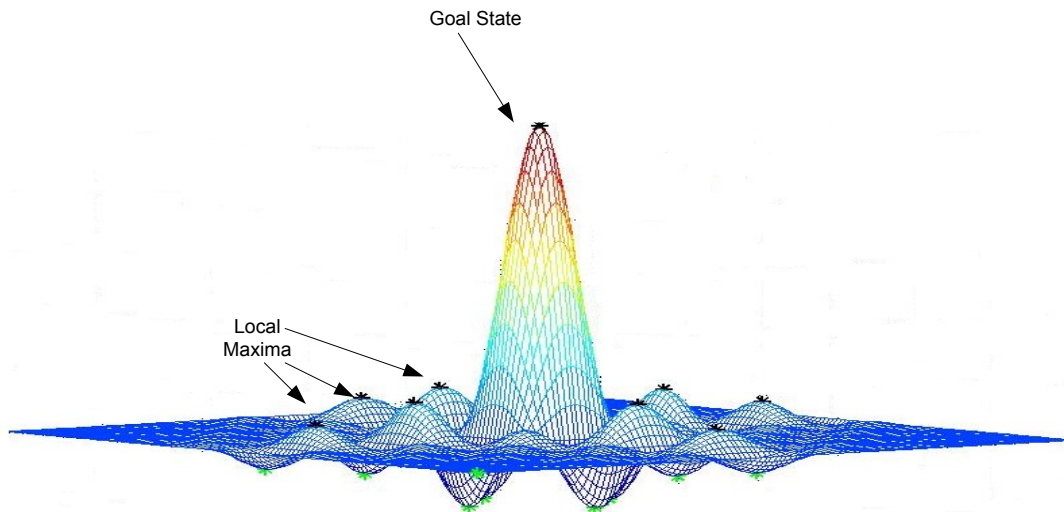
However, even for a moderate-sized board like the one in figure 129, the total number of possibilities is quite large:  $C_8^{64}$ , or about 4.4 billion possible arrangements but only 92 solutions [EQP 14]. The N-Queens problem is considered to be a nondeterministic polynomial (NP) class problem [Martinjak 07]. This means the problem has a computational complexity sufficiently high (such as  $O(2^n)$  or  $O(n!)$ ) such that brute force or deterministic methods are not really practical. These kinds of problems can be solved using *heuristic* methods instead [Russell 09]. A heuristic method is a procedure is guaranteed to find a *good* solution (if one exists) in a reasonable amount of time, although there is no guarantee it will be the *best* solution [Marshall 11]. What the approach gains in efficiency, it sacrifices in

completeness, however; with NP-class problems they are often the most *practical* approach to finding *a* solution.

The heuristic approach assumes little to no knowledge of the problem space beyond its current state and how to compare that to other possible states. Hence, virtually the entire problem space is unknown. When investigating an unknown state space, a software agent must engage in some form of search.

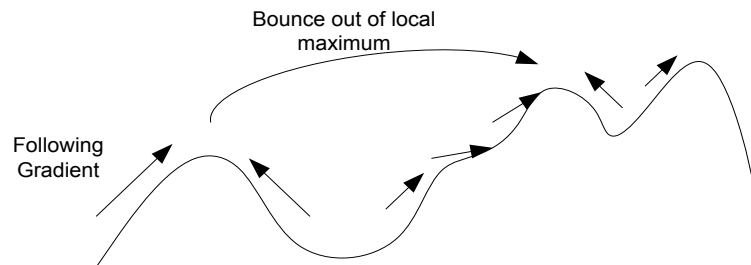
Brute-force or other comprehensive search simply looks at all of the possible permutations and selects the best one. For a heuristic search, an algorithm chooses (usually) a random point and simply begins “looking around”. Consider an ant searching for food; its tiny brain doesn’t have any concept of refrigerators, bread baskets, kitchens, dirty dishes, trash cans or any other place food might exist, it simply wanders around hoping to find food, *any* food. Likewise a heuristic search “wanders” around in search of a solution, *any* solution. In such cases a solution may still be found using only a “tiny brain”, i.e. limited computing resources or time. There are a class of heuristic search techniques where the search is limited to the local *neighborhood* around the process. These search techniques called Local Search [Russell 09].

So 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 [Martinjak 07].



**Figure 130 - 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 previously proved unfruitful [Zheng 06], [Marshall 11].

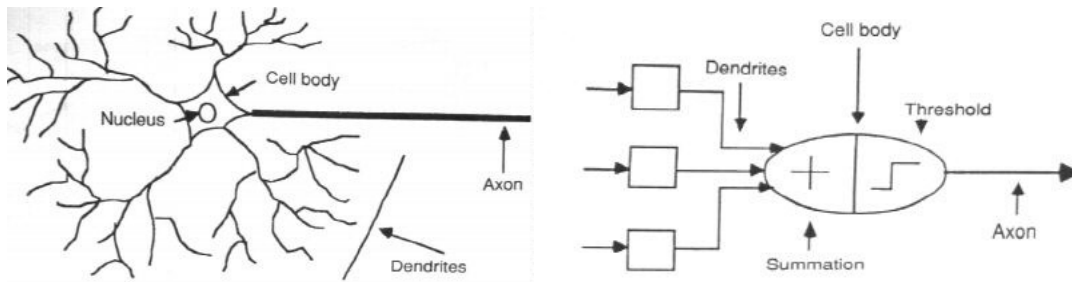


**Figure 131 - Local Search following gradient and “bounce” out of local maximum.**

Local search algorithms can be combined with additional heuristics to improve their effectiveness for a given search problem. Specific local search algorithms are discussed more fully with pseudo code examples in appendix H.

## APPENDIX D: NEURAL NETWORKS

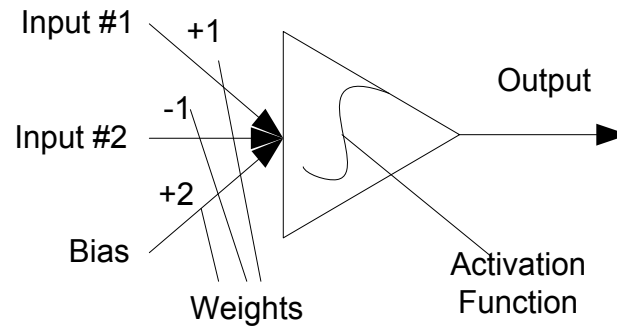
Neural networks arose from a study of biological neural systems, although only superficial similarities exist [Schalkoff 97]. The concept of a neural network centers on a single unit that receives input from one of more sources. The unit is called a neuron and a collection of neurons action in concert is called a neural network. 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 132.



**Figure 132 - A biological neuron and its computer-based equivalent.**

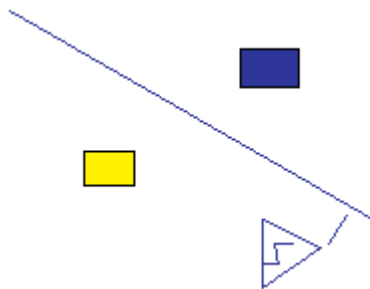
The input consists of a value along with "weights" and in the artificial neuron is combined with all other inputs and a bias. The total is then processed by the activation function, which outputs one of two values along the Axon, or output. This new model is shown in figure 133.





**Figure 133 - A simple artificial neuron**

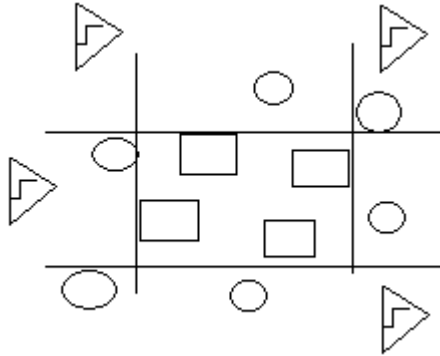
The output values allow a neuron to “classify” inputs into one of two categories. The classification allows a single neuron to distinguish characteristics between points in  $n$ -dimension space. By adjusting the weights on the individual inputs and bias, the neuron can “learn” to behave in a given fashion,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



**Figure 134 - 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 135.



**Figure 135 - 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 Networks 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 [Zurada 92].

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 11]. 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.

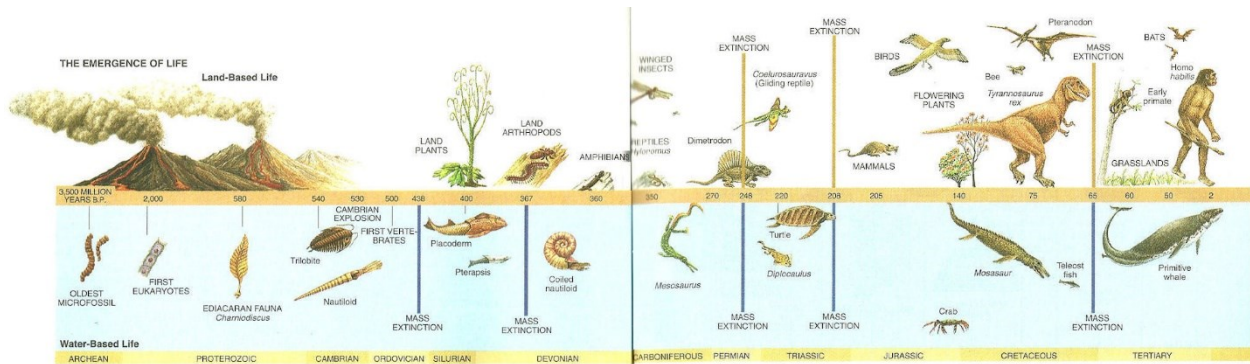
Neural networks are used in many commercial applications from image, character and voice recognition to medical diagnosis, stock market prediction and data mining [Yu 06], [Han 11].

## APPENDIX E: EVOLUTIONARY COMPUTATION

### E.1: Introduction

Evolution is defined as “the gradual development of something”. For example Cosmologists generally believe the universe “evolved”, first from an ultra-dense singularity to an expanding ultra-hot ball of energy to a giant cloud of hydrogen and finally to galaxies, stars, planets and the stuff that makes up our current universe. In the biological realm, evolution also means gradual development, but in an adaptive sense whereby less-competitive forms gradually evolve into more competitive forms within their environment.

At stake is survival itself. Each species must contend for limited resources in order to produce offspring and have those offspring survive long enough to reproduce and so on. Those that are successful, by and large, are the ones that are able to adapt in order to better meet a goal such as finding dinner or avoiding becoming dinner. However, because of the huge diversity of environments, both currently on the earth and over time as conditions change, there is a correspondingly huge diversity of living things. One such view of evolution is depicted below:



**Figure 136 – A View of Evolution**

The reason for this great biodiversity is the goals of different species affect how they evolve. When life first arose, it is believed the atmosphere had little oxygen but the seas were full of nutrients so life depended upon making best use of those nutrients. That was all well and good but the earth was also being bathed in sunlight so a new form of bacteria arose which was able to convert sunlight into food. This proved a huge competitive advantage for

that species and had the side effect of pumping huge amounts of oxygen into the atmosphere. New life forms arose to use this oxygen. Oxygen converted to ozone in the upper atmosphere which reduced harmful ultraviolet rays from the sun. Land was now available for use and so life evolved to use it. And so it goes. Opportunity arises in the form of a new resource (sunlight, oxygen, land, etc.) and species adapt to use it their advantage.

This is also the flipside to this. Hazards in the form of volcanic activity, gamma ray bursts, asteroids, ice ages and other catastrophes have come along and wiped out most species in a series of mass extinctions. In each case it was one or more constraints that arose requiring adaptation. Those species able to adapt eventually thrived while those species unable to adapt quickly enough passed away with the fossil record the only evidence of their existence.

Now consider a difficult computational problem with a goal and potentially many ways to achieve this goal. The problem can take the form of an opportunity, like a new type of sensor on a robot, or a constraint, such as operating on the surface of Mars, where communication is delayed. An algorithm, such as a Fuzzy Inference System (FIS), tailored to perform well in a slightly different set of opportunities/constraints now under-performs or fails when faced with this new challenge. Can the FIS be adapted in some way?

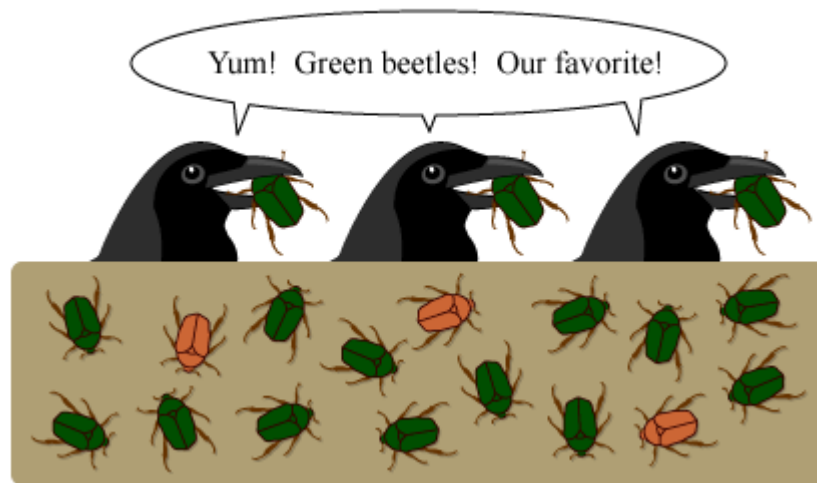
The answer, of course, is yes, but how? Brute-force methods only work when the number of possible permutations is small and heuristics may not adequately account for the changes that come along - so how is it possible to successfully adapt when the known information is so small and the possibilities so large? Biological organisms faced, and overcame, these same challenges countless times over the eons. Evolutionary Computation is a branch of computer science that tries to consider these kinds of problems in a similar way as did biological creatures in the past and understand how biological success came about. By emulating the adaptive methods of biology, Evolutionary Computation also hopes to emulate their success. Many tests of these adaptive methods over the years have shown this indeed to be the case.

So how do biological creatures adapt? There are three primary ways: 1) Natural Selection, 2) Recombination and 3) Mutation.

## E.2: Natural Selection

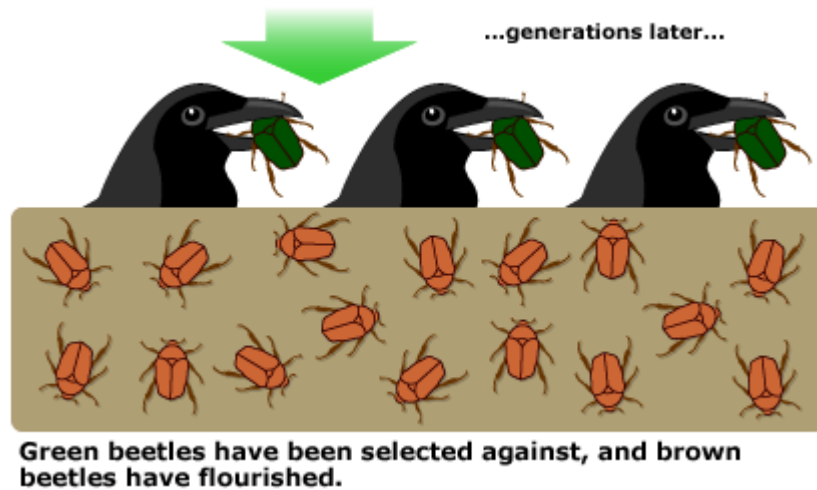
Natural Selection occurs in a competition for resources. In Darwin’s “Survival of the Fittest” theory, the species best able to compete for resources are those “most fit” in that competition. Most fit can be defined in many ways from the largest and strongest to the smartest or most nimble. Even the ability to simply prefer “what’s out there” often qualifies as most fit as predators who prefer the most available food are most likely to succeed as illustrated by figure 137.

### Natural selection, in a nutshell:



**Figure 137 – Predators adapt to a preponderance of green beetles**

On the other hand, the prospective prey may adapt as well as shown in figure 138. In each case, predator and prey strive to survive and adapt without actually consciously knowing what the goal is or how to get there.



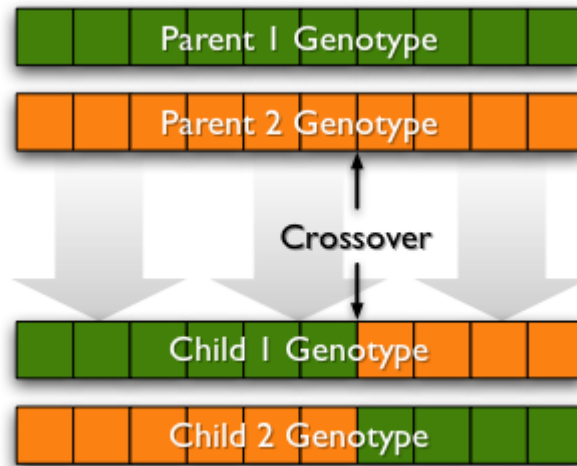
**Figure 138 – Beetles adapt to Predators Preferences by changing color**

Courtesy of University of California Museum of Paleontology's Understanding Evolution (<http://evolution.berkeley.edu>).

For an algorithm, determining “most fit” requires a fitness function to test it against competing algorithms in order to determine which algorithm “survives”. Chapter 4 describes fitness functions in some detail.

### E.3: Recombination

Recombination, also called Cross-Breeding or Crossover Mutation, is the process whereby parents get together and produce offspring. In the progeny, traits from both parents are recombined in different ways to produce a new set of traits reflecting both parents yet distinct on its own.



**Figure 139 – Recombination of Genotypes**

Courtesy of the University of Sydney

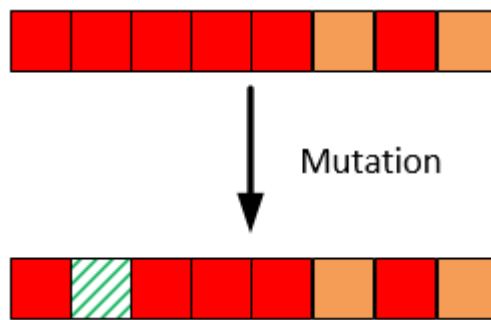
Among the purposes it serves is as a self-correcting mechanism in that it helps to prevent defective genes from being passed down as would be the case if a child were “cloned” from the parent. These “defects” are often recessive and get masked by the traits of the (presumably) dominant parent with the good gene. Also, this new set of traits helps to promote species diversity, allowing for competition and natural selection to be an ongoing process from generation to generation.

In Evolutionary Computation, recombination occurs by taking the configurations of two algorithms (the parent genotypes) and randomly or pseudo-randomly picking from both parents which configuration components to combine. This dissertation discusses the 8-Queens problem whereby chessboard configurations represent the genotypes. Recombination occurs by taking elements of two configurations (a winner and loser) and combining them to produce a new configuration that potentially can achieve a goal. Appendix A, Minor Contribution #3 describes the process in some detail and Appendix H provides the pseudo-code.

#### E.4: Mutation

Mutation is the final leg of the evolutionary stool. The history of biological evolution is replete with examples new traits introduced which are not possible through natural selection or recombination, such as eyes, vertebrae and sweat glands. Such a drastic change requires a more drastic technique.

While the cause of a biological mutation involves broken DNA, the mutation of an algorithm involves randomizing one of more of its traits as demonstrated in figure 140.



**Figure 140 – Genetic Mutation**

Mutation allows for dramatically different looks at an algorithm. While most mutations fail, the variability they introduce allows for exploration that is not possible via natural selection or recombination.

This dissertation uses Evolutionary Computation in the form of a genetic algorithm, memetic algorithm and a variation of the genetic algorithm in chapter 4 and Appendix A, Minor Contribution #3.



## APPENDIX F: RELATIONAL DATABASE CONCEPTS

### F.1: Introduction

Databases have become an indispensable component in almost all computerized systems. This has resulted in the development of ever more sophisticated tools that deliver a reliable and flexible tool set. While there has been some recent press talking about the negative impact of some databases, this is the result of unethical use of these tools, not the tools themselves [Johnson 13]. Databases and database software remains a valuable tool in the software development process.

As the use of databases has become ubiquitous there have been many theories as to what constitutes a database [Powell 05]. Consider the case of a company that, among other things, makes products, sells products, sends bills, etc. In the early day of computer automation each functionality would have been automated by different programmers. Each programmer would have defined their data requirements. Once those requirements were in place the data would be compiled in a flat file that was set up to meet the programmer's needs in the specific application. In extreme cases, each programmer would have assigned different identifiers to similar data (e.g. there would not have been one consistent product number or customer number) because they may not have known about other data sources, might not have had access to the other data sources or simply decided to do it that way.

In this type of environment, simply maintaining the data was a full time job. When there was a need for change to the data structure, a major effort was required. Consider what it might take to add one data element to the above environment:

- Data files would need to have the field added to their structure
- Programs would have to be written, tested and run to modify the file structure and then the data
- The question would remain “did we get it all”?

There was a clear need for a simpler, more manageable, cost efficient and secure way to manage data. A modern database is a resource to the entire enterprise. At a high level you can think of it like a central phone system:

- It is available to many different users
- It facilitates different users and applications
- It is centrally managed and maintained
- Access is safe and secure
- The user community has access only to appropriate functions
- A user expects that it will always be available and operate reliably

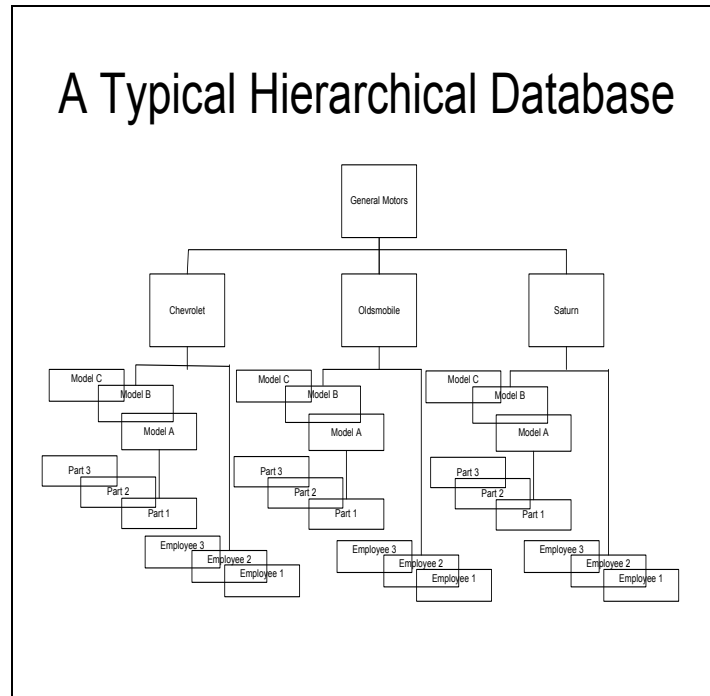
Hence in a functioning database:

- The data will be managed by the database team and used by the programmers
- Each customer, product, etc. would have only one identifier
- The programmers will have access to common data
- If a field needs to be added, the database team can do it very quickly. Once added the programmers can easily update the data

The database is a collection of data. In order to utilize the data a database management system (DBMS) is needed. Early DBMSs were very crude and difficult to use. They typically required a high degree of programming skill. These have evolved to modern systems (like SQL Server) that have visual tools and provide the database team with a rich and diverse tool set that allows for fast and easy access to the database to ensure optimal performance.

Most of the current DBMS product offerings go far beyond just managing the database. Tools exist for producing reports, analyzing data and even presenting different ways of viewing and analyzing the data.

In the beginning it wasn't that way. Early hierarchical databases addressed the issue of centralizing data, but they were difficult to work with as the programmer was forced to literally go through the hierarchy one record at a time.



**Figure 141 – A Hierarchical Database Design**

Consider the question where do we use a one inch screw, part number 12345? A programmer would need to write a program:

- Start at General Motors
- Move to the next level (Chevrolet)
- Within Chevrolet go through each model
- Within each model go through each part
- When the program finds part 12345 “note” which model of which brand

This is clearly a tedious process. Further, there is no guarantee that part 12345 will be the same across all branches of the hierarchy.

Another database came along called the Network Database. Networked databases are a type of database system, not to be confused with the relatively recent ability of some DBMSs to work across a network. A networked database contains a network of embedded pointers that allow for very rapid access to related data. While the network of pointers makes retrieving data very efficient, it comes with a very high cost in terms of maintaining that

network. For this reason networked databases are best used in scenarios where there are high demands for fast response times but limited update and maintenance requirements.

The relational model was proposed by E.F. Codd of IBM [Date 11]. Under this model the amount of data being stored is minimized to enable processing efficiency and maximize data storage resources.

The heart of a relational database is the Structured Query Language or SQL. SQL is a fourth generation language which means that commands tell what is to be done, not how to do it. One of SQL's major strengths is its ability to pull together (join) data from several tables. This capability allows us to store only foreign keys and include more details only when needed.

Referring back to the example from General Motors, we can instead create tables for cars and parts. The new structure doesn't need to be more than one level deep and much easier to read. Foreign keys in the various tables allow us to reliably refer to data in other tables. We can ask SQL to retrieve all data for part number 12345 instead of doing it ourselves. Because we know the model ID we can ask SQL to join in data about the Model. If needed, data about the brand can also be included. A competent SQL programmer can build a query to get the necessary data in minutes as compared to a much more extensive retrieval process against the hierarchical model in the earlier example.

Almost all computer systems that employ saved data now use a database system to manage that data. In many of those situations the database model used is the relational one.

## F.2: The Relational Model

The relational database model is an efficient mechanism for general purpose databases. Under this model, data are stored in tables. Tables are defined in terms of columns which represent the specific data elements within the table. Examples of columns might include name, product number or grade. Within the table data are stored in rows. Each row has data in some or all of the columns.

There is a well-defined methodology for building a relational database. The methodology is called Normalization. Similar to other data compression techniques, the idea behind Normalization is to find redundancies in the data and replace them with small, reusable “codes”. These codes take the form of “keys”. Normalization presents another advantage for Online Transaction Processing (OLTP) systems in that the use of codes means that the source data for the code only has to be written once for the entire system to be updated. For instance, suppose a very successful salesperson attaches her name to her invoices. Each invoice in the system has an entry with her name, “Mary Smith”. If Mary gets married to Joe Stacevokokious and wants to change her name to Mary Smith- Stacevokokious she or some poor data entry clerk will have their hands full changing each of Mary’s sales records, correctly, so she can get her commission. Better would be to simply have a code for Mary Smith, say 25 and a single entry that says essentially: Mary Smith = 25. Each of her invoices would contain a code of 25 for the sales person. Then the clerk would simply have to change a single entry to: Mary Smith- Stacevokokious = 25 and Mary’s invoices will all change and she can rest easy about her commission.

In order to manage the data efficiently, a key needs to be defined within each table. The key is made up of one or more columns which uniquely identify that row. The key structure makes relational databases very efficient at data storage and retrieval because the database can work with the smaller keys instead of the larger data the key represents. As a result, it is not unusual to store data about a similar object in several tables. For instance, in an automotive database, information about cars might supply one table. Each of those cars have individual parts, which reside in another table. The *cars* table, contains keys, or part numbers linking part information and usage to the *parts* table. There are two types of keys, a *primary* key which is a unique value identifying an object, and a *foreign* key, which serves as

a reference to the object. Primary keys are stored in *domain* tables, designed to provide data about objects of a specific kind or type, such as cars, colors, addresses, etc. Foreign keys are used by *fact* tables which contain transaction data about objects such as customer purchases (of cars) or an inventory.

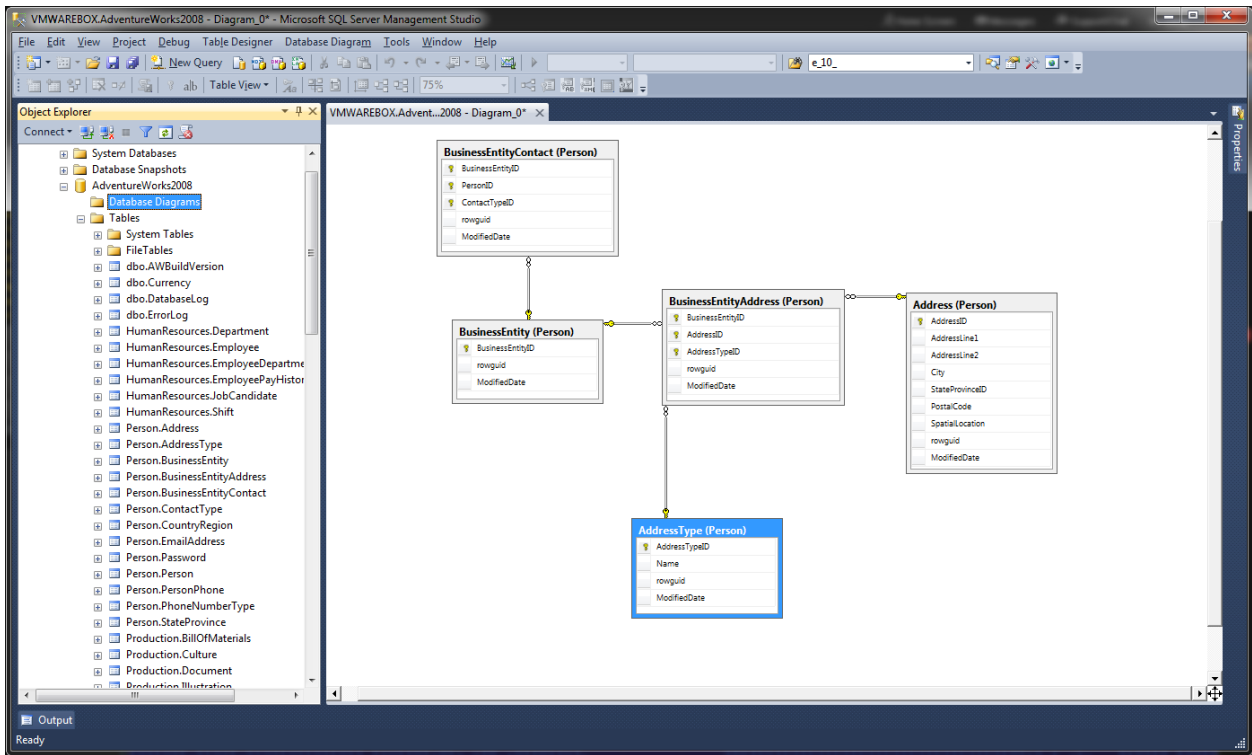
The foreign key enables the system to relate data in one (fact) table to data in another (domain) table. Using the foreign key also means redundancies are compressed into specific codes so the overall footprint of the data is smaller. This is because fact data tends to contain many rows, whereas domain data contains fewer rows. Think about a store selling cans of soup. There may be only a few dozen *types* (domain) of soup but many thousands of *purchases* (fact) of soup.

When data is required from two or more tables it is retrieved using a “join”. As the name implies, the join combines data from the specified tables by matching up data elements between them in a “relationship”. The easiest data elements to pair are key fields. Joins are an important component of the SQL SELECT query and part of the larger family SQL functions.

Cardinality is a term used to describe the relationships between tables and lies at the heart of performing joins. Cardinality helps a user or process to better understand the relationships between tables in order to create queries that will use them efficiently. There are three basic types of cardinality in relationships:

- One to One – for any record in the primary key table, there is one and only one record in the foreign key table. Think driver’s licenses. Each person (primary key) is only allowed to obtain one driver’s license (foreign key).
- One to Many – for any record in the primary key table, there can be one or more records in the foreign key table. Think credit cards. One person can have multiple credit cards, but each credit card belongs to one person only.
- Many to Many – any record in the primary key table can be associated to multiple records in the foreign key table and vice versa. Think bank accounts. Any one person can open up multiple accounts at a bank and each bank account can have multiple signers.

Relational databases require design and construction. Fortunately most commercial RDBMS's come with tools to aid this task. SQL Server Management Studio, which comes bundled with SQL Server 2012 is once such tool.



**Figure 142 – SQL Server Management Studio**

### F.3: Codd's Rules for Relational Databases

All commercial relational databases must conform to a large extent, if not completely, to E.F. Codd's rules for relational databases [Oracle 10], [Date 11], [Powell 05].

**Rule 1: The Information Rule:** *All information in a relational database including table names and column names are represented by values in tables.*

When Codd did his initial work on relational databases, most databases were built using hierarchical tools [Powell 05]. Each database had its own language and own way to storing metadata. The goal here was to have a uniform way to lookup information about

tables and column in order to know how to structure output. The database tool in this dissertation, SQL Server conforms to this rule by creating system tables in each database. These system tables are designed to hold metadata about all the objects in the database. By querying the system tables, processes can get information about tables, columns, constraints, data types, dependencies and more. SQL Server provides internal processes and code to maintain system tables so users need not concern themselves unless they need specific metadata, in which case they can query them directly or indirectly via views or system stored procedures as demonstrated in figure 143.

The screenshot shows a SQL query window with the following text: `select * from sys.objects`. Below the query window is a results grid with the following columns: name, object\_id, principal\_id, schema\_id, parent\_object\_id, type, type\_desc, create\_date, and modify\_date. The grid contains 10 rows of data for system tables.

	name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date
1	sysrscols	3	NULL	4	0	S	SYSTEM_TABLE	2012-02-10 20:16:00.707	2012-02-10 20:16:00.713
2	sysrowsets	5	NULL	4	0	S	SYSTEM_TABLE	2009-04-13 12:59:11.093	2012-02-10 20:16:01.943
3	sysclones	6	NULL	4	0	S	SYSTEM_TABLE	2012-02-10 20:16:01.523	2012-02-10 20:16:01.530
4	sysallocunits	7	NULL	4	0	S	SYSTEM_TABLE	2009-04-13 12:59:11.077	2012-02-10 20:16:00.730
5	sysfiles1	8	NULL	4	0	S	SYSTEM_TABLE	2003-04-08 09:13:38.093	2003-04-08 09:13:38.093
6	sysseobjvalues	9	NULL	4	0	S	SYSTEM_TABLE	2012-02-10 20:16:02.070	2012-02-10 20:16:02.077
7	syspriorities	17	NULL	4	0	S	SYSTEM_TABLE	2012-02-10 20:16:01.007	2012-02-10 20:16:01.020
8	sysdbfrag	18	NULL	4	0	S	SYSTEM_TABLE	2012-02-10 20:16:01.910	2012-02-10 20:16:01.920
9	sysfgfrag	19	NULL	4	0	S	SYSTEM_TABLE	2012-02-10 20:16:00.647	2012-02-10 20:16:00.653
10	sysdbfiles	20	NULL	4	0	S	SYSTEM_TABLE	2012-02-10 20:16:01.387	2012-02-10 20:16:01.393

**Figure 143 – SQL Server 2012 System Tables**

**Rule 2: The Guaranteed Access Rule:** *Every piece of data in a relational database can be accessed by using a combination of a table name, and a primary key value that identifies the row and column name which identifies a cell.*

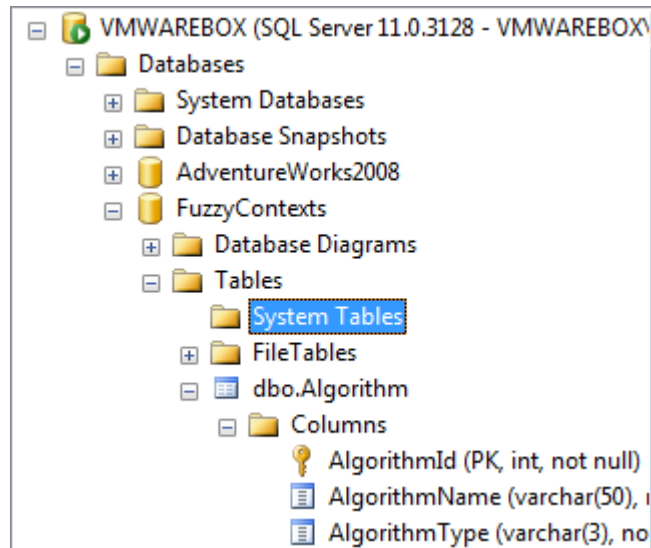
Critical in any database is the ability to find any specific piece of information stored. Since data is contained in tables, each table must provide the ability to label each row with a unique identifier. Normally referred to as the primary key, this identifier can be a single value (like an ID) or a composite value (such as and Customer/Purchase Date) but it must be unique in order to identify each row. SQL Server 2012 uses the primary key as an index pointer to a table row on a specific disk page.



Once a row is located, then SQL Server lays out the columns sequentially on disk in much the same way objects are laid out in memory. By calculating the amount of space used, based upon the column data type, the database engine knows how far to offset from the pointer to get to the beginning of the particular column value. In this way, all data elements are uniquely identified by the unique combination of database name, table name and column name. Column names are unique within the table and table names are unique within the database. Best practices require a unique primary key on tables which further reinforces this rule. SQL Server, like many other databases takes this a step further by the use of identifiers. Identifiers come in five parts:

1. Server name (implied as the current server if not specified)
2. Database name (implied as the current database if not specified)
3. Schema name (implied as current user if not specified, if not that, then dbo, otherwise query will fail)
4. Table name
5. Column name

Figure 144 shows a table whose primary key column has the 5-part designation VMWAREBOX.FuzzyContexts.dbo.Algorithm.AlgorithmId



**Figure 144 – 5 part identifier  
VMWAREBOX.FuzzyContexts.dbo.Algorithm.AlgorithmId**

**Rule 3: Systematic Treatment of Nulls Rule:** *The RDBMS handles records that have unknown or inapplicable values in a pre-defined fashion.*

The NULL is almost a rite of passage for SQL developers because it adds a frustrating twist to what is normally a fairly intuitive process. Because NULL is undefined, standard ANSI treatment of null values in SQL Server and other relational databases goes something like this:

- NULL is not equal to any regular value
- NULL is not equal to another NULL value
- NULL is not NOT equal to another NULL value

The first rule seems pretty straight-forward but the next two rules have caused errors in many a query. SQL Server provides a pair of functions: COALESCE and ISNULL to temporarily convert NULLs to values which can be compared, which also provide a specific test for NULL. Relational databases should allow for the possibility of null values and be able to

include/exclude these values in queries in some sort of manageable fashion as SQL Server does.

**Rule 4: Active Online Catalog Based on the Relational Model:** *The description of a database and its contents are in database tables and therefore must be able to be queried via the standard data manipulation language.*

This rule is saying that there must be an ability to find out things like

- What tables are in a database?
- What columns are in the various tables?
- What are the data types of the columns?
- Other related metadata such as dependencies, constraints, etc.

The database must make this information available via SQL. As mentioned earlier, SQL Server implements this rule by keeping all of this information in a set of system tables which can be queried in the same manner as user tables.

**Rule 5: Comprehensive Data Sublanguage Rule:** *A RDBMS may support multiple languages, but at least one of them should allow the user to do all of the following: define tables and views, query and update the data, set integrity constraints, set authorizations and define transactions.*

The database must have at least one language that enables creation of data definitions or objects (known as DDL), data manipulation (DML), data integrity, and transaction control. SQL or Standard Query Language has been the primary vehicle for implementing rule 5 in most traditional relational databases. SQL Server 2012 uses a SQL variant called Transact-SQL [Ben-Gan 06], Oracle databases use PL-SQL and other databases use other SQL variants which have been optimized for the particular platform, but generally conform to ANSI specifications [Davidson 12].

**Rule 6: View Updating Rule:** *Any view that is theoretically updateable can be updated using the RDBMS.*

This rule simply says the database should allow the same level of data manipulation to views that are available via direct access. “Views” are logical abstractions to the underlying physical tables. Views are useful because they can change the appearance and behavior of their underlying table or tables. As a result this can be a very difficult rule to implement. Imagine a view that has four or five inputs (tables or views) and several of the views are based on other views. The rule maintains we must be able to update the underlying tables correctly. SQL Server does not fully support this rule directly but in the case where the database engine is unable to determine the correct updates, developers can override the default behavior using triggers. Triggers are a special form of custom operation designed to fire when certain operations occur (like an update) against a table or view. When applied against a view, these triggers can then apply more advanced logic to put the appropriate changes to the underlying tables.

**Rule 7: High Level Insert, Update and Delete:** *The RDBMS supports insertions, updates and deletion at the table level.*

As in chapter 4, one of the great benefits of using a relational is the ability to perform set-based operations which allow working with multiple rows in a single operation. Rule 7 makes it possible. We normally retrieve data in sets which are made up of multiple rows. This rule states that we should be able to add/change/delete data based on sets and not just a single row of data. SQL Server 2012 gives users the ability to cross load data from one set to another as well as the ability to do add/change/delete operations based on an internal table, external table or even an input file. For SQL programmers, it is important to develop a skill for thinking in terms of set-based operations (or batches) in order to maximize the performance of database operations.

**Rule 8: Physical Data Independence:** *The execution of ad-hoc requests and application programs queries is not affected by changes in the physical data access and storage methods.*

There must be a layer of separation between the user and the physical devices and architecture that actually store the data so that users of the database need not be aware of the underlying physical architecture beyond such necessities as the network server name and/or IP address. Hence, for users working from a database, the experience should be the same (network bandwidth permitting) whether the actual physical server is sitting downstairs or on the other side of the Pacific Ocean. Furthermore, if the database server were to move from Cleveland to Manila, all else being equal, the users should not be able to detect a difference.

**Rule 9: Logical Data Independence:** *Logical changes in tables and views such as adding/deleting columns or changing field lengths or types need not necessitate modifications in the programs or in the format of ad-hoc requests.*

The rule states simply that the database should provide a mechanism so that a user's view of a piece of data remains static in the event the underlying table structure changes. This rule looks harder than it is in reality because SQL Server and other RDBMS provide views which can usually handle this task.

**Rule 10: Integrity Independence:** *Like the table/view definition, integrity constraints are stored in the online catalog and can therefore be changed without necessitating changes in the application programs.*

Constraints are a topic deserving of a separate chapter, but, in short, consider them as “checks” to make sure the data entered makes physical (the right type) and logical (the right value or values) sense. Like other object metadata, constraint information is stored in tables and can be changed using standard SQL. SQL Server provides a number of tables to store and retrieve information such as the one shown in figure 145.

The screenshot shows a SQL query window with the following query:

```
SELECT *
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
```

The results pane displays a table with 10 rows of primary key constraints. The columns are: CONSTRAINT\_CATALOG, CONSTRAINT\_SCHEMA, CONSTRAINT\_NAME, TABLE\_CATALOG, TABLE\_SCHEMA, and TABLE\_NAME.

	CONSTRAINT_CATALOG	CONSTRAINT_SCHEMA	CONSTRAINT_NAME	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME
1	FuzzyContexts	dbo	PK_Algorithm	FuzzyContexts	dbo	Algorithm
2	FuzzyContexts	dbo	PK_Defuzzifier	FuzzyContexts	dbo	Defuzzifier
3	FuzzyContexts	dbo	PK_DefuzzifierType	FuzzyContexts	dbo	DefuzzifierType
4	FuzzyContexts	dbo	PK_FuzzyContext	FuzzyContexts	dbo	FuzzyContext
5	FuzzyContexts	dbo	PK_FuzzyContextInput	FuzzyContexts	dbo	FuzzyContextInput
6	FuzzyContexts	dbo	PK_FuzzyContextOutput	FuzzyContexts	dbo	FuzzyContextOutput
7	FuzzyContexts	dbo	PK_FuzzyInferenceSystem	FuzzyContexts	dbo	FuzzyInferenceSystem
8	FuzzyContexts	dbo	PK_FuzzyInferenceSystem_FuzzyVariable_xRef	FuzzyContexts	dbo	FuzzyInferenceSystem_FuzzyVariable_x
9	FuzzyContexts	dbo	PK_FuzzyInferenceSystem_FuzzyVariable_xRef	FuzzyContexts	dbo	FuzzyInferenceSystem_FuzzyVariable_x
10	FuzzyContexts	dbo	PK_FuzzyOutputs	FuzzyContexts	dbo	FuzzyVariableOutput

**Figure 145 – Schema view listing primary key constraints**

**Rule 11: Distribution Independence:** *Application programs and ad-hoc request are not affected by changes in the distribution of the physical data.*

Similar to rule 8, but applies to the data *on* the server as opposed to the server itself. This allows data to reside on different disk drives, SAN (storage area network) or other devices without impacting the user interface. The database engine handles all the necessary low-level storage and retrieval operations under the covers in typical “black box” fashion.

**Rule 12: No Subversion Rule:** *If the RDBMS has a language that accesses the information of a record, this language should not be able to bypass the integrity constraints.*

Surprisingly this last rule might be the most important of them all. In order for users (say customers at a bank) to trust the database, they must trust that the database engine has the necessary safeguards for their data. If the database engine allowed for a mechanism to bypass integrity constraints then the data would be suspect and potentially useless. One the primary reasons for the ascendancy of relational databases has been the uncompromising way in which they prevent data from violating existing constraints. Any sort of data “sabotage” can only occur via deliberate means (such as removing a constraint and THEN adding bad data) and not through an accidental process.

Relational databases went from nearly non-existent to the preferred way to store data in just a couple of decades due to their ease of implementation, flexibility and power. As such they are growing more indispensable for generic applications and as demonstrated in chapter 4, add a new dimension to the power and intelligence of traditional (i.e. formerly non-database) applications approaches such as a Genetic algorithm or Fuzzy Inference System.

## APPENDIX G: 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 [Kitayama 02], [Ke-jun 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 11]. 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.

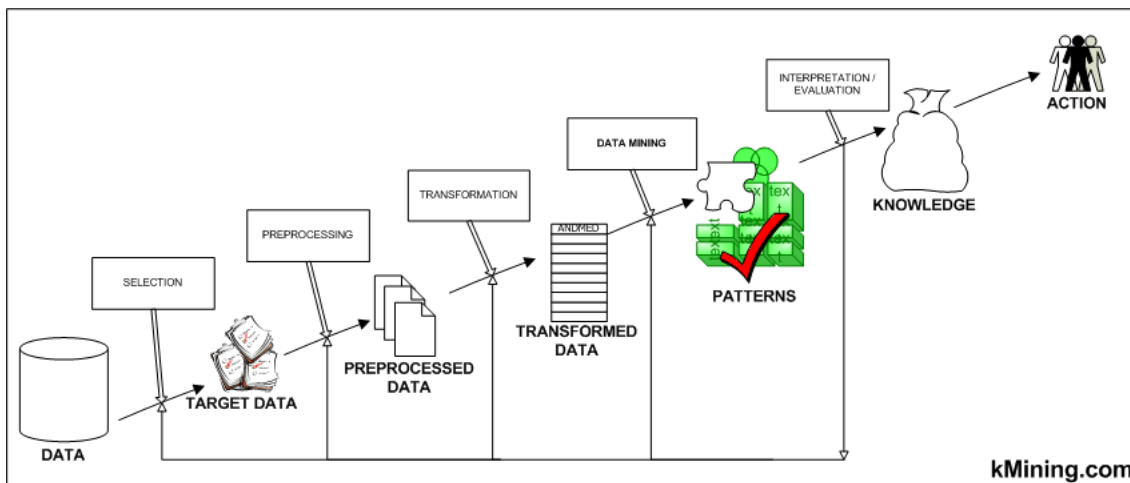
Problems often arise, however, in trying to determine what questions are appropriate. Datasets can become so large that even finding out where to begin presents significant challenges. Basic questions that answer “who” or “how many” do not easily lead to the more pressing and useful question of “why”. 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 11], [Ben-Gan 06], [Witten 11].

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 11], [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 [Haruechaiyasak 05], [Adomavicius 01].



**Figure 146 - Data Mining Process.**

Data mining starts with data, usually in the form of a large relational database [Han 11]. 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 systems, 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 the many classification techniques are Decision Trees, Neural Networks, Bayesian Networks, Rule-based systems, Support Vector Machines, K-Means and Fuzzy C-Means.

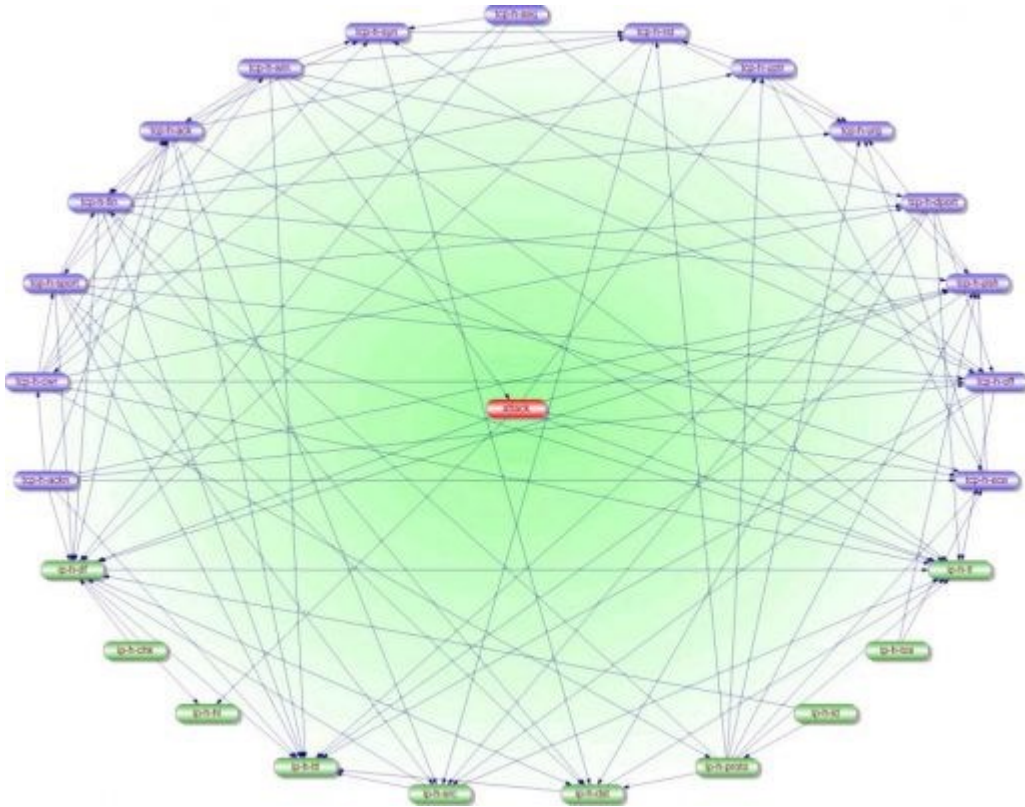


Figure 147 - Classification of relations among multiple elements.

2. Association, where relationships are drawn between objects.

Some association techniques include Association Rules, Decision Trees, Bayesian Networks, and K-Nearest Neighbor.

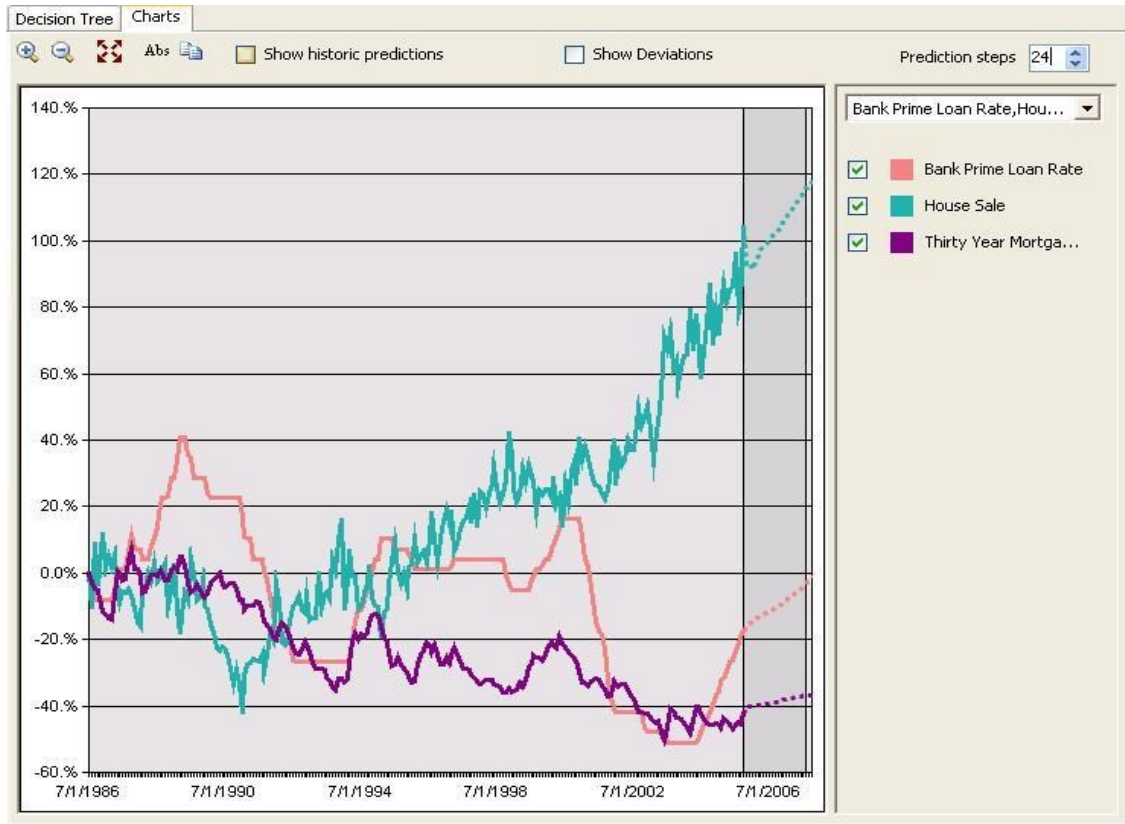
Customers Who Bought This Item Also Bought

A screenshot of an Amazon.com product page showing four book recommendations. Each recommendation includes a book cover, the title, author, star rating, and price.

Book Title	Author	Rating	Price
Fuzzy Sets and Fuzzy Logic: Theory and Applications	George J. Klir	★★★★ (4)	\$74.66
Schaum's Outline of Discrete Mathematics, 3rd Ed. (Schaum's Ou...)	Seymour Lipschutz	★★★★ (2)	\$12.89
An Introduction to Fuzzy Logic for Practical Applications	Kazuo Tanaka	★★★★☆ (7)	\$58.05
Fuzzy Logic: The Revolutionary Computer Technology That Is Cha...	Daniel Mcneill	★★★★☆ (10)	\$12.60

Figure 148 - Amazon.com association links other book titles to a book purchase.

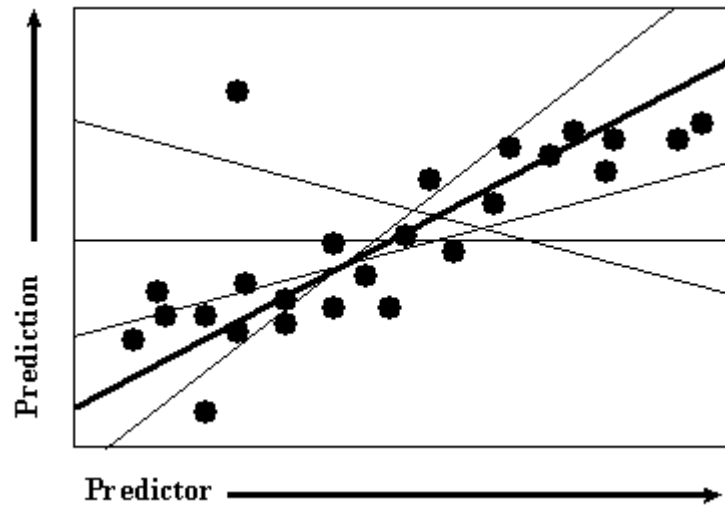
3. Prediction, where past behavior is extrapolated to anticipated future actions. Classification techniques can also serve as predictors.



**Figure 149 - Predictions (far right dotted lines) from existing data patterns**

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

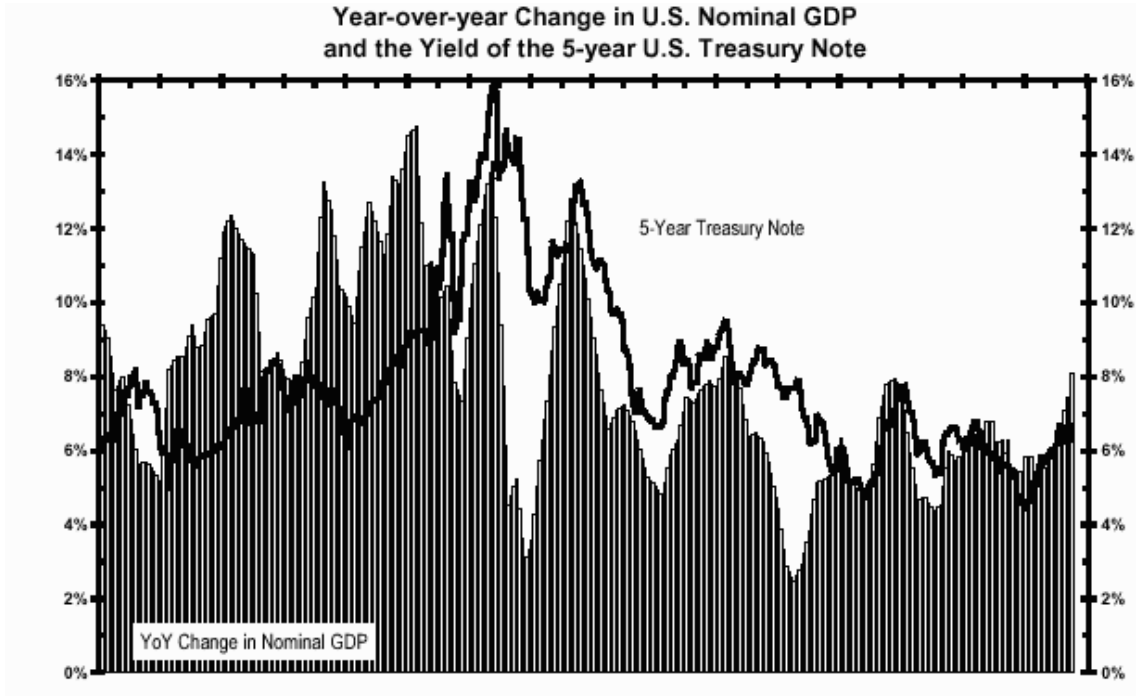
Among regression techniques are Neural Networks, Linear and Non-Linear regression techniques and Fuzzy-Set based approaches.



**Figure 150 - Linear Regression attempts over a series of data points.**

5. Time Series, where activity is grouped and trends established along 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. This is in order to create trend-based views that compare similar time periods or time spans or contrast with other data values. Figure 151 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 Nominal GDP as the lead. Time Series makes this otherwise vague relationship much more clear.



**Figure 151 - 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.

## APPENDIX H: LOCAL SEARCH ALGORITHMS PSEUDO-CODE

### H.1: Introduction

A number of local search algorithms (LSAs) were studied and programmed for simulation in the C# programming language. In the simulations explored, the algorithms attempt to find a solution to the 8-Queens problem. Eleven different local search techniques were developed, tried and compared:

1. Hill Climb
2. Stochastic Hill Climb
3. Stochastic Hill Climb using the Descending Deviations Technique
4. Random Restart Hill Climb
5. Simulated Annealing
6. Genetic Mutation
7. Minimum Conflicts
8. Tabu Search
9. Simulated Annealing using the Descending Deviations Technique
10. Memetic Mutation
11. Genetic Mutation using the Descending Deviations Technique

What follows is pseudo-code for the individual algorithms along with a general discussion of the merits and disadvantages of each. Appendix A, Minor Contribution #3 presents a comparative analysis of the various techniques.

### H.2: 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:

### Pseudo-code 16 – Hill Climbing Algorithm

**Algorithm: HillClimb(board)**

**Input:** *board* a random configuration of queens on a chessboard, 1 per column

**Output:** an chessboard solution if one can be found, otherwise NULL

**Begin**

```

1 DO
2   determine number of conflicts on board for each tile
3   IF board state = SUCCESS OR conflicts = 0, there are no conflicts, goal is reached
4     return board
5   ELSE IF board state = FAILURE, local maximum reached, unable to proceed further
6     return NULL
7   ELSE
8     set board to GetNextHillClimbBoard
9   END IF
10 LOOP

```

**End**

**Algorithm: GetNextHillClimbBoard(board)**

**Input:** *board* a chessboard to solve

**Output:** a new, more optimal chessboard if one can be found, otherwise, set state to FAILURE

**Begin**

```

1 FOR EACH column of tiles on board
2   get the queen for that column
3   get the lowest_tile for the column's row
4   IF queen is already on lowest_tile continue to next column
5   ELSE
6     add lowest_tile to array of candidate tiles
7   NEXT column
8   IF array of candidate tiles is empty set state of board to FAILURE and return board

9   pick best_tile from array of candidate tiles, if more than one tile is best, pick randomly
10  get column from best_tile
11  get queen from column
12  move queen from current position to best_tile
13  return board

```

**End**

It is a simple algorithm that examines the neighborhood of solutions and picks the best one (randomly if more than one option). 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.

### H.3: LSA #2 - Stochastic Hill Climbing

Stochastic Hill Climbing is a variant of the hill climb in which not just 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:



### Pseudo-code 17 – Stochastic Hill Climb Algorithm

**Algorithm: StochasticHillClimb(board)**

**Input:** *board* a random configuration of queens on a chessboard, 1 per column

**Output:** an chessboard solution if one can be found, otherwise NULL

**Begin**

```

1 DO
2   determine number of conflicts on board for each tile
3   IF board state = SUCCESS OR conflicts = 0, there are no conflicts, goal is reached
4     return board
5   ELSE IF board state = FAILURE, local maximum reached, unable to proceed further
6     return NULL
7   ELSE
8     set board to GetNextSHillClimbBoard
9   END IF
10 LOOP

```

**End**

**Algorithm: GetNextSHillClimbBoard(board)**

**Input:** *board* a chessboard to solve

**Output:** a new, more optimal chessboard if one can be found, otherwise, set state to FAILURE

**Begin**

```

1 set lowest_count = current number of board conflicts
2 FOR EACH column of tiles on board
3   FOR EACH row in column
4     get the tile for the column/row
5     IF tile conflicts < lowest_count
6       add tile to candidate tiles the number of times amount below lowest_count
7     END IF
5   NEXT row
6 NEXT column
9 IF array of candidate tiles is empty set state of board to FAILURE and return board

10 pick random tile from array of candidate tiles
11 get column from tile
12 get queen from column
13 move queen from current position to tile
14 return board

```

**End**

Strength of this approach is that it does provide some opportunity to allow an algorithm to escape a local maximum although weighting still favors the “best” options.

Weakness is that it still has a great tendency to be attracted to and trapped in a local maximum, only slightly less so than a purely greedy approach. The key significance of this approach is that it doesn't automatically throw out suboptimal candidates. This allows for greater latitude of picks while still maintaining a bias for those candidates which appear more qualified. It is also, relatively speaking, a resource light process.

#### H.4: 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:

### Pseudo-code 18 – Improved Stochastic Hill Climb Using DD Technique

**Algorithm: DD-StochasticHillClimb(board)**

**Input:** *board* a random configuration of queens on a chessboard, 1 per column

**Output:** an chessboard solution if one can be found, otherwise NULL

**Begin**

```

1 DO
2   determine number of conflicts on chessboard for each tile
3   IF board state = SUCCESS OR conflicts = 0, there are no conflicts, goal is reached
4     return board
5   ELSE IF board state = FAILURE, local maximum reached, unable to proceed further
6     return NULL
7   ELSE
8     set board to GetNextDDSHillClimbBoard
9   END IF
10 LOOP

```

**End**

**Algorithm: GetNextDDSHillClimbBoard(board, bounce)**

**Input:** *board* a chessboard to solve

*bounce* the ceiling size of the allowable bounce

**Output:** a new, more optimal chessboard if one can be found, otherwise, set state to FAILURE

**Begin**

```

1 set lowest_count = current number of board conflicts
2 FOR EACH column of tiles on board
3   FOR EACH row in column
4     get the tile for the column/row
5     IF tile conflicts < lowest_count
6       add tile to candidate tiles the number of times conflicts below lowest_count
7     END IF
8   NEXT row
9 NEXT column
10 IF array of candidate tiles is empty
11 IF bounce = 0 set state of board to FAILURE and return board
12 ELSE set board = BounceBoard, reduce value of bounce
13 END IF
14 ELSE
15   pick random tile from array of candidate tiles
16   get column from tile
17   get queen from column
18   move queen from current position to tile
19 return board

```

**End**

**Algorithm: BounceBoard(board, bounce)****Input:** *board* a chessboard to solve*bounce* the ceiling size of the allowable bounce**Output:** a new, bounced chessboard**Begin**

```

1  set lowest_count = current number of board conflicts
2  FOR EACH column of tiles on board
3    FOR EACH row in column
4      get the tile for the column/row
5      IF tile conflicts < lowest_count
6        add tile to candidate tiles the number of times conflicts below lowest_count + bounce
7      END IF
8    NEXT row
9  NEXT column
10 pick random tile from array of candidate tiles
11 get column from tile
12 get queen from column
13 move queen from current position to tile
14 return board

```

**End**

Strengths of this approach is that it is roughly equivalent to the standard stochastic hill climb, but has the ability to escape local maxima, via the “bounce”. The bounce allows suboptimal choices to be considered to whatever degree is indicated by the bounce. Because it is a Descending Derivation technique, the bounce is gradually decreased to zero and the algorithm behaves more like the traditional stochastic. 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.

#### H.5: 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. IF a solution exists, the probably of success is essentially one, meaning that if given enough retries the Random Restart will eventually randomize its way to

a solution, but depends upon just how many retries one wants to do before giving up. Pseudo-code implementation is as follows:

### Pseudo-code 19 – Random Restart Hill Climbing Algorithm

**Algorithm: RandomRestartHillClimb(Board, retries)**

**Input:** *board* a random configuration of queens on a chessboard, 1 per column

*retries* the number of times to restart algorithm

**Output:** an chessboard solution if one can be found, otherwise NULL

**Begin**

```

1 DO
2   determine number of conflicts on board for each tile
3   IF board state = SUCCESS OR conflicts = 0, there are no conflicts, goal is reached
4     return board
5   ELSE IF board state = FAILURE, AND retries = 0 local maximum reached, unable to
   proceed further
6     return NULL
7   ELSE IF board state = failure AND retries > 0
8     set board to random configuration of queens, 1 per column and reduce retries
9   ELSE
10    set board to GetNextHillClimbBoard
11  END IF
12 LOOP

```

**End**

**Algorithm: GetNextHillClimbBoard(board)**

**Input:** *board* a chessboard to solve

**Output:** a new, more optimal chessboard if one can be found, otherwise, set state to FAILURE

**Begin**

```

1  FOR EACH column of tiles on board
2    get the queen for that column
3    get the lowest_tile for the column's row
4    IF queen is already on lowest_tile continue to next column
5    ELSE
6      add lowest_tile to array of candidate tiles
7    NEXT column
8  IF array of candidate tiles is empty set state of board to FAILURE and return board

9  pick best_tile from array of candidate tiles, if more than one tile is best, pick randomly
10 get column from best_tile
11 get queen from column
12 move queen from current position to best_tile
13 return board

```

**End**

Strengths of this approach is that the algorithm should theoretically be able to find any solution, if it exists and given enough time or retries. The underlying Hill Climb technique is very fast and uses minimal resources. The weakness is that if global maxima are sparse and local maxima are dense, the algorithm is likely to repeat many failed attempts on the way to meeting its goal.

#### H.6: LSA #5 - Simulated Annealing (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. Pseudo-code implementation is as follows:

### Pseudo-code 20 – Simulated Annealing

**Algorithm: Simulated Annealing(board)**

**Input:** *board* a random configuration of queens on a chessboard, 1 per column

**Output:** an chessboard solution if one can be found, otherwise NULL

**Begin**

```

1  set temperature to a specific value
2  DO
3    determine number of conflicts on board for each tile
4    IF board state = SUCCESS, there are no conflicts, goal is reached
5      return board
6    ELSE IF board state = FAILURE, AND temperature = 0 local maximum reached,
unable to proceed further
7      return NULL
9    ELSE
10   AnnealBoard(board, temperature)
11   decrement temperature
12  END IF
LOOP

```

**End**
**Algorithm: AnnealBoard(board, temperature)**

**Input:** *board* a chessboard to solve

*temperature*, an annealing value used to calculate probability threshold

**Output:** an annealed chessboard if one is found, otherwise NULL

**Begin**

```

1  pick a random tile from the board
2  IF tile does not have queen
3    IF tile reduces conflicts
4      move queen from tile column to tile
5    ELSE
6      calculate annealing_error for temperature
7      generate random_error between 0 and 1
8      IF annealing_error < random_error
9        move queen from tile column to tile
10   END IF
11  END IF
12  END IF
13  return board

```

**End**

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.

### H.7: 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 the implementation from Chapter 4, 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:

#### Pseudo-code 21 – Genetic Mutation Algorithm

**Algorithm: GeneticMutation (chessboard, retries)**

**Input:** *board* a random configuration of queens on a chessboard, 1 per column

*retries* number of times to perform mutation before giving up

**Output:** an chessboard solution if one can be found, otherwise NULL

**Begin**

1 create random sample board collection

2 DO

3 determine number of conflicts on *board* for each tile

4 IF *board* state = SUCCESS, there are no conflicts, goal is reached

5 return *board*

6 ELSE IF *board* state = FAILURE, AND *retries* = 0 local maximum reached, unable to proceed further

7 return NULL

9 ELSE

10 pick random *sample\_board* from board collection

11 pick *winner* as best board between *board* and *sample\_board*

12 set *loser* as remaining board

13 crossover breed *winner* and *loser* to make new *loser*

14 mutate *loser*

15 add *loser* back into collection

16 set *board* to *winner*

17 decrement *retries*

18 END IF

19 LOOP

**End**



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 DD version limits 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.

#### H.8: 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:

### Pseudo-code 22 – Min Conflicts Algorithm

**Algorithm: MinConflicts(board)**

**Input:** *board* a random configuration of queens on a chessboard, 1 per column

**Output:** an chessboard solution if one can be found, otherwise NULL

**Begin**

```

1 DO
2   determine number of conflicts on board for each tile
3   IF board state = SUCCESS, there are no conflicts, goal is reached
4     return board
5   ELSE IF board state = FAILURE local maximum reached, unable to proceed further
6     return NULL
7   ELSE
8     FOR EACH column in board
9       IF tile = GetBestTile(column) != NULL
10        move queen to tile
11      END IF
12 END IF

```

**End**

**Algorithm: GetBestTile(column)**

**Input:** *column* one column of tiles on a chessboard

**Output:** the best tile without a queen if one can be found, otherwise NULL

**Begin**

```

1 set qTile to tile with queen
2 calculate the lowest conflict_count for column
3 IF qTile conflicts = conflict_count THEN return NULL
4 ELSE
5   FOR EACH tile in column
6     IF tile != qTile AND tile conflicts = conflict_count
7       Add tile to candidate_list
8     END IF
9   NEXT tile
10 IF candidate_list is empty return NULL
11 ELSE IF candidate_list = 1 return tile in candidate_list
12 ELSE return random tile from candidate_list

```

**End**

The strengths of this approach are that it generally finds a solution in only a few iterations and is fairly easy to implement. The weakness is that it may be susceptible to looping conditions as changes propagate and undo/redo various configurations that are interdependent. For variety, the column passes were randomized right-to-left and left-to-right.

### H.9: 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. This optimizes the search process at a cost of memory required to store previous attempts.

The strengths of this approach is that it explores many potential paths. The weakness is that it requires significant memory for large search problems (potential configuration size  $X$  # bits to represent configuration  $X$  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.

Pseudo-code implementation is as follows:

### Pseudo-code 23 – Tabu Search Algorithm

**Algorithm: TabuSearch (board)**

**Input:** *board* a random configuration of queens on a chessboard, 1 per column

**Output:** an chessboard solution if one can be found, otherwise NULL

**Begin**

```

1 DO
2   determine number of conflicts on board for each tile
3   IF board state = SUCCESS, there are no conflicts, goal is reached
4     return board
5   ELSE IF board state = FAILURE local maximum reached, unable to proceed further
6     return NULL
7   ELSE
8     set board to GetNextHillClimbBoardTabu
9   END IF
10 LOOP

```

**End**

**Algorithm: GetNextHillClimbBoardTabu(board)**

**Input:** *board* a chessboard to solve

**Output:** a new, more optimal chessboard if one can be found, otherwise, set state to FAILURE

**Begin**

```

1 FOR EACH column of tiles on board
2   get the queen for that column
3   get the lowest_tile for the column's row
4   IF queen is already on lowest_tile continue to next column
5   ELSE IF board configuration with lowest_tile not tried
6     add lowest_tile to array of candidate tiles
7 NEXT column
8 IF array of candidate tiles is empty set state of board to FAILURE and return board

9 pick best_tile from array of candidate tiles, if more than one tile is best, pick randomly
10 get column from best_tile
11 get queen from column
12 move queen from current position to best_tile
13 save board configuration with best_tile to configurations tried
13 return board

```

**End**

### H.10: 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 chooses that and completes successfully. A case can be made that this operation no longer classifies as a “local search” since the entire board is scanned. However, as opposed to scanning all states, as a comprehensive search does, this operation looks at the current state only. In the event the state space is very large and such a total scan becomes impractical, benefits might still be had by scanning only a “neighborhood” of the current position, in which case the algorithm fits the traditional definition of a local search.

This approach carries with it all the positives of traditional SA, with a couple of exceptions. 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, SA iterations are needed to reach a solution. The scan could be optimized further by adding elements to Tabu Search to prevent rescanning.

Pseudo-code implementation is as follows:

**Pseudo-code 24 – Improved Simulated Annealing Algorithm Using DD Technique****Algorithm: DDSimulatedAnnealing (chessboard)****Input:** *board* a random configuration of queens on a chessboard, 1 per column**Output:** an chessboard solution if one can be found, otherwise NULL**Begin**

```
1  set temperature to a specific value, dd_ceiling to a specific value or function of
   temperature
2  DO
3    determine number of conflicts on board for each tile
4    IF board state = SUCCESS, there are no conflicts, goal is reached
5      return board
6    ELSE IF board state = FAILURE, AND temperature = 0 local maximum reached,
   unable to proceed further
7      return NULL
9    ELSE
10     AnnealBoardDD(board, temperature, dd_ceiling)
11     decrement temperature
12  END IF
LOOP
End
```

**Algorithm: AnnealBoardDD(board, temperature, dd\_ceiling)**

**Input:** *board* a chessboard to solve

*temperature* an annealing value used to calculate probability threshold

*dd\_ceiling* a ceiling value (or a function of *temperature*)

**Output:** an annealed chessboard if one is found, otherwise NULL

**Begin**

1 scan board looking for *goal\_state\_tile*

2 IF *goal\_state\_tile* != NULL

3 move *queen* from *goal\_state\_tile* column to *goal\_state\_tile*

4 return *board*

5 ELSE

6 DO

7 pick a random *tile* from the *board*

8 IF *tile* does not have queen

9 IF *tile* reduces conflicts

10 move *queen* from *tile* column to *tile*

11 ELSE

12 calculate *annealing\_error* for *temperature*

13 generate *random\_error* between 0 and 1

14 IF *annealing\_error* < *random\_error* AND *tile* conflicts < *ceiling*

15 move *queen* from *tile* column to *tile*

16 END IF

17 END IF

18 END IF

19 LOOP UNTIL *tile* != NULL

20 return *board*

**End**

### H.11: LSA #10 – Memetic Mutation

As discussed in Chapter 2, 4 and Appendix E, the Memetic Mutation algorithm is a derivative of the Genetic Mutation algorithm. As such it has a similar design and operation but has been shown to more effective than a traditional Genetic Mutation, although with a correspondingly higher cost per iteration [Eiben 07]. In the case of the 8-Queens problem described in Appendix A, the Memetic Mutation algorithm is a combination of the Genetic Mutation and a simple Hill Climb algorithm.

Pseudo-code implementation is as follows:

#### Pseudo-code 25 – Memetic Mutation Algorithm

**Algorithm: MemeticMutation (board, retries)**

**Input:** *board* a random configuration of queens on a chessboard, 1 per column

*retries* number of times to perform mutation before giving up

**Output:** an chessboard solution if one can be found, otherwise NULL

**Begin**

1 create random sample board collection

2 DO

3 determine number of conflicts on *board* for each tile

4 IF *board* state = SUCCESS, there are no conflicts, goal is reached

5 return *board*

6 ELSE IF *board* state = FAILURE, AND *retries* = 0 local maximum reached, unable to proceed further

7 return NULL

9 ELSE

10 FOR EACH *sample\_board* in board collection

11 Apply Hill Climb to *sample\_board*

12 IF *board* state = SUCCESS, there are no conflicts, goal is reached

13 return *sample\_board*

14 NEXT *sample\_board*

15 pick random *sample\_board* from board collection

16 pick *winner* as best board between *board* and *sample\_board*

17 set *loser* as remaining board

18 crossover breed *winner* and *loser* to make new *loser*

19 mutate *loser*

20 add *loser* back into collection

21 set *board* to *winner*

22 decrement *retries*

23 END IF

24 LOOP

**End**



This algorithm could be improved by a Tabu-like mechanism to skip any boards which haven't changed, but in spite this it is a relatively easy way to get a significant improvement over the underlying genetic version.

#### H.12: LSA #11 – Genetic Mutation Using Descending Deviation Optimizations

Genetic Mutation traditionally doesn't discriminate between genes except when performing a fitness comparison. As a result it performs a lot of useless, even counter-productive mutations, often moving around local maxima for a number of iterations instead of spending fewer cycles on a more direct approach. Applying the Descending Deviation heuristic to Genetic Mutation means preventing mutations which have fitness values below a certain threshold. The reasoning for this is that as genes move further down the "fitness" scale, they become inherently more unviable and statistically less likely to produce viable offspring either through breeding, crossover mutation or random mutation. By limiting the numbers of these "unfit" genes, the overall sample is more likely to produce a solution within a given number of iterations. This was demonstrated in Appendix A, Minor Contribution #3.

Pseudo-code implementation is as follows:

**Pseudo-code 26 – Improved Genetic Mutation Algorithm Using DD Technique**

**Algorithm: DDGeneticMutation (chessboard, retries)**

**Input:** *board* a random configuration of queens on a chessboard, 1 per column

*retries* number of times to perform mutation before giving up

**Output:** an chessboard solution if one can be found, otherwise NULL

**Begin**

```

1 create random sample board collection
2 create fitness_threshold as value or function based upon retries
2 DO
3   determine number of conflicts on board for each tile
4   IF board state = SUCCESS, there are no conflicts, goal is reached
5     return board
6   ELSE IF board state = FAILURE, AND retries = 0 local maximum reached, unable to
   proceed further
7     return NULL
9   ELSE
10    pick random sample_board from board collection
11    pick winner as best board between board and sample_board
12    set loser as remaining board
13    crossover breed winner and loser to make new loser
14    mutate loser
15    IF loser fitness >= fitness_threshold
16      add loser back into collection
17    END IF
18    set board to winner
19    decrement retries
20  END IF
21 LOOP

```

**End**

## APPENDIX I: DATABASE, DATA WAREHOUSE AND DATA MINING TEST SOFTWARE

Microsoft's SQL Server 2012 Developer Edition provided the platform for the relational database along with the data mining test software used in this dissertation. 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 152.

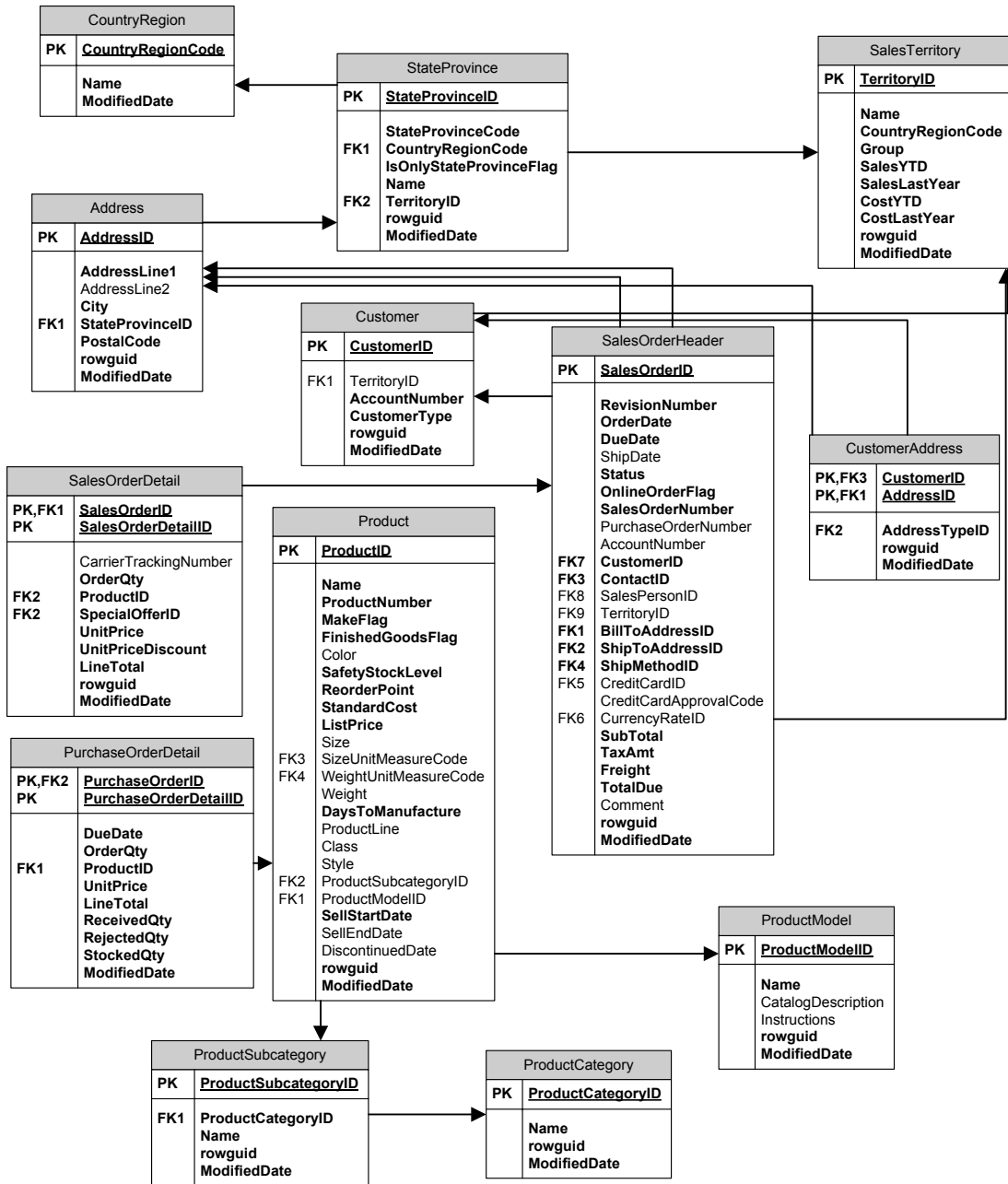


Figure 152 - Partial Schema of AdventureWorks sample database

Using SQL Server 2012 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 combining 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 153 and the Bayes mining model implementation is shown in Figure 154. 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 155.



Figure 153 – A Decision Tree in SQL Server Analysis Services

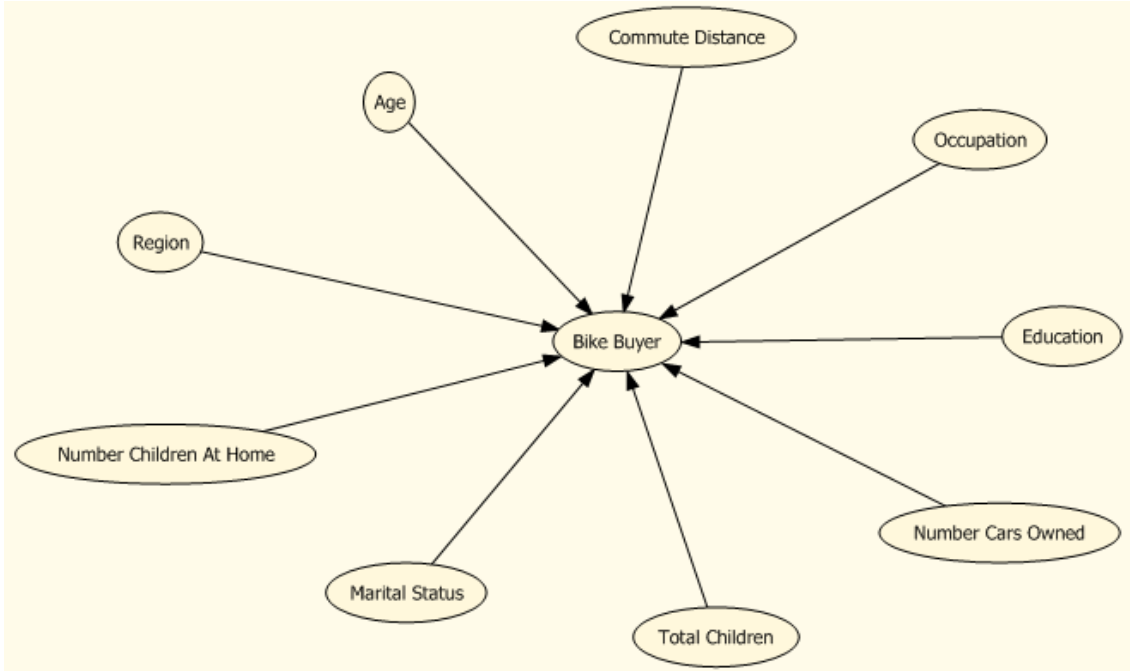


Figure 154 - The Bayes mining model

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 155 - The Neural Network mining model

## PUBLICATIONS

### JOURNAL PUBLICATIONS

1. McCarty, K., & Manic (2014), An Adaptive Framework for Contextual Algorithms, IEEE Transactions on Industrial Informatics (in review)

**Abstract:** Modern industrial systems are becoming increasingly more complex and tasked to perform an ever widening array of functions. State of the art implementations of hybrid functions or increasingly sophisticated algorithmic seek broader application in response, but often with correspondingly greater complexity and resource overhead. To alleviate this problem this paper proposes a robust and flexible software architecture designed to store, maintain and properly utilize contextual information and apply it to the problem at hand. This contextual architecture, called Fuzzy Logic Type-C (FLC) is composed of a number of integrated components: 1) a hierarchy of classes and interfaces supporting context creation and use, 2) implementations of swappable algorithms such Fuzzy Logic Type-1 and Type-2, 3) an extensible algorithm definition language to store, create and dynamically load algorithms and tools for maintaining algorithm configurations. Test results show that FLC applications 1) improved the performance of underlying Type-1 and Type-2 equivalents by over 20%, 2) reduced the number of fuzzy set evaluations 30%-40% 3) provided linguistic, human-readable descriptions of problem and solution behavior, 4) adapt readily to radically different behaviors and environments.



2. McCarty, K., & Manic (2014), A Hierarchical Technique for Diverse Contextual Solution Determination and Optimization, Applied Soft Computing (in review)

**Abstract:** Complex problems often require equally complex solutions. These solutions may require different algorithms depending upon the particular internal or operational state that exists. Fuzzy logic, Artificial Neural Networks, Radial Basis Functions and other techniques provide differing strengths and weaknesses for problem-solving. Traditional optimization techniques provide for ways to optimize an algorithm by determining optimal configurations, but do not naturally provide a mechanism to determine if the underlying technique itself is an optimal choice. This paper proposes a novel hierarchical technique for optimization under Fuzzy Contexts that allows for interchangeability and testing of algorithms as well internal optimization. This hierarchical technique is called Fuzzy Context Optimization Abstraction for Processes or FC-OAP and utilizes an algorithm definition language combined with random selection in order to determine optimal algorithmic techniques. Test results in problems as diverse as a robot navigation problem, simple sort and 8-Queens problem, the OAP implementation successfully picked the best algorithm, based upon a defined fitness function, from among a database of possible solutions.

### PEER-REVIEWED CONFERENCE PUBLICATIONS

3. McCarty, K., & Manic, M. (2008). Line-of-sight tracking based upon modern heuristics approach. 3rd IEEE Conference on Industrial Electronics and Applications, (pp. 40-45).

**Abstract:** Any autonomous vehicle must be able to successfully navigate a wide variety of situations and terrain conditions. As a result, proposed solutions usually involve a sophisticated and expensive implementation of both hardware and software. In many situations, however, truly autonomous operation may not be necessary or practical. Instead, equipping and training a vehicle to automatically follow a human-controlled lead vehicle is a viable alternative. While still autonomous, the vehicle relies upon its leader to handle the complex decisions with regards to course and speed. This paper presents a simple and elegant configuration, called FLoST for 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 the alternative approach where the follower needs to undergo a complex training process, the follower using the approach presented in this paper primarily relies upon a human leader to provide direction, allowing for a much simpler and less expensive vehicle implementation while still being able to match or exceed the effectiveness of the autonomous design under similar conditions. Finally, three boundary cases of lead vehicle maneuvers (circle, spiral and weave) are presented to show the efficacy of this approach.

4. McCarty, K., & Manic, M. (2008). Contextual Fuzzy Type-2 Hierarchies for Decision Trees (CoFuH-DT) - An Accelerated Data Mining Technique. Conference on Human System Interactions, (pp. 699-704). Krakow, Poland.

**Abstract:** Advanced data mining techniques (ADMT) are very powerful tools for classification, understanding and prediction of object behaviors, providing descriptive relationships between objects such as a customer and a product they intend to buy. ADMT typically consists of classifiers and association techniques, among them, Decision Trees (DT). However, some important relationships are not readily apparent in a traditional decision tree. In addition, decision trees can grow quite large as the number of dimensions and their corresponding elements increase, requiring significant resources for processing. In either case, rules governing these relationships can be difficult to construct. This paper presents CoFuH-DT, a new algorithm for capturing intrinsic relationships among the nodes of DT, based upon a proposed concept of type-2 fuzzy “contexts”. This algorithm modifies a decision tree, first by generating type-1 fuzzy extensions of the underlying DT criteria or “conditions”; combining further those extensions into new abstractions overlaid with type-2 contexts. The resulting fuzzy type-2 classification is then able to capture intrinsic relationships that are otherwise non-intuitive. In addition, performing fuzzy set-based operations simplifies the decision tree much faster than traditional search techniques in order to aid in rule construction. Testing presented on a virtual store example demonstrates savings of multiple orders of magnitude in terms of nodes and applicable conditions resulting in 1) reduced complexity of decision tree, 2) ability to data mine difficult to detect interrelationships, 3) substantial acceleration of decision tree search, making it applicable for 4) real-time data mining of new knowledge.

5. McCarty, K., & Manic, M. (2008). Descending Deviation Optimization Techniques for Scheduling Problems. IEEE International Conference on Emerging Technologies and Factory Automation, (pp. 257-260).

**Abstract:** In factory automation, production line scheduling entails a number of competing issues. Finding optimal configurations often requires use of local search techniques. Local search looks for a goal state employing heuristics and random local “probes” in order to move from state to state. All local search techniques, however, suffer from problems with local maxima, i.e. have the potential of getting “stuck” in a suboptimal state. While careful introduction of randomizations is certainly a recognized technique, it can also lead the algorithm even more astray. This paper describes a heuristic technique called Descending Deviation Optimizations (DDO) in which a gradually lowering-- randomization ceiling allows a local search technique to “bounce” randomly without going too far astray. An example applying the DDO to a local search technique and achieving significant improvement is shown.

6. McCarty, K., & Manic, M. (2009). Adaptive Behavioral Control of Collaborative Robots in Hazardous Environments. 2nd Conference on Human Systems Interactions, (pp. 10-15).

**Abstract:** Terrain exploration carries with it significant hazards. Robots attempting to map a piece of unknown terrain must be able to make decisions and react appropriately to dynamic and potentially hostile conditions. However, because of constraints on size and cost, robots may have limited ability to store and process necessary information. In addition, knowledge discovered by others may be difficult to share. This paper proposes a system using a powerful master controller, operating from a safe environment, directing the movements of numerous robots exploring a piece of terrain. The master controller processes the information from the robots, updates the decision process and distributes these updates back to the robots. This process allows for a cooperative, effective search environment while also maintaining a small processing footprint. It also allows the robot to employ adaptive, subsumptive behavioral modification as new information is made available. A test simulation of a hazardous environment demonstrates that even robots with little intrinsic intelligence can learn complex behaviors in order to reach their goal.

7. McCarty, K., Manic, M., Goodwin, P., & Piasecki, M. (2009). Submission and Querying Tools for a Hydrologic Information Systems Database. 8th International Conference on Hydroinformatics. Concepcion, Chile.

**Abstract:** The recent establishment of the WATERS information network in the US, has prompted a number of entities to join this network beyond the initial selected set of test bed nodes. The state of Idaho is supporting the creation of a IdahoWaters node through its EPSCoR program with them aim of not only providing a single access point for Idaho water information but also to make these data holdings accessible nationwide through participation in the network. Given the many individuals institutions that will participate in this effort, means of data submission are an extremely important aspect when developing an information node of this type. This paper demonstrates an architecture for the submission as well as querying and presentation of large datasets of hydrologic data via the internet. Discussed are the necessary hardware and software configurations used to create databases for staging, permanent storage, online analytical processing and distribution. In addition software and tools for decision support as well as automation for data extraction, transformation and loading are presented. Finally application of this architecture is shown for a wide-scale, distributed, hydrologic-based, collaborative information network.

8. McCarty, K., Manic, M., Cherry, S., & McQueen, M. (2010). A Temporal-Spatial Data Fusion Architecture for Monitoring Complex Systems. 3rd Conference on Human Systems Interactions, (pp. 101-106).

**Abstract:** Non-homogenous systems arise from the need to incorporate a variety of disparate systems into a cohesive functioning whole and may comprise many crucial elements of an industrialized, modern society. As a result they must be constantly monitored to ensure efficient functioning and avoid expensive breakdowns. In particular, inter-connected computer-based systems must increasingly be aware of cyber and physical threats that are dynamic and evolutionary in nature. However, difficulties arise in trying to ascertain threats and problems among the diverse sources of information generated by these systems. Finally, there is the question of how best to present this data to a human operator. Human systems require not just analysis, but presentation which encourages timely, proactive or corrective decisions. This paper presents a software architecture to solve these problems based upon data fusion using temporal-spatial relationships. As phase one of a three phase project, a prototype implementation of this architecture demonstrates application of this technique for a cohesive system. Test results showed the system capable of real-time fusion of physical, cyber and process data elements as well as analysis, display and interpretation of threats.

9. T.R. McJunkin, R.L. Boring, M.A. McQueen, L.P. Shunn, J.L. Wright, D.I. Gertman, O. Linda, K. McCarty, M. Manic, (2011) "Concept of operations for data fusion visualization," in Proc. of European Safety and Reliability Conference (ESREL2011), Troyes, France, Sept. 19-22, 2011

**Abstract:** Data fusion for process control involves the presentation of synthesized sensor data in a manner that highlights the most important system states to an operator. The design of a data fusion interface must strike a balance between providing a process overview to the operator while still helping the operator pinpoint anomalies as needed. With the inclusion of a predictor system in the process control interface, additional design requirements must be considered, including the need to convey uncertainty regarding the prediction and to minimize nuisance alarms. This paper reviews these issues and establishes a design process for data fusion interfaces centered on creating a concept of operations as the basis for a design style guide.

10. McCarty, K., & Manic, M. (2012). A Proposed Data Fusion Architecture for Micro-Zone Analysis and Data Mining. 5th International Symposium on Resilient Control Systems, (pp. 72-76).

**Abstract:** Micro-zone analysis involves use of data fusion and data mining techniques in order to understand the relative impact of many different variables. Data Fusion requires the ability to combine or “fuse” data from multiple data sources. Data mining involves the application of sophisticated algorithms, such as Time Series, to describe micro-zone behavior and predict future values based upon past values. One of the difficulties encountered in developing generic time series or other data mining techniques for micro-zone analysis is the wide variability of the data sets available for analysis. This presents challenges all the way from the data gathering stage to results presentation. This paper presents an architecture designed and used to facilitate the collection of disparate data sets well suited for data fusion and data mining. Results show this architecture provides a flexible, dynamic framework for the capture and storage of a myriad of dissimilar data sets and can serve as a foundation from which to build a complete data fusion architecture.

11. McCarty, K., Manic, M., & Gagnon, A. (2013). A Fuzzy Framework with Modeling Language for Fuzzy Logic Type 1 and Type 2 Application Development. The 6th International Conference on Human Systems Interaction (pp. 334-341). Gdansk, Poland: IEEE.

**Abstract:** Fuzzy logic, Type-1 and Type-2, are well suited for human systems interactions because they provides a natural way of implementing linguistic terms from human experts. Existing fuzzy frameworks, however, provide limited support for Type-2. They also tend to be fairly complicated and/or have limited portability. This paper introduces a fuzzy framework for building a Type-1 or Type-2 fuzzy controller. A “wizard” application and modeling language are supported to provide an easy-to-use interface for creating a fuzzy inference system. The benefits of this framework are: 1) Increased understanding of fuzzy systems implementation via easy-to-use visual tools; 2) Reduced development time; 3) A standardized and portable codebase; 4) Easy configuration via XML; 5) Support for both Type-1 and Type-2 fuzzy sets and rules. The framework is tested and solves a maze problem using both Type-1 and Type-2 implementations.



12. McCarty, K., & Manic, M. (2014). Fuzzy Contexts (Type C) and Fuzzymorphism to Solve Situational Discontinuity Problems. International World Congress on Computational Intelligence. Beijing, China: IEEE.

**Abstract:** Generalized solutions to complex problems often suffer from being overly complicated. The main contribution of this paper is to describe an architecture that allows for greater problem generalization without the traditional corresponding increase in complexity. The architecture extends traditional fuzzy logic and is called Fuzzy Contexts or Fuzzy Logic Type-C. Fuzzy logic permits partial membership and values can belong to multiple fuzzy sets. By breaking down a problem space into smaller contexts and allowing algorithms themselves to have relaxed memberships in those contexts, a Type-C solution can support multiple solutions to complex problems. This paper describes how problem spaces may be decomposed into smaller, more easily solvable components and fuzzified together under a Type-C hierarchy. Test results with a simulated robotic navigation system demonstrates how a Type-C implementation is able to improve upon a generalized fuzzy controller.

13. McCarty, K., & Manic, M. (2014). A Database Driven Memetic Algorithm for Fuzzy Set Optimization. 7th International Conference on Human System Interaction. Lisbon, Portugal: IEEE.

**Abstract:** Fuzzy logic provides a natural and precise way for humans to define and interact with systems. Optimizing a fuzzy inference system, however, presents some special challenges for the developer because of the imprecision that is inherent to fuzzy sets. This paper expands upon an earlier development of a fuzzy framework, adding components for dynamic self-optimization. What makes this approach unique is the use of relational database as a computational engine for the memetic algorithm and fitness function. The new architecture combines the power of fuzzy logic with the special properties of a relational database to create an efficient, flexible and self-optimizing combination. Database objects provide the fitness function, population sampling, gene crossover and mutation components allowing for superior batch processing and data mining potential. Results show the framework is able to improve the performance of a working configuration as well as fix a non-working configuration.

## BOOK CHAPTERS

14. McCarty, K., Manic, M., & Stan, D. (2009). Contextual Data Rule Generation for Autonomous Vehicle Control. In T. Sobh, & T. Sobh (Ed.), *Innovations and Advances in Computer Sciences and Engineering* (Vol. 1, pp. 123-128). Bridgeport, Connecticut, USA: Springer-Verlag.

**Abstract:** Traditional techniques for the construction of Decision Trees often create trees which are overly large, ambiguous or both. This paper builds upon prior research by the authors of an algorithm for using fuzzified trees and fuzzy type-2 contexts to improve searching and usability [1]. These type-2 contexts are derivable using a variety of advanced data mining techniques such as a back-propagation neural network, *K*-means and Bayesian algorithms. Applying these hierarchical classifiers can draw new and meaningful contextual information from an existing tree. This paper presents an algorithm and metrics for applying these techniques to the output of a decision tree, creating new and meaningful contexts for the underlying tree. A test example demonstrates acquisition of new knowledge and over 90% contextual reduction of a decision tree from a commercial algorithm.

## MASTER'S THESIS

15. McCarty, K (2008). Applications of Modern Heuristics and Advanced Data Mining Techniques, University of Idaho

**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.

## GLOSSARY OF TERMS

ADL	Algorithm Definition Language, XML or data record used to describe and configure a working algorithm
ADMT	Advanced Data Mining Technique
DBMS	Database Management System
DT	Decision Tree
FC	Fuzzy Context
FC-OAP	Fuzzy Context Optimization Abstractions for Processes
FLC	Fuzzy Logic Controller – an decision process built on a Fuzzy Inference System
FIS	Fuzzy Inference System
GA	Genetic Algorithm
LSA	Local Search Algorithm
LST	Local Search Technique
MA	Memetic Algorithm
NFS	Non-stationary Fuzzy Sets
PFS	Polymorphic Fuzzy Signatures
RDBMS	Relational Database Management System/Software
SD	Situational Discontinuity – a transition from one state to another, very different state
SDP	Situational Discontinuity Problem – a problem with one of more Situation Discontinuities
T1-FIS	Type-1 Fuzzy Inference System
T2-FIS	Type-2 Fuzzy Inference System
T1-FLC	Type-1 Fuzzy Logic Controller
T2-FLC	Type-2 Fuzzy Logic Controller
T1-C	Type-1 Contextual Implementation
T2-C	Type-2 Contextual Implementation
TCF	Type-C Framework