Understanding Search-Based Software Engineering

by

Paul William "Will" McBurney

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Arun A. Ross, Ph.D.
James D. Mooney, Ph.D.
Tim Menzies, Ph.D., Chair

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2012

Keywords: Multiobjective decision making, Search-based software engineering, Stochastic searching, Associative rule-learning, Pareto optimization

Copyright 2012 Paul William "Will" McBurney

للاستشارات المنارة
www.manaraa.com

UMI Number: 1520737

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

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

UMI 1520737

Published by ProQuest LLC (2012). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code

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

المنارة للاستشارات

www.manaraa.com

Abstract

Understanding Search-Based Software Engineering

by

Paul William "Will" McBurney
Master of Science in Computer Science

West Virginia University

Tim Menzies, Ph.D., Chair

Pareto Optimization is a method of finding the space of best solutions possible given multiple class dimensions. In practice, however, it is difficult for users to visualize how solutions are spaced across this multi-dimensional frontier. Existing algorithms like NSGA-II are capable of quickly finding the Pareto frontier. However, users cannot use these algorithms to understand how changes to a solution effect the qualities of that solution.

In response to these issues, this paper presents "HOW". Not only is "HOW" capable of finding the Pareto frontier across multiple dimensions, but HOW can also explain how each point was derived, and compare points along the frontier to their neighbors. HOW generates rules using a stochastic learner, then combines those rules into a subsumption network via formal concept analysis. The network is augmented with statistics on the training data selected by sub-networks. Users can navigate their decision space by walking that network, selecting regions containing properties they want to avoid or encourage.

المنارة للاستشارات

www.manaraa.com

Dedication

For Mary Lou "Grammy" Hobbs.

المنارة للاستشارات
www.manaraa.com

Acknowledgements

I would like to thank Dr. Menzies for giving me the opportunity to do something truly interesting and unique in my grad work. Dr. Menzies is the reason I have chosen to pursue academia as a career. His insight and knowledge of his field is beyond exemplary, and he was also able to apply that knowledge in coming up with new ideas to test.

I would also like to thank Dr. Mooney and Dr. Ross for being on my committee. I have been fortunate enough to take classes from these professors, and both of them were excellent instructors. Both truly wanted students not just to learn the material, but apply it. I feel I learned a lot from them.

I would like to thank Brian Powell, the Course Coordinator for Computer Science 101: Introduction to Computer Applications, for helping me learn to be an effective teacher. His guidance, along with Dr. Tim Menzies, has helped me develop the skills to match my passion for teaching.

Finally, I wish to thank my parents for continuing to postpone their retirement from parenthood, allowing me to be a college kid just a little while longer. Their support is the only reason I've been able to get this far.

المنارة للاستشارات
www.manaraa.com

Contents

Acknowledgements iii

List of Figures vii

List of Tables viii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Statement of thesis . . . 3
- 1.3 Contributions of this Thesis . . . 3
- 1.4 Structure of this Thesis . . . 4

2 Related Work 5
- 2.1 Multi-Objective Optimzation . . . 5
- 2.2 Data Mining Metrics . . . 7
- 2.3 Defect Prediction . . . 10
- 2.4 WHICH . . . 11
- 2.5 Next Release Problem . . . 13
- 2.6 Mathematical Models . . . 16
- 2.7 NSGA-II . . . 17
- 2.8 Discretization . . . 19
  - 2.8.1 Equal-Width . . . 19
  - 2.8.2 Equal-Frequency . . . 20
  - 2.8.3 Fayyad-Irani . . . 20
- 2.9 PROMISE data . . . 22
- 2.10 Formal Concept Analysis . . . 22
- 2.11 Lattice Miner . . . 24

3 How 31
- 3.1 Introduction . . . 31
- 3.2 Data Input . . . 32
- 3.3 Rules . . . 34
- 3.4 Score vs. Utilities . . . 34
- 3.5 Combining . . . 35

المنارة للاستشارات

www.manaraa.com

3.6 Algorithm . . . 37
3.6.1 Discretization . . . 37
3.6.2 Rounds . . . 37
3.6.3 Scoring . . . 39
3.7 Output . . . 39
3.8 Handling Mathematical Models . . . 40
3.8.1 Algorithm . . . 40
3.8.2 Crowd Pruning . . . 41

4 Experiments with HOW **43**
4.1 Different Test Models . . . 43
4.2 Mathematical Models . . . 44
4.3 Defect Data . . . 49
4.4 Fitness Heuristic . . . 54

5 Conclusion **59**
5.1 Overview . . . 59
5.2 Findings . . . 60
5.3 Future work . . . 60

A Source Code **62**
A.1 Defect Code . . . 62
A.1.1 3will.lisp . . . 62
A.1.2 boot.lisp . . . 67
A.1.3 defstruct.lisp . . . 68
A.1.4 how.lisp . . . 72
A.2 FrontierFinder . . . 79
A.2.1 3will.lisp . . . 79
A.2.2 boot.lisp . . . 83
A.2.3 defstructs.lisp . . . 83
A.2.4 ruletree.lisp . . . 88
A.2.5 testtables.lisp . . . 91
A.2.6 how.lisp . . . 102
A.3 Other . . . 107
A.3.1 angle.lisp . . . 107
A.3.2 bestof.lisp . . . 107
A.3.3 data.lisp . . . 108
A.3.4 deftest.lisp . . . 110
A.3.5 hash.lisp . . . 111
A.3.6 lib.lisp . . . 112
A.3.7 hash.lisp . . . 113
A.3.8 macros.lisp . . . 113
A.3.9 profile.lisp . . . 114
A.3.10 random.lisp . . . 118
A.3.11 structs.lisp . . . 119

المنارة للاستشارات
www.manaraa.com

A.3.12 trapezoid.py 120
A.4 Data 120
A.4.1 weather.lisp 120
A.4.2 pbeans.lisp 121
A.4.3 schaffer.lisp 121
A.4.4 poiwhich.txt 122

Bibliography **123**

المنارة للاستشارات
www.manaraa.com

List of Figures

2.1 A sample of a Pareto solution using the Schaffer problem 6
2.2 Concept Lattice of Table 2.3 23
2.3 A sample .lmb file 25
2.4 Lattice Miner User Interface 26
2.5 Lattice Miner: Concept Lattice View 27
2.6 XML output generate from previous Concept Lattice View 28
2.7 An example of a generated Concept Lattice output image. 28
2.8 Neighborhood printout 30

3.1 The Weather dataset using How's LISP data input 33
3.2 A sample rule 34
3.3 Two rules to be combined 36
3.4 The combined rule 36

4.1 Schaffer 46
4.2 Fonseca 47
4.3 Kursawe 48
4.4 Srinivas 50
4.5 Constr-Ex 51
4.6 Tanaka 51
4.7 Xerces 52
4.8 Velocity 52
4.9 Neighborhood printout 57
4.10 The Pareto frontier of the poi solution space found using B_1 and Sum as heuristics 58
4.11 The Pareto frontier of the velocity solution space found using B_1 and Sum as heuristics 58

www.manaraa.com

List of Tables

2.1 Confusion Matrix 8
2.2 different unsupervised discretization methods on the hypothetical set S. . . 21
2.3 Color Components 23

3.1 Symbols in the column names and their meanings 33

4.1 Unconstrained standard mathematical test problems. Note: all objectives minimized 45
4.2 Constrained standard mathematical test problems. Note: all objectives minimized. 49
4.3 Comparison of the Sum and B_1 hueristics 56

المنارة للاستشارات
www.manaraa.com

Chapter 1

Introduction

1.1 Motivation

Software Engineering is a problem of trade-offs. In software engineering, you can never perfectly meet the needs of every objective. A project is always going to have costs that a manager wants to minimize with goals that may break the budget. Thus, it is important to find the set of solutions, or space of solutions, that best optimize these multiple objectives. This process is referred to as Search Based Software Engineering. This thesis will focus on a methodology for finding this space of solutions.

According to Harman, solutions to a problem that reflect on the neighbourhood are more useful than precise solutions that do no reflect on the neighbourhood[12]. Harman refers to understanding a neighbourhood of solutions as an open and pressing issue in search-based software engineering. "[R]esearch on SBSE has tended to focus on the production of the fittest possible results. However, many application areas require solutions in a search space that may be subject to change. This makes robustness a natural second order property to which the research community could and should turn its attention [12]"

This tells us that when we find solutions in search-based software engineering, we want to properly explain the neighbourhood of our solution in detail. We want understanding of how multiple solutions in a neighbourhood are related to one another, and in what way.

المنارة للاستشارات

www.manaraa.com

This is especially important when discussing the Pareto frontier. The Pareto frontier is, by definition, a space of points which optimize multiple dimensions, not an individual point. By analysing how these points were arrived at using the HOW algorithm described within this document, we can create user understandable explanations of a space without the decreased performance typically expected with explanation.

Multi-objective optimization is key in requirements engineering. The process of requirements engineering is inherently multi-objective [26], since it involves trying to meet the needs of multiple stakeholders with different, often conflicting goals [26]. Heaven say multi-objective optimization "can support decision making among large numbers of design alternatives" [13]. Developing software is at minimum a bi-objective problem between the competing objectives of cost and value [14]. In software engineering, developers often have to choose which requirements to focus on to maximize customer value while minimizing developer cost [14].

The next-release problem is a common example of multi-objective decision making [30]. Zhang describes multi-objective analysis, explaining "requirements engineering is characterised by the presence of many complex and conflicting demands, for which the software engineer must nd a suitable balance[30]". The next-release problem is typically viewed as bi-objective [30]. The goal is to select a subset of features to include in the next release of a project that minimize cost to the developing firm, while maximizing the value of the product being delivered [30].

In order to more empower a user to make decisions, especially in requirements engineering, it will be useful to explain the frontier of multi-objective optimizations. When a user is given an understanding of search space's neighborhood, that user is capable of making informed decisions about how to move across Pareto frontier. Giving the user this information should help maximize their understanding and use of the solution space for a problem. HOW was developed to do just this: add explanation and comparison to the solution space of multi-objective optimization in order to help the user better understand the options available.

المنارة للاستشارات
www.manaraa.com

Harman, in describing the state of search-base software engineering, pointed out a need for a deeper understanding of problems and solutions when he said, "However, in order to develop the mature roots that allow the eld to grow, the second phase of exploration requires a deeper understanding of problem and solution characteristics[12]." HOW is an attempt to meet that need by explaining the solution space, offering a user explanation of how a particular solution was arrived at, and how that solution compares to related, or neighbouring, solutions. In doing so, HOW seeks to help realize Harman's goal [12] of providing an understanding of the searching process, in order to maximize understanding of the solution space.

1.2 Statement of thesis

HOW, an algorithm developed from WHICH[19], is designed to provide a solution space in multi-objective optimization. However, unlike existing multi-objective optimizers such as NSGA-II [1], HOW provides an explained general solution for each point found along the Pareto frontier of a particular search space.

This thesis will show the potential for multi-objective optimization to allow explanation without a degradation in results sometimes associated with explanation. It will show that HOW is a capable multi-objective optimization learner

1.3 Contributions of this Thesis

The HOW algorithm and this thesis serve four primary purposes:

1. HOW expands the existing WHICH [19] algorithm to allow for multi-objective optimization rather than single-objective optimization.

2. HOW can find the Pareto optimal frontier on existing test problems for multi-objective optimization, showing it is satisfactory as an optimization tool.

3. HOW can provide a solution space with that can serve to illustrate the neighbourhood of existing solutions, comparing them to similar or related solutions.

المنارة للاستشارات
www.manaraa.com

4. HOW's customizable fitness metric can be changed for different needs. This thesis will study the effects of changing the heuristic scoring aggregate fitness function on defect prediction.

We will show that HOW can be a high quality multi-objective optimization tool that is capable of explaining the neighbourhood of its solution space.

1.4 Structure of this Thesis

The rest of the paper is structured as follows. Chapter 2 gives relevant background information including where the data was acquired, a background on data mining and multi-objective optimizaion. Chapter 3 explains the HOW algorithm, including how to understand the output. Chapter 4 shows the results of various experiments on HOW, including mathematical problem performance and defect prediction results. Chapter 5 summarizes the findings and their impacts, and suggest future work.

المنارة للاستشارات

www.manaraa.com

Chapter 2

Related Work

2.1 Multi-Objective Optimzation

Multi-Objective optimization is a process where 2 or more objectives, frequently conflicting, are optimized. It is usually impossible to derive a single solution where all objectives are fully optimized[1]. Usually, there is a set of Pareto-optimal solutions, no solution of which optimizes all objectives better than any other solution along the frontier of Pareto-optimal solutions. A given point is Pareto-dominated when their exists another point that optimizes all dimensions at least as well and at least one dimension better than the original point. Most software engineering problems are multi-objective in nature[12] (including defect prediction). Yet, while mapping the Pareto frontier is a useful visual tool to help designers understand the trade-offs in the solution, there has not been as much work in connecting the outputs to the input parameters selected. Veerappa notes that it is very possible that two outputs which are similar may have wildly different input parameters, a possibility the Pareto frontier of the solution space does not express well on its own[26].

In Figure 2.1, we see an illustration of the Schaffer bi-objective problem's Pareto-optimal curve[2, 23]. In Schaffer, the goal is to minimize the two given objective functions[1, 23]. The labeled point "A" is along the Pareto optimal curve. "A" does not dominate, nor is not dominated by, any solution along the curve. Point "B" fails to optimize either objective of Schaffer as well as "A", and is thus not part of the Pareto frontier. "C" would dominate

المنارة للاستشارات

www.manaraa.com

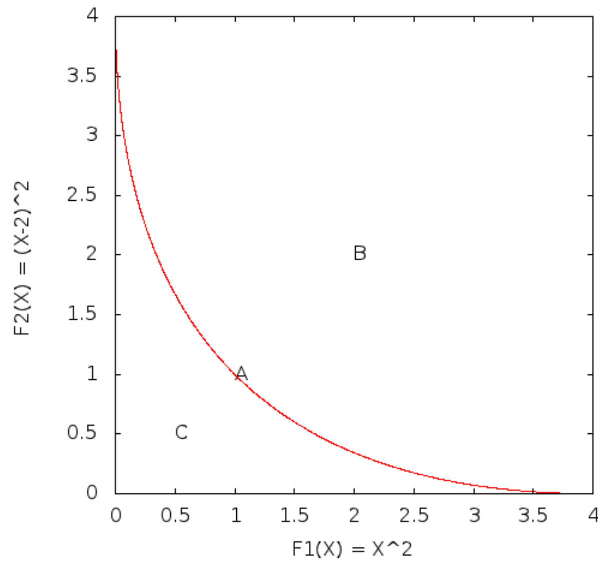

Figure 2.1: A sample of a Pareto solution using the Schaffer problem

"A", as well as much of the Pareto optimal curve. However, "C" is not possible to acheive with the given objective functions. A theoretical perfect point in multi-objective, where all objectives are completely optimized, is frequently called "heaven". It's very rare that a point at heaven actually exists. Rather, the goal of finding new points in the Pareto optimal region is to approach heaven. By contrast, the point where all objectives are completely non-optimized can be called "hell". In Figure 2.1, "heaven" would be the point of origin, and "hell" would be (∞, ∞).

The major advantage of Pareto optimization is that a Pareto optimal frontier can clearly illustrate the relationships between the objective functions. This frontier will help a user understand the trade-offs in the objectis caused by altering the input vector. HOW will take this one step farther by showing how the input maps to the output, allowing a deeper understanding of how to move across the Pareto frontier.

Pareto optimization forms the basis for HOW's multi-objective selection process. This is in fitting with most existing multi-dimensional optimization techniques[1, 12, 22, 27, 32]. Pareto optimization is not capable of saying how much "better" one solution is than

www.manaraa.com

another, only that one solution is better than another[12].

Pareto optimization provides a space of solutions, allowing a human to choose which point along the frontier may best optimize their needs in real world settings. HOW seeks to empower this process by provided an explanation of the points along the frontier to the user.

Pareto optimization is not the only solution to multi-objective optimization. Another common solution is the aggregate fitness function[12], which takes the multiple objective functions $f_1, ..., f_n$ and adding them together with constant weights on each objective to denote relative importance.

$$F = \sum_{i=1}^{n} c_i f_i(\vec{x})$$

The goal would be to find the input vector $\vec{x}$ that maximize F. Objectives considered more important will have a larger value of c_i. For example, in a hypothetical situation with two objective functions, the fitness function may look like:

$$F = f_1(\vec{x}) + 3 * f_2(vecx)$$

This would imply that optimizing f_2 is three times as important as optimizing f_1. This function, however would mean that a large improvement in f_1 could be overshadowed by a small drop in f_2. This is the primary reason why Pareto optimization is generally considered superior to an aggregate fitness algorithm [12]. By providing a space of solutions, Pareto optimization is more likely to give a user better understanding the possible solutions of the problem. The aggregate function can't give this same picture.

2.2 Data Mining Metrics

Standard metrics for data mining need to be used to explain the performance of HOW. This section will show the common measurements that can be used to explain performance of classification. This section will not focus on the metrics of non-classification optimization problems, since metrics used in those problems are typically specialized for

المنارة للاستشارات
www.manaraa.com

		Predicted	
		False	True
Actual	False	a	b
	True	c	d

Table 2.1: Confusion Matrix

the data, or simply an average of an existing field. Any numeric metrics will be explained alongside their relevant data.

It is first important to understand the confusion matrix. A confusion matrix breaks down a test dataset where a classification method was applied. For each class, the number of true negatives, false negatives, false positive, and true positives are calculated.[29] Table 2.1 is an example of a confusion matrix. Each cell in the table is assigned a letter. The variables a,b,c, and d are assigned respectively to true negative, false negative, false positive, and true positive. [29]

Now that we have established the values, we can define our metrics[29] using the variables a,b,c, and d.

$$pd = recall = d/(d + b)$$

$$pf = calsealarm = c/(a + c)$$

$$prec = precision = d/(c + d)$$

$$accuracy = (a + d)/(a + b + c + d)$$

$$support = (c + d)$$

For the purposes of this paper (and the HOW algorithm), support is simply the sum of c and d. Support will somtimes be mentioned as a percentage or records predicted to be true over the number of all records, which can be calculated using the formula below.

$$support = (c + d)/(a + b + c + d)$$

المنارة للاستشارات

www.manaraa.com

When working with classification problems, one generally seeks to optimize *pd*, *prec*, and *accuracy*. By contrast, *pf* is typically minimized, since you generally want a low false alarm rate.

Pd and *pf* tend to be directly related. Generally, as *pd* increase, so too does *pf*[19, 28]. Since we want to minimize *pf*, but maximize *pd*, these objectives tend to be in direct conflict. The importance of optimizing individual metricss depends on the situation. For example, on a security system, a low *pd* may be acceptable in order to keep *pf* low, since false positives are more damaging than a false negative. Generally, projects can fall into two groups: risk averse and cost averse. A risk averse system may tolerate a large false alarm rate in order to ensure catch any major faults, such as human safety systems. Cost averse systems are more limited by budget than by safety risks, and will usually tolerate diminished *pd* in order to not waste energy and finances on false alarms.[19]

The advantage HOW provides is an allowance to maximize or minimize whichever fields we deem important, and ignore those we don't find important.

In defect prediction, we generally want to measure the number of defects and number of lines of code[19]. While other static features can be use [19], I generally chose these two for many of the experiments in Chapter 4. If we seek to classify all records in a defect prediction data set, we want to divide the records into two groups: those that have zero defects, and those that have one or more defects. When using classification on defect prediction datasets, measure such as *pd* and *pf* typically refer only to classifying the class with one or more defects. Unless otherwise mentioned in this article, assume all metrics on classification refer only to classifying the class with one or more defects.

We can also use numeric metrics. For instance, we can get a sum of the number of defects found, which we would seek to maximize, and a sum of the lines of code found, which we would seek to minimize. The goal is to find as many defects as possible while having to read through as few lines of code as possible.

المنارة للاستشارات
www.manaraa.com

2.3 Defect Prediction

Defect Prediction is the process of trying to automate finding the location of defects before a piece of software is used. Finding and correcting defects can be a very expensive process in software engineering, so we want to automate this process in an attempt to reduce cost of debugging. This has been attempted using data miners on historical examples, like those found in the PROMISE repository[7]. Defect prediction can be defined as a multi-objective optimization problem with several objectives. Ideally, a perfect rule (or, "heaven") would be a rule that finds all defects without reading any lines of code, with zero false alarm and perfect recall, precision, and accuracy. Of course, such a rule couldn't reasonably exist, since you would have to read at least some lines of code to find any defects. But this hypothetical perfect lets us define which parameters we seek to maximize and which we seek to minimize. We want to minimize false alarm and lines of code read (or effort). We want to maximize the number of defects found, recall, precision, and accuracy.

There is no reason we have to use every single one of these traits in optimization. We can select certain subsets of these objectives to serve as the utilities in an multi-objective optimization.

Research into better methods of using machine learners for defect predictions is wide-spread and varied. Yet despite this, overall performance has been relatively static. Even in the midst of several modern, advanced learning algorithms, Naive Bayes tends to provide benchmark results. [20].

The primary method of defect prediction is to learn on tables structured with a defects column treated as a boolean value (where "true" means the record has defects). Then, in standard data mining practice, the non-target attributes are analyzed to see which set of features may best predict the defect column.This form of classification learning will not help show how many defects there are, only help isolate which modules the defects are likely to occur in[20].

المنارة للاستشارات
www.manaraa.com

2.4 WHICH

WHICH[19, 21] is aan associative rule-based learning algorithm with user configurable searching criteria. WHICH was developed to improve on the stochastic associative rule learner TAR3[11, 21]. Milton criticized TAR3 as "over-elaborate", claiming that the algorithm WHICH is more generalized, while still maintaining TAR3's linear scalability [21]. HOW is an extension of Which that uses multi-objective optimization to find and explain the Pareto-optimal solution space. The weakness of Which that HOW improves on is that Which relies solely on a total aggregate function for optimization, effectively representing several fields as one. Which makes no account of Pareto dominance, and thus isn't capable of showing a broad range of trade-offs. HOW, by contrast, provides the Pareto frontier while clearly illustrating the trade-offs.

WHICH, like HOW, requires data to be discretized. In [19] Equal-Width discretization was used. In that study, the best results were found using 2-bin, 4-bin, and 8-bin discretization.[19].

WHICH maintains a stack of associative rules, which feature combination of ranges in features. A scoring metric B_1 is used to sort the effectiveness of the rules. B_1 is user customizable[19, 21]. In [19], Which is used for defect prediction. The objective function of the search, B_1 is define as

$$B_1 = 1 - \frac{\sqrt{PD^2 * \alpha + (1 - PF)^2 * \beta + (1 - effort)^2 * \gamma}}{\sqrt{\alpha + \beta + \gamma}}$$

This optimization is a form of aggregate fitness function for multi-objective optimization. Unlike HOW, Which does not use Pareto optimization as a tool for multi-objective optimization. This function output increases as PD increases, PF decreases, and effort decreases. PD and PF are defined the same was as they are in Section 2.2. Effort is defined as the percentage of the code base predicted to be faulty, and which must be analyzed in more detail by more expensive QA methods [19]. α, β, and γ serve as user-defined constants to either emphasize or ignore individual attributes. [19] found the most success with

المنارة للاستشارات
www.manaraa.com

$$\alpha = 1;\ \beta = 1000;\ \gamma = 0$$

This design is to heavily penalize false alarms while still attempting to optimize PD. Effort is ignored, as PF serves as a "surrogate measure" (since false alarms have to be analyzed in great detail along with the true positives).[19]

At the initial stage of Which, all rules are "atomic,"[19] that is, they contain only one element of one feature. Initially, which creates a list of atomic rules on all attributes (but not on the class) to classify each possible target class. These rules are all initially scored using the described multi-objective aggregate function. These rules can then be combined (see Section) to form more complex rules, which are in turn scored. The rules are not combined exhaustively, as that would lead to an exponential runtime[21]. Rather, the rules are combined stochastically, using the score of the aggregate fitness function as a weighting feature. Rules with a larger score on the aggregate fitness function are more likely to be selected than rules with lower scores. This is an attempt to bias the search towards things believed to be more successful without creating complete blind spots that may contain useful knowledge.

Which continues to combine rules together until one of two stopping conditions are met. The first condition is after a very large number of total rounds have elapsed, since it has been shown machine learning stops learning relevant new information after a time, referred to as the ceiling effect[20]. The other stopping condition is to check if you have gone a certain number of generations without improvement. This measure serves primarily for efficiency, as if a learning algorithm is no longer improving, there is no reason to go as many rounds as the first stopping condition requires.[21]

Which is structured as follows:

1) Discretize the input data.

2) Create a stack of rules. Generate possible atomic rules for each target class.

3) Stochastically select two rules from the stack, weighting your selection using

www.manaraa.com

the B_1 metric. Rules with a larger B_1 metric are more likely to be selected.

4) Combine the selected rules, score the combination, and put the combination into the stack.

5) Return to Step 3 until one of the exit condtions is met. The output is the stack of generated rules.

HOW preserves this same basic algorithm layout. HOW primarily changes the scoring function to allow for a score to be defined by multiple numeric values and metrics, not just one. HOW does use an aggregate fitness function to determine the weighting heuristic of a rule, but Pareto optimization is performed on the utilities vector which contains the set of all objective values. How also allows a user to specificy multiple numeric fields as target classes to be optimized or minimized. In defect prediction, success was found by minimizing the lines of code read while maximizing the number of defects found.

Which generally performed very well across a wide variety of datasets using $AUC(pd, effort)$ (That is, the area under the pd-effort curve generated) from the stack of rules Which generated. Which-2 (that is, which using 2-bin discretization) "won", in that it either had the best results or it's results were statistically comparable to the best results, in nine out of ten of the dataset tests reported. It is worth noting that on the last dataset, Which-2 performed very poorly.[19] This poor performance, though an outlier, may explain a similar phenomena that occurs in HOW, implemented later.

2.5 Next Release Problem

The Next Release Problem is an excellent example of a multi-object optimization problem in which a firm designing a product tries to select the features to include in the next release to maximize value and minimize cost[30]. All software firms have finite resources, so finding how to best allocate these resources to maximize the value of a product is a critical problem. Cost and value are conflicting objectives. The implementation of the Next Release Problem is an excellent example of syntactically defining a multiple-objective

المنارة للاستشارات
www.manaraa.com

problem, which is why, even though HOW does not test on the Next Release Problem, it was included in this document. Using HOW to solve the Next Release Problem could prove to be a worthwhile endeavour in the future. HOW's use of stochastically combined rules could provide very interesting insight into a given case of a Next Release Problem.

While it is possible to solve this problem using a single optimization "Knapsack Problem" implementation[30], there are several downsides to this approach. First, the knapsack problem is hard-NP[6], which means very large knapsack problems become very computationally expensive. Second, because the knapsack problem is, in essence, a single-objective optimizer, the knapsack problem has to be recalculated for each set of parameters tested. Multi-objective optimization, and Pareto dominance, provides a space of solution, so accounting for possible parameter change becomes easier[30]. This is especially true with HOW, which, in addition to finding the Pareto frontier of a space, explains the relationships in the frontier neighborhood.

Zhang, et al. were the first to publish a paper formally notating the Next-Release Problem as a multi-objective optimization problem[30]. The Next Release Problem was defined as follows:

Assume for a given software system there exist a set of customers C such that

$$C = \{c_1, c_2, ..., c_m\}$$

and a set of requirements R such that

$$R = \{r_1, r_2, ..., r_n\}$$

All requirements are assumed to be independent.[30] Each requirement has an associated cost, which is represented by

$$Cost = \{cost_1, cost_2, ..., cost_n\}$$

Each customer is weighted by the company differently.[30] The weight a company put on each customer is represented by

المنارة للاستشارات
www.manaraa.com

$$Weight = \{w_1, w_2, ..., w_m\}$$

where

$$\sum_{j=1}^{m} w_j = 1$$

Each customer uniquely weights the importance of each requirement. This value is represented by $value(r_i, c_j)$. If requirement customer c_j is provided r_i, then $value(r_i, c_j) > 0$. Otherwise, $value(r_i, c_j) = 0$.[30]

From these defintion, we can define a scoring function as

$$score_i = \sum_{j=1}^{m} w_j * value(r_i, c_j)$$

The input vector is the set of requirements to be met. This is represented by $\vec{x} = \{x_1, x_2, ...x_n\} \epsilon \{0, 1\}$ where $x_i = 1$ denotes that r_i will be included in the next release.[30].

We can now define the two competing abilities of value and score[30].

$$f_1(\vec{x}) = \sum_{i=1}^{n} score_i * x_i$$

$$f_2(\vec{x}) = \sum_{i=1}^{n} cost_i * x_i$$

The value of a particular solution is denoted by $f_1(\vec{x}$, and the cost of that solution is $f_2(\vec{x})$. The goal, as stated before, is to maximize the value while minizing cost.

HOW is not tested using any Next Release Problems, however the principle of multi-objective optimization to find a Pareto frontier is present. Zhang, et al,[30] were very effective at creating a well-defined syntax and equation set for the Next Release Problem. This serves as an excellent example of how to define existing problems not previously seen having multiple objectives as multi-objective optimization problems.

Zhang's, et al. solution is not the only attempt to solve the requirements problem, of course. Another solution provided by Karlson and Ryan relied on prioritizing the

المنارة للاستشارات
www.manaraa.com

requirements base on the relationship between the cost and the value[14]. This method derives the a cost and value measure, and plots it onto a chart. The section of the chart with high value and lost cost is labelled "high" priority, whereas high cost and low value is labelled "low" priority. However, because the borders of these regions are definied by the proportion of cost to value linearally, this isn't a Pareto dominance solution. This is an example of an aggregate fitness function solution.

2.6 Mathematical Models

There exist several mathematically defined multi-objective optimization problems which serve as a proving ground for multi-objective optimizers[1, 5, 16, 23, 24, 25, 27, 31]. Tables defining these problem sets can be seen in Table 4.1 for unconstrained problems and Table 4.2 for constrained problems. The difference between constrained and unconstrained problems is constrained problems have one or more constraint functions which the input vector must satisfy, in addition to limits on the individual inputs [1, 27]. These models are tested with HOW in Chapter 4.

The unconstrained problems we have specific results for are Schaffer [23], Fonseca[5], and Kursawe [16]. There are also results for the ZDT family of problems[31] not in this document. Also included is an analysis of some constrained problems: Constr-Ex[1], Srinivas[24], and Tanaka[25]..

All of these problems have two or more objective functions derived from an input vector. In every multi-objective problem we analyze, the objective functions are to be minimized[1, 27] The input vector can be as small as one dimension such as in Schaffer[23], or as large as 30 dimensions in ZDT1 and ZDT2[31]. The size of the input vector affects the complexity of finding the optimal frontier.

The dimensionality of the problem solution space, equal to the number of objective functions[27] also affects complexity of finding the Pareto optimal frontier[1]. While all problems cited above are bi-objective, there are other problems, like Water[33], which have

المنارة للاستشارات
www.manaraa.com

more than 2 objectives. Water has 5 objective functions, and thus has a five-dimensional solution space[33]. This thesis focuses on bi-objective functions because they are the easiest to represent in charts and graphs.

These mathematical multi-objective problems were coded within the framework of HOW. The data is generated randomly based on the constraints generated by HOW to find the solution space. More detail on how this is accomplished is in Chapter 3.

2.7 NSGA-II

Several multi-objective optimizers exist[1, 2, 10, 12, 15, 18, 22, 32] that operate on a variety of methodologies and principles. NSGA-II has become a very popular benchmark in multiobjective optimization.[10, 15, 22]. It is that popularity that warrants a section devoted to explaining how NSGA-II functions as a multi-objective optimizer.

NSGA-II (Nondominated sorting-base multi-objective evolutionary algorithm II) is an evolutionary algorithm designed to improve on NSGA[1]. NSGA was one of the first evolutionary algorithms designed to find a space of Pareto optimal solutions in a single simulation run[24].

NSGA used a nondominated sorting procedure level to rank Pareto dominance of a dataset into a nondominated ranking. A ranking of 1 was a solution that was not dominated. A ranking of 2 meant the point was dominated exactly once, a ranking of 3 meant it was dominated exactly twice, etc. The larger the ranking, the less fit the solution was. NSGA essentially identifies the best Pareto optimization candidates using this nondomination ranking method.[1]

NSGA, however, had three main problems that NSGA-2 sought to correct. The first problem was a runtime on the order of $O(MN^3)$ where M is the number of objectives and and N is the size of the input data. This was because in order to find the highest ranked values, you had to compare every item in the data with everything other item for each of M objects, which translated to $O(MN^2)$. To find the second highest ranked values,

www.manaraa.com

it was also $O(MN^2)$. Each ranking coulde be traced to that same upper bound. In a worst case, there would be be an N number or rankings (where each ranking would only have one element), meaning the process would be $O(MN^3)$.[1]

NSGA-II avoids this by initially, for each point in the data, creating two variables. All the points are placed in the set P. The first variable, n_p is initialized to zero. n_p keeps track of how many points in the data dominates the point p. S_p is the set of all points which p dominates. To populate n_p and S_p for all points in P takes $O(MN^2)$ comparisons. After this is done, each point in P with $n_p = 0$ is non-dominated and in the first front. For each p where $n_p = 0$, S_p is iterated through and all points in S_p have n_p decremented. If after a decrement an n_p becomes zero, that solution is moved to the set Q. Once this is complete on the set P, the process is repeated on Q to find the third front, and continues from there. This reduces the complexity of NSGA to $O(MN^2)$.[1]

NSGA lacked elitism, which was the second criticism that NSGA-II addressed. Elitism is the practice in evolutionary algorithms of maintaining a "best" population from generation to generation. If a solution in a current generation is inferior to one or more solutions in a previous generation, having elitism allows you to maintain the previous solution and "throw out" the less valuable new one.[1]

The third criticism that NSGA-II tried to respond to was to remove the need for specifying a share parameter, which was used to improve the "spread" of the data. It was desirable to remove this parameter, as it was a required user-defined value that greatly impacted the results of NSGA. Automating the spreading was accomplished by using a crowding-distance measure defined in NSGA-II. The crowding distance measure normalized each objective function, sorting the data on that objective, and using the distance to the two nearest neighbors as a crowding measure.[1]

The NSGA-II algorithm initially creates a random parent population of size n, and assigns each member a fitness rank (as described above). From there, standard genetic algorithm operators (such as combine, mutate, etc) are used to create the offspring population of size n. The parent and offspring population are combined and sorted by non-

المنارة للاستشارات
www.manaraa.com

denomination. After sorting by non-denomination, we sort each nondenomination rank by the crowding-distance measure. We then take the top n items, choosing first the items with the best nondenomination rank, then when the remaining spots in the next generation is smaller than the size of the next nondenomiation rank, we selecte the items with the large crowd-distance measure (this is to encourage spread across the frontier). NSGA-II can be run for however many generations the user chooses. [1]

2.8 Discretization

HOW will require discretized input. Descritization is the process of taking numeric data and converting the data into non-scalar discreet clusters. There are three primary methods of discretization: equal-width, equal-frequency, and Fayyad-Irani[4].

Discretizing data adds another dynamic that can be measured. If you do not discretize your data with a large enough number of bins, it's possible you will lack the detail to be able to accurately classify certain objects. However, if you discretize with large numbers of bins, you run the risk of overfitting your solutions[28].

2.8.1 Equal-Width

Equal-width discretization is arguably the easiest to implement. Equal-width is an unsupervised discretization method, meaning each attribute is discretized independently of other attributes in the dataset, including the target class.[28]. To discretize one numeric attribute, you first have to iterate through the data in that attribute to find the minimum and maximum items. You can then assign that attribute to one of n bins. This process takes linear time, since you only need to pass through the data twice. This is done by using the following formula:

$$bin = \frac{val - min}{max - min} * n$$

. Equal width does not generate equally populated bins unless the data is distributed evenly. Bins can sometimes be completely empty depending on the data distribution . Having vast

المنارة للاستشارات
www.manaraa.com

outliers can cause this form of discretization problems, as you can end up with several very sparsely populated bins, incapable of providing enough support to reliably learn on. You can also end up with a small number of vastly populated bins, which crowd out the rest of the fields in the attribute.[28]

2.8.2 Equal-Frequency

Equal frequency is another form of unsupervised discretization. As the name implies, the goal of equal frequency is to discretize an attribute in such as way that each discretized bin is as near the same size to the other bins as possible. For example, if you had 100 samples, and discretized your data to 10 bins, each bin would contain 10 samples regardless of the data distribution.[28]

This is done by first sorting the data. Then, you simply find your bin size by dividing the number of records in the attribute by the number of bins you want to have:

$$sizeofbins(insamples) = \frac{n}{\#ofbins}$$

This approach leaves you with more bins of consistent size. However if there are natural "gaps" in the data between multiple classification types, this form of discretization may miss those. That said, it is still relatively easy to implement[28].

Table 2.2 gives a hypothetical case, showing how a given a set of numbers would be discretized in both equal-width and equal-frequency. The table makes the assumption that the given numeric set is being discretized to 2-bins.

2.8.3 Fayyad-Irani

Fayyad-Irani is the only supervised discretizer this paper will give attention to. This form of discretization is supervised because it takes the target class into account in its discretization[4]. That is, we don't discretize a single numeric attribute in a vaccuum, we take other attributes into account.

المنارة للاستشارات
www.manaraa.com

Discretization		
S_i	Equal-Width (n = 2)	Equal-Frequency (n = 2)
0	Bin 1	Bin 1
15	Bin 1	Bin 1
22	Bin 1	Bin 1
23	Bin 1	Bin 1
27	Bin 1	Bin 1
31	Bin 1	Bin 1
40	Bin 1	Bin 1
44	Bin 1	Bin 2
46	Bin 1	Bin 2
61	Bin 2	Bin 2
63	Bin 2	Bin 2
80	Bin 2	Bin 2
92	Bin 2	Bin 2
100	Bin 2	Bin 2

Table 2.2: different unsupervised discretization methods on the hypothetical set S.

In Fayyad-Irani discretization, we sort the existing numeric attribute, then try to find a spliting point that best reduces the entropy on both sides of the split of the target class in the data[4]. The entropy is, more or less, a measure of how "mixed up" the target class in the data is. The entropy formula is:

$$Ent(S) = -\sum_{i=1}^{k} P(C_i, S)\log(P(C_i, S))$$

In this formula, $P(C_i, S)$ means the proporation of samples in S that are of class C_i. k is the number of classes in the dataset S belongs too. Fayyad-Irani seeks to find the split point to minimize the weighted sum of entropy on either side of the split. The goal is to find the point that divides S into S_1 and S_2[4].

$$E(A, T;\ S) = \frac{|S_1|}{|S|}Ent(S_1) + \frac{|S_2|}{|S|}Ent(S_2)$$

Fayyad-Irani then recurses on S_1 and S_2, repeating the process until one of two conditions are met. The first condition is having a "pure" set (where all elements in the set are of the same class). The second condition is the Minimum Description Length Principle

المنارة للاستشارات
www.manaraa.com

(or MDLP)[4]. In short, we stop when the Gain, the improvement in weighted entropy generated by the split, is no longer substantial enough to warrant the split.

2.9 PROMISE data

The PROMISE[1] dataset provided the defect prediction data for all experiments. PROMISE is a large software engineering database of volunteered project data. This data is available for experimentation in various aspects of software engineering, including defect prediction and effort estimation.

2.10 Formal Concept Analysis

Formal Concept Analysis[8, 9] is a way of mathematically grouping concepts into a form of ordered structure. Formal Concept Analysis can be used to group objects together based on common properties in order to better understand the structure of a space of objects. A concept lattice is a product that can be derived using formal concept analysis. The idea of a concept lattice is to graphically repreesnt objects and how they relate to other attributes using a web structure.

For example, say we had the data in Table 2.3, which shows how common colors are represented in an RGB scale. A check mark in a cell means that a color will need at least some level of color from the column given (for example, Cyan is combined using Green and Blue, but not red. Thus, Green and Blue are marked, and red is blank). From this table, we can use a Concept Lattice to illustrate how each object (or color) is related to other colors by the component colors. A concept lattice in Figure 2.2 generated using Lattice Miner (described in the following section) illustrates these relationships.

[1] G. Boetticher, T. Menzies and T. Ostrand, PROMISE Repository of empirical software engineering data http://promisedata.org/ repository, West Virginia University, Department of Computer Science, 2007

www.manaraa.com

Colors			
Color	Red	Green	Blue
Red	X		
Orange	X	X	
Yellow	X	X	
Green		X	
Blue			X
Violet	X		X
Magenta	X		X
Black			
White	X	X	X

Table 2.3: Color Components

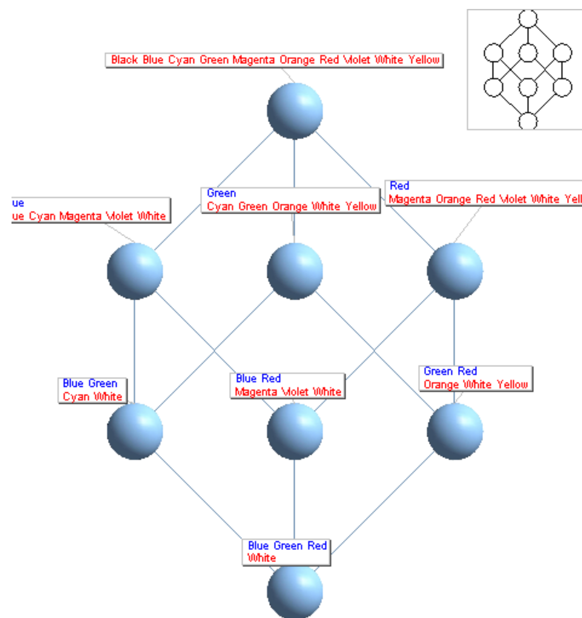

Figure 2.2: Concept Lattice of Table 2.3

المنارة للاستشارات
www.manaraa.com

2.11 Lattice Miner

Lattice Miner is a Java-based formal concept analysis tool designed to generate concept lattices of an input database[17]. By simply writing a script that translates the output rules into the proper Lattice Miner input file format, Lattice Miner can be used to create a concept lattice based on our rules[17]. Advantages to using Lattice Miner include user simplicity as well as easy to reproduce file structures (a major advantage in scripting). It is also advantageous that Lattice Miner is Java-based, as this can run on many platforms. Most importantly, Lattice Miner is a very fast way to generate a concept lattice structure[17]. In using Lattice Miner, however, we have discovered that it limits the number of rules and number of rule attributes, which indirectly limits the number of dimensions you can optimize with HOW and still be able to produce a concept Lattice. This is because optimizing on a larger number of dimensions creates a larger output rule set, due to an increase in the size of the Pareto frontier of the rule space.

The input for Lattice Miner is a .lmb file. This file contains several key portions.

The first line, "LM_BINARY_CONTEXT" refers to the type of context our Lattice Miner will use. Binary Context means that each attribute for a rule is defined by 1 if the attribute is present, and 0 if that attribute is not. There are other contexts in Lattice Miner, but we will only focus on Binary, since that is the only context that is relevant to this project.

The next line list the name of all our records. We replace the name with parseable information about the PD (recall), PF (false alarm rate), and PREC (precision) of our rules, since these were the three dimensions we optimized on. This allows us to access a rule's score by using the name feature, which means there is no modification to Lattice Miner needed. Each row name is separated by a | symbol.

The next line [which, in Figure 2.3, starts "|$CAM:less-than-median" is the name of each column. Each column refers to one atomic rule (that is, a rule with one element in one attribute). For each row, any column where the rule contains that attribute is set to 1.

المنارة للاستشارات
www.manaraa.com

```
LM_BINARY_CONTEXT
| PD:100 PF:100 PREC:71| PD:59 PF:57 PREC:71| PD:73 PF:62 PREC
    :74| PD:12 PF:4 PREC:88| PD:53 PF:33 PREC:79| PD:92 PF:69 PREC
    :76| PD:19 PF:6 PREC:87| PD:56 PF:35 PREC:79| PD:18 PF:5 PREC
    :88| PD:11 PF:1 PREC:94| PD:28 PF:9 PREC:88| PD:7 PF:0 PREC
    :100| PD:33 PF:10 PREC:88| PD:51 PF:13 PREC:91
| $CAM: less-than-median | $AVG_CC: more-than-median | $RFC: more-
    than-median | $MOA: more-than-median | $CE: more-than-median |
    $CBO: more-than-median | $LOC: more-than-median | $MFA: less-than
    -median | $DIT: less-than-median | $NPM: more-than-median | $NOC
    : more-than-median | $CBM: less-than-median | $MAX_CC: more-than-
    median | $MAX_CC: less-than-median
0 0 0 0 0 0 1 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0 1 0 0 0 1 0
1 0 0 0 0 0 0 0 1 0 1 0 0 0
0 1 0 0 0 0 1 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 1 0 0 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 1 1
```

Figure 2.3: A sample .lmb file

www.manaraa.com

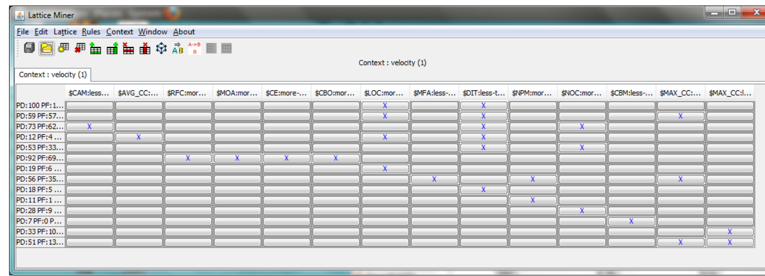

Figure 2.4: Lattice Miner User Interface

All other columns are set to zero. This setting of ones and zeros is handled in the bottom portion of the file.

When the .lmb file is loaded in, the user interface will look like Figure 2.4. Within this user interface, you can edit any existing information by simple commands such as double clicking in order to toggle a boolean option, or type in a text option.

When Lattice Miner is run to generate the concept lattice, it brings up another user interface, the Lattice View, which can be seen in Figure 2.5. Within this interface, you can edit the appearance of the lattice with a wide variety of options. You can modify what information is present in the labels, how the colors of the chart reflect the density of a particular lattice point, change the zoom size, and many other standard options. You can export an image of the current lattice view (which you can manipulate in the lattice view window) as a .bmp, .jpg, or .png image file[17].

Within the concept lattice view of Lattice Miner, you can also generate an .XML output file[17] (such as the one in figure 2.6 which can be parsed to create user friendly displays of information. The XML file generated by Lattice Miner can be parsed relatively simply. When we parse the file, we can create user friendly outputs (like those seen in Figure 4.9).

Each point on the Concept Lattice is treated like a node in a linked lattice structure. All parents of a particular node contain some subset of the that same node. All children of a node contain at least all rule elements of that original node, and additionally

المنارة للاستشارات
www.manaraa.com

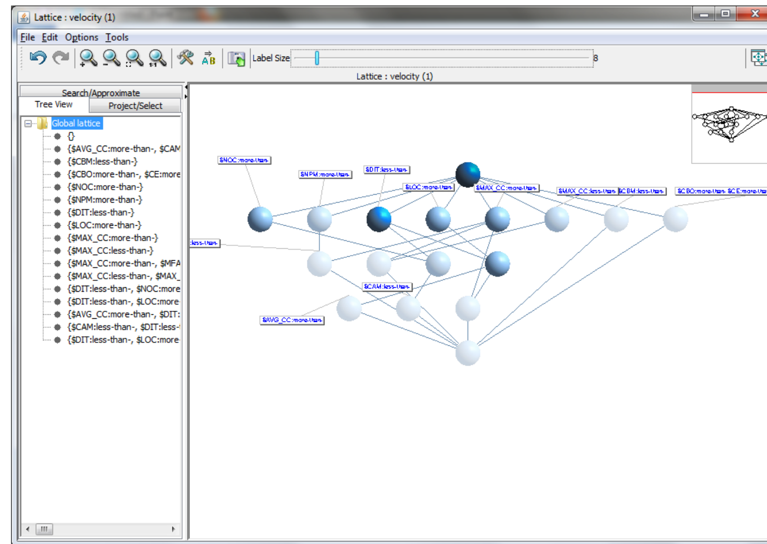

Figure 2.5: Lattice Miner: Concept Lattice View

one or more rule elements not in the parent. Each node's "score" is calculated by simply averaging the score elements of each rule that matches the nodes conditions.

If the user specifies what Node they are "starting from" , which can be defined by the rules that particular node represents, we can explain related nodes, and how traversing from one node to another can affect their expected averages for target classes.

These outputs will illustrate the the user the average class values of the rules that meet the nodes condition. It will also compare those averages to neighbouring nodes, allowing a user to understand how they can move through the lattice structure.

We simply parse the existing XML file looking for relationships between nodes. For example, if a user selected a given node, the printout would display the relationship with the parent and child nodes, illustrating which objectives the parent and children improve upon and which objectives they degrade on. In our output, we simply average all the rules in a single node together to generate the node's objective scores.

The parsing algorithm simply matches the rule identification number to the rule name in order to access the scores. Then by accessing the ID numbers of rules in parent and child nodes, we can easily create this printout by averaging together the values in each

المنارة للاستشارات
www.manaraa.com

temp.lat.xml


```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <LAT Desc="velocity (1) - temp.slf - #OfNodes = 17" type="ConceptLattice">
3    <MINSUPP>0.0</MINSUPP>
4    <OBJS>
5      <OBJ id="0">PD:100 PF:100 PREC:71</OBJ>
6      <OBJ id="1">PD:11 PF:1 PREC:94</OBJ>
7      <OBJ id="2">PD:12 PF:4 PREC:88</OBJ>
8      <OBJ id="3">PD:18 PF:5 PREC:88</OBJ>
9      <OBJ id="4">PD:19 PF:6 PREC:87</OBJ>
10     <OBJ id="5">PD:28 PF:9 PREC:88</OBJ>
11     <OBJ id="6">PD:33 PF:10 PREC:88</OBJ>
12     <OBJ id="7">PD:51 PF:13 PREC:91</OBJ>
13     <OBJ id="8">PD:53 PF:33 PREC:79</OBJ>
14     <OBJ id="9">PD:56 PF:35 PREC:79</OBJ>
15     <OBJ id="10">PD:59 PF:57 PREC:71</OBJ>
16     <OBJ id="11">PD:7 PF:0 PREC:100</OBJ>
17     <OBJ id="12">PD:73 PF:62 PREC:74</OBJ>
18     <OBJ id="13">PD:92 PF:69 PREC:76</OBJ>
19   </OBJS>
20   <ATTS>
21     <ATT id="0">$AVG_CC:more-than-</ATT>
22     <ATT id="1">$CAM:less-than-</ATT>
23     <ATT id="2">$CBM:less-than-</ATT>
24     <ATT id="3">$CBO:more-than-</ATT>
25     <ATT id="4">$CE:more-than-</ATT>
26     <ATT id="5">$DIT:less-than-</ATT>
27     <ATT id="6">$LOC:more-than-</ATT>
28     <ATT id="7">$MAX_CC:less-than-</ATT>
29     <ATT id="8">$MAX_CC:more-than-</ATT>
30     <ATT id="9">$MFA:less-than-</ATT>
```


Figure 2.6: XML output generate from previous Concept Lattice View

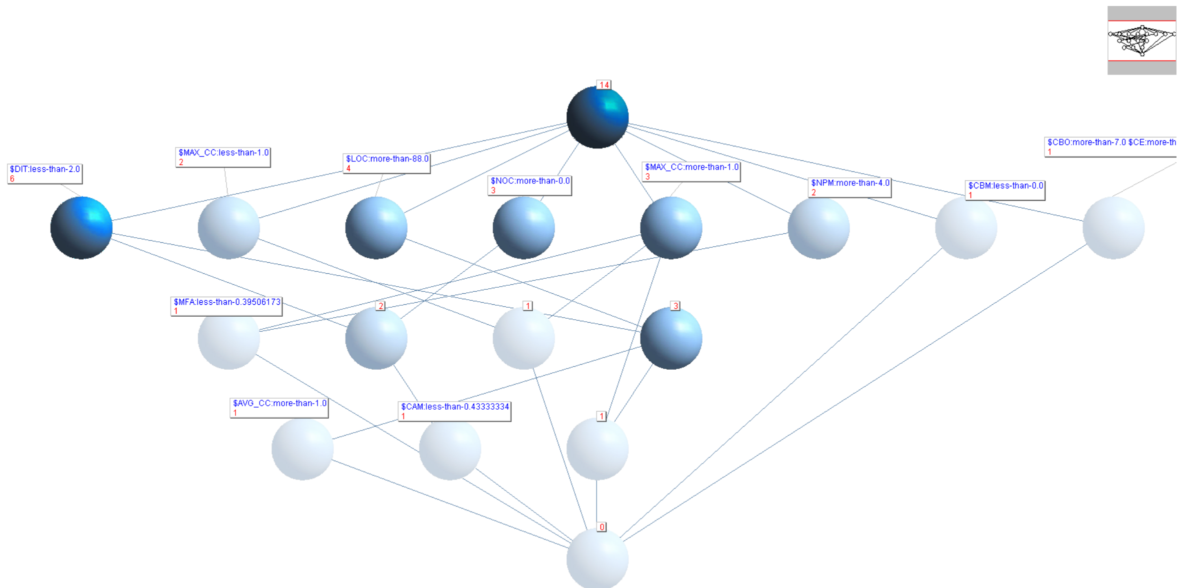

Figure 2.7: An example of a generated Concept Lattice output image.

المنارة للاستشارات
www.manaraa.com

node.

This printout parsed from the XML is a very simple version of explaining the Pareto frontier. While in the future, more complex methods may be developed which produce better information yield, HOW with Formal Concept Analysis shows getting this type of information at present is possible and not costly, and thus satisfies the goals of this thesis.

المنارة للاستشارات
www.manaraa.com

```
At Node 9

Conditions
$DIT:less-than-median
$LOC:more-than-median

Averages
PD: 57
PF: 54
PREC: 77

Parents


Node 1

$DIT:less-than-median

PD: 53 (-)
PF: 44 (-)
PREC: 79 (+)

...
Children

Node 13

$DIT:less-than-median
$LOC:more-than-median
$MAX_CC:more-than-median
PD: 59 (+)
PF: 57 (+)
PREC: 71 (-)
```

Figure 2.8: Neighborhood printout

للاستشارات المنارة
www.manaraa.com

Chapter 3

How

3.1 Introduction

HOW is an adaptation of WHICH [19]. WHICH is a stochastic associative rule learning algorithm designed to optimize a single objective function. Which combines rules together in order to form new rules. Which would run several rounds of stochastic generation, until it had gone through a set number of generations without improvement. Upon completion, the top rules would becomes the theories generated. The method of scoring the rule was based on context. In the case of detecting defects, measures like number of bugs per lines of code would determine the score. In class prediction databases, measure like PD, precision, or f-measure could serve as the dependent class.

HOW was developed using Common LISP. It was primarily developed in Ubuntu 11.04 (Natty Narwhal) using SLIME within emacs. Some of the code was written by Dr. Tim Menzies (the portions of code written by him are noted in the Appendix), though I wrote the vast majority.

HOW defers from Which primarily because HOW can serve to optimize more than one dimension. The algorithm for HOW is not limited to any specific number of dependent variables. The primary difference between HOW and WHICH is in the scoring function, as the mechanisms for stochastically combining associative rules are basically unaffected. Because WHICH only scores one attribute[19], it cannot find an optimal Pareto frontier in

المنارة للاستشارات
www.manaraa.com

multi-dimensional space. HOW's objective function is highly customizable, and allows for any number of objectives.

HOW takes in a database with either a single discrete target class or one or multiple numeric target classes. HOW then generates a list of rules, returning those rules that form a Pareto frontier in the target space. In the case of a single discrete target class, HOW can optimize accuracy, f-measure, and other measures of a rule's effectiveness.

3.2 Data Input

HOW, which was developed in LISP, uses LISP-designed structures and macros to load in the table. A sample of the "Weather" dataset can be seen in Figure 3.2. This data type can be adapted from .arff files easily, as the structure of the data is relatively unchange. Formatting only has to be done to column names, and the records and column headings have to be wrapped in LISP procedure calls. The first line, as in .arff files, will define the table. The call to deftable will create a table with the name given as the first argument (in Figure 3.2, the name is weather). The "!" macro at the beginning of each record calls a LISP procedure that loads the record into a global table variable

The column name can be given meaningful symbols so the user can control how their data will be parsed. For example, if one column is named "$humidity", then this column is treated as a numeric class, meaning it will be discretized using Equal-Width discretization. The user can control the level of discretization in the function call of HOW. There are limitations on using these symbols. A dataset can only have one discrete target class feature (though there can be any number of classes within this feature) or can have at least one numeric target class. A dataset cannot mix a discrete class with any number of numeric classes without heavy specialization. A specialization has been written for defect prediction, which treats the number of defects as both a numeric and a discrete value (discretized number of defects being zero or number of defects being non-zero).

المنارة للاستشارات

www.manaraa.com

```
( deftable weather
      forecast $temperature
  $humidity windy !class)


( ! sunny 85 85 FALSE no)
( ! sunny 80 90 TRUE no)
( ! overcast 83 86 FALSE yes)
( ! rainy 70 96 FALSE yes)
( ! rainy 68 80 FALSE yes)
( ! rainy 65 70 TRUE no)
( ! overcast 64 65 TRUE yes)
( ! sunny 72 95 FALSE no)
( ! sunny 69 70 FALSE yes)
( ! rainy 75 80 FALSE yes)
( ! sunny 75 70 TRUE yes)
( ! overcast 72 90 TRUE yes)
( ! overcast 81 75 FALSE yes)
( ! rainy 71 91 TRUE no)
```

Figure 3.1: The Weather dataset using How's LISP data input

Symbol meanings	
Symbol	Meaning
!	Discrete target class
$	Numeric attribute
$!	Numeric target class (maximize)
$#	Numeric target class (minimize)

Table 3.1: Symbols in the column names and their meanings

المنارة للاستشارات
www.manaraa.com

3.3 Rules

The Rule data structure is the most important part of HOW. A Rule is a data representation of an associative rule, connecting certain occurrences in the features of the data with the target classes. For example, a rule for a discrete target class may look like figure 3.2. This rule data structure is the backbone of HOW's learning algorithm.

```
CLASS: BREAKFAST
Given that:
   MEAT is in (SAUSAGE BACON)
   SIDE in in (EGGS)
SCORE: x
UTILS: PD, (1-PF), PREC
```

Figure 3.2: A sample rule

The rule in Figure 3.2 means that if either Sausage or Bacon are the attribute for Meat, *and* Eggs is the attribute for side, then the projected classification for that record is Breakfast. The score would denote how optimal the rule was, based on its utilities (utils) are used. In this case, the utilities are made up of the sum of *recall* (PD), *inverse false alarm* (PF), and *precision* (PREC). It is worth noting that datasets with numeric targets do not have a CLASS element, since the goal is optimization, not classification.

3.4 Score vs. Utilities

The score of the rule is used to weight the stochastic selection of rules, causing rules with higher scores to be selected for combination more than rules with lower scores. This is done because, logically, combining poor scoring rules is unlikely to produce a high scoring rule. Thus, score is only a heuristic tool used for weighting selection, where utilities actually specifically define where in multi-dimensional space a rule is located.

The utilities serve as the dependent variables of the rule. In the case of rules to

www.manaraa.com

determine discrete classes, where the goal is correct classification, utilities like recall, false alarm, precision, accuracy, et al., prove to be the most useful. Using n such measures allows you to find rules along the Pareto frontier in n-dimensional space that optimize those n utilities. In numeric datasets, you can use the average or median dependent variables in the training data that the rule covers. This allows you to similarly find the Pareto frontier of a numeric space.

Each of these items represents a different form of multi-objective optimization. Score is a form of aggregate fitness function, which combines the numeric metrics of the objectives into one metric. Utilities, however, uses Pareto dominance for multi-objective optimization. The actual "success" of HOW is measured by the results of the output rules' utilities, since we are trying to find the Pareto frontier of our objective functions with HOW.

In short, when defining the Pareto frontier, a rule's utilities are used. When stochastically selecting which rules to combine, the score hueristic is used.

3.5 Combining

Rules are combined stochastically (with their weighting of selection being based on score) two at a time. This is implemented by creating a list of rules, duplicating each rule several times (or exploding the list), and then selecting two items at random from the exploded list. The number of duplications for a given rule is proportional to the rule's score element.

When two rules are combined, like fields are merged. For example, say we have two rules like in Figure 3.5. The result of combining these two rules into one is in Figure 3.5.

This illustrates several important points about rule combination. First, you can only combine rules with the same class in classification problems (in problems with numeric target classes, the "class" element of a rule is irrelevant). Also, when you combine rules with a common attribute, but different elements, that portion of the rule becomes more general

المنارة للاستشارات

www.manaraa.com

```
RULE 1 =
CLASS: BREAKFAST
Given that:
MEAT is in (BACON)

RULE 2 =
CLASS: BREAKFAST
Given that:
MEAT is in (SAUSAGE)
SIDE is in (EGGS)
```

Figure 3.3: Two rules to be combined

```
CLASS: Breakfast
Given that:
MEAT is in (BACON SAUSAGE)
SIDE is in (EGGS)
```

Figure 3.4: The combined rule

للاستشارات المنارة

www.manaraa.com

(that is, in the example provided, now the meat can be either BACON or SAUSAGE, so more elements in MEAT will meet the criteria of this rule). A rule only gets more specific when the parent rules have different attributes.

3.6 Algorithm

3.6.1 Discretization

Even when evaluating numeric target classes, it is still vital in HOW to discretize the data of non-target attributes. For this process, we use Equal-Width discretization. This method involves breaking the data up into ordered "bins" which are equal in range.

The number of bins chosen in discretization affected the output. Generally, for defect prediction, 2-bin discretization worked very well. However, for numeric problems, 4-bin, 8-bin, and 16-bin would frequently perform better than 2-bin on several problems, such as Schaffer.

3.6.2 Rounds

A distinction needs to be made between two different processes in HOW: round0 and rounds. Round0 generates all atomic rules, which is an associative rule that contains only one condition (if Field X is Attribute Y). Round0 generates the initial list of rules which become the basis on which all combined rules are created. Rounds is responsible for combining these rules.

The Rounds algorithm is the looping process in which rules are stochastically created. This process loops, creating several rules per round, until a specific number of rounds passes without improvement. The rules given as input to rounds are generated within round zero.

Several constants have been set for these experiments. For example, five is considered the minimum number of rows that need to apply to a rule for that rule to be valid. Also, in each round, twenty rules are generated. Lives refers to the amount of tolerated

المنارة للاستشارات
www.manaraa.com

"failed" rounds, or rounds without finding a new "best rule", before rounds terminates.

Rounds

```
begin
  comment: "rules" is the existing list of rules
  lives := 5;
  while (lives > 0)
      newrule;
      frontier := getParetoFrontier(rules);
      for i := 0 to 20 step 1 do
         rule1 := random-element(explode(rules))
         rule2 := random-element(explode(rules))
         newrule := combine(rule1, rule2)
         score(newrule);
         if (newrule not in rules) ∧
           then
               (newrule− > support > 5)
               push(newrule, rules)
         fi
      end
      if frontier! = getParetofrontier(rules)
        then
             frontier := getParetoFrontier(rules);
             lives := 5
        else
             lives := lives − 1
      fi
end
```

المنارة للاستشارات
www.manaraa.com

```
  return rules;
end
```

3.6.3 Scoring

Rules can be scored based on a number of objectives. In classification data, is it common to score rules as attributes like recall, accuracy, false alarm rate, et al. This is done by simply finding these measurements on all the records of data the rule applies to.

Numeric target-class problems are scored in a similar fashion. The averages are taken of all numeric target-classes for all records in the dataset that meet the conditions of the rule being scored. These target averages become the dimensions of the rule in the utility space. The score is frequently the weighted sum of the utility measures.

Each numeric target class can either be optimized or minimized, depending on user preference. For example, when we examine defect prediction data, our goal is to find as many faults as possible while minimizing the number of line of code that need to be examined. In this case, we would seek to maximize the number of defects, and minimize the lines of code that our rules find.

Each average is then represented as a utility, weighted from 0 to 1, where 1 is optimal (if we are maximizing, 1 is the maximum; if we are minimizing, 1 is the minimum). Usually, the sum of these utilities serves as the rule's score, which is used as a heueristic to determine how often that rule is used to combine with other rules.

3.7 Output

HOW outputs a list of rules. If you are working with a numeric database, the rules outline the Pareto frontier of your target class utilities. For example, in the problem Schaffer 4.1, optimum is considered where f1(x) and f2(x) are smallest. In discreet datasets (like Weather and Diabetes), the dependent variables can be PD, Precision, F-Measure, etc. are used to form the Pareto frontier. Regardless which type of database is used, the rules

المنارة للاستشارات
www.manaraa.com

that are returned are all along the Pareto frontier, and thus no rule in the final output is Pareto dominant over another rule in the output.

3.8 Handling Mathematical Models

Standard mathematical models exist where you try to find the sets of input variables $\vec{x}$ that optimize multiple functions $f(\vec{x})$. This presents a challenge to HOW, as HOW requires discretized input. When the data is discretized it becomes impossible to find precise answers. Therefore, an iterative generations model is used to handle mathematical models such as Schaffer and Tanaka.

3.8.1 Algorithm

First, a dataset is generated based of the model being tested. If the data is constrained in any way, such as in Tanaka[25], the constraints are checked at data generation, and that data is not added to the data set. HOW is then run on this data set, and the output rules that form the Pareto frontier are then used as constraints to generate another data set.

This is effectively a breadth first search, where the rules are used as nodes in the tree. At each point, we run HOW, use the output rules to constrain data, and then run HOW again. After all the rules are recursed on, all the generated rules are evaluated for Pareto dominance, and then each rule along the frontier is evaluated similarly to form the next generation.

Using this breadth-first search style method does lead to a very large set of solutions. In order to maintain a reasonable run time, and to limit the data, crowd-pruning is used. A linear method of pruning is used, where the rules are selected such that their target vectors are some angle apart. This pruning method is called at the conclusion of each generation.

Summarily, the algorithm can be conveyed this way

المنارة للاستشارات
www.manaraa.com

1. Generate an initial dataset based on the given problem set

2. Discretize the data and run HOW on the discretized dataset.

3. For each output rule from the previous generation, generate a new dataset, discretize, and then run HOW

4. Combine all the current generation and previous generation output rules into one group, and find the Pareto dominant set. This uses an elitism principle similar to what is seen in NSGA-II[1].

5. Prune the set using the described method.

6. Return to Step 3.

This algorithm works because on each subsequent run of the data, the range of input variables becomes tighter, and therefore the discretized data has more tightly bound ranges. This allows HOW to analyze much more precise areas than a run across a full data set.

3.8.2 Crowd Pruning

Crowd Pruning is needed to prevent the size of the solution space from increasing exponentially from one generation to the next. Without any crowd pruning, runs of HOW's mathematical model algorithm would take unreasonably long, and would require very large amounts of memory.

In practice, crowd pruning has a minimal effect on the "quality" of the Pareto frontier discovered. This is because most of the rules generated in later generations are only very slightly different from existing rules, due to a very minor change in the input variable changes. Summarily, the benefits of crowd pruning (reasonable run times, smaller memory cost) far exceed any cost, which at most is a minimal to negligible loss in performance.

Crowd pruning for mathematical models is found by using the FastMap[3] algorithm. FastMap can be effectively be used to reduce the dimensionality of a problem.

1. Given an array of points S, arbitrarily select y such that $y \epsilon$ S.

2. Find the point $a \epsilon$ S maximizes the distance $d(a, y)$.

المنارة للاستشارات
www.manaraa.com

3. Find the point $b \epsilon$ S maximizes the distance $d(b, a)$.

4. Using a and b as the pivots.

The distance measure $d(x, y)$ measures the cosine similarity between x and y in order to find the angle. The actual distance is measured as the absolute value of the angle between x and y.

For crowd pruning, we select the point a, and sort all the data. For each point i ϵ, we find the angle between the line from a to the point of origin, and the line from i to the point of origin. This measure of the angle from a to i is used to sort the data so that each point is ordered based on distance from a.

Once we have this sorted list of points, we remove from the set S all point that are within some angle θ of a. We then select the closest point p to a that is not within the angle θ and then, remove all points within θ of p. This process repeats until the end of the data. The algorithm is illustrated below.

```
begin
  Prune S
  comment: "S" is the sorted list of points, sorted by angle from a.
  a := S[0];
  theta := .5comment: "in degrees"
  for i := 0;  to length(S) step 1 do
     if abs(angle(a, i)) > theta then
                                  remove(i, S);
                              else a := i;  fi
  end
  returnS;
end
```

www.manaraa.com

Chapter 4

Experiments with HOW

4.1 Different Test Models

Because our approaches to the mathematical test cases and to the defect data were very different, and our output is processed in different ways, we have broken the results up into two sections and will analyse them separately.

The Mathematical Models section will illustrate that the HOW algorithm can function adequately as a general multi-objective optimization tool. The Defect Data section will illustrate the "neighbourhood finding" mechanic developed to handle HOW's output.

Each test model had different utilities and a different scoring heuristic fitness function, which are outlined at the beginning of the section. Unless otherwise stated, assume Equal-Width discretization was used.

المنارة للاستشارات

www.manaraa.com

4.2 Mathematical Models

For handling mathematical modes, the utilities were set to the output of the objectives function such that:

$$Util_1 = \frac{Max(F_1) - avg(f_1)}{Max(F_1) - Min(F_1)}$$

$$Util_2 = \frac{(F_2) - avg(f_2)}{Max(F_2) - Min(F_2)}$$

$$...$$

$$Util_n = \frac{Max(F_n) - avg(f_n)}{Max(F_n) - Min(F_n)}$$

Where $Max(F_i)$ and $Min(F_i)$ are defined as the maximum and minimum of the objective function $f_i(\vec{x})$ observed in the data. The notation $avg(f_i)$ is defined as the average of the objective function $f_i(\vec{x})$ across all records that match the rule.

It is worth noting that in the above formulas $Util_i$ is small where $avg(f_1)$ is large. This is because all observed mathematical models seek to minimize the objective functions. If a function $f_{opt}(\vec{x})$ existed that was to be maximized, the Util measure would look like this:

$$Util_{opt} = \frac{avg(f_{opt}) - Min(F_{opt}}{Max(F_{opt}) - Min(F_{opt}}$$

These $Util$ functions are used to weight the averages found for the objective function results from 1 to 0, where 1 is optimal (which in the mathematical models, means closest to the minimum). This is to balance out the utility functions in problems such as Constr-Ex, where the range of one objective function is very different than the range of the other[1].

HOW can function as a standard multi-objective optimizer. However, due the level of precision required to perform this task, and due to HOW's limitation of requiring discretized data as an input, HOW had to be wrapped inside a larger breadth-first search-style algorithm outlined in section 3.8. All results shown are after nine generations of the wrapped HOW.

www.manaraa.com

Unconstrained Multi-Objective Functions			
Name	n	Objectives	Variable Bounds
Schaffer	1	$f_1(x) = x^2$ $f_2(x) = (x-2)^2$	$-10^5 <= x <= 10^5$
Fonseca	3	$f_1(\vec{x}) = 1 - e^{\sum_{i=1}^{n}(x_i - \frac{1}{\sqrt{n}})^2}$ $f_2(\vec{x}) = 1 - e^{\sum_{i=1}^{n}(x_i + \frac{1}{\sqrt{n}})^2}$	$-4 <= x_i <= 4$
Kursawe	3	$f_1(\vec{x}) = \sum_{i=1}^{n-1}(10\exp(-0.2\sqrt{x_i^2 + x_{i+1}^2}))$ $f_2(\vec{x}) = \sum_{i=1}^{n}(\lvert x_i\rvert^{.8} + 5\sin x_i^3)$	$-5 <= x_i <= 5$

Table 4.1: Unconstrained standard mathematical test problems. Note: all objectives minimized

This means that for these mathematical models, where precision of answer is very important to have results close to the Pareto frontier, we saw a dramatic increase in run-time and memory requirements over the stand-alone HOW. While absolute run-times for the wrapped HOW were generally reasonable (several seconds to several minutes), this is relatively much larger than the run-times for existing algorithms.

The mathematical models presented will all illustrated the points found by HOW's mathematical models handling. The charts will also show a line where the "ideal" Pareto front is. The presented Pareto frontier was found using the jMetal[2] solution space repository[1]. These solution spaces represnt the findings of several different algorithms including NSGA-II[1], AbYSS[22], SPEA2[32], and more[2].

The Mathematical Models can be divided into two categories. The first is unconstrained problems. A table of unconstrained problems used in this study can be seen in figure 4.1. These problems give two or more objective functions, and define the input vector, both in how many input variables there are, and what numeric range each variable exists in.

Constrained problems have all the features of unconstrained problems. However, they additionally have several constraints (represented as $g_i(\vec{x})$ on the input variables. So while some combination of points may better optimize the objective functions, they may

[1] http://jmetal.sourceforge.net/problems.html

المنارة للاستشارات
www.manaraa.com

violate one or more of the constraints, and therefore can not be considered part of the solution space.

Figures 4.1 and 4.2 show the Fonseca and Schaffer problem respectively. These charts show off the better solutions acheived. Different levels of data discretization proved more or less effective. In the case of Fonseca, using 4-bin discretization proved the most effective. In Schaffer, 16-bin discretization.

HOW's solutions on the Schaffer[23] problem in Figure 4.1 have a nice, even spread that reaches across the entire frontier[2]. It matches very closely with the ideal line. However, on low-levels of discretization (low meaning 2, 4, and 8) HOW failed to come close to the Pareto frontier. This was because early rules would often eliminate the space $x > 0$, when the Pareto dominant range is $0 <= x <= 2$ [1]

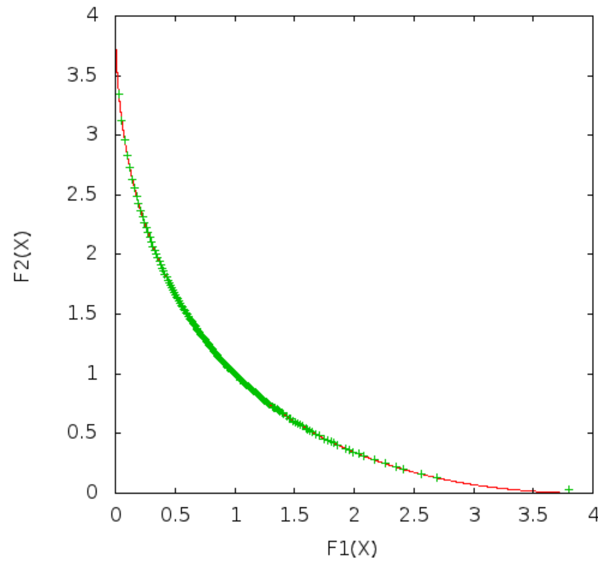

Figure 4.1: Schaffer

On Fonseca[5] in Figure 4.2, HOW also has a good spread and convergence on the frontier[2], but HOW fails to find the boundary along the bottom right. Still, this is a very good result, especially the spread.

On some problems, HOW just simply does not succeed in finding the frontier[2].

www.manaraa.com

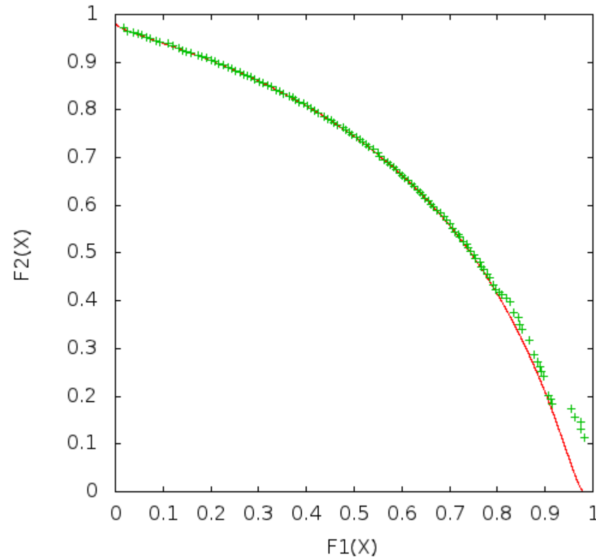

Figure 4.2: Fonseca

A noteworthy example is Kursawe[16], as seen in figure 4.3. Worse yet, at some levels of discretization, "impossible" answers were generated. Impossible, in this case, meaning a solution that is actual dominant over the Pareto frontier This is possible because HOW's rules operate on averages, so even though the averages of a rule may not be possible, the data the rule reflects isn't incorrect.

Constrained problems presented a unique problem for HOW. HOW does not, by itself, handle constraints in any fashion. Constraint handling is managed at the time of data generation. In the same way as unconstrained data, each record is created by randomizing the input variables. However, if the randomized input produces an output violating one or more constraints, the record is simply not added to the data set.

The first constrained problems we will look at is Srinivas[24]. The results of HOW on Srivinas can be seen in Figure 4.4. The frontier[2] on Srivinas is relatively linear, with curves at each end. HOW does very well in converging on the frontier, and the solution is acceptable.

HOW's performance on Constr-Ex[1] is very mixed. While the solutions HOW

المنارة للاستشارات
www.manaraa.com

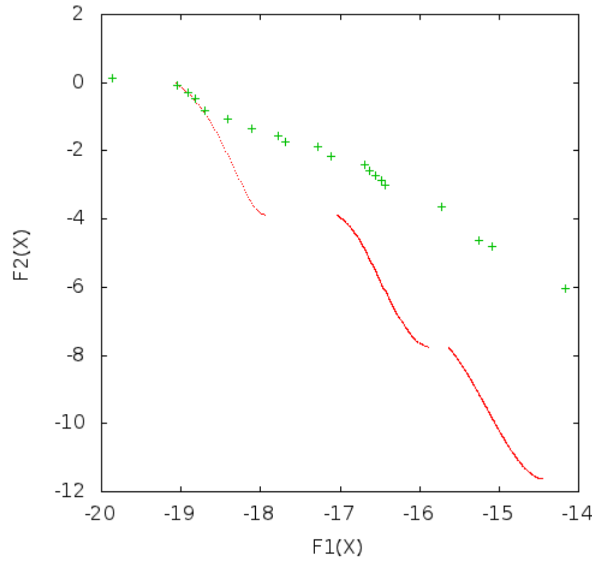

Figure 4.3: Kursawe

finds are generally very close to the ideal Pareto frontier[2], meaning a good convergence, the spread of the solution space is somewhat poor. Still, HOW has good convergence on both sides of the "bend" in this problem, so again our answer is acceptable.

Tanaka is very unique as a problem. The objective functions are actually equal to the input variables[25]. The constraints form a curving Pareto frontier which is disconnected and non-convex[25, 2], similar to Kursawe[1]. How performs fairly well on the curve with excellent convergence. The spread is good, but not excellent. Still, given the general "difficulty" associated with this problem, this is a very good solution.

From these results, we can see that HOW is very capable of finding the frontier of a mathematical problem set. HOW had reasonable solutions on five of the six selected datasets. While it may not have performed as well as NSGA-II[1] on the given datasets, HOW's solutions still show it to be a very capable multi-objective optimizer.

It is important to draw the distinction that HOW isn't just an optimizer. HOW also explains the frontier. Each of the points along the frontier is a child in a large tree of generated rules. An origin back to an initial rule in the first generation of HOW can be

المنارة للاستشارات
www.manaraa.com

Constrained Multi-Objective Functions			
Name	Objectives	Constraints	Variable Bounds
Tanaka	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = x_2$	$g_1(\vec{x}) = -x_1^2 - x_2^2 + 1 + 0.1\cos(16\arctan(\frac{x_1}{x_2})) <= 0$ $g_2(\vec{x}) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 <= .5$	$-\pi <= x_i <= \pi$
ConstrEx	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = (1 + x_2)/x_1$	$g_1(\vec{x}) = x_2 + 9x_1 >= 6$ $g_2(\vec{x}) = -x_2 + 9x_1 >= 1$	$0.1 <= x_1 <= 1.0$ $0 <= x_2 <= 5$
Srinivas	$f_1(\vec{x}) = (x_1 - 2)^2 + (x_2 - 1)^2 + 2$ $f_2(\vec{x}) = 9x_1 - (x_2 - 1)^2$	$g_1(\vec{x}) = x_2 + 9x_1 >= 6$ $g_2(\vec{x}) = -x_2 + 9x_1 >= 1$	$-20 <= x <= 20$

Table 4.2: Constrained standard mathematical test problems. Note: all objectives minimized.

traced, allowing a user to understand how a solution was derived over time. This explanation is the key separation of HOW from other optimizers. NSGA-II features no such explanation ability.

4.3 Defect Data

For the tests in this section, equal-frequency 2-bin discretization was used. The Pareto dominance utilites were set to

$$Util_1 = frac(defects\, found)(total\, defects)]$$

$$Util_2 = 1 - \frac{lines\, of\, code\, read}{total\, lines\, of\, code}$$

The scoring heuristic fitness function was set to:

$$F = Util_1 + Util_2$$

Much like it's precursor Which [19], HOW was primarily tested using defect prediction data. Because HOW is able to optimize multiple dimensions, HOW can be used in several ways here. HOW can be used to optimize various statistical measures for predicting

المنارة للاستشارات
www.manaraa.com

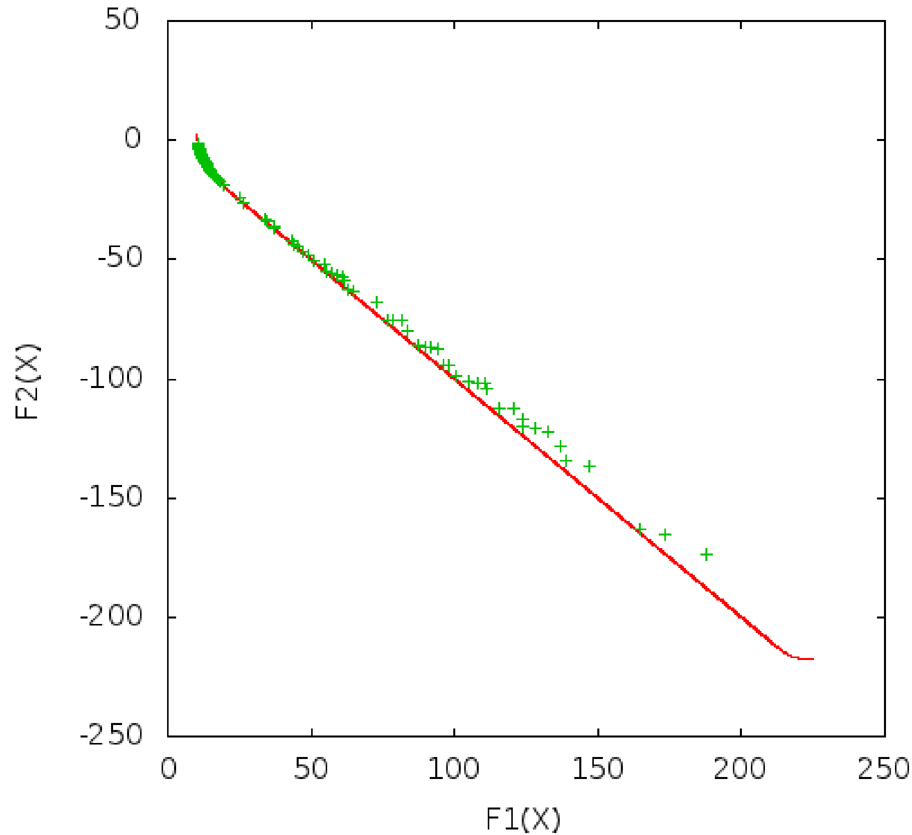

Figure 4.4: Srinivas

where defects occur such as recall, false alarm, and precision. Figure 2.7 shows the concept lattice generated from such an optimization on the Velocity dataset[2]

HOW can also be used to optimize fields within the defect prediction data. For example, in figure 4.7, you can see the space of rules generated for the Xerces dataset[3] that optimize the percentage of defects found while minimizing the percentage of lines of code (LOC) read. Figure 4.7 illustrates the rules found as points. The diagonal line would be the result of finding a percentage of bugs equal to the percentage of lines of code read, which would be a "worst-case" scenario.

The space of rules generated for Xerces shows a typical curve found across most

[2]Data available at http://promisedata.org/?p=333
[3]Xerces dataset available at http://promisedata.org/?p=345

www.manaraa.com

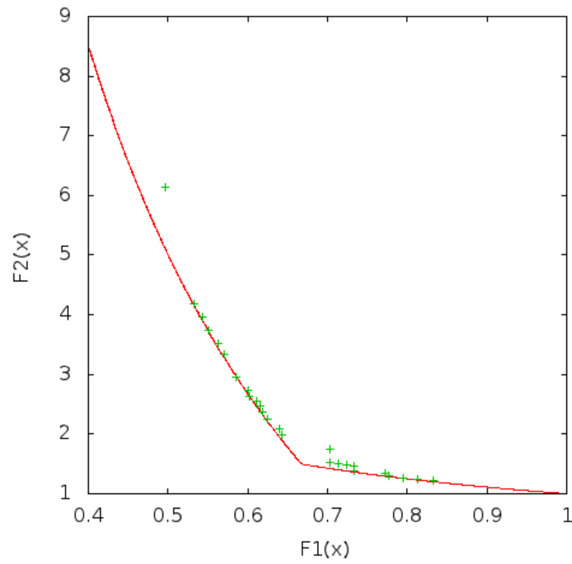

Figure 4.5: Constr-Ex

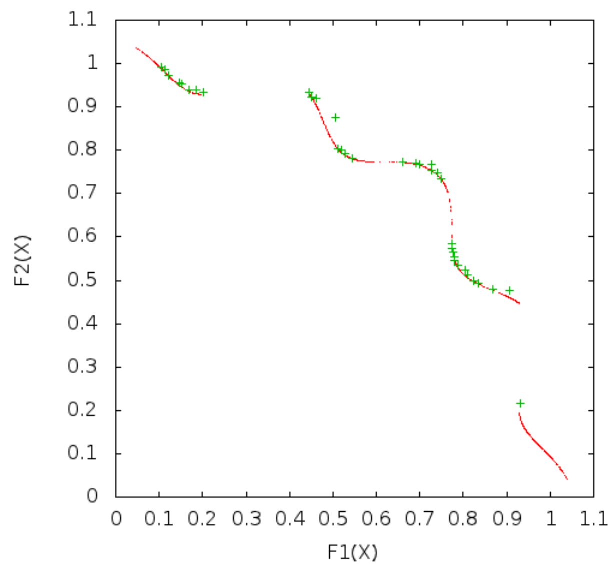

Figure 4.6: Tanaka

datasets. Several rules are found that locate a large percentage of the bugs in a relatively small portion of the code. For example, one rule finds roughly half the bugs looking at less than 20% of the lines of code. As the rules increase in generality, the increase in bugs found

المنارة للاستشارات
www.manaraa.com

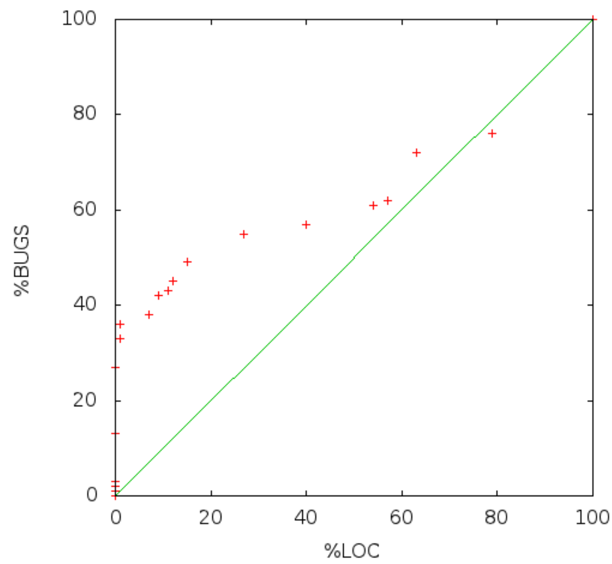

Figure 4.7: Xerces

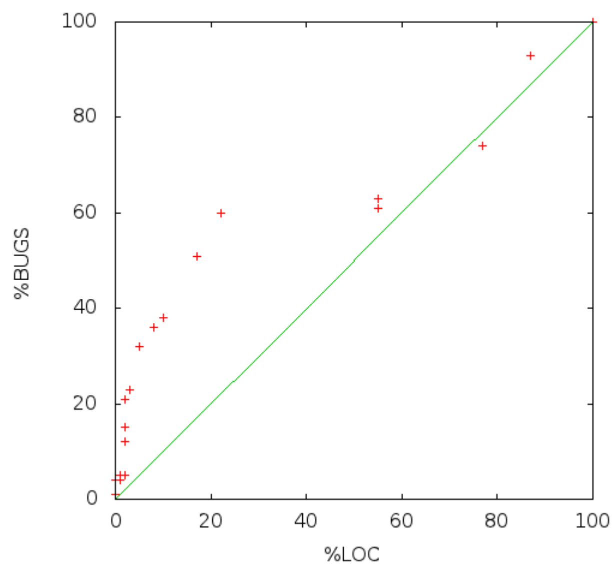

Figure 4.8: Velocity

becomes much slower than the increase in lines of code found. Eventually, in most cases, the rules with the highest percentage bugs found have to read a larger percentage of lines of code (illustrated by the point underneath the worst-case line at roungly 80% of lines of

www.manaraa.com

code read). Figure 4.8 for the Velocity data set shows a very similar performance.

Based on the rules in the Velocity dataset, we can create the lattice structure in figure 2.7 using Lattice Miner[17]. This concept lattice is relatively simple compared to the lattice generated from most datasets. This lattice was generated from HOW optimizing recall, precision, and false alarm rate on classifying records as having defects. Recall and Precision are optimal when maximized, and false alarm is optimal when minimized.

The key to HOW's ability to explain the neighborhood is in the ability to parse the generated concept lattice to produce an output, which is illustrated in figure 4.9. This particular figure shows the immediate neighborhood (that is, parent and child nodes) of node 9 (which is the dark, further right node on the third row of figure 2.7). This figure does illustrate the common pattern in rules generated on the defect data that as recall increase, false alarm increases and precision decreases.

The defect data allows several optimization options. In figure 4.9, an output optimization of recall (PD), false alarm (PF), and precision (PREC) is shown. Each of these measures is calculate by first labelling all records with defects as "bugs" and records without as "none". These numbers refect how successful a rule is in classifying records as "bugs". Note that in order to simplify the rules to make them easier to understand, the rules generated from velocity for this demonstration were generated using 2-bin Equal-Frequency discretization, not equal width.

This user output can be produced from the perspective of any node, and shows the "cost" (represented by the expected change in performance) of moving from one point in the neighbourhood to another. This can be helpful for showing the difference between a general solution, where performance among the neighbours is relatively consistent, and from less stable solutions, where the neighbours' objective scores vary greatly. This is especially useful when we understand that HOW performs as well on defect data as Which, which does not offer the same neighbourhood explanation and cannot directly optimize multiple dimensions[19] as HOW can.

المنارة للاستشارات
www.manaraa.com

4.4 Fitness Heuristic

As explained before, while the results of HOW uses Pareto dominance, the scoring hueristic for weighting the stochastic selection uses an aggregate fitness function. For all tests in the Mathematical models section, the hueristic was simply the sum of the utilities, where a larger sum was "better" than a smaller sum. For the examples shown in Defect Prediction data, the fitness function was the sum of the percentage of bugs found and the percentage lines of code *not* read. In both cases, the fitness function was simply a sum of the utilities used.

However, in the original experiments with Which, it was found that having heavier penalties on high PFs (false alarm rate) improved the quality of the data [19]. Therefore, I wanted to compare two heuristic functions within HOW and their effect on runs of several defect prediction datasets. The first heuristic function was

$$Sum = PD + (1 - PF) + (1 - \%LOC)$$

The second formula was adapted from Which's B_1 function, however effort was replaced by percentage lines of code, since the two were more or less the same measure [19].

$$B_1 = 1 - \frac{\sqrt{PD^2 * \alpha + (1 - PF)^2 * \beta + (1 - \%LOC)^2 * \gamma}}{\sqrt{\alpha + \beta + \gamma}}$$

where

$$\alpha = 1;\ \beta = 1000;\ \gamma = 0$$

.

Too compare these two hueristics, we ran HOW using each hueristic over multiple datasets, calculating the $AUC(\%LOC, PD)$ (The area under the curve where the x-axis is the lines of code, and the y-axis is the PD). $AUC(\%LOC, PD$ was calculated using the Trapezoid rule, given a set of points $S(x, y)$ where $|S| = n$ x represents %LOC and y represents PD, the area can be calculated with the following summation:

المنارة للاستشارات
www.manaraa.com

$$AUC(\%LOC, PD) = \sum_{i=1}^{n-1} \frac{1}{2}(y_i + y_{i+1}) * (x_{i+1} - x_i)$$

One could argue that the trapezoid rule, which will underestimate the concave curve expected from the (%LOC, PD) graph, will unfairly favor a more dense frontier, since there is less loss from underestimated space. In reponse to this, I would note that density of the frontier is relevant, so it is actually advantageous to punish more sparsely populated frontiers. A problem could arise if the curve were convex, rather than concave, since then more sparsely populated frontiers would actually over-estimate the area of the frontier, but this simply is not the case here.

When using the Trapezoid Rule, we first sort the data using PD as the sorting attribute such that the rule with the smallest PD appears first in the list. Since we are working with two dimensions, and since we've limited our rule output to the Pareto frontier, we can make the assumption that the second dimension, LOC, is also sorted similarly. We can also make the assumption in all cases that the point (0,0) is in the rule solution space. This is the point where a rule has found 0% of the defects after reading 0% of the lines of code. A null rule would meet this condition, so it is always included. Additionally, the last point on a curve (x_n, y_n) is considerd a worst case for extrapolation where x_n is the %LOC and y_n is the PD. Because HOW limits the output to those along the Pareto frontier, no output rule with utilities (x_m, y_m)could ever exist such that $y_m < y_n$ and $x_m > x_n$, since in this case the point (x_m, y_m) would be Pareto dominated by point (x_n, y_n). Thus, we can extrapolate the point (x_n, y_n) to at worst $(100\%, y_n)$

Table 4.4 illustrates our findings for $AUC(\%LOC, PD)$ across multiple defect prediction datasets[7]. Ideally, a large area under the curve is desirable, as it would show you are finding more defects in fewer lines of code read. Therefore, in this table, larger numbers are "better."

From Table 4.4, we can see that across most datasets the area under the curve of B_1 nnd Sum are very similar. In all but three of the datasets, they are within 3% of each

المنارة للاستشارات
www.manaraa.com

Sum vs. B_1				
Dataset	*Sum*	B_1	$Sum - B_1$	% difference
ant	0.5587	0.5380	0.0207	-3.71%
camel	0.5095	0.5607	-0.0.512	10.05%
jedit	0.6963	0.6869	0.0094	-1.35%
log4j	0.542	0.5411	0.0009	-0.17%
lucene	0.5993	0.6348	-0.0355	5.92%
pbeans	0.8431	0.8511	-0.008	0.95%
poi	0.5659	0.6627	-0.0968	17.11%
synapse	0.5589	0.56	-0.0011	0.20%
velocity	0.7259	0.7264	-0.0005	0.07%
xalan	0.5904	0.6013	-0.0109	1.85%
xerces	0.7395	0.7558	-0.0163	2.20%

Table 4.3: Comparison of the Sum and B_1 hueristics

other. However, on the datasets where there is seperation (camel, lucene, and poi), B_1. It is then reasonable to see that it is generally worth using B_1. The scoring heuristic, while not directly affecting the Pareto frontier utilities, has an indirect effect on how the search space finds the frontier. Further research into improving the heuristic would be useful beyond just these two solutions. A better heuristic may help address some of the issues HOW has with various mathematical model sets.

I have included two charts illustrating the results on a dataset where B_1 beats Sum substantially, poi (Figure 4.10), and one where the two finish relatively equally, velocity (Figure 4.11). In poi, you can visibly see there is an area of solutions using the sum just simply doesn't find. This results in Sum performing dramatically worse than B_1. In velocity, both hueristics perform nearly identically. Both models find roughly rules that find 50% of the defects within the 15% of the LOC, the shallow out before having a steep jump at around 50% of lines of code read to nearly 100% of defects found.There are some points B_1 find that dominate points Sum finds, but vice versa is also true. In the end, our area under the curve measure is very similar for both metrics.

www.manaraa.com

```
At Node 9

Conditions
$DIT:less-than-median
$LOC:more-than-median

Averages
PD: 57
PF: 54
PREC: 77

Parents
Node 1

$DIT:less-than-median

PD: 53 (-)
PF: 44 (-)
PREC: 79 (+)
...
Children
Node 13

$DIT:less-than-median
$LOC:more-than-median
$MAX_CC:more-than-median
PD: 59 (+)
PF: 57 (+)
PREC: 71 (-)
```

Figure 4.9: Neighborhood printout

المنارة للاستشارات
www.manaraa.com

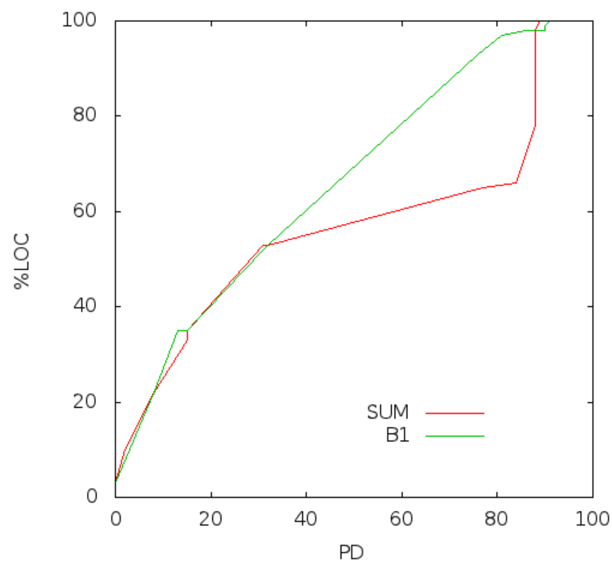

Figure 4.10: The Pareto frontier of the poi solution space found using B_1 and Sum as heuristics

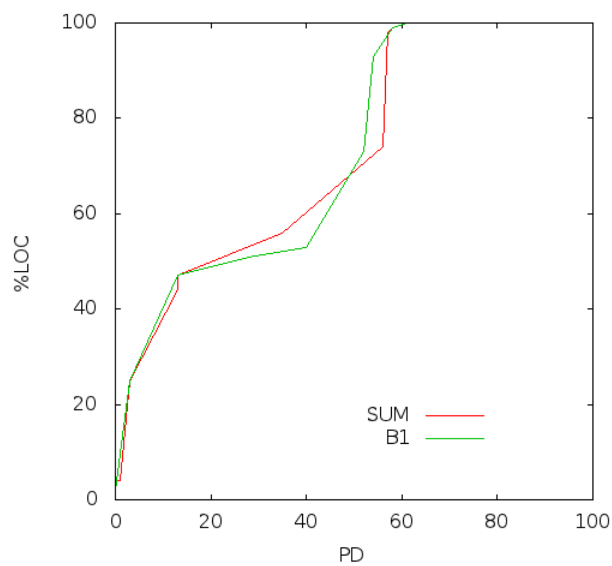

Figure 4.11: The Pareto frontier of the velocity solution space found using B_1 and Sum as heuristics

المنارة للاستشارات
www.manaraa.com

Chapter 5

Conclusion

This chapter is broken into three parts. Section 5.1 will give an overview of the findings of this thesis. Section 5.2 will summarize the findings. Section 5.3 will provide future work to be done on the HOW algorithm.

5.1 Overview

According to Harman, "in order to develop the mature roots that allow the eld [of search-based software engineering] to grow, the second phase of exploration requires a deeper understanding of problem and solution characteristics."[12]. HOW is a stochastic associative rule learning algorithm developed from Which [19, 21] that can meet the needs of the second phase mentioned by Harman. HOW is highly customizable for use in multi-dimensional optimization, and can work or be made to work on several different types of multi-objective optimization problems. H

OW uses primarily Pareto dominance optimization, though an aggregate fitness function is used as a heueristic for searching. As we have shown, HOW not only finds a solution, but also keeps the perspective as to how each solution is derived, and where it fits in the neighbourhood of solutions. HOW hopes to fill this need by trying to explain the search and solution space to the user. By giving the user more information on the Pareto frontier, we hope to empower the user with better decision-making ability when analyzing

المنارة للاستشارات
www.manaraa.com

software engineering problems.

HOW improves on the Which algorithm in several ways. Where Which can only directly optimize one objective function (the objective function itself could be a combinate of several objectives, as seen in [19], but is ultimately a single-optimization problem), HOW can directly optimize multiple dimensions and find a space of Pareto optimal solutions. More importantly, HOW provides knowledge of a search space with explanation, without a decrease in performance from Which. With HOW, we have shown that explanation of a search-space is possible, and that it can come without a loss in performance.

5.2 Findings

HOW can quickly learn on defect data and numeric data. On the mathematical test problems, HOW was able to converge on the frontier in five of six cases. This pattern of succes with one failure is similar to Which's initial study, where Which performed in the top ranking all but one time, where it finished near the bottom. These failures clearly can be present, but seem to be unlikely in most cases, as they are observed rarely.

On Defect Prediction, HOW was able to produce useful results that were on par with results of Which's fitness aggregate function. However, unlike Which, HOW was able to produce a neighborhood finding output that explained the Pareto frontier. The ability to explain the Pareto frontier is the key finding from this thesis.

5.3 Future work

Two enhancements should be able to be made in order to decrease the run time of HOW. Currently, HOW uses list structures to access the data. By changing from list structures to indexed vector structures, access time for particular elements in the table will decrease from linear to constant time operations. Additionally, if a hash table were created for the attributes and used to access the rows, scoring a rule would no longer require iterating through the entire table.

المنارة للاستشارات
www.manaraa.com

Efforts can be spent finding different formal concept analysis software outside of Lattice Miner. Lattice Miner is advantageous because it is open-sourced[17]. However, it is possible other FCA software[1] might be more suited for HOW than Lattice Miner.

Reworking HOW to solve the Next Release Problem would be a very interesting a useful experiment. The next-release problem as defined by [30] presents a complex but solvable multi-objective optimization problem. HOW may require a fair amount of modification in the scoring algorithm, but the basic structure of stochastically combining rules into more rules would seem very apt for finding solutions to the Next Release Problem.

[1] An excellent collection of FCA tools can be found at http://www.fcahome.org.uk/fcasoftware.html

www.manaraa.com

Appendix A

Source Code

Because there are differences in HOW's implementation for the purposes of Defect Prediction or for Handling Mathematical Models, we have seperate sections for Defect Code and "Frontier Finder", which was the name giving to the wrapping of Which for hanling mathematical models. Any code in the section Other can be assumed to be identical for both forms of How. The Data section gives samples of the input data formats for various functions.

A.1 Defect Code

A.1.1 3will.lisp

```
;3will.lisp was used for discretizing the data
;
;

(defun discretizer (column numbins table)
  (let ((out '()))
    (reverse
     (dotimes (x numbins out)
       (if (= x (- numbins 1))
              (push (float (+ (num-max column) 1.0)) out)
              (push (float (+ (num-min column)
                              (* (+ x 1)
                                 (/ (- (num-max column) (num-min column)) numbins))))
                    out))))))

(defun percentile (column x table)
  (let ((pos (position column (table-cols table) :test #'equalp)) (lst '()))
    (dolist (row (table-rows table))
      (push (nth pos (row-cells row)) lst))
    (setf lst (sort (copy-list lst) #'<))
```

المنارة للاستشارات
www.manaraa.com

```
    (nth (round (* x (length lst))) lst)))

(deftest !discr1 (&aux (column (make-num :max 280 :min 10)) (numbins 3))
  (test (discretizer column numbins) '(100 190 280)))

(defun discretizetbl (tbl &optional (numbins 2))
  (minmax tbl)
  (let ((newtbl (make-table :name (table-name tbl) :klasses(list (make-klass :name 'BUGS)
                                                                  (make-klass :name 'NONE)))))
    (dolist (row (table-rows tbl))
      (let ((newrow (make-row :old (row-cells (copy-row row)))))
        (cond
          ((>= (nth 20 (row-cells row)) 1)
           (progn
             (setf (row-class newrow) 'BUGS)
             (dolist (k (table-klasses newtbl))
               (if (equal 'BUGS (klass-name k))
                   (incf (klass-n k))))))
          (t
           (progn
             (setf (row-class newrow) 'NONE)
             (dolist (k (table-klasses newtbl))
               (if (equal 'NONE (klass-name k))
                   (incf (klass-n k)))))))
        (setf (row-loc newrow) (nth 10 (row-cells row))
              (row-bugs newrow) (nth 20 (row-cells row))
              (row-cells newrow) (row-old newrow))
        (push newrow (table-rows newtbl))))
    (dolist (col (table-cols tbl))
      (if (typep col 'num)
          (let ((pos (position col (table-cols tbl))) (dlist (discretizer col numbins tbl)))
            (dolist (r (table-rows newtbl))
              (setf (nth pos (row-cells r)) (num-to-descr (nth pos (row-cells r)) dlist)))
            (setf (table-cols newtbl) (append (table-cols newtbl) (list (make-sym :name (col-name
                col) :goalp (col-goalp col))))))
          (let ((pos (position col (table-cols tbl))))
            (dolist (r (table-rows newtbl))
              (if (numberp (nth pos (row-cells r)))
                  (setf (nth pos (row-cells r)) (intern (write-to-string (nth pos (row-cells r))))
                      )))
            (setf (table-cols newtbl) (append (table-cols newtbl) (list col))))))
    (dolist (col (table-cols newtbl) newtbl)
      (let ((pos (position col (table-cols newtbl))))
        (dolist (row (table-rows newtbl))
          (unless (member (nth pos (row-cells row)) (col-vals col))
            (push (nth pos (row-cells row)) (col-vals col))))
        (sort (copy-list (col-vals col)) #'string-lessp))))

#|(defun discretizetbln (tbl &optional (numbins 10))


  (minmax tbl)
  (let ((newtbl (make-table :name (table-name tbl) :klasses (table-klasses tbl))))
    (dolist (thisrow (table-rows tbl))
      (let ((thisrow2 (copy-row thisrow)))
        (push thisrow2 (table-rows newtbl))))
    (dolist (this (table-cols tbl))
      (if (col-goalp this)
          (setf (table-cols newtbl) (append (table-cols newtbl) (list this)))
          (if (typep this 'num)
              (let ((pos (position this (table-cols tbl))) (dlist (discretizer this numbins)))
                (dolist (r (table-rows newtbl))
                  (setf (nth pos (row-cells r)) (num-to-descr (nth pos (row-cells r)) dlist)))
```

www.manaraa.com

```
                (setf (table-cols newtbl) (append (table-cols newtbl) (list (make-sym :name (
                    col-name this) :goalp (col-goalp this))))))
              (let ((pos (position this (table-cols tbl))))
                (dolist (r (table-rows newtbl))
                  (if (numberp (nth pos (row-cells r)))
                      (setf (nth pos (row-cells r)) (intern (write-to-string (nth pos (row-cells r
                          )))))))
                (setf (table-cols newtbl) (append (table-cols newtbl) (list this)))))))
    newtbl))
|#

#|(defun discretizetbl (tbl &optional (numbins 10))
  (minmax tbl)
  (let ((newtbl (make-table :name (table-name tbl) :klasses (table-klasses tbl))))
    (dolist (thisrow (table-rows tbl))
      (let ((thisrow2 (copy-row thisrow)))
        (push thisrow2 (table-rows newtbl))))
    (dolist (this (table-cols tbl))
      (if (typep this 'num)
          (let ((pos (position this (table-cols tbl))) (dlist (discretizer this numbins)))
            (dolist (r (table-rows newtbl))
              (setf (nth pos (row-cells r)) (num-to-descr (nth pos (row-cells r)) dlist)))
            (setf (table-cols newtbl) (append (table-cols newtbl) (list (make-sym :name (col-name
                this) :goalp (col-goalp this))))))
          (let ((pos (position this (table-cols tbl))))
            (dolist (r (table-rows newtbl))
              (if (numberp (nth pos (row-cells r)))
                  (setf (nth pos (row-cells r)) (intern (write-to-string (nth pos (row-cells r))))
                      )))
            (setf (table-cols newtbl) (append (table-cols newtbl) (list this))))))
    newtbl))|#

(defun minmax (tbl)
  (dolist (this (table-cols tbl))
    (if (typep this 'num)
        (let ((pos (position this (table-cols tbl))))
          (dolist (thisrow (table-rows tbl))
            (if (> (nth pos (row-cells thisrow)) (num-max this))
                (progn
                  (setf (num-max this) (nth pos (row-cells thisrow)))))
            (if (< (nth pos (row-cells thisrow)) (num-min this))
                (progn
                  (setf (num-min this) (nth pos (row-cells thisrow))))))))))


#|(defun num-to-descr (x dlist &optional (n 0)) ;dlist is output from discretzer function
  (let ((nthn (nth n dlist)))
    (if (<= x nthn)
        (cond
          ((= n (- (length dlist) 1)) (intern (concatenate 'string "more-than-" (write-to-string (
              nth (- n 1) dlist)))))
          ((> n 0) (intern (concatenate 'string "less-than-" (write-to-string nthn) "-
              and-more-than-" (write-to-string (nth (- n 1) dlist)))))
          (t (intern (concatenate 'string "less-than-" (write-to-string nthn)))))
        (num-to-descr x dlist (+ n 1)))))
|#
(defun num-to-descr (x dlist &optional (n 0)) ;dlist is output from discretzer function
  (let ((nthn (nth n dlist)))
    (if (<= x nthn)
        (cond
          ((= n (- (length dlist) 1)) (intern (concatenate 'string "more-than-" )))
          ((> n 0) (intern (concatenate 'string "less-than-" (write-to-string nthn) "-
              and-more-than-" (write-to-string (nth (- n 1) dlist)))))
```

www.manaraa.com

```
          (t (intern (concatenate 'string "less-than-"))))
        (num-to-descr x dlist (+ n 1)))))

(deftest !tester3will1 ()
  (data "../data/discrete-lisp/weather.lisp")
  (test (table-cols (thetable)) (table-cols (discretizetbl (thetable)))))

(deftest !tester3will2 ()
  (data "../data/numeric-lisp/weather.lisp")
  (test (discretizetbl (thetable))
        "#S(TABLE
                     :NAME $WEATHER
                     :ROWS (#S(ROW
                               :CELLS (SUNNY
                                       less-than-76.6-and-more-than-74.5
                                       less-than-71.2-and-more-than-68.1
                                       TRUE YES)
                               :CLASS YES
                               :UTILITY 0
                               :SORTKEY 1.4869120755548817d0)
                            #S(ROW
                               :CELLS (OVERCAST less-than-66.1
                                       less-than-68.1 TRUE YES)
                               :CLASS YES
                               :UTILITY 0
                               :SORTKEY 1.4617063527177772d0)
                            #S(ROW
                               :CELLS (RAINY
                                       less-than-76.6-and-more-than-74.5
                                       less-than-80.5-and-more-than-77.4
                                       FALSE YES)
                               :CLASS YES
                               :UTILITY 0
                               :SORTKEY 1.4508161515756934d0)
                            #S(ROW
                               :CELLS (OVERCAST
                                       less-than-82.9-and-more-than-80.8
                                       less-than-77.4-and-more-than-74.3
                                       FALSE YES)
                               :CLASS YES
                               :UTILITY 0
                               :SORTKEY 1.266554799861022d0)
                            #S(ROW
                               :CELLS (RAINY
                                       less-than-70.3-and-more-than-68.2
                                       less-than-96.0-and-more-than-92.9
                                       FALSE YES)
                               :CLASS YES
                               :UTILITY 0
                               :SORTKEY 1.2627522398599997d0)
                            #S(ROW
                               :CELLS (RAINY
                                       less-than-68.2-and-more-than-66.1
                                       less-than-80.5-and-more-than-77.4
                                       FALSE YES)
                               :CLASS YES
                               :UTILITY 0
                               :SORTKEY 1.1968093818857513d0)
                            #S(ROW
                               :CELLS (SUNNY
                                       less-than-70.3-and-more-than-68.2
                                       less-than-71.2-and-more-than-68.1
                                       FALSE YES)
```

www.manaraa.com

```
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.1004932973769972d0)
                             #S(ROW
                                 :CELLS (OVERCAST
                                          less-than-85.0-and-more-than-82.9
                                          less-than-86.7-and-more-than-83.6
                                          FALSE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.0607153431365122d0)
                             #S(ROW
                                 :CELLS (OVERCAST
                                          less-than-72.4-and-more-than-70.3
                                          less-than-92.9-and-more-than-89.8
                                          TRUE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.0410945777495766d0)
                             #S(ROW
                                 :CELLS (SUNNY
                                          less-than-85.0-and-more-than-82.9
                                          less-than-86.7-and-more-than-83.6
                                          FALSE NO)
                                 :CLASS NO
                                 :UTILITY 0
                                 :SORTKEY 0.45160094544602414d0)
                             #S(ROW
                                 :CELLS (SUNNY
                                          less-than-80.8-and-more-than-78.7
                                          less-than-92.9-and-more-than-89.8
                                          TRUE NO)
                                 :CLASS NO
                                 :UTILITY 0
                                 :SORTKEY 0.44694239671216807d0)
                             #S(ROW
                                 :CELLS (RAINY
                                          less-than-72.4-and-more-than-70.3
                                          less-than-92.9-and-more-than-89.8
                                          TRUE NO)
                                 :CLASS NO
                                 :UTILITY 0
                                 :SORTKEY 0.40643407928930564d0)
                             #S(ROW
                                 :CELLS (RAINY less-than-66.1
                                          less-than-71.2-and-more-than-68.1
                                          TRUE NO)
                                 :CLASS NO
                                 :UTILITY 0
                                 :SORTKEY 0.27521698080383394d0)
                             #S(ROW
                                 :CELLS (SUNNY
                                          less-than-72.4-and-more-than-70.3
                                          less-than-96.0-and-more-than-92.9
                                          FALSE NO)
                                 :CLASS NO
                                 :UTILITY 0
                                 :SORTKEY 0.2585191038174379d0))
                      :KLASSES (#S(KLASS :NAME NO :N 5)
                                #S(KLASS :NAME YES :N 9))
                      :COLS (#S(SYM
                                 :NAME FORECAST
                                 :GOALP NIL
```

www.manaraa.com

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:COUNTS␣{hash␣of␣0␣items})
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#S(SYM
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:NAME␣$TEMPERATURE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:GOALP␣NIL
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:COUNTS␣{hash␣of␣0␣items})
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#S(SYM
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:NAME␣$HUMIDITY
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:GOALP␣NIL
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:COUNTS␣{hash␣of␣0␣items})
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#S(SYM
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:NAME␣WINDY
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:GOALP␣NIL
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:COUNTS␣{hash␣of␣0␣items})
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#S(SYM
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:NAME␣!CLASS
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:GOALP␣!
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:COUNTS␣{hash␣of␣0␣items}))
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:RESULTS␣NIL)"))

(deftest !tester3will4 ()
  (reset-seed)
  (data "../data/numeric-lisp/weather.lisp")
  (test (which2 (discretizetbl (thetable) 3) t 3) nil))


(deftest !testingposition3will (&aux (col1 (make-sym :name 'apple))
                              (col2 (make-sym :name 'banana))
                              (col3 (make-sym :name 'cat)))
  (test (position col1 (list col1 col2 col3)) 0))

(deftest !3b1 ()
  (data "../data/proj3/iris.lisp")
  (discretizetbl (thetable)))



(deftest !disctabletest2 ()
  (data "../data/numeric-lisp/weather.lisp")
  (discretizetbl (thetable) 3)
  (print (thetable)))

(deftest !3a1 ()
  (data "../data/proj3/iris.lisp")
  (print (fss (discretizetbl (thetable) 3))))
```

A.1.2 boot.lisp

Some portions written by Dr. Menzies

```
;;;; config
;;;; start slime (M-x slime <RET>)
;;;; (load "boot.lisp")
;;;; Loads all other files


(let (files)
  (defun make (&optional verbosep &rest new )
    (labels ((make0 (x)
               (format t "~a␣" x) (load x)))
      (format t "~&;")
      (if new
          (setf files new))
      (if verbosep
```

المنارة للاستشارات
www.manaraa.com

```
            (mapcar #'make0 files)
            (handler-bind ; SBCL-specific
                ((style-warning #'muffle-warning))
              (mapcar #'make0 files)))
      (terpri)
      t))
 )


(make nil
      "../lib/deftest.lisp"
      "../lib/macros.lisp"
      "../lib/lib.lisp"
      "../lib/hash.lisp"
      "../lib/bestof.lisp"
      "../lib/random.lisp"
      "../lib/profile.lisp"
      "lib1.lisp"
      "bestof.lisp"
      "structs.lisp"
      "data.lisp"
      "how.lisp"
      "defstructs.lisp"
      "angle.lisp"
      "3will.lisp"
      )
```

A.1.3 defstruct.lisp

```
;All operations that edit, modify, or create the rules defstruct are here.
;
;

(defstruct rule
  class ; "Yes/No"
  ands;list of ors
  (score 0)
  avgs
  utils
  (marked nil)
  support
  (pd 0)
  (pf 0)
  (prec 0)
  (bugs 0)
  (loc 0))

#|(defmethod print-object ((r rule) stream)
  (unless (equal (rule-class r) 'blandqxzy)
    (format t "Class: ~A~%" (rule-class r)))
  (format t  "Given that:~%")
  (dolist (this (rule-ands r))
    (format t "~T~T~A is in ~A ~%" (ors-for this) (ors-values this)))
  (format t "Score: ~F~%" (rule-score r))
  (format t "Support: ~D~%" (rule-support r))
  (if (null (rule-avgs r))
      (progn
        (format t "Pd: ~D~%" (* 100 (first (rule-utils r))))
        (format t "Pf: ~D~%" (* 100 (- 1 (second (rule-utils r)))))
        (format t "Prec: ~D~%" ( * 100 (third (rule-utils r)))))
```

www.manaraa.com

```
    (format t "Averages: ~A~%~%" (rule-avgs r)))
  r)|#


(defstruct ors
  for;              weather
  values);       sunny/cloudy


(defstruct centroid
  center
  rows
  classcount)

(defun square (x)
  (* x x))

(defun combine (rlist) ;combine two rules
  (let ((r1 (first rlist)) (r2 (second rlist)))
    (let ((r3 (make-rule :class (rule-class r1) :score 0)))
      (dolist (r1ands (rule-ands r1))
        (push (copy-ors r1ands) (rule-ands r3)));deep copy to avoid errors
      (dolist (r2ands (rule-ands r2))
        (let ((no-match t))
          (dolist (r3ands (rule-ands r3))
            (progn
              (if (eq (ors-for r2ands) (ors-for r3ands))
                  (progn
                    (setf (ors-values r3ands) (sort (copy-list (remove-duplicates (concatenate '
                         list (ors-values r2ands) (ors-values r3ands)) :test #'equalp)) #'
                         string-lessp)
                      no-match nil)))))
          (if no-match
              (push r2ands (rule-ands r3)))))
      (setf (rule-ands r3) (sort (copy-list (rule-ands r3)) #'string-lessp :key #'ors-for))
      r3)))

(defun rule-check (rule table)
  (dolist (ors (rule-ands rule))
    (dolist (col (table-cols table))
      (if (equal (ors-for ors) (col-name col))
          (if (equal (ors-values ors) (col-vals col))
              (setf (ors-for ors) 'killme)))))
  (setf (rule-ands rule) (remove-kills (rule-ands rule)))
  rule)

(defun remove-kills (lst)
  (if (null lst)
      '()
      (if (equal (ors-for (first lst)) 'killme)
          (remove-kills (rest lst))
          (cons (first lst) (remove-kills (rest lst))))))



(defun to-rule (klass lst) ;take something from which in round0 and alters it to match rules
     structure
  (make-rule :class klass :ands (list (make-ors
                                       :for (second lst)
                                       :values (list (intern (string (third lst))))))
             :score (first lst)));;change to score function later)

(defun twos (lst)
```

المنارة للاستشارات
www.manaraa.com

```
  (let ((r1 (nth (randi (length lst)) lst))
        (r2 (nth (randi (length lst)) lst)))
    (if (eq r1 r2);change equals function
        (twos0 lst r1 19 361 r1)
        (let ((diff (diff-angle (rule-utils r1) (rule-utils r2))))
              (if (> 9999999 diff)
                  (list r1 r2)
                  (twos0 lst r1 19 diff r2))))))

(defun twos0 (lst r1 x min-diff min-r)
  (let ((r2 (nth (randi (length lst)) lst)))
    (if (<= x 0)
        (list r1 r2)
        (if (eq r1 r2)
            (twos0 lst r1 (- x 1) min-diff min-r)
            (let ((diff (diff-angle (rule-utils r1) (rule-utils r2))))
              (if (> 15 diff)
                  (list r1 r2)
                  (if (< diff min-diff)
                      (twos0 lst r1 (- x 1) diff r2)
                      (twos0 lst r1 (- x 1) min-diff r2))))))))


(defun explode (lst)
  (let ((out '()))
    (dolist (this lst)
      (dotimes (i (first this))
        (push (second this) out)))
    out))

(defun normalize (lst) ;lst or rules, returns ((num rule),(num rule)...)
  (let ((out '())(sum 0)) ;testing if this new style works
    (dolist (this lst)
      (if (< sum (rule-score this))
          (setf sum (rule-score this))))
    (dolist (this lst)
      (push (list (floor (* 100 (/ (rule-score this) sum))) this) out))
    out))

(defun prune (lstin)
  (let ((out1 '()) (max 0)(cntr 0))
    (dolist (this lstin)
      (unless (= 0 (rule-support this))
        (push this out1)))
    (setf out1 (sort (copy-list out1) #'> :key #'rule-score))
    (reverse out1)))


(defun score (r tbl s)
  (let ((colnums '()) (countr 0) (rowlistin (therows tbl))(a 0)(b 0)(c 0)(d 0)(loc 0)(bugs 0)(
       locyes 0)(bugsyes 0))
    (dolist (this (thecols tbl))
      (push (list (col-name this) countr) colnums)
      (incf countr))
    (dolist (this rowlistin)
      (incf loc (row-loc this))
      (incf bugs (row-bugs this))
      (if (rule-match this r tbl)
          (progn
            (incf bugsyes (row-bugs this))
            (incf locyes (row-loc this))
            (if (equal (rule-class r) (row-class this))
                (incf d)
```

www.manaraa.com

```
              (incf c)))
          (if (equal (rule-class r) (row-class this))
              (incf b)
              (incf a))))
    (setf (rule-support r) (support a b c d))
    (setf (rule-utils r) (list (pd a b c d) (- 1 (pf a b c d))(- 1 (/ locyes loc)))
          (rule-pd r) (ceiling (* 100 (pd a b c d)))
          (rule-pf r) (floor (* 100 (- 1 (pf a b c d))))
          (rule-prec r) (ceiling (* 100 (prec a b c d)))
          (rule-bugs r) (ceiling (* 100 (/ bugsyes bugs)))
          (rule-loc r) (floor (* 100 (/ locyes loc))))

    ;(setf (rule-utils r) (rule-avgs r))
    ;(sum (rule-utils r))))
    (- 1 (/ (sqrt (+ (square (pd a b c d)) (* 1000 (square (- 1 (pf a b c d)))))) (sqrt 1001)))))

(defun scoren (r tbl)
  (let ((colnums '()) (countr 0) (rowlistin (therows tbl)) (rowlistout '()) (goals '()))
    (dolist (this (thecols tbl))
      (push (list (col-name this) countr) colnums)
      (if (col-goalp this)
          (push countr goals))
      (incf countr))
    (dolist (this rowlistin)
      (if (rule-match this r tbl)
          (push this rowlistout)))
    (let ((goalsums '()) (goalavgs '()))
      (dotimes (x (length goals))
        (push 0 goalsums))
      (dolist (thisrow rowlistout)
        (let ((counter1 0))
          (dolist (g goals)
            (progn
              (incf (nth counter1 goalsums) (nth g (row-cells thisrow)))
              (incf counter1)))))
      (dolist (this goalsums)
        (if (= this 0)
            (push 0 goalavgs)
            (push (float (/ this (length rowlistout))) goalavgs)))
      (setf (rule-support r) (length rowlistout))
      (setf (rule-avgs r) goalavgs)
      (setf goals (reverse goals))
      (let ((weighted-avgs '()))
        (dolist (i goals)
          (if (eq #\! (col-goalp (nth i (table-cols tbl))))
              (if (= 0 (nth (position i goals) goalavgs))
                  (push 0 weighted-avgs)
                  (push (/ (- (nth (position i goals) goalavgs) (num-min (nth i (table-cols tbl)))
                        )
                           (- (num-max (nth i (table-cols tbl))) (num-min (nth i (table-cols tbl))
                              ))) weighted-avgs))
              (if (= 0 (nth (position i goals) goalavgs))
                  (push 0 weighted-avgs)
                  (push (/ (- (num-max (nth i (table-cols tbl))) (nth (position i goals) goalavgs)
                        )
                           (- (num-max (nth i (table-cols tbl))) (num-min (nth i (table-cols tbl))
                              ))) weighted-avgs))))
        (setf (rule-utils r) weighted-avgs)
        (magnitude weighted-avgs)))))

(defun pd (a b  c d)
  (/ (float d) (float (+ d b .0000000001))))
```

المنارة للاستشارات
www.manaraa.com

```
(defun pf (a b c d)
  (/ (float c) (float (+ a c .0000000001))))

(defun prec (a b c d)
  (/ (float d) (float (+ c d .0000000001))))

(defun support (a b c d)
  (/ (float (+ c d)) (float (+ a b c d))))

(defun accuracy (a b c d)
  (/ (float (+ a d)) (float (+ a b c d))))

(defun sum (l)
  (if (null l)
      0
      (+ (first l) (sum (rest l)))))

(defun entropy (tbl lst)
  (let ((sum 0) (l (length lst)))
    (dolist (class (table-klasses tbl) sum)
      (let ((n 0))
        (dolist (row lst)
          (if (eql (row-class row) (klass-name class))
              (incf n)))
        (unless (= n 0)
          (incf sum (* (/ n l) (- 0 (log (/ n l))))))))))

(defun getcolnum (col colnums)
  (unless (null colnums)
    (if (eq col (first (first colnums)))
        (second (first colnums))
        (getcolnum col (rest colnums)))))


(defun final-prune (lst)
  (dolist (this (copy-list lst))
    (unless (rule-marked this)
      (dolist (that (copy-list lst))
        (unless (or (equalp this that) (rule-marked that))
          (if (pareto-dominate (rule-utils that) (rule-utils this))
              (progn
                (setf (rule-marked this) t)))))))
  (remove-marks lst))

(defun pareto-dominate (lst1 lst2)
  (if (null lst1)
      t
      (if (< (first lst1) (first lst2))
          nil
          (pareto-dominate (rest lst1) (rest lst2)))))

(defun remove-marks (lst)
  (if (null lst)
      '()
      (if (rule-marked (first lst))
          (remove-marks (rest lst))
          (cons (first lst) (remove-marks (rest lst))))))
```

A.1.4 how.lisp

Some portions written by Dr. Menzies

```
;This implements the How loop (in the code still referred to as which2,
```

www.manaraa.com

```
;since which2 was the original name for this extension)
;
;


(defparameter *every* 'allqzjx)


(defun which2 (&optional (tbl (thetable)) (s 2))
  (train tbl)
  (learn tbl s)
  )

(defun which2n (&optional (tbl (thetable)))
  (learnn tbl))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 (defun train (tbl)
   (dolist (row (therows tbl))
     (how-manys (thecols tbl)     ; get the column headers
                (row-cells row)   ; get the cells
                (row-class row)   ; get the class of this row
                )))


(defun how-manys (cols cells class)
  (labels ((worker (col cell)
             (how-many class
                       (col-name col)
                       cell
                       (sym-counts col))))
    (mapcar #'worker cols cells))) ; run down cols and cells in parallel

(defun how-many (class what cell hash)
  (when (knownp cell) ; skip any cell labelled "?"
    (inch `(,class   ,what ,cell) hash)
    (inch `(,*every* ,what ,cell) hash)))

(defun inch (key hash)
  "increment a hash bucket from zero"
  (incf (gethash key hash 0)))

(defun !how-manys1 ()
  (reset-seed)
  (data "../data/discrete-lisp/weather.lisp")
  (train (thetable))
  (with-output-to-string (s)
    (dolist (col (thecols))
      (showh (sym-counts col) :stream s))))

(deftest !how-manys ()
  (test (!how-manys1)
        "(ALLQZJX FORECAST OVERCAST) = 4
        (ALLQZJX FORECAST RAINY) = 5
        (ALLQZJX FORECAST SUNNY) = 5
        (NO FORECAST RAINY) = 2
        (NO FORECAST SUNNY) = 3
        (YES FORECAST OVERCAST) = 4
        (YES FORECAST RAINY) = 3
        (YES FORECAST SUNNY) = 2
        (ALLQZJX TEMP COOL) = 4
        (ALLQZJX TEMP HOT) = 4
        (ALLQZJX TEMP MILD) = 6
```

www.manaraa.com

```
␣␣␣␣␣␣␣␣(NO_TEMP_COOL)␣=␣1
␣␣␣␣␣␣␣␣(NO_TEMP_HOT)␣=␣2
␣␣␣␣␣␣␣␣(NO_TEMP_MILD)␣=␣2
␣␣␣␣␣␣␣␣(YES_TEMP_COOL)␣=␣3
␣␣␣␣␣␣␣␣(YES_TEMP_HOT)␣=␣2
␣␣␣␣␣␣␣␣(YES_TEMP_MILD)␣=␣4
␣␣␣␣␣␣␣␣(ALLQZJX_HUMIDTY_HIGH)␣=␣7
␣␣␣␣␣␣␣␣(ALLQZJX_HUMIDTY_NORMAL)␣=␣7
␣␣␣␣␣␣␣␣(NO_HUMIDTY_HIGH)␣=␣4
␣␣␣␣␣␣␣␣(NO_HUMIDTY_NORMAL)␣=␣1
␣␣␣␣␣␣␣␣(YES_HUMIDTY_HIGH)␣=␣3
␣␣␣␣␣␣␣␣(YES_HUMIDTY_NORMAL)␣=␣6
␣␣␣␣␣␣␣␣(ALLQZJX_WINDY_FALSE)␣=␣8
␣␣␣␣␣␣␣␣(ALLQZJX_WINDY_TRUE)␣=␣6
␣␣␣␣␣␣␣␣(NO_WINDY_FALSE)␣=␣2
␣␣␣␣␣␣␣␣(NO_WINDY_TRUE)␣=␣3
␣␣␣␣␣␣␣␣(YES_WINDY_FALSE)␣=␣6
␣␣␣␣␣␣␣␣(YES_WINDY_TRUE)␣=␣3
␣␣␣␣␣␣␣␣(ALLQZJX_!PLAY_NO)␣=␣5
␣␣␣␣␣␣␣␣(ALLQZJX_!PLAY_YES)␣=␣9
␣␣␣␣␣␣␣␣(NO_!PLAY_NO)␣=␣5
␣␣␣␣␣␣␣␣(YES_!PLAY_YES)␣=␣9"))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun learn (tbl s)
  (let ((out '()))
    (dolist (target (theklasses tbl) out)
      (push (learn1  target tbl s) out))))

(defun learnn (tbl)
  (roundsn (round0n tbl) tbl))

(defun learn1 (target tbl s)
  (let ((which (round0 target tbl)))
    (rounds target which tbl s)))

(defun round0 (target tbl)
  (let ((out '()))
    (dolist (col (thecols tbl))
      (unless (col-goalp col)
        (dokeys (key (sym-counts col))
          (push (list 0 (second key) (third key)) out))))
    (remove-duplicates out :test #'equal)))

(defun init-lattices (tbl)
  (let ((out '()) (init '()))
    (dolist (col (thecols tbl))
      (unless (col-goalp col)
        (dokeys (key (sym-counts col))
          (push (list (second key) (third key)) init))))
    (dolist (this init)
      (push (make-latvertex :conditionals (list (make-conditional :for (first this)
                                                      :ors (list (second this))))) out))
    (remove-duplicates out :test #'equalp)))

(defun round0n (tbl)
  (let ((out '()))
    (dolist (row (therows tbl))
      (let ((countr 0))
        (dolist (this (row-cells row))
          (unless (col-goalp (nth countr (table-cols tbl)))
            (push (list 0 (col-name (nth countr (table-cols tbl))) this) out))
```

المنارة للاستشارات
www.manaraa.com

```
        (incf countr))))
    (remove-duplicates out :test #'equal)))

(defun rule-match (row rule table)
  (let ((colnums '()) (countr 0) (out t))
    (dolist (this (table-cols table))
      (push (list (col-name this) countr) colnums)
      (incf countr))
    (dolist (thisors (rule-ands rule) out)
      (unless (member (nth (getcolnum (ors-for thisors) colnums) (row-cells row)) (ors-values
          thisors))
        (setf out nil)))))

;(defun round0 (target tbl)
;  (let (out
;       (n (length (therows tbl))))
;    (labels
;       ((worker (hash want m what class value &aux s)
;          (if (eql class want)
;            (if (setf s (b^2/b+r hash want m n what value))
;                (push (list (round s 0.01)
;                            what value)
;                      out)))))
;     (dolist (col (thecols tbl))          ; for every column
;       (unless (col-goalp col)            ; that's not the goal
;         (dokeys (key  (sym-counts col))
;           (print key)                    ; for everything counted in that col
;           (worker (sym-counts col)        ; get the hash table counds
;                   (klass-name target)    ; what class are we targetting?
;                   (klass-n target)       ; how many of them do we have?
;                   (col-name col)         ; what is the col name?
;                   (first key)            ; what class is being counted?
;                   (third key)            ; what value we looking at?
;                   ))))
;     (sort out #'> :key #'first))))

(defun b^2/b+r  (hash want m n what value)
  (let* ((every (gethash `(,*every* ,what ,value) hash 0))
         (b0    (gethash `(,want    ,what, value) hash 0))
         (r0    (- every b0))
         (b     (/ b0 m))          ; ratio in target
         (r     (/ r0 (- n m)))) ; ration everwhere else
    (if (> b r)                   ; in more better than rester
        (/ (* b b)                ; b^2/(b+r)
           (+ b r (randf 0.0000001))))))

(defun roundsn (which tbl)
  (let ((lives 5) (rules '()) (round 1) (max 0) (zero-score t))
    (dolist (this which)
      (let ((that (to-rule 'blandqxzy this)))
        (setf (rule-score that) (scoren that tbl))
        (unless (> 16 (rule-support that))
          (push that rules))))
    (loop while (> lives 0) do
         (let ((this nil) (found nil))
           (dotimes (number 50)
             (setf this (combine (twos (explode (normalize rules)))))
             (setf (rule-score this) (scoren this tbl))
             (dolist (that rules)
               (if (equalp this that)
                   (setf found t)))
             (unless (or found (> 5 (rule-support this)))
               (push this rules))))
```

المنارة للاستشارات
www.manaraa.com

```
        (setf rules (prune rules))
        (if (> (rule-score (first rules)) max)
            (setf max (rule-score (first rules))
                  lives 6))
        (incf round)
        (decf lives))
    (final-prune rules)))

(defun rounds (class which tbl s)
  (let ((lives 5) (rules '()) (round 1) (max 0) (zero-score t))
    (dolist (this which)
      (push (to-rule (klass-name class) this) rules))
    (dolist (r rules)
      (setf (rule-score r) (score r tbl s))
      (if (> (rule-score r) 0)
          (setf zero-score nil)))
    (if zero-score
        rules
        (progn
          (loop while (> lives 0) do
               (let ((this nil) (found nil))
                 (dotimes (number 20)
                   (setf this (combine (twos (explode (normalize rules)))))
                   (setf this (rule-check this tbl))
                   (setf (rule-score this) (score this tbl s))
                   (dolist (that rules)
                     (if (equalp this that)
                         (setf found t)))
                   (unless (or found (= 0 (rule-support this)))
                     (push this rules)))
                 (setf rules (prune rules))
                 (if (> (rule-score (first rules)) max)
                     (setf max (rule-score (first rules))
                           lives 6))
                 (incf round)
                 (decf lives)))
          (final-prune rules)))))

(defun !table1 ()
  (data "../data/LispData/xerces/05-xerces-abc.lisp")
  (discretizetbl (thetable) 2))


(defun !lattice ()
  (data "../data/LispData/velocity/03-velocity-ab.lisp")
  (let ((tbl (discretizetbl (thetable) 2)))
    (let (rules)
      (dolist (this (which2 tbl))
        (if (equal (rule-class (first this)) 'BUGS)
            (setf rules this)))
      (let ((outfile (open "velocity.lmb" :direction :output :if-exists :supersede)))
        (write-line "LM_BINARY_CONTEXT" outfile)
        (let ((column-labels '()) (row-labels "") (rowline '()) (rows '()) (n 1))
          (dolist (r rules)
            (dolist (this (rule-ands r))
              (dolist (that (ors-values this))
                (push (intern (concatenate 'string (symbol-name (ors-for this)) ":" (symbol-name
                          that))) column-labels)))
            (setf row-labels (concatenate 'string row-labels "|_PD:" (write-to-string (rule-pd r))
                       "_PF:" (write-to-string (rule-pf r)) "_PREC:" (write-to-string (rule-prec r)))
                       n (+ n 1)))
          (setf column-labels (remove-duplicates column-labels))
          (dolist (r rules)
```

www.manaraa.com

```
            (setf rowline '())
            (dotimes (i (length column-labels))
              (push 0 rowline))
            (dolist (this (rule-ands r))
              (dolist (that (ors-values this))
                (if (position (intern (concatenate 'string (symbol-name (ors-for this)) ":" (
                    symbol-name that))) column-labels)
                    (setf (nth (position (intern (concatenate 'string (symbol-name (ors-for this))
                         ":" (symbol-name that))) column-labels)
                              rowline) 1))))
            (push rowline rows))
          (let ((column-line ""))
            (dolist (c column-labels)
              (setf column-line (concatenate 'string column-line "|␣" (symbol-name c) "␣")))
            (write-line row-labels outfile)
            (write-line column-line outfile))
          (dolist (row rows)
            (let ((row-string ""))
              (dolist (this row)
                (setf row-string (concatenate 'string row-string (write-to-string this) "␣")))
              (write-line row-string outfile))))
        (close outfile)))))


(defun !stability ()
  (let ((whichout (which2 (!table1))))
    (let ((rules '()))
      (dolist (r whichout)
        (if (equal 'BUGS (rule-class (first r)))
            (setf rules r)))
      (data "../data/LispData/velocity/03-velocity-ab.lisp")
      (let ((tbl (discretizetbl (thetable) 2)) (out '()))

        (dolist (r rules)
          (unless (equal 'KILLME (ors-for (first (rule-ands r))))
            (let ((temp (make-rule :class 'BUGS :ands (copy-list (rule-ands r)))))
              (score temp tbl 2)
              (print "*****************")
                                   ;(print r)
                                   ;(print temp)))))))
              (print "Original")
              (print r)
              (print "New")
              (print temp)
              )))))))
(defun !learn1 ()
  (let ((rules (which2 (!table1) 2)))
    (dolist (this rules)
      (if (equal (rule-class (first this)) 'BUGS)
          (print this)
          (print (length this))))))


;(data "../data/LispData/velocity/03-velocity-ab.lisp")
;     (dolist (that rules)
;       (print that)
;       (score that (thetable) 0)
;       (print that))))

(defun !table2 ()
  (data "../data/LispData/velocity/03-velocity-ab.lisp")
  (discretizetbl (thetable) 2))
```

www.manaraa.com

```
(defun !learn2 ()
  (print-output (which2 (!table2) 4)))

(defun !table3 ()
  (data "../data/LispData/pbeans/01-pbeans-a.lisp")
  (discretizetbl (thetable) 2))

(defun !learn3 ()
  (which2 (!table3) 4))

(defun !table4 ()
  (data "../data/LispData/xerces/05-xerces-abc.lisp")
  (discretizetbl (thetable) 2))

(defun !learn4 ()
  (which2 (!table4) 4))

(defun !run1 ()
  (data "../data/LispData//05-xerces-abc.lisp")
  (let ((table (discretizetbl (thetable) 2)))
    (let ((rules (which2 table 4)))
      (dolist (r rules)
        (let ((vertices (init-lattices table)))
          (print "****************************************")
          (print "****************************************")
          (print "****************************************")
          (lattice-builder vertices r))))))

      ;1)))


(defun table-averages (tbl)
  (let ((out '()) (l (length (row-cells (first (table-rows tbl))))))
    (dolist (this (table-rows tbl) out)
      (push (list (nth (- l 1) (row-cells this)) (nth (- l 2) (row-cells this))) out))))

(defun print-output (rlst)
  (dolist (rules rlst)
    (if (equal (rule-class (first rules)) 'BUGS)
        (dolist (rule rules)
          (print (concatenate 'string (write-to-string (rule-pd rule))
                              "␣" (write-to-string (rule-loc rule)))))))))
#|defun first-util (r1 r2)
  (if (= (ceiling (first (rule-utils r1))) (ceiling * 1(second (rule-utils r2))))
       (> (third (rule-utils r1)) (third (rule-utils r2)))
       (< (first (rule-utils r1)) (first (rule-utils r2)))))

(defun second-util (r1 r2)
  (> (third (rule-utils r1)) (third (rule-utils r2))))
                                        ;(deftest !learn ()|#
;  (test (!learn1)
;        ";;; NO
;           (56 HUMIDTY HIGH)
;           (44 FORECAST SUNNY)
;           (39 WINDY TRUE)
;           (26 TEMP HOT)
;           (22 FORECAST RAINY)
        ;;; YES
;           (51 HUMIDTY NORMAL)
;           (44 FORECAST OVERCAST)
;           (42 WINDY FALSE)
;           (23 TEMP MILD)
;           (21 TEMP COOL)"))
```

المنارة للاستشارات
www.manaraa.com

A.2 FrontierFinder

A.2.1 3will.lisp

```
;Handles discretization
;
;

(defun discretizer (column numbins)
  (let ((out '()))
    (push (float (- (num-min column) .00001)) out)
    (dotimes (x numbins out)
      (if (= x (- numbins 1))
          (push (float (+ (num-max column) .00001)) out)
          (push (float (+ (num-min column)
                          (* (+ x 1)
                             (/ (- (num-max column) (num-min column)) numbins))))
                out)))
    (reverse out)))

(deftest !discr1 (&aux (column (make-num :max 280 :min 10)) (numbins 3))
  (test (discretizer column numbins) '(100 190 280)))

(defun discretizetbln (tbl &optional (numbins 10))
  (minmax tbl)
  (let ((newtbl (make-table :name (table-name tbl) :klasses (table-klasses tbl))))
    (dolist (thisrow (table-rows tbl))
      (let ((thisrow2 (copy-row thisrow)))
        (push thisrow2 (table-rows newtbl))))
    (dolist (this (table-cols tbl))
      (if (col-goalp this)
          (setf (table-cols newtbl) (append (table-cols newtbl) (list this)))
          (if (typep this 'num)
              (let ((pos (position this (table-cols tbl))) (dlist (discretizer this numbins)))
                (dolist (r (table-rows newtbl))
                  (setf (nth pos (row-cells r)) (num-to-descr (nth pos (row-cells r)) dlist)))
                (setf (table-cols newtbl) (append (table-cols newtbl) (list (make-sym :name (
                    col-name this) :goalp (col-goalp this))))))
              (let ((pos (position this (table-cols tbl))))
                (dolist (r (table-rows newtbl))
                  (if (numberp (nth pos (row-cells r)))
                      (setf (nth pos (row-cells r)) (intern (write-to-string (nth pos (row-cells r
                          )))))))
                (setf (table-cols newtbl) (append (table-cols newtbl) (list this))))))))
    newtbl))


(defun discretizetbl (tbl &optional (numbins 10))
  (minmax tbl)
  (let ((newtbl (make-table :name (table-name tbl) :klasses (table-klasses tbl))))
    (dolist (thisrow (table-rows tbl))
      (let ((thisrow2 (copy-row thisrow)))
        (push thisrow2 (table-rows newtbl))))
    (dolist (this (table-cols tbl))
      (if (typep this 'num)
          (let ((pos (position this (table-cols tbl))) (dlist (discretizer this numbins)))
            (dolist (r (table-rows newtbl))
              (setf (nth pos (row-cells r)) (num-to-descr (nth pos (row-cells r)) dlist)))
            (setf (table-cols newtbl) (append (table-cols newtbl) (list (make-sym :name (col-name
                this) :goalp (col-goalp this))))))
          (let ((pos (position this (table-cols tbl))))
```

www.manaraa.com

```
            (dolist (r (table-rows newtbl))
              (if (numberp (nth pos (row-cells r)))
                  (setf (nth pos (row-cells r)) (intern (write-to-string (nth pos (row-cells r))))
                      )))
            (setf (table-cols newtbl) (append (table-cols newtbl) (list this))))))
    newtbl))


(defun minmax (tbl)
  (dolist (this (table-cols tbl))
    (if (typep this 'num)
        (let ((pos (position this (table-cols tbl))))
          (dolist (thisrow (table-rows tbl))
            (if (> (nth pos (row-cells thisrow)) (num-max this))
                (progn
                  (setf (num-max this) (nth pos (row-cells thisrow)))))
            (if (< (nth pos (row-cells thisrow)) (num-min this))
                (progn
                  (setf (num-min this) (nth pos (row-cells thisrow))))))))))


(defun num-to-descr (x dlist &optional (n 1)) ;dlist is output from discretzer function
  (let ((nthn (nth n dlist)))
    (if (<= x nthn)
        (intern (concatenate 'string  (write-to-string nthn) "," (write-to-string (nth (- n 1)
             dlist))))
        (num-to-descr x dlist (+ n 1)))))


(deftest !tester3will1 ()
  (data "../data/discrete-lisp/weather.lisp")
  (test (table-cols (thetable)) (table-cols (discretizetbl (thetable)))))

(deftest !tester3will2 ()
  (data "../data/numeric-lisp/weather.lisp")
  (test (discretizetbl (thetable))
        "#S(TABLE
                      :NAME $WEATHER
                      :ROWS (#S(ROW
                                 :CELLS (SUNNY
                                         less-than-76.6-and-more-than-74.5
                                         less-than-71.2-and-more-than-68.1
                                         TRUE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.4869120755548817d0)
                             #S(ROW
                                 :CELLS (OVERCAST less-than-66.1
                                         less-than-68.1 TRUE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.4617063527177772d0)
                             #S(ROW
                                 :CELLS (RAINY
                                         less-than-76.6-and-more-than-74.5
                                         less-than-80.5-and-more-than-77.4
                                         FALSE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.4508161515756934d0)
                             #S(ROW
                                 :CELLS (OVERCAST
                                         less-than-82.9-and-more-than-80.8
```

www.manaraa.com

```
                                         less-than-77.4-and-more-than-74.3
                                         FALSE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.266554799861022d0)
                             #S(ROW
                                 :CELLS (RAINY
                                         less-than-70.3-and-more-than-68.2
                                         less-than-96.0-and-more-than-92.9
                                         FALSE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.2627522398599997d0)
                             #S(ROW
                                 :CELLS (RAINY
                                         less-than-68.2-and-more-than-66.1
                                         less-than-80.5-and-more-than-77.4
                                         FALSE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.1968093818857513d0)
                             #S(ROW
                                 :CELLS (SUNNY
                                         less-than-70.3-and-more-than-68.2
                                         less-than-71.2-and-more-than-68.1
                                         FALSE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.1004932973769972d0)
                             #S(ROW
                                 :CELLS (OVERCAST
                                         less-than-85.0-and-more-than-82.9
                                         less-than-86.7-and-more-than-83.6
                                         FALSE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.0607153431365122d0)
                             #S(ROW
                                 :CELLS (OVERCAST
                                         less-than-72.4-and-more-than-70.3
                                         less-than-92.9-and-more-than-89.8
                                         TRUE YES)
                                 :CLASS YES
                                 :UTILITY 0
                                 :SORTKEY 1.0410945777495766d0)
                             #S(ROW
                                 :CELLS (SUNNY
                                         less-than-85.0-and-more-than-82.9
                                         less-than-86.7-and-more-than-83.6
                                         FALSE NO)
                                 :CLASS NO
                                 :UTILITY 0
                                 :SORTKEY 0.45160094544602414d0)
                             #S(ROW
                                 :CELLS (SUNNY
                                         less-than-80.8-and-more-than-78.7
                                         less-than-92.9-and-more-than-89.8
                                         TRUE NO)
                                 :CLASS NO
                                 :UTILITY 0
                                 :SORTKEY 0.44694239671216807d0)
                             #S(ROW
                                 :CELLS (RAINY
```

المنارة للاستشارات
www.manaraa.com

```
                                        less-than-72.4-and-more-than-70.3
                                        less-than-92.9-and-more-than-89.8
                                        TRUE NO)
                                :CLASS NO
                                :UTILITY 0
                                :SORTKEY 0.40643407928930564d0)
                             #S(ROW
                                :CELLS (RAINY less-than-66.1
                                        less-than-71.2-and-more-than-68.1
                                        TRUE NO)
                                :CLASS NO
                                :UTILITY 0
                                :SORTKEY 0.27521698080383394d0)
                             #S(ROW
                                :CELLS (SUNNY
                                        less-than-72.4-and-more-than-70.3
                                        less-than-96.0-and-more-than-92.9
                                        FALSE NO)
                                :CLASS NO
                                :UTILITY 0
                                :SORTKEY 0.2585191038174379d0))
                     :KLASSES (#S(KLASS :NAME NO :N 5)
                                #S(KLASS :NAME YES :N 9))
                     :COLS (#S(SYM
                                :NAME FORECAST
                                :GOALP NIL
                                :COUNTS {hash of 0 items})
                             #S(SYM
                                :NAME $TEMPERATURE
                                :GOALP NIL
                                :COUNTS {hash of 0 items})
                             #S(SYM
                                :NAME $HUMIDITY
                                :GOALP NIL
                                :COUNTS {hash of 0 items})
                             #S(SYM
                                :NAME WINDY
                                :GOALP NIL
                                :COUNTS {hash of 0 items})
                             #S(SYM
                                :NAME !CLASS
                                :GOALP !
                                :COUNTS {hash of 0 items}))
                     :RESULTS NIL)"))

(deftest !tester3will4 ()
  (reset-seed)
  (data "../data/numeric-lisp/weather.lisp")
  (test (which2 (discretizetbl (thetable) 3) t 3) nil))


(deftest !testingposition3will (&aux (col1 (make-sym :name 'apple))
                              (col2 (make-sym :name 'banana))
                              (col3 (make-sym :name 'cat)))
  (test (position col1 (list col1 col2 col3)) 0))

(deftest !3b1 ()
  (data "../data/proj3/iris.lisp")
  (discretizetbl (thetable)))



(deftest !disctabletest2 ()
```

المنارة للاستشارات
www.manaraa.com

```
  (data "../data/numeric-lisp/weather.lisp")
  (discretizetbl (thetable) 3)
  (print (thetable)))

(deftest !3a1 ()
  (data "../data/proj3/iris.lisp")
  (print (fss (discretizetbl (thetable) 3))))
```

A.2.2 boot.lisp

Some portions written by Dr. Menzies

```
;;;; config
;;;; start slime (M-x slime <RET>)
;;;; (load "boot.lisp")
; Loads all other files in FrontierFinder
;
(let (files)
  (defun make (&optional verbosep &rest new )
    (labels ((make0 (x)
                (format t "~a_" x) (load x)))
      (format t "~&;")
      (if new
          (setf files new))
      (if verbosep
          (mapcar #'make0 files)
          (handler-bind ; SBCL-specific
              ((style-warning #'muffle-warning))
            (mapcar #'make0 files)))
      (terpri)
      t))
 )


(make nil
      "../lib/deftest.lisp"
      "../lib/macros.lisp"
      "../lib/lib.lisp"
      "../lib/hash.lisp"
      "../lib/bestof.lisp"
      "../lib/random.lisp"
      "../lib/profile.lisp"
      "lib1.lisp"
      "bestof.lisp"
      "structs.lisp"
      "data.lisp"
      "how.lisp"
      "defstructs.lisp"
      "angle.lisp"
      "3will.lisp"
      "testtables.lisp"
      "ruletree.lisp"
      )
```

A.2.3 defstructs.lisp

```
; Controls the rule defstruct, including creates, edits, combines,
; scoring, et al
;
;
```

المنارة للاستشارات
www.manaraa.com

```
(defstruct rule
  class ; "Yes/No"
  ands;list of ors
  score
  avgs
  utils
  (marked nil)
  support)

;(defmethod print-object ((r rule) stream)
;  (unless (equal (rule-class r) 'blandqxzy)
;    (format t "Class: ~A~%" (rule-class r)))
;  (format t  "Given that:~%")
;  (dolist (this (rule-ands r))
;    (format t "~T~T~A is in ~A ~%" (ors-for this) (ors-values this)))
;  (format t "Score: ~F~%" (rule-score r))
;  (format t "Support: ~D~%" (rule-support r))
;  (if (null (rule-avgs r))
;      (progn
;        (format t "Pd: ~D~%" (* 100 (first (rule-utils r))))
;        (format t "Pf: ~D~%" (* 100 (- 1 (second (rule-utils r)))))
;        (format t "Prec: ~D~%" ( * 100 (third (rule-utils r)))))
;      (format t "Averages: ~A~%~%" (rule-avgs r)))
;  r)


(defstruct ors
  for;           weather
  values);       sunny/cloudy


(defstruct centroid
  center
  rows
  classcount)

(defun square (x)
  (* x x))

(defun combine (rlist) ;combine two rules
  (let ((r1 (first rlist)) (r2 (second rlist)))
    (let ((r3 (make-rule :class (rule-class r1) :score 0)))
      (dolist (r1ands (rule-ands r1))
        (push (copy-ors r1ands) (rule-ands r3)));deep copy to avoid errors
      (dolist (r2ands (rule-ands r2))
        (let ((no-match t))
          (dolist (r3ands (rule-ands r3))
            (progn
              (if (eq (ors-for r2ands) (ors-for r3ands))
                  (progn
                    (setf (ors-values r3ands) (sort (copy-list (remove-duplicates (concatenate '
                         list (ors-values r2ands) (ors-values r3ands)) :test #'equalp)) #'
                         string-lessp)
                          no-match nil)))))
          (if no-match
              (push r2ands (rule-ands r3)))))
      (setf (rule-ands r3) (sort (copy-list (rule-ands r3)) #'string-lessp :key #'ors-for))
      r3)))

(defun to-rule (klass lst) ;take something from which in round0 and alters it to match rules
     structure
```

www.manaraa.com

```
  (make-rule :class klass :ands (list (make-ors
                                        :for (second lst)
                                        :values (list (intern (string (third lst))))))
            :score (first lst)));;change to score function later)

(defun twos (lst)
  (let ((r1 (nth (randi (length lst)) lst))
        (r2 (nth (randi (length lst)) lst)))
    (if (eq r1 r2);change equals function
        (twos0 lst r1 19 361 r1)
        (let ((diff (diff-angle (rule-utils r1) (rule-utils r2))))
            (if (> 9999 diff)
                (list r1 r2)
                (twos0 lst r1 19 diff r2))))))

(defun twos0 (lst r1 x min-diff min-r)
  (let ((r2 (nth (randi (length lst)) lst)))
    (if (<= x 0)
        (list r1 r2)
        (if (eq r1 r2)
            (twos0 lst r1 (- x 1) min-diff min-r)
            (let ((diff (diff-angle (rule-utils r1) (rule-utils r2))))
              (if (> 15 diff)
                  (list r1 r2)
                  (if (< diff min-diff)
                      (twos0 lst r1 (- x 1) diff r2)
                      (twos0 lst r1 (- x 1) min-diff r2))))))))


(defun explode (lst)
  (let ((out '()))
    (dolist (this lst)
      (dotimes (i (first this))
        (push (second this) out)))
    out))

(defun normalize (lst) ;lst or rules, returns ((num rule),(num rule)...)
  (let ((out '())(sum 0)) ;testing if this new style works
    (dolist (this lst)
      (if (< sum (rule-score this))
          (setf sum (rule-score this))))
    (dolist (this lst)
      (push (list (floor (* 100 (/ (rule-score this) sum))) this) out))
    out))

(defun prune (lstin)
  (let ((out1 '()) (max 0)(cntr 0))
    (dolist (this lstin)
      (unless (= 0 (rule-support this))
        (push this out1)))
    (setf out1 (sort (copy-list out1) #'> :key #'rule-score))
    (reverse out1)))


(defun score (r tbl s)
  (let ((colnums '()) (countr 0) (rowlistin (therows tbl))(a 0)(b 0)(c 0)(d 0)(loc 0)(bugs 0))
    (dolist (this (thecols tbl))
      (push (list (col-name this) countr) colnums)
      (incf countr))
    (dolist (this rowlistin)
      (if (rule-match this r tbl)
          (progn
            (incf bugs 0)
```

المنارة للاستشارات
www.manaraa.com

```
            (if (equal (rule-class r) (row-class this))
                (incf d)
                (incf c)))
          (if (equal (rule-class r) (row-class this))
              (incf b)
              (incf a))))
    (setf (rule-support r) (support a b c d))
    (setf (rule-avgs r) (list (* 100 (pd a b c d)) (* 100 (- 1 (pf a b c d))) (* 100 (prec a b c d
        ))))
    (print (rule-avgs r))
    (setf (rule-utils r) (rule-avgs r))
    (sum (rule-avgs r))))

(defun scoren (r tbl)
  (let ((colnums '()) (countr 0) (rowlistin (therows tbl)) (rowlistout '()) (goals '()))
    (dolist (this (thecols tbl))
      (push (list (col-name this) countr) colnums)
      (if (col-goalp this)
          (push countr goals))
      (incf countr))
    (dolist (this rowlistin)
      (if (rule-match this r tbl)
          (push this rowlistout)))
    (let ((goalsums '()) (goalavgs '()))
      (dotimes (x (length goals))
        (push 0 goalsums))
      (dolist (thisrow rowlistout)
        (let ((counter1 0))
          (dolist (g goals)
            (progn
              (incf (nth counter1 goalsums) (nth g (row-cells thisrow)))
              (incf counter1)))))
      (dolist (this goalsums)
        (if (= this 0)
            (push 0 goalavgs)
            (push (float (/ this (length rowlistout))) goalavgs)))
      (setf (rule-support r) (length rowlistout))
      (setf (rule-avgs r) goalavgs)
      (setf goals (reverse goals))
      (let ((weighted-avgs '()))
        (dolist (i goals)
          (if (eq #\! (col-goalp (nth i (table-cols tbl))))
              (if (= 0 (nth (position i goals) goalavgs))
                  (push 0 weighted-avgs)
                  (push (/ (- (nth (position i goals) goalavgs) (num-min (nth i (table-cols tbl)))
                      )
                           (- (num-max (nth i (table-cols tbl))) (num-min (nth i (table-cols tbl))
                               ))) weighted-avgs))
              (if (= 0 (nth (position i goals) goalavgs))
                  (push 0 weighted-avgs)
                  (push (/ (- (num-max (nth i (table-cols tbl))) (nth (position i goals) goalavgs)
                      )
                           (- (num-max (nth i (table-cols tbl))) (num-min (nth i (table-cols tbl))
                               ))) weighted-avgs))))
        (setf (rule-utils r) weighted-avgs)
        (magnitude weighted-avgs)))))

(defun pd (a b  c d)
  (/ (float d) (float (+ d b .0000000001))))

(defun pf (a b c d)
  (/ (float c) (float (+ a c .0000000001))))
```

www.manaraa.com

```
(defun prec (a b c d)
  (/ (float d) (float (+ c d .0000000001))))

(defun support (a b c d)
  (/ (float (+ c d)) (float (+ a b c d))))

(defun accuracy (a b c d)
  (/ (float (+ a d)) (float (+ a b c d))))

(defun sum (l)
  (if (null l)
      0
      (+ (first l) (sum (rest l)))))

(defun entropy (tbl lst)
  (let ((sum 0) (l (length lst)))
    (dolist (class (table-klasses tbl) sum)
      (let ((n 0))
        (dolist (row lst)
          (if (eql (row-class row) (klass-name class))
              (incf n)))
        (unless (= n 0)
          (incf sum (* (/ n l) (- 0 (log (/ n l))))))))))

(defun getcolnum (col colnums)
  (unless (null colnums)
    (if (eq col (first (first colnums)))
        (second (first colnums))
        (getcolnum col (rest colnums)))))


(defun final-prune (lst)
  (dolist (this (copy-list lst))
    (unless (rule-marked this)
      (dolist (that (copy-list lst))
        (unless (or (equalp this that) (rule-marked that))
          (if (pareto-dominate (rule-utils that) (rule-utils this))
              (progn
                (setf (rule-marked this) t)))))))
  (remove-marks lst))

(defun final-prune-2 (lst)
  (dolist (this (copy-list lst))
    (unless (rule-marked this)
      (dolist (that (copy-list lst))
        (unless (or (equalp this that) (rule-marked that))
          (if (pareto-dominate-min (rule-avgs that) (rule-avgs this))
              (progn
                (setf (rule-marked this) t)))))))
  (remove-marks lst))

(defun pareto-dominate (lst1 lst2)
  (if (null lst1)
      t
      (if (< (first lst1) (first lst2))
          nil
          (pareto-dominate (rest lst1) (rest lst2)))))

(defun pareto-dominate-min (lst1 lst2)
  (if (null lst1)
      t
      (if (> (first lst1) (first lst2))
          nil
```

www.manaraa.com

```
        (pareto-dominate-min (rest lst1) (rest lst2)))))

(defun remove-marks (lst)
  (if (null lst)
      '()
      (if (rule-marked (first lst))
          (remove-marks (rest lst))
          (cons (first lst) (remove-marks (rest lst))))))
```

A.2.4 ruletree.lisp

```
; The "breadth-first" style search for handling math models
;
;

(defun rule-to-constraint (rule table constraints) ;output = '('('(0 1) '(1 2)) '('(0 .5)))
  (dolist (this (rule-ands rule) constraints)
    (unless (> (length (ors-values this)) 1)
      (let ((in-col (ors-for this)) new-constraints)
        (dolist (that (ors-values this))
          (let ((new-min (get-min-from-string (symbol-name that)))
                (new-max (get-max-from-string (symbol-name that))))
            (push (list new-min new-max) new-constraints)))
        (setf (nth (get-table-col-num in-col table) constraints) new-constraints)))))

(defun get-max-from-string (str)
  (read-from-string (subseq str 1 (search "," str))))

(defun get-min-from-string (str)
  (read-from-string (subseq str (+ 1 (search "," str)))))

(defun any (lst)
  (nth (randi (length lst)) lst))

(defun get-table-col-num (in-col table)
  (let ((out -1) (n 0))
    (dolist (col (table-cols table) out)
      (if (eq (col-name col) in-col)
          (progn
            (setf out n)
            (return))
          (incf n)))
    out))

(defun test-rule-to-constraint ()
  (fonseca-search (list (list -4 4)) (list (list -4 4)) (list (list -4 4))))
  (data "../data/tests/kursawe.lisp")
  (dolist (rule (which2n (discretizetbln (thetable) 2)))
    (print (rule-to-constraint rule (thetable) (list (list (list -4 4)) (list (list -4 4)) (list (
         list -4 4)))))))

(defun tree-search-test ()
  (let ((initial-constraints (list (list (list -4 4)) (list (list -4 4)) (list (list -4 4)))))
    (fonseca-search initial-constraints 1000)
    (data "../data/tests/fonseca.lisp")
    (let ((rules (which2n (discretizetbln (thetable) 16))))
      (dotimes (n 20)
        (let ((newrules rules)
              (outfile (open (concatenate 'string "generation" (write-to-string n) ".dat") :
                   direction :output :if-exists :supersede)))
          (dolist (rule rules)
            (fonseca-search (rule-to-constraint rule (thetable) initial-constraints))
```

المنارة للاستشارات
www.manaraa.com

```
          (data "../data/tests/fonseca.lisp")
          (setf newrules (append newrules
                                 (which2n (discretizetbln (thetable) 16)))))
        (setf rules (final-prune-2 newrules))
        (setf rules (crowd-prune-3 rules .9))
        (print (length rules))
        (print (concatenate 'string "******" (write-to-string n) "******"))
        (let (ruleavgs)
          (dolist (rule rules)
            (push (rule-avgs rule) ruleavgs))
          (setf ruleavgs (copy-list (sort (copy-list ruleavgs) #'first-avg)))
          (dolist (that ruleavgs)
            (write-line (concatenate 'string (write-to-string (first that)) "_" (write-to-string
                 (second that))) outfile)))
        (close outfile)))))
  (print "done"))

(defun frontier-finder (file func init-constraints d &optional (samples 100) (alpha .9)(nrounds
     10))
  (funcall func init-constraints samples)
  (data (concatenate 'string "../data/tests/" file ".lisp"))
  (let ((rules (which2n (discretizetbln (thetable) d))))
    (dotimes (n nrounds)
      (let ((newrules rules)
            (outfile (open (concatenate 'string "output/" file "/" (write-to-string d) "/
                 generation" (write-to-string n) ".dat") :direction :output :if-exists :supersede :
                 if-does-not-exist :create)))
        (dolist (rule rules)
          (funcall func (rule-to-constraint rule (thetable) init-constraints) samples)
          (data (concatenate 'string "../data/tests/" file ".lisp"))
          (unless (<= (length (table-rows (thetable))) 20)
            (setf newrules (append newrules (which2n (discretizetbln (thetable) d))))))
        (setf rules (final-prune-2 newrules))
        (setf rules (crowd-prune-3 rules alpha))
        (let (ruleavgs)
          (dolist (rule rules)
            (push (rule-avgs rule) ruleavgs))
          (setf ruleavgs (copy-list (sort (copy-list ruleavgs) #'first-avg)))
          (dolist (that ruleavgs)
            (write-string (write-to-string (first that)) outfile)
            (dolist (item (rest that))
              (write-string (concatenate 'string "_" (write-to-string item)) outfile)
              (write-line "" outfile))))
        (print (concatenate 'string "*****" (write-to-string n) "*****"))
        (close outfile))))
  (print "done"))

(defun ff-test ()
  (frontier-finder "fonseca"
                   #'fonseca-search
                   (list (list (list -4 4)) (list (list -4 4)) (list (list -4 4)))
                   8
                   1000
                   .6
                   10))

(defun ff-fonseca ()
  (let ((dlst (list 2 4 8 16 32 64)))
    (dolist (d dlst)
      (frontier-finder "fonseca"
                   #'fonseca-search
                   (list (list (list -4 4)) (list (list -4 4)) (list (list -4 4)))
                   d
```

المنارة للاستشارات
www.manaraa.com

```
                   1000
                   .5))))

(defun ff-kursawe ()
  (let ((dlst (list 2 4 8 16 32 64)))
    (dolist (d dlst)
      (frontier-finder "kursawe"
                   #'kursawe-search
                   (list (list (list -5 5)) (list (list -5 5)) (list (list -5 5)))
                   d
                   1000
                   .5))))

(defun ff-schaffer ()
  (let ((dlst (list 2 4 8 16 32 64)))
    (dolist (d dlst)
      (frontier-finder "schaffer"
                       #'schaffer-search
                       (list (list (list -100000 100000)))
                       d
                       1000
                       .5))))

(defun ff-constrex ()
  (let ((dlst (list 2 4 8 16)))
    (dolist (d dlst)
      (frontier-finder "constrex"
                       #'constrex-search
                       (list (list (list .1 1.0)) (list (list 0.0 5.0)))
                       d
                       1000
                       .5))))

(defun ff-srinivas ()
  (let ((dlst (list 2 4 8 16)))
    (dolist (d dlst)
      (frontier-finder "srinivas"
                       #'srinivas-search
                       (list (list (list -20 20)) (list (list -20 20)))
                       d
                       1000
                       .5))))

(defun ff-tanaka ()
  (let ((dlst (list 2 4 8 16 32 64)))
    (dolist (d dlst)
      (frontier-finder "tanaka"
                       #'tanaka-search
                       (list (list (list (- 0 pi) pi)) (list (list (- 0 pi) pi)))
                       d
                       1000
                       .5))))


(defun weekend-run ()
  (ff-fonseca)
  (ff-kursawe))

(defun crowd-prune-3 (rules alpha)
  (let (out current)
    (setf rules (sort (copy-list rules) #'first-avg-2))
    (dolist (rule rules)
      (unless current
```

المنارة للاستشارات
www.manaraa.com

```
      (setf current rule)
      (push current out))
    (if (> (diff-angle (rule-avgs rule) (rule-avgs current)) alpha)
      (progn
        (push rule out)
        (setf current rule))))
  out))


(defun crowd-prune-2 (rules n)
  (let (out)
    (if (< n (length rules))
        (progn
          (setf rules (sort (copy-list rules) #'first-avg-2))
          (let ((incr (/ (float n) (float (length rules)))) (step 0.0))
            (dolist (rule rules)
              (if (<= 0 step)
                  (progn
                    (push rule out)
                    (decf step 1.0)))
              (incf step incr)))
          out)
        rules)))

(defun crowd-prune (rules n)
  (let (out)
    (if (< n (length rules))
        (progn
          (setf rules (sort (copy-list rules) #'first-avg-2))
          (let ((chance-kept (/ (float n) (length rules))))
            (dolist (rule rules)
              (if (< (randf 1.0) chance-kept)
                  (push rule out)))))
        (setf out rules))
    out))


(defun first-avg (tuple1 tuple2)
  (< (first tuple1) (first tuple2)))

(defun first-avg-2 (rule1 rule2)
  (< (first (rule-avgs rule1)) (first (rule-avgs rule2))))
```

A.2.5 testtables.lisp

```
; Generates the table data  in a .lisp file format for the math models testing.
;

(defun schaffer (&optional (samples 10000))
  (let ((outfile (open "../data/tests/schaffer.lisp" :direction :output :if-exists :supersede)))
    (write-line "(deftable_$schaffer_$x0_$#f1_$#f2)" outfile)
    (dotimes (number samples)
      (let ((x1 (- (randi 20000) 10000)))
        (let ((f1 (square x1))(f2 (square (- x1 2))))
          (write-line (concatenate 'string "(!_" (write-to-string x1) "_"
                                   (write-to-string f1) "_"
                                   (write-to-string f2) ")") outfile))))
    (close outfile)))

(defun schaffer-search (constraints &optional (samples 1000))
  (let ((outfile (open "../data/tests/schaffer.lisp" :direction :output :if-exists :supersede)))
    (write-line "(deftable_$schaffer_$x0_$#f1_$#f2)" outfile)
```

www.manaraa.com

```
    (dotimes (number samples)
      (let ((that (any (first constraints))))
        (let ((x (+ (randf (- (second that) (first that))) (first that))))
          (let ((f1 (square x)) (f2 (square (- x 2))))
            (write-line (concatenate 'string "(! " (write-to-string x) " "
                                     (write-to-string f1) " "
                                     (write-to-string f2) ")") outfile)))))
    (close outfile)))

(defun fonseca (&optional (samples 10000))
  (let ((outfile (open "../data/tests/fonseca.lisp" :direction :output :if-exists :supersede)))
    (write-line "(deftable $fonseca $x0 $x1 $x2 $#f1 $#f2)" outfile)
    (dotimes (number samples)
      (let ((x '()))
        (dotimes (index-x 3)
          (push (- (randf 8) 4) x))
        (let ((catstr "(! "))
          (dolist (xi x)
            (setf catstr (concatenate 'string catstr (write-to-string xi) " ")))
          (setf catstr (concatenate 'string catstr (write-to-string (fonseca-f1 x)) " "))
          (setf catstr (concatenate 'string catstr (write-to-string (fonseca-f2 x)) ")"))
          (Write-line catstr outfile))))
    (close outfile)))

(defun fonseca-search (constraints &optional (samples 1000))
  (let ((outfile (open "../data/tests/fonseca.lisp" :direction :output :if-exists :supersede)))
    (write-line "(deftable $fonseca $x0 $x1 $x2 $#f1 $#f2)" outfile)
    (dotimes (number samples)
      (let ((x '()))
        (dolist (this constraints)
          (let ((that (any this)))
            (push (+ (randf (- (second that) (first that))) (first that)) x)))
        (let ((catstr "(! "))
          (dolist (xi x)
            (setf catstr (concatenate 'string catstr (write-to-string xi) " ")))
          (setf catstr (concatenate 'string catstr (write-to-string (fonseca-f1 x)) " "))
          (setf catstr (concatenate 'string catstr (write-to-string (fonseca-f2 x)) ")"))
          (Write-line catstr outfile))))
    (close outfile)))

;(defun fonseca-f1 (x)
;  (let ((sum 0) (n (length x)))
;    (- 1 (exp (- 0 (dolist (xi x sum)
;      (incf sum (square (- xi (/ 1 (sqrt n)))))))))))

(defun fonseca-f1 (x)
  (let ((sum 0) (n (length x)))
    (dolist (xi x)
      (incf sum (square (- xi (/ 1 (sqrt n))))))
    (- 1 (exp (* -1 sum)))))

(defun fonseca-f2 (x)
  (let ((sum 0) (n (length x)))
    (- 1 (exp (- 0 (dolist (xi x sum)
      (incf sum (square (+ xi (/ 1 (sqrt n))))))))))))

(defun kursawe (&optional (samples 10000))
  (let ((outfile (open "../data/tests/kursawe.lisp" :direction :output :if-exists :supersede)))
    (write-line "(deftable $kurwasa $x0 $x1 $x2 $#f1 $#f2)" outfile)
    (dotimes (number samples)
      (let ((x '()))
        (dotimes (index-x 3)
          (push (- (randf 10) 5) x))
```

www.manaraa.com

```
        (let ((catstr "(!␣"))
          (dolist (xi x)
            (setf catstr (concatenate 'string catstr (write-to-string xi) "␣")))
          (setf catstr (concatenate 'string catstr (write-to-string (kursawe-f1 x)) "␣"))
          (setf catstr (concatenate 'string catstr (write-to-string (kursawe-f2 x)) ")"))
          (write-line catstr outfile))))
    (close outfile)))

(defun kursawe-search (constraints &optional (samples 1000))
  (let ((outfile (open "../data/tests/kursawe.lisp" :direction :output :if-exists :supersede)))
    (write-line "(deftable␣$kursawe␣$x0␣$x1␣$x2␣$#f1␣$#f2)" outfile)
    (dotimes (number samples)
      (let ((x '()))
        (dolist (this constraints)
          (let ((that (any this)))
            (push (+ (randf (- (second that) (first that))) (first that)) x)))
        (let ((catstr "(!␣"))
          (dolist (xi x)
            (setf catstr (concatenate 'string catstr (write-to-string xi) "␣")))
          (setf catstr (concatenate 'string catstr (write-to-string (kursawe-f1 x)) "␣"))
          (setf catstr (concatenate 'string catstr (write-to-string (kursawe-f2 x)) ")"))
          (Write-line catstr outfile))))
    (close outfile)))

(defun kursawe-f1 (x)
  (+ (* -10.0 (exp (* -.2 (sqrt (+ (square (first x)) (square (second x)))))))
     (* -10.0 (exp (* -.2 (sqrt (+ (square (second x)) (square (third x)))))))))

(defun kursawe-f2 (x) ;a = .8, b = 3
  (let ((sum 0) (n (length x)) (a .8) (b 3.0))
    (dolist (xi x sum)
      (incf sum (+ (expt (abs xi) a) (* 5.0 (sin (expt xi b))))))))

(defun zdt1 (&optional (samples 10000))
  (let ((outfile (open "../data/tests/zdt1.lisp" :direction :output :if-exists :supersede)))
    (let ((catstr "(deftable␣$zdt1"))
      (dotimes (index-x 30)
        (setf catstr (concatenate 'string catstr "␣$x" (write-to-string index-x))))
      (setf catstr (concatenate 'string catstr "␣$#f1␣$#f2)"))
      (write-line catstr outfile))
    (dotimes (number samples)
      (let ((x '()))
        (dotimes (index-x 30)
          (push (randf 1.0) x))
        (let ((catstr "(!␣"))
          (dolist (xi x)
            (setf catstr (concatenate 'string catstr (write-to-string xi) "␣")))
          (setf catstr (concatenate 'string catstr (write-to-string (zdt1-f1 x)) "␣"))
          (setf catstr (concatenate 'string catstr (write-to-string (zdt1-f2 x)) ")"))
          (write-line catstr outfile))))
    (close outfile)))



(defun zdt1-f1 (x)
  (let ((sum 0) (n (length x)))
    (first x)))


(defun zdt1-f2 (x)
  (let ((sum 0) (n (length x)))
    (* (zdt1-g x) (- 1 (sqrt (/ (first x) (zdt1-g x)))))))
```

www.manaraa.com

```
(defun zdt1-g (x)
  (let ((sum 0) (n (length x)))
    (+ 1 (* 9.0 (/ (dolist (xi (rest x) sum)
                     (incf sum xi))
                   (- n 1))))))

(defun zdt2 (&optional (samples 10000))
  (let ((outfile (open "../data/tests/zdt2.lisp" :direction :output :if-exists :supersede)))
    (let ((catstr "(deftable $zdt2"))
      (dotimes (index-x 30)
        (setf catstr (concatenate 'string catstr " $x" (write-to-string index-x))))
      (setf catstr (concatenate 'string catstr " $#f1 $#f2)"))
      (write-line catstr outfile))
    (dotimes (number samples)
      (let ((x '()))
        (dotimes (index-x 30)
          (push (randf 1.0) x))
        (let ((catstr "(! "))
          (dolist (xi x)
            (setf catstr (concatenate 'string catstr (write-to-string xi) " ")))
          (setf catstr (concatenate 'string catstr (write-to-string (zdt2-f1 x)) " "))
          (setf catstr (concatenate 'string catstr (write-to-string (zdt2-f2 x)) ")"))
          (write-line catstr outfile))))
    (close outfile)))

(defun zdt2-f1 (x)
  (let ((sum 0) (n (length x)))
    (first x)))

(defun zdt2-f2 (x)
  (let ((sum 0) (n (length x)))
    (* (zdt2-g x) (- 1 (square (/ (first x) (zdt2-g x)))))))

(defun zdt2-g (x)
  (let ((sum 0) (n (length x)))
    (+ 1 (* 9.0 (/ (dolist (xi (rest x) sum)
                     (incf sum xi))
                   (- n 1))))))

(defun zdt3 (&optional (samples 10000))
  (let ((outfile (open "../data/tests/zdt3.lisp" :direction :output :if-exists :supersede)))
    (let ((catstr "(deftable $zdt3"))
      (dotimes (index-x 30)
        (setf catstr (concatenate 'string catstr " $x" (write-to-string index-x))))
      (setf catstr (concatenate 'string catstr " $#f1 $#f2)"))
      (write-line catstr outfile))
    (dotimes (number samples)
      (let ((x '()))
        (dotimes (index-x 30)
          (push (randf 1.0) x))
        (let ((catstr "(! "))
          (dolist (xi x)
            (setf catstr (concatenate 'string catstr (write-to-string xi) " ")))
          (setf catstr (concatenate 'string catstr (write-to-string (zdt3-f1 x)) " "))
          (setf catstr (concatenate 'string catstr (write-to-string (zdt3-f2 x)) ")"))
          (write-line catstr outfile))))
    (close outfile)))

(defun zdt3-f1 (x)
  (let ((sum 0) (n (length x)))
    (first x)))
```

المنارة للاستشارات
www.manaraa.com

```
(defun zdt3-f2 (x)
  (let ((sum 0) (n (length x)))
    (*  (zdt3-g x) (- 1 (sqrt (/ (first x) (zdt3-g x)))
                    (* (/ (first x) (zdt3-g x)) (sin (* 10.0 pi (first x))))))))

(defun zdt3-g (x)
  (let ((sum 0) (n (length x)))
    (+ 1 (* 9.0 (/ (dolist (xi (rest x) sum)
                     (incf sum xi))
                   (- n 1))))))

(defun zdt4 (&optional (samples 10000))
  (let ((outfile (open "../data/tests/zdt4.lisp" :direction :output :if-exists :supersede)))
    (let ((catstr "(deftable $zdt4"))
      (dotimes (index-x 10)
        (setf catstr (concatenate 'string catstr " $x" (write-to-string index-x))))
      (setf catstr (concatenate 'string catstr " $#f1 $#f2)"))
      (write-line catstr outfile))
    (dotimes (number samples)
      (let ((x '()))
        (push (randf 1.0) x)
        (dotimes (index-x 9)
          (push (- (randf 10) 5) x))
        (let ((catstr "(! "))
          (dolist (xi x)
            (setf catstr (concatenate 'string catstr (write-to-string xi) " ")))
          (setf catstr (concatenate 'string catstr (write-to-string (zdt4-f1 x)) " "))
          (setf catstr (concatenate 'string catstr (write-to-string (zdt4-f2 x)) ")"))
          (write-line catstr outfile))))
    (close outfile)))

(defun zdt4-f1 (x)
  (let ((sum 0) (n (length x)))
    (first x)))

(defun zdt4-f2 (x)
  (let ((sum 0) (n (length x)))
    (* (zdt4-g x) (- 1 (square (/ (first x) (zdt4-g x)))))))

(defun zdt4-g (x)
  (let ((sum 0) (n (length x)))
    (+ 1 (* 10 (- n 1)) (dolist (xi (rest x) sum)
                          (incf sum (- (square xi) (* 10 (cos (* 4 pi xi)))))))))



(defun zdt6 (&optional (samples 10000))
  (let ((outfile (open "../data/tests/zdt6.lisp" :direction :output :if-exists :supersede)))
    (let ((catstr "(deftable $zdt6"))
      (dotimes (index-x 10)
        (setf catstr (concatenate 'string catstr " $x" (write-to-string index-x))))
      (setf catstr (concatenate 'string catstr " $#f1 $#f2)"))
      (write-line catstr outfile))
    (dotimes (number samples)
      (let ((x '()))
        (dotimes (index-x 10)
          (push (randf 1.0) x))
        (let ((catstr "(! "))
          (dolist (xi x)
            (setf catstr (concatenate 'string catstr (write-to-string xi) " ")))
          (setf catstr (concatenate 'string catstr (write-to-string (zdt6-f1 x)) " "))
          (setf catstr (concatenate 'string catstr (write-to-string (zdt6-f2 x)) ")"))
```

www.manaraa.com

```
        (write-line catstr outfile))))
    (close outfile)))

(defun zdt6-f1 (x)
  (let ((sum 0) (n (length x)))
    (- 1 (* (exp (* -4 (first x))) (expt (sin (* 6 pi (first x))) 6)))))

(defun zdt6-f2 (x)
  (let ((sum 0) (n (length x)))
    (* (zdt6-g x) (- 1 (square (/ (zdt6-f1 x) (zdt6-g x)))))))

(defun zdt6-g (x)
  (let ((sum 0) (n (length x)))
    (+ 1 (* 9.0 (expt (/ (dolist (xi (rest x) sum)
                   (incf sum xi))
                 (- n 1)) .25)))))


(defun viennet2 (&optional (samples 10000))
  (let ((outfile (open "../data/tests/viennet2.lisp" :direction :output :if-exists :supersede)))
    (write-line "(deftable $viennet2 $x1 $x2 $#f1 $#f2 $#f3)" outfile)
    (dotimes (number samples)
      (let ((x1 (- (randf 8.0) 4)) (x2 (- (randf 8.0) 4)))
        (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                 (write-to-string x2) " "
                                 (write-to-string (viennet2-f1 x1 x2)) " "
                                 (write-to-string (viennet2-f2 x1 x2)) " "
                                 (write-to-string (viennet2-f3 x1 x2)) ")")
                    outfile)))
    (close outfile)))

(defun viennet2-f1 (x1 x2)
  (+ 3.0 (/ (square (- x1 2)) 2) (/ (square (+ x1 1)) 13)))

(defun viennet2-f2 (x1 x2)
  (+ -17.0 (/ (square (+ x1 x2 -3)) 36) (/ (square (- x2 x1 -2)) 8)))

(defun viennet2-f3 (x1 x2)
  (+ -13.0 (/ (square (+ x1 (* 2 x2) -1)) 175) (/ (square (+ x1 (* 2 x2))) 17)))


(defun viennet3 (&optional (samples 10000))
  (let ((outfile (open "../data/tests/viennet3.lisp" :direction :output :if-exists :supersede)))
    (write-line "(deftable $viennet3 $x1 $x2 $#f1 $#f2 $#f3)" outfile)
    (dotimes (number samples)
      (let ((x1 (- (randf 6.0) 3)) (x2 (- (randf 6.0) 3)))
        (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                 (write-to-string x2) " "
                                 (write-to-string (viennet3-f1 x1 x2)) " "
                                 (write-to-string (viennet3-f2 x1 x2)) " "
                                 (write-to-string (viennet3-f3 x1 x2)) ")")
                    outfile)))
    (close outfile)))

(defun viennet3-f1 (x1 x2)
  (+ (* .5 (square x1)) (square x2) (sin (+ (square x1) (square x2)))))

(defun viennet3-f2 (x1 x2)
  (+ 15.0 (/ (square (+ (* 3 x1) (* -2 x2) 4)) 8) (/ (square (+ x1 (* -1 x2) 1)) 27)))

(defun viennet3-f3 (x1 x2)
  (- (/ 1 (+ (square x1) (square x2) 1)) (* 1.1 (exp (* -1 (+ (square x1) (square x2)))))))
```

www.manaraa.com

```
;BEGIN CONSTRAINTS

(defun constr-ex (&optional (samples 10000))
  (let ((outfile (open "../data/tests/constrex.lisp" :direction :output :if-exists :supersede))
        (rowcount 0))
    (write-line "(deftable $constrex $x1 $x2 $#f1 $#f2)" outfile)
    (loop while (< rowcount samples) do
         (let ((x1 (+ (randf .9) .1)) (x2 (randf 5.0)))
           (if (and (>= (+ x2 (* 9 x1)) 6)
                    (>= (+ (* -1 x2) (* 9 x1)) 1))
               (progn
                 (incf rowcount)
                 (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                          (write-to-string x2) " "
                                          (write-to-string (constr-ex-f1 x1 x2)) " "
                                          (write-to-string (constr-ex-f2 x1 x2)) ")")
                             outfile)))))
    (close outfile)))

(defun constr-ex-f1 (x1 x2)
  x1)

(defun constr-ex-f2 (x1 x2)
  (/ (+ 1 x2) x1))

(defun constrex-search (constraints &optional (samples 1000))
  (let ((outfile (open "../data/tests/constrex.lisp" :direction :output :if-exists :supersede))
        (rowcount 0) (attempts 0) (maxsamples (* 10 samples)))
    (write-line "(deftable $constrex $x1 $x2 $#f1 $#f2)" outfile)
    (loop while (and (< rowcount samples) (< attempts maxsamples)) do
         (let ((thatx1 (any (first constraints)))
               (thatx2 (any (second constraints))))
           (let ((x1 (+ (randf (- (second thatx1) (first thatx1))) (first thatx1)))
                 (x2 (+ (randf (- (second thatx2) (first thatx2))) (first thatx2))))
             (incf attempts)
             (if (and (>= (+ x2 (* 9 x1)) 6)
                      (>= (- (* 9 x1) x2) 1))
                 (progn
                   (incf rowcount)
                   (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                            (write-to-string x2) " "
                                            (write-to-string (constr-ex-f1 x1 x2)) " "
                                            (write-to-string (constr-ex-f2 x1 x2)) ")")
                               outfile))))))
    (close outfile)))

(defun tanaka (&optional (samples 10000))
  (let ((outfile (open "../data/tests/tanaka.lisp" :direction :output :if-exists :supersede))
        (rowcount 0))
    (write-line "(deftable $tanaka $x1 $x2 $#f1 $#f2)" outfile)
    (loop while (< rowcount samples) do
         (let ((x1 (- (randf (* 2 pi)) pi)) (x2 (- (randf (* 2 pi)) pi)))
           (if (and (<= (- 1.0 (square x1) (square x2) (* -0.1 (cos (* 16 (atan (/ x1 x2)))))) 0)
                    (<= (+ (square (- x1 .5)) (square (- x2 .5))) .5))
               (progn
                 (incf rowcount)
                 (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                          (write-to-string x2) " "
                                          (write-to-string (tanaka-f1 x1 x2)) " "
                                          (write-to-string (tanaka-f2 x1 x2)) ")")
                             outfile)))))
    (close outfile)))
```

المنارة للاستشارات
www.manaraa.com

```
(defun tanaka-search (constraints &optional (samples 1000))
  (let ((outfile (open "../data/tests/tanaka.lisp" :direction :output :if-exists :supersede))
        (rowcount 0) (attempts 0) (maxsamples (* 10 samples)))
    (write-line "(deftable $tanaka $x1 $x2 $#f1 $#f2)" outfile)
    (loop while (and (< rowcount samples) (< attempts maxsamples)) do
         (let ((thatx1 (any (first constraints)))
               (thatx2 (any (second constraints))))
           (let ((x1 (+ (randf (- (second thatx1) (first thatx1))) (first thatx1)))
                 (x2 (+ (randf (- (second thatx2) (first thatx2))) (first thatx2))))
             (incf attempts)
             (if (and (<= (- 1.0 (square x1) (square x2) (* -0.1 (cos (* 16 (atan (/ x1 x2))))))
                      0)
                      (<= (+ (square (- x1 .5)) (square (- x2 .5))) .5))
                 (progn
                   (incf rowcount)
                   (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                            (write-to-string x2) " "
                                            (write-to-string (tanaka-f1 x1 x2)) " "
                                            (write-to-string (tanaka-f2 x1 x2)) ")")
                               outfile))))))
    (close outfile)))



(defun tanaka-f1 (x1 x2)
  x1)

(defun tanaka-f2 (x1 x2)
  x2)

(defun osyczka2 (&optional (samples 10000))
  (let ((outfile (open "../data/tests/osyczka2.lisp" :direction :output :if-exists :supersede))
        (rowcount 0))
    (write-line "(deftable $osyczka2 $x1 $x2 $x3 $x4 $x5 $x6 $#f1 $#f2)" outfile)
    (loop while (< rowcount samples) do
         (let ((x1 (randf 10.0)) (x2 (randf 10.0))
               (x3 (+ (randf 4.0) 1.0)) (x4 (randf 6.0))
               (x5 (+ (randf 4.0) 1.0)) (x6 (randf 10.0)))
           (if (and (<= 0 (+ x1 x2 -2.0))
                    (<= 0 (- 6.0 x1 x2))
                    (<= 0 (- x1 x2 -2.0))
                    (<= 0 (+ 2 (* -1.0 x1) (* 3.0 x2)))
                    (<= 0 (- 4 (square (- x3 3)) x4))
                    (<= 0 (+ (square (- x5 3)) x6 -4.0)))
               (progn
                 (incf rowcount)
                 (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                          (write-to-string x2) " "
                                          (write-to-string x3) " "
                                          (write-to-string x4) " "
                                          (write-to-string x5) " "
                                          (write-to-string x6) " "
                                          (write-to-string (osyczka2-f1 x1 x2 x3 x4 x5 x6)) " "
                                          (write-to-string (osyczka2-f2 x1 x2 x3 x4 x5 x6)) ")")
                             outfile)))))
    (close outfile)))

(defun osyczka2-f1 (x1 x2 x3 x4 x5 x6)
  (+ (square (- x2 2))
     (* -25.0 (square (- x1 2)))
     (* (square (- x3 1)) (square (- x4 4)))
     (square (- x5 1))))
```

المنارة للاستشارات
www.manaraa.com

```
(defun osyczka2-f2 (x1 x2 x3 x4 x5 x6)
  (+ (square x1) (square x2) (square x3) (square x4) (square x5) (square x6)))


(defun srinivas (&optional (samples 10000))
  (let ((outfile (open "../data/tests/srinivas.lisp" :direction :output :if-exists :supersede))
        (rowcount 0))
    (write-line "(deftable $srinivas $x1 $x2 $#f1 $#f2)" outfile)
    (loop while (< rowcount samples) do
         (let ((x1 (- (randf 40.0) 20))
               (x2 (- (randf 40.0) 20)))
           (if (and (<= (+ (square x1) (square x2)) 225)
                    (<= (- x1 (* 3 x2)) -10))
               (progn
                 (incf rowcount)
                 (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                          (write-to-string x2) " "
                                          (write-to-string (srinivas-f1 x1 x2)) " "
                                          (write-to-string (srinivas-f2 x1 x2)) ")")
                             outfile)))))
    (close outfile)))

(defun srinivas-search (constraints &optional (samples 1000))
  (let ((outfile (open "../data/tests/srinivas.lisp" :direction :output :if-exists :supersede))
        (rowcount 0) (attempts 0) (maxsamples (* 10 samples)))
    (write-line "(deftable $srinivas $x1 $x2 $#f1 $#f2)" outfile)
    (loop while (and (< rowcount samples) (< attempts maxsamples))  do
         (let ((thatx1 (any (first constraints)))
               (thatx2 (any (second constraints))))
           (let ((x1 (+ (randf (- (second thatx1) (first thatx1))) (first thatx1)))
                 (x2 (+ (randf (- (second thatx2) (first thatx2))) (first thatx2))))
             (incf attempts)
             (if (and (<= (+ (square x1) (square x2)) 225)
                      (<= (- x1 (* 3 x2)) -10))
                 (progn
                   (incf rowcount)
                   (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                            (write-to-string x2) " "
                                            (write-to-string (srinivas-f1 x1 x2)) " "
                                            (write-to-string (srinivas-f2 x1 x2)) ")")
                               outfile))))))
    (close outfile)))


(defun srinivas-f1 (x1 x2)
  (+ (square (- x1 2)) (square (- x2 1)) 2))

(defun srinivas-f2 (x1 x2)
  (- (* 9 x1 ) (square (- x2 1))))

(defun golinski (&optional (samples 10000))
  (let ((outfile (open "../data/tests/golinski.lisp" :direction :output :if-exists :supersede))
        (rowcount 0))
    (write-line "(deftable $golinski $x1 $x2 $x3 $x4 $x5 $x6 $x7 $#f1 $#f2)" outfile)
    (loop while (< rowcount samples) do
         (let ((x1 (+ (randf 1.0) 2.6))
               (x2 (+ (randf 0.1) 0.7))
               (x3 (+ (randf 11.0) 17.0))
               (x4 (+ (randf 1.0) 7.3))
               (x5 (+ (randf 1.0) 7.3))
               (x6 (+ (randf 1.0) 2.9))
               (x7 (+ (randf 0.5) 5.5)))
```

المنارة للاستشارات
www.manaraa.com

```
          (if (and (<= (- (/ 1.0 (* x1 (square x2) x3)) (/ 1.0 27.0)) 0)
                   (<= (- (/ 1.0 (* x1 (square x2) (square x3))) (/ 1.0 397.5)) 0)
                   (<= (- (/ (* x4 x4 x4) (* x2 x3 x3 x6 x6 x6 x6)) (/ 1.0 1.93)) 0)
                   (<= (- (/ (* x5 x5 x5) (* x2 x3 x7 x7 x7 x7)) (/ 1.0 1.93)) 0)
                   (<= (- (* x2 x3) 40.0) 0)
                   (<= (- (/ x1 x2) 12.0) 0)
                   (<= (- 5.0 (/ x1 x2)) 0)
                   (<= (+ 1.9 (* -1.0 x4) (* 1.5 x6)) 0)
                   (<= (+ 1.9 (* -1.0 x5) (* 1.1 x7)) 0)
                   (<= (golinski-f2 x1 x2 x3 x4 x5 x6 x6) 1300.0)
                   (<= (/ (sqrt (+ (square (golinski-a x1 x2 x3 x4 x5 x6 x7))
                                   (golinski-b x1 x2 x3 x4 x5 x6 x7)))
                          (* 0.1 x7 x7 x7)) 1100.0))
              (progn
                (incf rowcount)
                (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                         (write-to-string x2) " "
                                         (write-to-string x3) " "
                                         (write-to-string x4) " "
                                         (write-to-string x5) " "
                                         (write-to-string x6) " "
                                         (write-to-string x7) " "
                                         (write-to-string (golinski-f1 x1 x2 x3 x4 x5 x6 x7)) " "
                                         (write-to-string (golinski-f2 x1 x2 x3 x4 x5 x6 x7)) ")"
                                             )
                            outfile)))))
    (close outfile)))

(defun golinski-f1 (x1 x2 x3 x4 x5 x6 x7)
  (+ (* .7854 x1 (square x2) (+ (* 10 (square x3) (/ 1 3)) (* 14.933 x3) -49.0934))
     (* -1.508 x1 (+ (square x6) (square x7)))
     (* 7.477 (+ (expt x6 3) (expt x7 3)))
     (* 0.7854 (+ (* x4 x6 x6) (* x5 x7 x7)))))

(defun golinski-f2 (x1 x2 x3 x4 x5 x6 x7)
  (/  (sqrt (+ (square (golinski-a2 x1 x2 x3 x4 x5 x6 x7))
               (* 1.69 (expt 10.0 7))))
      (* 0.1 x6 x6 x6)))

(defun golinski-a (x1 x2 x3 x4 x5 x6 x7)
  (/ (* 745.0 x5) (* x2 x3)))

(defun golinski-a2 (x1 x2 x3 x4 x5 x6 x7)
  (/ (* 745.0 x4) (* x2 x3)))

(defun golinski-b (x1 x2 x3 x4 x5 x6 x7)
  (* 1.575 (expt 10.0 8)))

(defun viennet4 (&optional (samples 10000))
  (let ((outfile (open "../data/tests/viennet4.lisp" :direction :output :if-exists :supersede))
        (rowcount 0))
    (write-line "(deftable $viennet4 $x1 $x2 $#f1 $#f2 $#f3)" outfile)
    (loop while (< rowcount samples) do
         (let ((x1 (- (randf 8.0) 4))
               (x2 (- (randf 8.0) 4)))
          (if (and (>= (- 4 x2 (* 4 x1)) 0)
                   (>= (+ x1 1) 0)
                   (>= (+ x2 (* -1.0 x1) 2) 0))
              (progn
                (incf rowcount)
                (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                         (write-to-string x2) " "
                                         (write-to-string (viennet4-f1 x1 x2)) " "
```

المنارة للاستشارات
www.manaraa.com

```lisp
                                        (write-to-string (viennet4-f2 x1 x2)) " "
                                        (write-to-string (viennet4-f3 x1 x2)) ")")
                                outfile)))))
    (close outfile)))

(defun viennet4-f1 (x1 x2)
  (+ (/ (square (- x1 2)) 2) (/ (square (+ x2 1)) 13) 3))

(defun viennet4-f2 (x1 x2)
  (+ (/ (square (+ x1 x2 -3)) 175)
     (/ (square (- (* 2 x2) x1)) 17)
     -13))

(defun viennet4-f3 (x1 x2)
  (+ (/ (square (+ (* 3 x2) (* -2 x2) 4)) 8)
     (/ (square (+ x1 (* -1.0 x2) 1)) 27)
     15))

(defun water (&optional (samples 10000))
  (let ((outfile (open "../data/tests/water.lisp" :direction :output :if-exists :supersede))
        (rowcount 0))
    (write-line "(deftable $water $x1 $x2 $x3 $#f1 $#f2 $#f3 $#f4 $#f5)" outfile)
    (loop while (< rowcount samples) do
         (let ((x1 (+ (randf .44) .01))
               (x2 (+ (randf .09) .01))
               (x3 (+ (randf .09) .01)))
           (if (and (>= (+ 1 (/ -.00139 (* x1 x2)) (* 4.94 x3) -.08) 0)
                    (>= (+ 1 (/ -.000306 (* x1 x2)) (* 1.082 x3) -.0986) 0)
                    (>= (+ 5000 (/ -12.307 (* x1 x2)) (* 4.9408 x3) 4051.02) 0)
                    (>= (+ 16000 (/ -2.09 (* x1 x2)) (* 8046.33 x3) -696.71) 0)
                    (>= (+ 10000 (/ -2.138 (* x1 x2)) (* 7883.39 x3) -705.04) 0)
                    (>= (+ 2000 (/ -.417 (* x1 x2)) (* 1721.26 x3) -136.54) 0)
                    (>= (+ 550 (/ -.164 (* x1 x2)) (* 631.13 x3) -54.48) 0))
               (progn
                 (incf rowcount)
                 (write-line (concatenate 'string "(! " (write-to-string x1) " "
                                          (write-to-string x2) " "
                                          (write-to-string x3) " "
                                          (write-to-string (water-f1 x1 x2 x3)) " "
                                          (write-to-string (water-f2 x1 x2 x3)) " "
                                          (write-to-string (water-f3 x1 x2 x3)) " "
                                          (write-to-string (water-f4 x1 x2 x3)) " "
                                          (write-to-string (water-f5 x1 x2 x3)) ")")
                             outfile)))))
    (close outfile)))

(defun water-f1 (x1 x2 x3)
  (+ (* 106780.37 (+ x2 x3)) 61704.67))

(defun water-f2 (x1 x2 x3)
  (* 3000 x1))

(defun water-f3 (x1 x2 x3)
  (/ (* 305700 2289 x2) (expt (* .06 2289) .65)))

(defun water-f4 (x1 x2 x3)
  (* 250 2289 x2 (exp (+ (* -39.75 x2) (* 9.9 x3) 2.74))))

(defun water-f5 (x1 x2 x3)
  (/ (* 25 1.39) (+ (* x1 x2) (* 4940 x3) -80.0)))

(defun build-all-test-datasets ()
  (reset-seed)
```

المنارة للاستشارات
www.manaraa.com

```
  (schaffer)
  (fonseca)
  (kursawe)
  (zdt1)
  (zdt2)
  (zdt3)
  (zdt4)
  (zdt6)
  (osyczka2)
  (tanaka)
  (constr-ex)
  (srinivas)
  (golinski)
  (viennet2)
  (viennet3)
  (viennet4)
  (water)
)

(defun learn-all-test-datasets ()
  (!learn1 "schaffer")
  (!learn1 "fonseca")
  (!learn1 "kursawe")
  (!learn1 "zdt1")
  (!learn1 "zdt2")
  (!learn1 "zdt3")
  (!learn1 "zdt4")
  (!learn1 "zdt6")
  (!learn1 "osyczka2")
  (!learn1 "tanaka")
  (!learn1 "constr-ex")
  (!learn1 "srinivas")
  (!learn1 "viennet2")
  (!learn1 "viennet3")
  (!learn1 "viennet4")
  (!learn1 "water")
)
```

A.2.6 how.lisp

Some portions written by Dr. Menzies

```
;This implements the How loop (in the code still referred to as which2,
;since which2 was the original name for this extension)
;
;
(defparameter *every* 'allqzjx)


(defun which2 (&optional (tbl (thetable)) (s 2))
  (train tbl)
  (learn tbl s)
  )

(defun which2n (&optional (tbl (thetable)))
  (learnn tbl))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 (defun train (tbl)
   (dolist (row (therows tbl))
     (how-manys (thecols tbl)     ; get the column headers
                (row-cells row)   ; get the cells
                (row-class row)   ; get the class of this row
```

المنارة للاستشارات
www.manaraa.com

```
                )))


(defun how-manys (cols cells class)
  (labels ((worker (col cell)
             (how-many class
                       (col-name col)
                       cell
                       (sym-counts col))))
    (mapcar #'worker cols cells))) ; run down cols and cells in parallel

(defun how-many (class what cell hash)
  (when (knownp cell) ; skip any cell labelled "?"
    (inch `(,class   ,what ,cell) hash)
    (inch `(,*every* ,what ,cell) hash)))

(defun inch (key hash)
  "increment␣a␣hash␣bucket␣from␣zero"
  (incf (gethash key hash 0)))

(defun !how-manys1 ()
  (reset-seed)
  (data "../data/discrete-lisp/weather.lisp")
  (train (thetable))
  (with-output-to-string (s)
    (dolist (col (thecols))
      (showh (sym-counts col) :stream s))))

(deftest !how-manys ()
  (test (!how-manys1)
        "(ALLQZJX␣FORECAST␣OVERCAST)␣=␣4
␣␣␣␣␣␣␣␣(ALLQZJX␣FORECAST␣RAINY)␣=␣5
␣␣␣␣␣␣␣␣(ALLQZJX␣FORECAST␣SUNNY)␣=␣5
␣␣␣␣␣␣␣␣(NO␣FORECAST␣RAINY)␣=␣2
␣␣␣␣␣␣␣␣(NO␣FORECAST␣SUNNY)␣=␣3
␣␣␣␣␣␣␣␣(YES␣FORECAST␣OVERCAST)␣=␣4
␣␣␣␣␣␣␣␣(YES␣FORECAST␣RAINY)␣=␣3
␣␣␣␣␣␣␣␣(YES␣FORECAST␣SUNNY)␣=␣2
␣␣␣␣␣␣␣␣(ALLQZJX␣TEMP␣COOL)␣=␣4
␣␣␣␣␣␣␣␣(ALLQZJX␣TEMP␣HOT)␣=␣4
␣␣␣␣␣␣␣␣(ALLQZJX␣TEMP␣MILD)␣=␣6
␣␣␣␣␣␣␣␣(NO␣TEMP␣COOL)␣=␣1
␣␣␣␣␣␣␣␣(NO␣TEMP␣HOT)␣=␣2
␣␣␣␣␣␣␣␣(NO␣TEMP␣MILD)␣=␣2
␣␣␣␣␣␣␣␣(YES␣TEMP␣COOL)␣=␣3
␣␣␣␣␣␣␣␣(YES␣TEMP␣HOT)␣=␣2
␣␣␣␣␣␣␣␣(YES␣TEMP␣MILD)␣=␣4
␣␣␣␣␣␣␣␣(ALLQZJX␣HUMIDTY␣HIGH)␣=␣7
␣␣␣␣␣␣␣␣(ALLQZJX␣HUMIDTY␣NORMAL)␣=␣7
␣␣␣␣␣␣␣␣(NO␣HUMIDTY␣HIGH)␣=␣4
␣␣␣␣␣␣␣␣(NO␣HUMIDTY␣NORMAL)␣=␣1
␣␣␣␣␣␣␣␣(YES␣HUMIDTY␣HIGH)␣=␣3
␣␣␣␣␣␣␣␣(YES␣HUMIDTY␣NORMAL)␣=␣6
␣␣␣␣␣␣␣␣(ALLQZJX␣WINDY␣FALSE)␣=␣8
␣␣␣␣␣␣␣␣(ALLQZJX␣WINDY␣TRUE)␣=␣6
␣␣␣␣␣␣␣␣(NO␣WINDY␣FALSE)␣=␣2
␣␣␣␣␣␣␣␣(NO␣WINDY␣TRUE)␣=␣3
␣␣␣␣␣␣␣␣(YES␣WINDY␣FALSE)␣=␣6
␣␣␣␣␣␣␣␣(YES␣WINDY␣TRUE)␣=␣3
␣␣␣␣␣␣␣␣(ALLQZJX␣!PLAY␣NO)␣=␣5
␣␣␣␣␣␣␣␣(ALLQZJX␣!PLAY␣YES)␣=␣9
␣␣␣␣␣␣␣␣(NO␣!PLAY␣NO)␣=␣5
␣␣␣␣␣␣␣␣(YES␣!PLAY␣YES)␣=␣9"))
```

www.manaraa.com

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun learn (tbl s)
  (let ((out '()))
    (dolist (target (theklasses tbl) out)
      (push (learn1  target tbl s) out))))

(defun learnn (tbl)
  (roundsn (round0n tbl) tbl))

(defun learn1 (target tbl s)
  (let ((which (round0 target tbl)))
    (rounds target which tbl s)))

(defun round0 (target tbl)
  (let ((out '()))
    (dolist (col (thecols tbl))
      (unless (col-goalp col)
        (dokeys (key (sym-counts col))
          (push (list 0 (second key) (third key)) out))))
    (remove-duplicates out :test #'equal)))

(defun round0n (tbl)
  (let ((out '()))
    (dolist (row (therows tbl))
      (let ((countr 0))
        (dolist (this (row-cells row))
          (unless (col-goalp (nth countr (table-cols tbl)))
            (push (list 0 (col-name (nth countr (table-cols tbl))) this) out))
            (incf countr))))
    (remove-duplicates out :test #'equal)))

(defun rule-match (row rule table)
  (let ((colnums '()) (countr 0) (out t))
    (dolist (this (table-cols table))
      (push (list (col-name this) countr) colnums)
      (incf countr))
    (dolist (thisors (rule-ands rule) out)
      (unless (member (nth (getcolnum (ors-for thisors) colnums) (row-cells row)) (ors-values
             thisors))
        (setf out nil)))))

;(defun round0 (target tbl)
;  (let (out
;       (n (length (therows tbl))))
;    (labels
;       ((worker (hash want m what class value &aux s)
;          (if (eql class want)
;            (if (setf s (b^2/b+r hash want m n what value))
;                (push (list (round s 0.01)
;                            what value)
;                      out)))))
;      (dolist (col (thecols tbl))          ; for every column
;        (unless (col-goalp col)            ; that's not the goal
;          (dokeys (key  (sym-counts col))
;            (print key)                   ; for everything counted in that col
;            (worker (sym-counts col)       ; get the hash table counds
;                    (klass-name target)    ; what class are we targetting?
;                    (klass-n target)       ; how many of them do we have?
;                    (col-name col)         ; what is the col name?
;                    (first key)            ; what class is being counted?
;                    (third key)            ; what value we looking at?
```

المنارة للاستشارات
www.manaraa.com

```
;                    ))))
;       (sort out #'> :key #'first))))

(defun b^2/b+r (hash want m n what value)
  (let* ((every (gethash `(,*every* ,what ,value) hash 0))
         (b0    (gethash `(,want     ,what, value) hash 0))
         (r0    (- every b0))
         (b     (/ b0 m))          ; ratio in target
         (r     (/ r0 (- n m)))) ; ration everwhere else
    (if (> b r)                   ; in more better than rester
        (/ (* b b)                ; b^2/(b+r)
           (+ b r (randf 0.0000001))))))

(defun roundsn (which tbl)
  (let ((lives 5) (rules '()) (round 1) (max 0) (zero-score t))
    (dolist (this which)
      (let ((that (to-rule 'blandqxzy this)))
        (setf (rule-score that) (scoren that tbl))
        (unless (> 4 (rule-support that))
          (push that rules))))
    (loop while (> lives 0) do
         (let ((this nil) (found nil))
           (dotimes (number 20)
             (setf this (combine (twos (explode (normalize rules)))))
             (setf (rule-score this) (scoren this tbl))
             (dolist (that rules)
               (if (equalp this that)
                   (setf found t)))
             (unless (or found (>= 4 (rule-support this)))
               (push this rules))))
         (setf rules (prune rules))
         (if (> (rule-score (first rules)) max)
             (setf max (rule-score (first rules))
                   lives 6))
         (incf round)
         (decf lives))
    (final-prune rules)))

(defun rounds (class which tbl s)
  (let ((lives 5) (rules '()) (round 1) (max 0) (zero-score t))
    (dolist (this which)
      (push (to-rule (klass-name class) this) rules))
    (dolist (r rules)
      (setf (rule-score r) (score r tbl s))
      (if (> (rule-score r) 0)
          (setf zero-score nil)))
    (if zero-score
        rules
        (progn
          (loop while (> lives 0) do
               (let ((this nil) (found nil))
                 (dotimes (number 20)
                   (setf this (combine (twos (explode (normalize rules)))))
                   (setf (rule-score this) (score this tbl s))
                   (dolist (that rules)
                     (if (equalp this that)
                         (setf found t)))
                   (unless (or found (= 0 (rule-support this)))
                     (push this rules)))
                 (setf rules (prune rules))
                 (if (> (rule-score (first rules)) max)
                     (setf max (rule-score (first rules))
                           lives 6))
```

www.manaraa.com

```
                (incf round)
                (decf lives)))
          (final-prune rules)))))

(defun !table1 ()
  (data "../data/discrete-lisp/iris.lisp")
  (print (thetable)))

(defun !table2 ()
  (data "../data/numeric/servo.lisp")
  (print (discretizetbln (thetable) 3)))

(defun !table3 ()
  (data "../data/LispData/ant/01-ant-a.lisp")
  (print (discretizetblcode (thetable) 10)))

(defun !learn1 (str)
  (reset-seed)
  (data (concatenate 'string "../data/tests/" str ".lisp"))
  (which2n (discretizetbln (thetable) 2)))

(defun !learn2 ()
  (reset-seed)
  (data "../data/proj3/schaffer.lisp")
  (which2 (discretizetbl (thetable))))

(defun !learn3 ()
  (data "../data/proj3/iris.lisp")
  (which2 (discretizetbl (thetable))))

(defun !learn4 ()
  (reset-seed)
  (data "../data/numeric-lisp/weather.lisp")
  (which2n (discretizetbln (thetable))))

(defun !learn5 ()
  (reset-seed)
  (data "../data/xerces/xerces12.lisp")
  (which2n (discretizetbln (thetable))))

(defun !learnandrew ()
  (reset-seed)
  (data "../data/china.lisp")
  (which2n (discretizetbln (thetable))))

(defun !learnlqn ()
  (reset-seed)
  (data "lqn/lqn.lisp")
  (print-output (thetable))
  (which2n (discretizetbln (thetable))))

(defun !averageslqn ()
  (reset-seed)
  (data "lqn/lqn.lisp")
  (table-averages (thetable)))

(defun !printtablelqn ()
  (data "lqn/lqn.lisp")
  (print (thetable)))

(defun table-averages (tbl)
  (let ((out '()) (l (length (row-cells (first (table-rows tbl))))))
    (dolist (this (table-rows tbl) out)
```

المنارة للاستشارات
www.manaraa.com

```
      (push (list (nth (- l 1) (row-cells this)) (nth (- l 2) (row-cells this))) out))))

(defun print-output (tbl)
  (let ((out '()) (l (length (row-cells (first (table-rows tbl))))))
    (dolist (this (table-rows tbl) out)
      (print (list (nth (- l 1) (row-cells this)) (nth (- l 2) (row-cells this)) 2 )))))


                                        ;(deftest !learn ()
;   (test (!learn1)
;        ";;; NO
;            (56 HUMIDTY HIGH)
;            (44 FORECAST SUNNY)
;            (39 WINDY TRUE)
;            (26 TEMP HOT)
;            (22 FORECAST RAINY)
        ;;; YES
;            (51 HUMIDTY NORMAL)
;            (44 FORECAST OVERCAST)
;            (42 WINDY FALSE)
;            (23 TEMP MILD)
;            (21 TEMP COOL)"))
```

A.3 Other

A.3.1 angle.lisp

```
(defun similarity (a b)
  (/ (float (dot-product a b)) (float (* (magnitude a) (magnitude b)))))

(defun dot-product (a b)
  (if (null a)
      0
      (+ (* (first a) (first b)) (dot-product (rest a) (rest b)))))

(defun magnitude (a)
  (sqrt (sum-of-squares a)))

(defun sum-of-squares (a)
  (if (null a)
      0
      (+ (square (first a)) (sum-of-squares (rest a)))))

(defun diff-angle (a b)
  (let ((k (similarity a b)))
    (if (> k 1)
        (setf k 1.0))
    (radians-to-degrees (acos k))))

(defun radians-to-degrees (theta)
  (* theta (/ 180.0 pi)))
```

A.3.2 bestof.lisp

Written by Dr. Menzies

```
(defun maxof (l &key (test #'>) (key #'identity) (result #'identity))
  (bestof l test key result most-negative-fixnum nil))
```

المنارة للاستشارات
www.manaraa.com

```
(defun minof (l &key (test #'<) (key #'identity) (result #'identity))
  (bestof l test key result most-positive-fixnum nil))

(defun bestof (l > key result max out)
  (if (null l)
      (values out max)
      (let* ((head (first l))
             (max1 (funcall key head)))
        (if (funcall > max1 max)
            (bestof (rest l) > key result max1 (funcall result head))
            (bestof (rest l) > key result max  out)))))

(defun best-worst-of (l &key (key #'identity) (result #'identity))
  (if (null l)
      (values)
      (let* (min-ist
             max-ist
             (min most-positive-fixnum)
             (max most-negative-fixnum))
        (labels ((best-worst (one)
                   (let ((x (funcall key one)))
                     (when (< x min)
                       (setf min x
                             min-ist (funcall result one)))
                     (when (> x max)
                       (setf max x
                             max-ist (funcall result one))))))
          (visit #'best-worst l)
          (values min-ist max-ist min max)))))
```

A.3.3 data.lisp

Written by Dr. Menzies

```
;;;; utils
;;; about symbols
(defun thingp (x y) (and (symbolp x) (find y (symbol-name x))))
(defun goalp  (x)
  (if (thingp x (wme-goal *w*))
      (wme-goal *w*)
      (if (thingp x (wme-neggoal *w*))
          (wme-neggoal *w*)
          nil)))
(defun nump   (x)   (thingp x (wme-num *w*)))
(defun knownp (x)   (not (eql x (wme-unknown *w*))))

;;; misc
(defun zero (l)        (declare (ignore l)) 0)
(defun noop (&rest l) (declare (ignore l)) t)

;;;; main
;;; reader
(defun data (&optional f)
  (w0)
  (load (or f (thefile)))
  (funcall (wme-ready *w*))
  (funcall (wme-run *w*))
  (funcall (wme-report *w*))
)
```

www.manaraa.com

```
(defmacro deftable (name &rest cols)
  `(setf (wme-table *w*)
         (make-table :name ',name
                     :cols (mapcar #'make-col ',cols))))

(defun make-col (col)
  (if (nump col)
      (make-num :name col :goalp (goalp col))
      (make-sym :name col :goalp (goalp col))))

(defun sort-rows ()
  (setf (therows) (sort (therows) #'< :key #'row-sortkey)))

(defun defklass (class &optional (tbl (thetable)))
  (let ((k (first (member class (theklasses tbl) :key #'klass-name))))
    (unless k
      (setf  (theklasses tbl)
             (appendl (theklasses tbl) (setf k (make-klass :name class)))))
    k))

(defun defrow (l &optional (tbl (thetable)) &aux (class (car (goals-in-list l))))
  (incf (klass-n (defklass class tbl)))
  (push (make-row :cells   l
                  :class   class
                  :utility (funcall (theu) class)
                  :sortkey  (+ (randf 0.49)
                               (position class (theklasses tbl)
                                         :key #'klass-name)))
        (therows tbl)))

(defun goals-in-list (l)   (mapcan #'1goal-in-list l (thecols)))
(defun 1goal-in-list (x c) (and (col-goalp c) (knownp x) (list x)))

;;;; testss

(deftest !data ()
  (reset-seed)
  (data)
  (test (thetable)
        "#S(TABLE
␣␣␣:NAME␣WEATHER
␣␣␣:ROWS␣(#S(ROW
␣␣␣␣␣␣␣␣␣␣␣␣␣:CELLS␣(SUNNY␣MILD␣HIGH␣FALSE␣NO)
␣␣␣␣␣␣␣␣␣␣␣␣␣:CLASS␣NO
␣␣␣␣␣␣␣␣␣␣␣␣␣:UTILITY␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣:SORTKEY␣0.2585191038174379d0)
␣␣␣␣␣␣␣␣␣␣#S(ROW
␣␣␣␣␣␣␣␣␣␣␣␣␣:CELLS␣(RAINY␣COOL␣NORMAL␣TRUE␣NO)
␣␣␣␣␣␣␣␣␣␣␣␣␣:CLASS␣NO
␣␣␣␣␣␣␣␣␣␣␣␣␣:UTILITY␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣:SORTKEY␣0.27521698080383394d0)
␣␣␣␣␣␣␣␣␣␣#S(ROW
␣␣␣␣␣␣␣␣␣␣␣␣␣:CELLS␣(RAINY␣MILD␣HIGH␣TRUE␣NO)
␣␣␣␣␣␣␣␣␣␣␣␣␣:CLASS␣NO
␣␣␣␣␣␣␣␣␣␣␣␣␣:UTILITY␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣:SORTKEY␣0.40643407928930564d0)
␣␣␣␣␣␣␣␣␣␣#S(ROW
␣␣␣␣␣␣␣␣␣␣␣␣␣:CELLS␣(SUNNY␣HOT␣HIGH␣TRUE␣NO)
␣␣␣␣␣␣␣␣␣␣␣␣␣:CLASS␣NO
␣␣␣␣␣␣␣␣␣␣␣␣␣:UTILITY␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣:SORTKEY␣0.44694239671216807d0)
␣␣␣␣␣␣␣␣␣␣#S(ROW
␣␣␣␣␣␣␣␣␣␣␣␣␣:CELLS␣(SUNNY␣HOT␣HIGH␣FALSE␣NO)
```

المنارة للاستشارات
www.manaraa.com

```
              :CLASS NO
              :UTILITY 0
              :SORTKEY 0.45160094544602414d0)
          #S(ROW
              :CELLS (OVERCAST MILD HIGH TRUE YES)
              :CLASS YES
              :UTILITY 0
              :SORTKEY 1.0410945777495766d0)
          #S(ROW
              :CELLS (OVERCAST HOT HIGH FALSE YES)
              :CLASS YES
              :UTILITY 0
              :SORTKEY 1.0607153431365122d0)
          #S(ROW
              :CELLS (SUNNY COOL NORMAL FALSE YES)
              :CLASS YES
              :UTILITY 0
              :SORTKEY 1.1004932973769972d0)
          #S(ROW
              :CELLS (RAINY COOL NORMAL FALSE YES)
              :CLASS YES
              :UTILITY 0
              :SORTKEY 1.1968093818857513d0)
          #S(ROW
              :CELLS (RAINY MILD HIGH FALSE YES)
              :CLASS YES
              :UTILITY 0
              :SORTKEY 1.2627522398599997d0)
          #S(ROW
              :CELLS (OVERCAST HOT NORMAL FALSE YES)
              :CLASS YES
              :UTILITY 0
              :SORTKEY 1.266554799861022d0)
          #S(ROW
              :CELLS (RAINY MILD NORMAL FALSE YES)
              :CLASS YES
              :UTILITY 0
              :SORTKEY 1.4508161515756934d0)
          #S(ROW
              :CELLS (OVERCAST COOL NORMAL TRUE YES)
              :CLASS YES
              :UTILITY 0
              :SORTKEY 1.4617063527177772d0)
          #S(ROW
              :CELLS (SUNNY MILD NORMAL TRUE YES)
              :CLASS YES
              :UTILITY 0
              :SORTKEY 1.4869120755548817d0))
   :KLASSES (#S(KLASS :NAME NO :N 5) #S(KLASS :NAME YES :N 9))
   :COLS (#S(SYM :NAME FORECAST :GOALP NIL :COUNTS {hash of 0 items})
          #S(SYM :NAME TEMP :GOALP NIL :COUNTS {hash of 0 items})
          #S(SYM :NAME HUMIDTY :GOALP NIL :COUNTS {hash of 0 items})
          #S(SYM :NAME WINDY :GOALP NIL :COUNTS {hash of 0 items})
          #S(SYM :NAME !PLAY :GOALP #\! :COUNTS {hash of 0 items}))
   :RESULTS NIL)"))
```

A.3.4 deftest.lisp

Written by Dr. Menzies

```
;;; test engine
```

المنارة للاستشارات
www.manaraa.com

```
(defparameter *tests* nil)
(defmacro deftest (name params &body body)
  `(progn (unless (member ',name *tests*) (push ',name *tests*))
          (defun ,name ,params ,@body)))

(let ((pass 0)
      (fail 0))
  (defun test (want got)
    (labels
        ((white      (c)    (member c '(#\# #\\ #\Space #\Tab #\Newline
                                        #\Linefeed #\Return #\Page) :test #'char=))
         (whiteout (s)    ;(print (remove-if #'white s))
                          (remove-if #'white s))
         (samep      (x y) (string= (whiteout (format nil "~(~a~)" x))
                                    (whiteout (format nil "~(~a~)" y)))))
      (cond ((samep want got) (incf pass))
            (t               (incf fail)
                             (format t "~&; fail : expected ~a~%" want)))
      got))
  (defun tests ()
    (labels ((run (x) (format t "~&;testing  ~a~%" x) (funcall x)))
      (when *tests*
        (setf fail 0 pass 0)
        (mapcar #'run (reverse *tests*))
        (format t "~&; pass : ~a = ~5,1f% ~%; fail : ~a = ~5,1f% ~%"
                pass (* 100 (/ pass (+ pass fail)))
                fail (* 100 (/ fail (+ pass fail)))))))
)

(deftest !deftest1 (&aux (a 1))
  (test (+ a 1) 2))

(deftest !deftest2 (&aux (a 1))
  (test (+ a 1) 3))
```

A.3.5 hash.lisp

Written by Dr. Menzies

```
(defmacro dohash ((key value hash &optional end) &body body)
  `(progn (maphash #'(lambda (,key ,value)
                       ,@body)
                   ,hash)
          ,end))

(defmacro dovalues ((value hash &optional end) &body body)
  (let ((key (gensym)))
    `(progn (maphash #'(lambda (,key ,value)
                         (declare (ignore ,key))
                         ,@body)
                     ,hash)
            ,end)))

(defmethod print-object ((h hash-table) str)
  (format str "{hash of ~a items}" (hash-table-count h)))

(defun showh (h &key
             (indent 0) (stream t) (before "") (after "")
             (if-empty "empty")
             (show #'(lambda (x)
                       (format stream "~a~a = ~a~%"
```

المنارة للاستشارات
www.manaraa.com

```
                              (nchars indent) (first x) (rest x))))
               (lt #'lt))
 (if (zerop (hash-table-count h))
     (format stream "~a~a~a" before if-empty after)
     (let (l)
       (format stream "~a" before)
       (maphash #'(lambda (k v) (push (cons k v) l)) h)
       (mapc show
             (sort l #'(lambda (a b)
                         (funcall lt (car a) (car b)))))
       (format stream "~a" after)
       h)))
```

A.3.6 lib.lisp

Written by Dr. Menzies

```
(defun appendl (a b) (append a (list b)))

;(defun abs (x) (if (< x 0) (* -1 x) x))

(defun sum (l)
  "Computes the sum of a list of numbers."
  (let ((sum 0))
    (dolist (x l sum)
      (incf sum x))))

(defun mean (&rest nums)
  (/ (apply #'+ nums)
     (length nums)))

(defun median (l &optional (n (length l)))
  (let* ((sorted (sort l #'<))
         (midv   (floor (/ n 2)))
         (midpos (nth midv sorted)))
    (if (oddp n)
        midpos
        (mean  midpos
               (nth (1- midv) sorted)))))

(defun lt (x y)
  (string< (format nil "~a" x) (format nil "~a" y)))

(defun flatten (lis)
  "Removes nestings from a list."
  (cond
    ((atom lis)         lis)
    ((listp (car lis)) (append (flatten (car lis))
                               (flatten (cdr lis))))
    (t                 (append (list (car lis))
                               (flatten (cdr lis))))))

(defun nchars (n &optional (char #\Space))
  (with-output-to-string (s)
    (dotimes (i n)
      (format s "~a" char))))

(defun visit (f l)
  (if (atom l)
      (funcall f l)
      (dolist (x l)
```

المنارة للاستشارات
www.manaraa.com

```
        (visit f x))))

(defun percentiles (l &optional (collect '(0 25 50 75 100)))
  "one pass through a list of nums to 'collect' some percentiles"
  (let* (out
         last
         (size (length l)))
    (doitems (one pos l out)
      (if (null collect)
          (return-from percentiles out))
      (let ((want (first collect))
            (progress (* 100.0 (/ (1+ pos) size))))
        (if (>= progress want)
            (push (cons (pop collect)
                        (if (= progress want)
                            one
                            (mean one (or last one))))
                  out)))
      (setf last one))))

(deftest !percentiles ()
  (test
   (percentiles '(1 2 3 4 5 6 7 8 9 10)
                '(0 25 50 75 100)
                )
   '((100 . 10)
     (75 . 15/2)
     (50 . 5)
     (25 . 5/2)
     (0 . 1))))
```

A.3.7 hash.lisp

Written by Dr. Menzies

```
; list of tricks that,net year, i will move into lib

(defun rands (&optional (thing "thing"))
  (intern
   (format nil "~:@(~a~)-~a" thing (randi 1000000000))))

(defmacro dokeys ((key hash &optional end) &body body)
  (let ((value (gensym)))
    `(progn (maphash #'(lambda (,key ,value)
                         (declare (ignore ,value))
                         ,@body)
                     ,hash)
            ,end)))
```

A.3.8 macros.lisp

Written by Dr. Menzies

```
(defmacro oo (&rest l)
  `(progn (terpri) (o ,@l)))

(defmacro o (&rest l)
  (let ((last (gensym)))
    `(let (,last)
       ,@(mapcar #'(lambda(x) `(setf ,last (oprim ,x))) l)
```

المنارة للاستشارات
www.manaraa.com

```
      (terpri)
      ,last)))

(defmacro oprim (x)
  `(progn (format t "~&[~a]=[~a] " (quote ,x) ,x) ,x))

(Defmacro doitems ((one n list &optional out) &body body )
  `(let ((,n -1))
     (dolist (,one ,list ,out)
       (incf ,n)
       ,@body)))


(defmacro do12 ((one two list &optional out) &body body)
  `(let ((,one (car ,list)))
     (dolist (,two (cdr ,list) ,out)
       ,@body
       (setf ,one ,two))))
```

A.3.9 profile.lisp

Written by Dr. Menzies

```
;;;; profiling tricks
(defmacro time-it (n &body body)
  (let ((n1 (gensym))
        (i  (gensym))
        (t1 (gensym)))
    `(let ((,n1 ,n)
           (,t1 (get-internal-run-time)))
       (dotimes (,i ,n1) ,@body)
       (float (/ (- (get-internal-run-time) ,t1)
                 (* ,n1 internal-time-units-per-second))))))

(defun test-time-it (&key (repeats 100) (loops 100) (max 10))
  (let (out)
    (dotimes (i loops)
      (push (random max) out))
    (time-it repeats
      (apply #'+ out))))

(defmacro watch (code)
  `(progn
    (sb-profile:unprofile)
    (sb-profile:reset)
    (sb-profile:profile ,@(my-funs))
    (eval ,code)
    (sb-profile:report)
    (sb-profile:unprofile)
    t)
)

(defun my-funs ()
  (let ((out '()))
    (do-symbols  (s)
      (if (and (fboundp s)
               (find-symbol  (format nil "~a" s) *package*)
               (not (member s (lisp-funs))))
          (push s out)))
    out))
```

المنارة للاستشارات
www.manaraa.com

```
(defun lisp-funs () '(
    * * ** *** + + ++ +++ - - / / // /// 1+ 1- < <= = > >= abort abs
    acons acos acosh add-method adjoin adjust-array adjustable-array-p
    alpha-char-p alphanumericp alter always and append append appending
    apply applyhook *applyhook* apropos apropos-list aref arithmetic-error
    arithmetic-error-operands arithmetic-error-operation array-dimension
    array-dimension-limit array-dimensions array-element-type
    array-has-fill-pointer-p array-in-bounds-p array-rank array-rank-limit
    array-row-major-index array-total-size array-total-size-limit arrayp
    as ash asin asinh assert assert assoc assoc-if assoc-if-not atan
    atanh atom augment-environment bit bit-and bit-andc1 bit-andc2
    bit-eqv bit-ior bit-nand bit-nor bit-not bit-orc1 bit-orc2 bit-vector-p
    bit-xor block boole both-case-p boundp break break *break-on-signals*
    *break-on-warnings* broadcast-stream-streams butlast byte byte-position
    byte-size caaaar caaadr caaar caadar caaddr caadr caar cadaar cadadr
    cadar caddar cadddr caddr cadr call-arguments-limit call-method
    call-next-method car case catch catenate ccase ccase cdaaar cdaadr
    cdaar cdadar cdaddr cdadr cdar cddaar cddadr cddar cdddar cddddr
    cdddr cddr cdr ceiling cell-error cell-error-name cerror cerror
    change-class char char-bit char-bits char-bits-limit char-code
    char-code-limit char-control-bit char-downcase char-equal char-font
    char-font-limit char-greaterp char-hyper-bit char-int char-lessp
    char-meta-bit char-name char-not-equal char-not-greaterp char-not-lessp
    char-super-bit char-upcase char/= char< char<= char= char> char>=
    character characterp check-type check-type choose choose-if chunk
    cis class-name class-of clear-input close clrhash code-char coerce
    collect collect collect-alist collect-and collect-append collect-file
    collect-first collect-fn collect-hash collect-last collect-length
    collect-max collect-min collect-nconc collect-nth collect-or
    collect-plist collect-sum collecting collecting-fn commonp compile
    compile-file compile-file-pathname *compile-file-pathname*
    *compile-file-truename* *compile-print* *compile-verbose*
    compiled-function-p compiler-let compiler-let compiler-macro-function
    compiler-macroexpand compiler-macroexpand-1 complement complex
    complexp compute-applicable-methods compute-restarts concatenate
    concatenated-stream-streams cond condition conjugate cons consp
    constantp continue control-error copy-alist copy-list copy-pprint-dispatch
    copy-readtable copy-seq copy-symbol copy-tree cos cosh cotruncate
    count count count-if count-if-not counting ctypecase ctypecase
    *debug-io* *debugger-hook* decf declaim declaration-information
    declare decode-float decode-universal-time *default-pathname-defaults*
    defclass defgeneric define-compiler-macro define-condition
    define-declaration define-method-combination define-modify-macro
    define-setf-method defmacro defmethod defpackage defstruct deftype
    defun defvar delete delete-duplicates delete-file delete-if
    delete-if-not delete-package denominator deposit-field describe
    describe-object destructuring-bind digit-char digit-char-p directory
    directory-namestring disassemble division-by-zero do do do*
    do-all-symbols do-external-symbols do-symbols documentation
    documentation doing dolist dotimes double-float-epsilon
    double-float-negative-epsilon dpb dribble ecase echo-stream-input-stream
    echo-stream-output-stream ed eighth elt encapsulated enclose
    encode-universal-time end-of-file endp enough-namestring
    ensure-generic-function eq eql equal equalp error error error
    *error-output* etypecase etypecase eval eval-when evalhook *evalhook*
    evenp every exp expand export expt fboundp fdefinition *features*
    ffloor fifth file-author file-error file-error-pathname file-length
    file-namestring file-position file-string-length file-write-date
    fill fill-pointer finally find find-all-symbols find-class find-if
    find-if-not find-method find-package find-restart find-symbol
    finish-output first flet float float-digits float-precision float-radix
    float-sign floating-point-overflow floating-point-underflow floatp
    floor for format formatter fourth funcall function function-information
```

www.manaraa.com

```
function-keywords function-lambda-expression functionp gatherer
gathering gcd generator generic-flet generic-function generic-labels
gensym *gensym-counter* gentemp get get-decoded-time get-internal-real-time
get-internal-run-time get-output-stream-string get-properties
get-setf-method get-setf-method-multiple-value get-universal-time
getf gethash go graphic-char-p handler-bind handler-case hash-table-count
hash-table-p hash-table-rehash-size hash-table-rehash-threshold
hash-table-size hash-table-test host-namestring identity if if
ignore-errors imagpart import in-package in-package incf
initialize-instance initially input-stream-p inspect int-char
integer-decode-float integer-length integerp interactive-stream-p
intern internal-time-units-per-second intersection invalid-method-error
invoke-debugger invoke-restart isqrt iterate keywordp lambda-list-keywords
lambda-parameters-limit last latch lcm ldb ldb-test ldiff
least-negative-double-float least-negative-long-float
least-negative-normalized-double-float least-negative-normalized-long-float
least-negative-normalized-short-float
least-negative-normalized-single-float least-negative-short-float
least-negative-single-float least-positive-double-float
least-positive-long-float least-positive-normalized-double-float
least-positive-normalized-long-float least-positive-normalized-short-float
least-positive-normalized-single-float least-positive-short-float
least-positive-single-float length let let* lisp-implementation-type
lisp-implementation-version list list* list-all-packages list-length
listen listp load load-logical-pathname-translations *load-pathname*
*load-print* load-time-value *load-truename* *load-verbose* locally
locally log logand logandc1 logandc2 logbitp logcount logeqv
logical-pathname logical-pathname logical-pathname-translations
logior lognand lognor lognot logorc1 logorc2 logtest logxor
long-float-epsilon long-float-negative-epsilon long-site-name loop
loop-finish lower-case-p machine-instance machine-type machine-version
macro-function macroexpand macroexpand-1 *macroexpand-hook* make-array
make-broadcast-stream make-char make-concatenated-stream make-condition
make-dispatch-macro-character make-echo-stream make-hash-table
make-instance make-instances-obsolete make-list make-load-form
make-load-form-saving-slots make-package make-pathname make-random-state
make-sequence make-string make-string-input-stream
make-string-output-stream make-symbol make-synonym-stream
make-two-way-stream makunbound map map-fn map-into mapc mapcan
mapcar mapcon maphash mapl maplist mapping mask mask-field max
maximize maximizing member member-if member-if-not merge merge-pathnames
method-combination-error method-qualifiers min mingle minimize
minimizing minusp mismatch mod *modules* most-negative-double-float
most-negative-fixnum most-negative-long-float most-negative-short-float
most-negative-single-float most-positive-double-float most-positive-fixnum
most-positive-long-float most-positive-short-float
most-positive-single-float muffle-warning multiple-value-bind
multiple-value-call multiple-value-list multiple-value-prog1
multiple-value-setq multiple-values-limit name-char named namestring
nbutlast nconc nconc nconcing never next-in next-method-p next-out
nil nintersection ninth no-applicable-method no-next-method not
notany notevery nreconc nreverse nset-difference nset-exclusive-or
nstring-capitalize nstring-downcase nstring-upcase nsublis nsubst
nsubst-if nsubst-if-not nsubstitute nsubstitute-if nsubstitute-if-not
nth nth-value nthcdr null numberp numerator nunion oddp off-line-port
open open-stream-p optimizable-series-function or output-stream-p
*package* package-error package-error-package package-name
package-nicknames package-shadowing-symbols package-use-list
package-used-by-list packagep pairlis parse-integer parse-macro
parse-namestring pathname pathname-device pathname-directory
pathname-host pathname-match-p pathname-name pathname-type
pathname-version pathnamep peek-char phase pi plusp pop position
position-if position-if-not positions pprint-dispatch
```

www.manaraa.com

```
pprint-exit-if-list-exhausted pprint-fill pprint-indent pprint-linear
pprint-logical-block pprint-newline pprint-pop pprint-tab pprint-tabular
previous prin1 *print-array* *print-base* *print-case* *print-circle*
*print-escape* *print-gensym* *print-length* *print-level* *print-lines*
*print-miser-width* print-object *print-pprint-dispatch* *print-pretty*
*print-radix* *print-readably* *print-right-margin* print-unreadable-object
probe-file proclaim producing prog prog* prog1 prog2 progn program-error
progv propagate-alterability provide psetf psetq push pushnew
*query-io* quote random *random-state* random-state-p rassoc rassoc-if
rassoc-if-not rational rationalize rationalp read *read-base*
read-byte read-char read-char-no-hang *read-default-float-format*
read-delimited-list *read-eval* read-from-string read-line
read-preserving-whitespace *read-suppress* *readtable* readtable-case
readtablep realp realpart reduce reinitialize-instance rem remf
remhash remove remove-duplicates remove-method remprop rename-file
rename-package repeat replace require rest restart restart-bind
restart-case restart-name result-of return return return-from
revappend reverse room rotatef round row-major-aref rplaca rplacd
sbit scale-float scan scan-alist scan-file scan-fn scan-fn-inclusive
scan-hash scan-lists-of-lists scan-lists-of-lists-fringe scan-multiple
scan-plist scan-range scan-sublists scan-symbols schar search second
series series series-element-type serious-condition set set-char-bit
set-difference set-dispatch-macro-character set-exclusive-or
set-macro-character set-pprint-dispatch set-syntax-from-char setf
setq seventh shadow shadowing-import shared-initialize shiftf
short-float-epsilon short-float-negative-epsilon short-site-name
signal signum simple-bit-vector-p simple-condition
simple-condition-format-arguments simple-condition-format-string
simple-error simple-string-p simple-type-error simple-vector-p
simple-warning sin single-float-epsilon single-float-negative-epsilon
sinh sixth sleep slot-boundp slot-exists-p slot-makunbound slot-missing
slot-unbound slot-value software-type software-version some sort
special-form-p split split-if sqrt stable-sort standard-char-p
*standard-input* *standard-output* step storage-condition store-value
stream-element-type stream-error stream-error-stream stream-external-format
streamp string string-capitalize string-char-p string-downcase
string-equal string-greaterp string-left-trim string-lessp
string-not-equal string-not-greaterp string-not-lessp string-right-trim
string-trim string-upcase string/= string< string<= string= string>
string>= stringp sublis subseq subseries subsetp subst subst-if
subst-if-not substitute substitute-if substitute-if-not subtypep
sum summing *suppress-series-warnings* svref sxhash symbol-function
symbol-macrolet symbol-name symbol-package symbol-plist symbol-value
symbolp synonym-stream-symbol t tagbody tailp tan tanh tenth
*terminal-io* terminate-producing terpri the thereis third throw
time to-alter trace *trace-output* translate-logical-pathname
translate-pathname tree-equal truename truncate two-way-stream-input-stream
two-way-stream-output-stream type-error type-error-datum
type-error-expected-type type-of typecase typep unbound-variable
undefined-function unexport unintern union unless unless unread-char
until until-if untrace unuse-package unwind-protect
update-instance-for-different-class update-instance-for-redefined-class
upgraded-array-element-type upgraded-complex-part-type upper-case-p
use-package use-value user-homedir-pathname values values-list
variable-information vector vector-pop vector-push vector-push-extend
warn warning when when while wild-pathname-p with with-accessors
with-added-methods with-compilation-unit with-condition-restarts
with-hash-table-iterator with-input-from-string with-open-file
with-open-stream with-output-to-string with-package-iterator
with-simple-restart with-slots with-standard-io-syntax write
write-byte write-char write-string write-to-string y-or-n-p yes-or-no-p
zerop unprofile reset report profile stream-read-char-no-hang
stream-fresh-line stream-peek-char stream-write-char stream-write-byte
```

www.manaraa.com

```
    stream-write-string stream-line-column stream-write-sequence
    stream-read-byte stream-read-line stream-line-length stream-read-sequence
    stream-read-char stream-clear-output stream-unread-char stream-clear-input
    stream-finish-output stream-start-line-p stream-force-output
    stream-terpri stream-advance-to-column stream-file-position
    stream-listen weak-pointer-p package-locked-p step-condition-result
    native-pathname defconstant-uneql-new-value defconstant-uneql-name
    cancel-finalization purify process-status-hook process-output
    timer-scheduled-p package-lock-violation process-plist interactive-eval
    list-all-timers process-p process-status get-bytes-consed process-error
    defconstant-uneql-old-value hash-table-weakness step-next
    package-implements-list float-nan-p octets-to-string with-unlocked-packages
    enable-debugger float-denormalized-p with-timeout
    package-locked-error-symbol process-pid package-implemented-by-list
    process-pty posix-getenv step-condition-args gc-off finalize
    without-package-locks unschedule-timer schedule-timer make-timer
    native-namestring parse-native-namestring float-infinity-p lock-package
    process-kill process-exit-code step-continue string-to-octets
    unlock-package quit process-alive-p remove-implementation-package
    find-executable-in-search-path weak-pointer-value process-wait
    disable-debugger process-core-dumped define-source-context
    add-implementation-package run-program process-close step-condition-form
    posix-environ timer-name process-input bytes-consed-between-gcs
    gc-on make-weak-pointer save-lisp-and-die describe-compiler-policy
    step-into gc float-trapping-nan-p truly-the internal-debug
    frame-has-debug-tag-p backtrace-as-list arg var backtrace
    unwind-to-frame-and-call slot alien-funcall def-alien-variable deref
    addr with-alien load-shared-object define-alien-routine def-alien-routine
    make-alien free-alien alien-sap cast get-errno load-foreign sap-alien
    def-alien-type null-alien define-alien-type define-alien-variable
    extern-alien load-1-foreign alien-size clear-output print princ-to-string
    defsetf remove-if-not vectorp print-not-readable-object copy-structure
    read-sequence get-dispatch-macro-character define-setf-expander
    fmakunbound write-sequence constantly labels prin1-to-string
    get-setf-expansion defconstant simple-condition-format-control
    ensure-directories-exist unbound-slot-instance /= get-macro-character
    allocate-instance remove-if array-displacement fceiling special-operator-p
    force-output princ lambda invoke-restart-interactively ftruncate
    fround write-line macrolet define-symbol-macro pprint fresh-line
    defparameter
    )
)
```

A.3.10 random.lisp

Written by Dr. Menzies

```
;;; random stuff
(let* ((seed0      10013)
       (seed       seed0)
       (multiplier 16807.0d0)
       (modulus    2147483647.0d0))
  (defun reset-seed ()  (setf seed seed0))
  (defun randf      (n) (* n (- 1.0d0 (park-miller-randomizer))))
  (defun randi      (n) (floor (* n (/ (randf 1000.0) 1000))))
  (defun park-miller-randomizer ()
       "cycle= 2,147,483,646 numbers"
    (setf seed (mod (* multiplier seed) modulus))
    (/ seed modulus))
)
```

www.manaraa.com

```
(deftest !rands ()
  (reset-seed)
  (dotimes (i 11) (randf 100))
  (test (randf 100) 8.386648357094572d0)
  (reset-seed)
  (test (randf 100) 92.16345646053713d0))
```

A.3.11 structs.lisp

Written by Dr. Menzies

```
;;;; macros

(defmacro !         (&rest l) `(funcall (wme-! *w*) ',l))
(defmacro theu       ()        `(wme-utility-function *w*))
(defmacro thefile    ()        `(wme-file *w*))
(defmacro thetable   ()        `(wme-table *w*))
(defmacro thecols    (&optional tbl) `(table-cols    (or ,tbl (wme-table *w*))))
(defmacro thename    (&optional tbl) `(table-name    (or ,tbl (wme-table *w*))))
(defmacro therows    (&optional tbl) `(table-rows    (or ,tbl (wme-table *w*))))
(defmacro theklasses (&optional tbl) `(table-klasses (or ,tbl (wme-table *w*))))

;;;; structs
(defstruct result  target (a 0) (b 0) (c 0) (d 0) acc pf prec pd f details)
(defstruct table name rows klasses cols results hpipes)
(defstruct row   cells class utility sortkey)
(defstruct col   name goalp)
(defstruct klass (name '()) (n 0))
(defstruct (sym  (:include col)) (counts (make-hash-table :test 'equal)))
(defstruct (num  (:include col))
  (n 0)
  (sum 0)
  (sumsq 0)
  (min most-positive-fixnum)
  (max most-negative-fixnum)
 )
(defstruct wme
  (goal             #\!)
  (neggoal          #\#)
  (num              #\$)
  (unknown          #\?)
  (file             "../data/discrete-lisp/weather.lisp")
  (utility-function #'zero)
  (!                #'defrow)
  (ready            #'sort-rows)
  (run              #'noop)
  (report           #'noop)
  table
)
(defstruct rig
  (preprocess #'noop)
  (train #'noop)
  (ready #'noop)
  (tester #'noop)
  (reporter #'noop))

;;;; globals
(defparameter *w* nil)
(defun w0 () (setf *w* (make-wme)))
```

المنارة للاستشارات
www.manaraa.com

```
;;; utils

(defun klass.majority (tbl)
  "return the symbol of the largest class"
  (maxof (table-klasses tbl) :key #'klass-n :result #'klass-name))

(defun klass.all (tbl &aux out)
  (mapcar #'klass-name (table-klasses tbl)))
```

A.3.12 trapezoid.py

```
# Quick and dirty script to calculate area under the curve
#


def parse(line):
    out = [];
    if len(line.split(" "))<2:
        return -99;
    else:
        for x in line.split(" ")[0:2]:
                out.append(int(x)/100.0);
        return out;


#IT'S A
def trap(x1, y1, x2, y2):
    if (x1 > x2):
        print "Danger Will Robinson! Danger!"
        print x1;
    return .5*(y1+y2)*(x2-x1);

infile = open(raw_input("File name: "));
data = [[0,0]];
s = 0;
for line in infile:
    if not (parse(line)) == -99:
        data.append(parse(line));
data.append([data[-1][0],1.0]);

for i in range(0, (len(data)-1)):
    s = s + trap(data[i][1], data[i][0], data[i+1][1], data[i+1][0]);

print s;
```

A.4 Data

A sample of Lisp data.

A.4.1 weather.lisp

```
 ( deftable $weather
        forecast $temperature
    $humidity windy !class)
```

المنارة للاستشارات
www.manaraa.com

```
( ! sunny 85 85 FALSE no)
( ! sunny 80 90 TRUE no)
( ! overcast 83 86 FALSE yes)
( ! rainy 70 96 FALSE yes)
( ! rainy 68 80 FALSE yes)
( ! rainy 65 70 TRUE no)
( ! overcast 64 65 TRUE yes)
( ! sunny 72 95 FALSE no)
( ! sunny 69 70 FALSE yes)
( ! rainy 75 80 FALSE yes)
( ! sunny 75 70 TRUE yes)
( ! overcast 72 90 TRUE yes)
( ! overcast 81 75 FALSE yes)
( ! rainy 71 91 TRUE no)
```

A sample of defect prediction data.

A.4.2 pbeans.lisp

```
(deftable pbeans−a $wmc $dit $noc $cbo $rfc $lcom $ca $ce $npm $lcom3 $loc $dam $moa $mfa $cam $ic
     $cbm $amc $max_cc $avg_cc !bug)
(! 2 2 0 3 7 1 1 2 1 2 33 0 0 0.981132075 0.6 1 1 15.5 1 0.5 1)
(! 4 1 0 7 21 0 1 6 4 0.333333333 96 1 0 0 0.583333333 0 0 22.5 1 0.75 2)
(! 2 2 0 2 4 1 0 2 2 2 25 0 0 0.944444444 1 0 0 11.5 1 0.5 1)
(! 1 1 0 3 1 0 2 1 1 2 1 0 0 0 1 0 0 0 1 1 0)
(! 1 1 0 5 1 0 5 0 1 2 1 0 0 0 1 0 0 0 1 1 0)
(! 8 1 0 7 8 28 5 2 8 2 8 0 0 0 0.339285714 0 0 0 1 1 1)
(! 6 1 0 8 7 9 8 0 6 0.4 43 1 0 0 0.433333333 0 0 5.666666667 1 0.5 1)
(! 27 1 3 13 122 283 7 9 11 0.880769231 1394 1 1 0 0.190883191 0 0 50.25925926 2 1.037 4)
(! 8 1 0 2 24 0 2 1 4 0.642857143 235 1 0 0 0.375 0 0 27.625 1 0.875 1)
(! 7 1 0 4 15 5 4 0 7 0.25 97 1 0 0 0.371428571 1 1 12.57142857 4 1.1429 1)
(! 3 4 0 3 6 3 2 1 3 2 18 0 0 1 0.777777778 0 0 5 0 0 0)
(! 6 1 0 6 6 15 2 5 6 2 6 0 0 0 0.5 0 0 0 1 1 1)
(! 1 2 0 2 2 0 1 1 1 2 5 0 0 1 1 0 0 4 0 0 2)
(! 7 1 0 2 8 15 1 2 6 0.833333333 34 1 2 0 0.357142857 0 0 3.428571429 1 0.8571 2)
(! 74 1 0 18 185 2517 6 16 21 0.894216134 2390 1 2 0 0.152296535 0 0 31.05405405 5 1.2297 4)
(! 7 1 0 5 24 9 3 4 7 0 202 1 1 0 0.428571429 0 0 27.71428571 1 0.8571 2)
(! 3 1 0 0 17 3 0 0 3 2 75 0 0 0 0.5 0 0 24 1 0.6667 0)
(! 3 3 1 7 6 3 7 0 3 2 18 0 0 1 0.777777778 0 0 5 0 0 0)
(! 4 2 0 3 13 6 1 2 2 2 62 0 0 0.945454545 0.357142857 0 0 14.5 3 1.5 2)
(! 9 1 0 2 15 10 2 0 9 0.416666667 106 1 0 0 0.4 1 1 10.44444444 5 1.1111 1)
(! 12 1 0 0 25 48 0 0 12 0.787878788 159 1 0 0 0.333333333 0 0 12 1 0.8333 1)
(! 4 2 0 3 13 6 1 2 2 2 73 0 0 0.945454545 0.458333333 0 0 17.25 2 1 3)
(! 7 1 3 9 39 13 3 6 6 0.75 446 1 0 0 0.404761905 0 0 62.42857143 27 5.1429 4)
(! 6 1 0 3 7 3 2 1 6 0.6 34 1 0 0 0.555555556 0 0 4.333333333 1 0.6667 0)
(! 2 2 0 4 4 1 0 4 2 2 11 0 0 0.944444444 0.75 0 0 4.5 1 0.5 1)
(! 0 1 0 9 0 0 9 0 0 2 0 0 0 0 0 0 0 0 0 0 1)
```

A.4.3 schaffer.lisp

A sample of mathematical model data. generation.

```
;Note: The size of this file was decrease dramatically for space

(deftable $schaffer $x0 $#f1 $#f2)
(! −25.094492134905096d0 629.7335355088137d0 734.1115040484341d0)
(! −73.8554229182979d0 5454.623494440643d0 5754.045186113835d0)
(! −38.871580355058654d0 1510.999759299782d0 1670.4860807200166d0)
(! −57.892814628356 8d0 3351.5779855932833d0 3587.1492441067107d0)
(! −120.73587864032363d0 14577.152391050957d0 15064.095905612252d0)
```

المنارة للاستشارات

www.manaraa.com

```
(! -161.0116283966359d0 25924.74447893637d0 26572.79099252291d0)
(! -101.17416859971824d0 10236.212391844212d0 10644.909066243086d0)
(! -142.2732371245225d0 20241.674001890606d0 20814.766950388697d0)
(! -93.331372316166d0 8710.745058418797d0 9088.070547683461d0)
(! -106.89018582181814d0 11425.511825022812d0 11857.072568310083d0)
(! -110.38906575795832d0 12185.745838914847d0 12631.30210194668d0)
(! -48.22683878313629d0 2325.8279790146184d0 2522.7353341471635d0)
```

A.4.4 poiwhich.txt

A sample input to the Area Under Curve Python Script. "poiwhich" means the results from using the B_1 heuristic measure on the poi dataset.

```
1 0
3 0
22 8
35 13
35 15
52 31
53 32
93 76
97 81
98 86
98 90
99 90
100 91
100 91
100 98
```

www.manaraa.com

Bibliography

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182 –197, apr 2002.

[2] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[3] Christos Faloutsos and King-Ip Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *SIGMOD Rec.*, 24(2):163–174, May 1995.

[4] Usama M Fayyad and Keki B Irani. *Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning*, volume 2, pages 1022–1027. Morgan Kaufmann, 1993.

[5] C.M. Fonseca and P.J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. ii. application example. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 28(1):38 –47, jan 1998.

[6] Arnaud Frville and Sad Hanafi. The multidimensional 0-1 knapsack problembounds and computational aspects. *Annals of Operations Research*, 139:195–227, 2005. 10.1007/s10479-005-3448-8.

[7] T. Menzies G. Boetticher and T. Ostrand. Promise repository of empirical software engineering data. West Virginia University, Department of Computer Science.

[8] B. Ganter, G. Stumme, and R. Wille. *Formal Concept Analysis: Foundations And Applications*. Lecture Notes in Computer Science. Springer, 2005.

[9] B. Ganter, G. Stumme, and R. Wille. *Formal Concept Analysis: Foundations And Applications*. Lecture Notes in Computer Science. Springer, 2005.

[10] A.C. Godi andnez, L.E.M. Espinosa, and E.M. Montes. An experimental comparison of multiobjective algorithms: Nsga-ii and omopso. In *Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2010*, pages 28 –33, 28 2010-oct. 1 2010.

المنارة للاستشارات

www.manaraa.com

[11] Karen Gundy-Burlet, Johann Schumann, Tim Menzies, and Tony Barrett. Parametric analysis of antares re-entry guidance algorithms using advanced test generation and data analysis, 2008.

[12] Mark Harman. The current state and future of search based software engineering. In *2007 Future of Software Engineering*, FOSE '07, pages 342–357, Washington, DC, USA, 2007. IEEE Computer Society.

[13] W. Heaven and E. Letier. Simulating and optimising design decisions in quantitative goal models. In *Requirements Engineering Conference (RE), 2011 19th IEEE International*, pages 79 –88, 29 2011-sept. 2 2011.

[14] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *Software, IEEE*, 14(5):67 –74, sep/oct 1997.

[15] S. Kukkonen and J. Lampinen. Gde3: the third evolution step of generalized differential evolution. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 443 –450 Vol.1, sept. 2005.

[16] Frank Kursawe. A variant of evolution strategies for vector optimization. In *Parallel Problem Solving from Nature. 1st Workshop, PPSN I, volume 496 of Lecture Notes in Computer Science*, pages 193–197. Springer-Verlag, 1991.

[17] Boumedjout Lahcen and Leonard Kwuida. Lattice miner: A tool for concept lattice construction and exploration, 2010.

[18] Bo Liu, F.V. Fernandez, Qingfu Zhang, M. Pak, S. Sipahi, and G. Gielen. An enhanced moea/d-de and its application to multiobjective analog cell sizing. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1 –7, july 2010.

[19] Tim Menzies, Zach Milton, Burak Turhan, Bojan Cukic, Yue Jiang, and Aye Bener. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, 17:375–407, 2010. 10.1007/s10515-010-0069-5.

[20] Tim Menzies, Burak Turhan, Ayse Bener, Gregory Gay, Bojan Cukic, and Yue Jiang. Implications of ceiling effects in defect predictors. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, PROMISE '08, pages 47–54, New York, NY, USA, 2008. ACM.

[21] Zach Milton. Which: A stochastic bestfirst search learner. Master's thesis, West Virginia University, 2008.

[22] A. J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. J. Durillo, and A. Beham. Abyss: Adapting scatter search to multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 12(4):439 –457, aug. 2008.

[23] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.

المنارة للاستشارات

www.manaraa.com

[24] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2:221–248, 1994.

[25] M. Tanaka, H. Watanabe, Y. Furukawa, and T. Tanino. Ga-based decision support system for multicriteria optimization. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 2, pages 1556 –1561 vol.2, oct 1995.

[26] V. Veerappa and E. Letier. Understanding clusters of optimal solutions in multi-objective decision problems. In *Requirements Engineering Conference (RE), 2011 19th IEEE International*, pages 89 –98, 29 2011-sept. 2 2011.

[27] David A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.* PhD thesis, Air Force Institute OF Technology, 1999.

[28] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Series in Data Management Sys. Morgan Kaufmann, second edition, June 2005.

[29] Hongyu Zhang, Adam Nelson, and Tim Menzies. On the value of learning from defect dense components for software defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, PROMISE '10, pages 14:1–14:9, New York, NY, USA, 2010. ACM.

[30] Yuanyuan Zhang, Mark Harman, and S. Afshin Mansouri. The multi-objective next release problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1129–1137, New York, NY, USA, 2007. ACM.

[31] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evol. Comput.*, 8(2):173–195, June 2000.

[32] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical report, 2001.

[33] Eckart Zitzler and Lothar Thiele. An evolutionary algorithm for multiobjective optimization: The strength pareto approach, 1998.

المنارة للاستشارات

www.manaraa.com