Cross Trees: Visualizing Estimations using Decision Trees

Naveen Kumar Lekkalapudi
Cross Trees: Visualizing Estimations using Decision Trees

Naveen Kumar Lekkalapudi

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

Tim Menzies, Ph.D., Chair
Roy S Nutter, Jr., Ph.D.
Ray Morehead, MD.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2014

Keywords: FASTMAP, WHERE4, Data Mining, SBSE, Contrast set learning

© 2014 Naveen Kumar Lekkalapudi
Abstract

Cross Trees: Visualizing Estimations using Decision Trees

Naveen Kumar Lekkalapudi

Optimization has been the goal of almost every human thought and action. With the growing computational capabilities, solutions to problems are also exponentially increasing. With rising demand for data and analytics on this big data, solutions to problems are also multiplied. Although, strong statistical and analytical reasoning has helped narrow down many solutions into set of feasible applicable solutions; Experts also seek knowledge of how these solutions have improved and what decisions are needed to be taken for remaining solutions to improve. This problem persists in software engineering with managers and domain experts taking decisions deciding future course of the project.

This thesis proposes a method for optimizing solutions along with providing decisions that help improve a solution. Literature supports that landscape visualization gives inside details on data behavior. A method is proposed which takes benefit of visualizing data and improving solutions based on their position in landscape. CrossTrees are built by grouping data based on their similarities and generating contrast sets between these groups, which help experts to gain insight from knowledge. This thesis tests CrossTrees with two models which simulate software projects: POM3 and XOMO. Models are simulated with a widely used genetic algorithm NSGA-II, to generate set of highly optimized solutions. When compared to NSGA-II, CrossTrees recommendations are more effective and faster to generate. Further as problem complexity increases, NSGA-II takes (worst-case) polynomial time while CrossTrees runs in linear time.
Success is not only work of one person but also of very many talented people guiding and supporting through the tough endeavor. First off, I would like to thank Dr. Tim Menzies for guiding me through complications of research and also for being extremely supportive. This thesis couldn’t have been put together without his expertise and skills in data mining and machine learning. I would like to thank him for bearing with me, answering trivial questions whenever I was in need. Also, I would take this opportunity to thank Dr. Roy Nutter for advising me throughout the masters program, and pushing me into CyberWVU. I could not have started to build myself up without his patience and extreme support right from first month of my masters. I would also like to thank Dr. Morehead for being with me whenever I was need and also teaching databases which has proven to be one of many skills in recent demand. I appreciate the support of Dr. Jairus Hihn for his valuable suggestions and also funding me for completing this research. I should also mention the support of my talented colleague Dr. Joe Krall for providing tools that aided in proposing this thesis. Lastly and more importantly, I would like to thank my family and friends for being extremely supportive throughout highs and lows of research and motivating me to work harder and smarter.
Contents

1 Introduction 1
  1.1 Statement of Thesis .................................................. 5
  1.2 Contributions of the Thesis ........................................ 5
  1.3 Research Questions .................................................. 5
  1.4 Structure of the thesis .............................................. 6

2 Background 8
  2.1 Buzz of big data ..................................................... 8
  2.2 Classic Old Techniques ............................................. 11
    2.2.1 Linear Programming ............................................. 12
  2.3 Heuristic Techniques ............................................... 12
    2.3.1 Hill Climbing .................................................. 13
    2.3.2 Stochastic Algorithms Early Work: Simulated Annealing ....... 16
    2.3.3 Stochastic Algorithms Later Work: Genetic Algorithms ....... 19
    2.3.4 Stopping criteria .............................................. 25
    2.3.5 Time complexity of NSGA-II .................................. 26
    2.3.6 Why NSGA-II? .................................................. 27
  2.4 Summary ............................................................. 28

3 CrossTree (CT1): Big Trees 29
  3.1 Introduction ......................................................... 29
  3.2 Projecting data on 2D Plane ....................................... 30
  3.3 Clustering 2D Plane ................................................ 32
  3.4 Decision Tree of clusters and Contrast Sets ...................... 33
    3.4.1 Scoring .......................................................... 36
    3.4.2 Generating Contrast sets .................................... 37
  3.5 Time Complexity .................................................... 40
  3.6 Summary ............................................................. 41

4 CrossTree0 (CT0): Smaller Better Trees 42
  4.1 Problems of using CrossTree (CT1) ................................. 42
  4.2 Proposed Solution for issue of Big Trees ......................... 43
    4.2.1 Feature subset selection using Infogain ..................... 43
    4.2.2 Range reduction by entropy discretization .................. 43
# Model: XOMO

5.1 XOMO

5.2 Models inside XOMO

5.2.1 The COCOMO Effort Model

5.2.2 SCED-RISK model

5.2.3 COQUALMO

5.3 Implementing NSGA-II with XOMO

5.3.1 Parameters

5.3.2 Disadvantages of NSGA-II on XOMO

---

# Model: POM3

6.1 POM3

6.2 History

6.2.1 POM1

6.2.2 POM2

6.3 POM3 attributes

6.4 POM3

6.5 POM3 submodels

6.6 Implementing NSGA-II with POM3

6.6.1 Disadvantages of NSGA-II

---

# Experiments

7.1 Overview

7.2 Design and Specifications

7.3 How to read output

7.4 Experiment 1: CrossTrees on XOMO

7.5 Experiment 2: CrossTrees on POM3

7.6 Summary of Experiments

7.7 Overcoming problems of NSGA-II

7.7.1 Time for generations

7.7.2 Single point solutions

7.7.3 Provides Insight

---

# Threats to Validity

8.1 Internal Threats

8.2 External Threats

---

# Conclusion

9.1 Addressing Research Questions

9.2 Impact on landscape visualization and multi-objective optimization

9.3 Future work
# List of Figures

1.1 DIKW pyramid ................................................................. 2

2.1 Trends in Google 2004-2014 .............................................. 9
2.2 Future prediction of Data .................................................. 10
2.3 DIKW pyramid .............................................................. 10
2.4 Search Problems - Many Local Extrema .............................. 14
2.5 Simulated Annealing Example .......................................... 18
2.6 Generic Genetic Algorithm .............................................. 20
2.7 The NSGA-II Process ...................................................... 21
2.8 The NSGA-II Fast Sort ................................................... 22
2.9 The NSGA-II Generational Process .................................... 23
2.10 The NSGA-II Crowding Distance Operator ......................... 23
2.11 Python Code for Random Mutation ................................... 25
2.12 Why NSGA-II Overall Sayyad ......................................... 27
2.13 Principal component analysis ......................................... 28

3.1 Principal component analysis ........................................... 31
3.2 Splitting data with FASTMAP .......................................... 32
3.3 Recursing Spectrally in quadrants using WHERE4 Algorithm .... 33
3.4 Tree showing clustering of 500 samples into 24 groups. ........... 34
3.5 Sample Decision Tree of Iris Dataset ................................. 35
3.6 Generating Deltas between clusters using CrossTree .............. 39

4.1 Sample Decision Tree generated from CrossTree. (Sample CT0) .... 45
4.2 Sample Decision Tree generated from CrossTree0. (Sample CT0) .. 45

5.1 Sample Flight software ranges ......................................... 47
5.2 COCOMO: co-efficients tunings. Taken from [60] .................. 50
5.3 SCED-RISK: an example risk table. Taken from [60] ............. 51
5.4 SCED-RISK: the calculations ........................................... 52
5.5 SCED-RISK: the details .................................................. 53
5.6 COQUALMO: effort multipliers and defect introduction. Taken from [60] .. 54
5.7 COQUALMO: scale factors and defect introduction. Taken from [60] .... 54
5.8 COQUALMO: defects introduced. Taken from [60] ................ 54
5.9 COQUALMO: defect removal ......................................... 55
5.10 COQUALMO: defects added and removed ......................... 55
Chapter 1

Introduction

With the advent of cheaper computational costs, analyzing of big data has increased exponentially in last decade. The searches for the term itself shows the growing interest in last couple of years as shown in Figure 2.1 and Figure 2.2. In the market of big data, to get the wisdom from the data in large quantities, four steps are followed, also known as the DIKW pyramid [1] as shown in Figure 1.1. The problem with big data is it lacks true insight into knowledge being extracted. Analyzing zeta bytes of records will give thousands and millions of patterns which each have similar impact on the objectives proposed by end clients [8]. By lacking this step of analytics, this vast knowledge being extracted may not be transformed into wisdom.

Multi-Objective Evolutionary Algorithms (MOEA) try to optimize this solution space when multiple criteria are involved in making decisions [17]. In such a case, improvement in one objective can lead to decrease in performance of other objective. MOEAs are population based meta heuristic optimization algorithms that provide solutions for multiple objective problems along the pareto optimal front [17, 44]. Initial experiments by the Author suggested that complications in multi objective optimization restricted evolutionary algorithms in getting an insight of converting knowledge to wisdom. However, this problem can be overcomed by exploring vast solution space with novel techniques in data mining.

In this thesis mining of software data is proposed and an attempt is made in generating feasible
solutions from a large set of available solutions for multi-objective problems. Techniques from statistics and artificial intelligence are used in data mining to find small, yet relevant, patterns in large sets of data [26, 31]. Another major problem faced while generating optimized solutions is instead of producing a landscape of better solutions, several single point solutions are generated [17, 23, 43]. These rigid-better solutions might be surrounded by space of worse solutions. This restricts the end client to take decisions proposed precisely.

Visualization of solution spaces in form landscape can solve this problem by providing the end client with a group of better solution sets at peak of landscape than one solution point [11, 26, 78]. This thesis proposes data mining to generalize solutions and visualize them using decision trees. Visualization of data not only helps end clients but also assists busy businessmen/designers/engineers/managers to understand the behavior of decision variables and help them improve their objectives.
CrossTree uses the following steps to improve the solution space:

- **Projecting on 2D Plane.**
  Firstly, CrossTree projects instance space on two directions of greatest variability using FASTMAP heuristic, which is a linear approximation of PCA. Dimensionality reduction using Multi-variate Data Analysis, such as PCA (Principal Component Analysis), is the conventional approach for visualization [35].

- **Clustering 2D Plane.**
  WHERE4 clustering algorithm is applied on visualized data, which recursively partitions projected space by mean value of projected dimensions. The generated clusters are groups of data with closer dimension values.

- **Tree of Decisions.**
  A Classification And Regression Tree (CART) is built from the clustered data. For that decision tree learning, the ids of clusters found above are used as “classes” explored by CART.

- **Envy.**
  The principle of envy is then applied: each cluster identifies the near-by cluster where some performance score is better.

- **Contrast set learning.**
  For each leaf cluster \( C_1 \), CrossTrees builds a contrast set showing the difference between this cluster and the other one \( C_2 \) that it envies. The decision tree branches down to \( C_1,C_2 \) are inspected and the recommendation is generated: “if you are in \( C_1 \), then what you need to do is \( C_2-C_1 \)”

CrossTree also uses several, (0) WHERE4: recursive FASTMAP (1) range reduction (entropy discretization) (2) feature subset selection (entropy-based infogain), (3) contrast sets; (4) CART,
(5) tree reduction (reduce sub-trees with the same majority class down to one); (6) envy; (7) non-parametric statistical tests and other techniques for optimizing solution space even more. Initial experiments showed CrossTrees to have large decision trees which are hard to comprehend for busy business users. To reduce the size of decision trees a second variant of CrossTree(CT0) is proposed which uses range reduction and entropy based feature subset selection to reduce size of decision tree. The two versions of CrossTrees are:

- **CrossTree CT1** uses (0),(3),(4),(5),(6),(7)
- **CrossTree CT0** uses CT1 + (1),(2)

CrossTree has been tested on two software models:

- **XOMO**
  A software model that is adapted from COCOMO model which helps CrossTrees explore solution space. XOMO is a monte-carlo simulator that generates samples for the solution space and pronounced as x-o-mow [60]. Few contrainted adaptions of XOMO are used to experiment this thesis.

- **POM3**
  POM3 is based on Turner & Boehm model that simulates software projects designed by Port, Oklov and Menzies. POM is customized into few sub models on which this thesis is tested [46].

To test this method, the recommendations found by CrossTree from POM3 and XOMO are compared against a standard genetic algorithm NSGA-II for determining its efficiency. Our experiments show that CrossTree is not only does as good as NSGA-II but also (1) offers a succinct business-ready representation of the space of decisions; and (2) CrossTrees runs much faster than NSGA-II (linear time vs polynomial time). Hence, we can recommend CrossTree as an scalable optimizer that offer succinct explanations.
1.1 Statement of Thesis

The research work of this thesis is summarized in the following statement:

*If optimization tasks are implemented using data miners then the resulting system can provide concise visualizations, scale to larger problems, and offer succinct recommendations on how to change and improve a system.*

1.2 Contributions of the Thesis

The main contributions of this thesis are summarized as follows:

- Described the implementation of *CrossTree* as a method to explore solution space for optimization.
- Showed how *CART* Trees can be used to find contrast sets of solution spaces which assist business users to take decisions.
- Demonstrated that feature subset selection and range reduction helps in mapping better solution space.

1.3 Research Questions

The objective of this thesis is to evaluate whether optimizing tasks with data miners provide concise visualizations, can scale to large problems and offer succinct recommendations for busy end user.

For that purpose, here are few research questions (Q) and related hypothesis (H):

*Q1. Does applying data mining methods provide visualizations on how to improve a system?*

*H1. Data mining methods can be used to produce visualizations that recommend changes and improvements to busy human experts.*

*Q2. Does data mining methods scale to big problems?*
H2. Data mining methods can provide solutions faster than existing stochastic methods and hence can scale big problems more efficiently.

Q3. Can data mining methods provide flexibility in prescribed solutions?

H3. Instead of proposing rigid single solution for a problem, a range of solutions can be proposed using data mining methods.

Q4. Can concise visualizations affect performance compared to broad visualizations?

H3. Yes, concise readable visualizations does affect performance as they may not capture the required decisions for improvement. Whereas, broad visualizations may be due to over fitting of information that is irrelevant to end user.

1.4 Structure of the thesis

The rest of this thesis is organized as follows:

• Chapter 2 starts with the background of landscape visualization and work that has been pursued to determine how effective it is to learn from projected data. To to end of chapter, NSGA-II, one of the most widely used genetic algorithm is explained and parameters used in this thesis are also mentioned.

• Chapter 3 CrossTree (CT1) method is introduced in this chapter. FASTMAP, WHERE4, and other methods used in this thesis are explained in detail. Also, the parameters maintained for experiments are briefly covered.

• Chapter 4 CrossTree (CT0) variation of smaller decision trees is introduced in this chapter. Problems with having bigger decision trees is discussed. Also, this chapter shows how we can reduce size of decision tree using techniques like range reduction and infogain.

• Chapter 5 introduces model XOMO and its several variants. One of the most widely used genetic algorithm NSGA-II is applied on XOMO model and its results are discussed.
• Chapter 6 introduces model POM, its sub models and also adaption of the model. Results of applying genetic algorithm NSGA-II are also discussed.

• Chapter 7 explains the experiments performed to test performance of CrossTree. Results thus obtained are also explained in detail.

• Chapter 8 discusses the threats to this thesis.

• Finally, Chapter 9 summarizes the conclusions of this thesis and also the scope of future work.
Chapter 2

Background

This section starts with the motivation behind proposing this thesis and it then proceeds with an overview of broad range of decision support methods ranging from classical techniques to complex meta heuristic Techniques.

2.1 Buzz of big data

Big data has been a buzzword of industry for last couple of years [8]. Preliminary statistics on Google searches show the exponential rise of search for the term “big data”. By following the technology industry closely, the drift towards big data can be observed. Figure 2.1 and Figure 2.2 shows exponential increase of searches in 2012-2014. The numbers on the graph reflect how many searches have been done for a particular term, relative to the total number of searches done on Google search engine over time (source: www.google.com). Though this represents a portion of opinion on big data, it supports the initial claim of exponential increase in its popularity over last decade. Manyika [50], Boyd [9], Schönberger [52] have supported this claim in recent years.

Big data when analyzed with cheap computational capabilities of cloud computing solve real time problems with less cost and much faster than existing methods [25, 50, 52]. This association of big data with big CPU enables solutions that have previously been impossible. A recent study
indicated the use of cheap computation for automated bug fixing [45]. It was observed that 55 out of 105 bugs were fixed in 1.6 hours with a minimal cost of less than $8 dollars. In comparison manual bug fixing run for about 5 to 15 hours and cost from $25 to $500. [25], [50], [52], [32] propose several domains that could use this advantage of big data computations. We will see more widespread use of CPU-intensive analysis of big data in very recent future.

The DIKW pyramid shown in Figure 2.3 forms the base for big data analysis. The data is collected from any and possibly all sources is processed for relations to get information. This information is checked for patterns to get knowledge. Knowledge converts into wisdom by predicting the outcome formed from applying proposed knowledge [1]. The important step of analytics is handled in knowledge block of DIKW pyramid. Analytics can produce several such patterns which achieve different objectives of end clients. If there are multiple objectives to be met, conflicting objectives complicate the already large set of solution base [51]. A set of solutions along the pareto optimal front can be considered as the best way to tackle the problem [72]. Since, solutions along the pareto optimal front are also inform of single points, they may not fulfill the flexibility of end-user decisions. Therefore, instead of one best solution, an ideal method should produce a set of decisions having flexibility for end user to tune according to their model implementations and objectives.
Figure 2.2: Big data future in coming years. Source: www.reuters.com

Figure 2.3: The DIKW pyramid
Search and optimization technologies underpin the development of decision support systems in a wide variety of applications across industry, commerce, science and government [27, 31, 72]. There is a significant level of diversity among optimization and computational search applications. This can be evidenced by noting that a small selection of applications includes transport scheduling, bioinformatics optimization, personnel rostering, medical decision support and timetabling [13].

By designing more efficient decision support methodologies, few of the potential benefits are:

- More efficient production scheduling can lead to significant financial savings [13, 26, 76].
- Higher-quality personnel rosters lead to a more contented workforce [13, 47].
- Efficient health care scheduling will lead to faster treatment (potentially saving lives) [13].
- More effective cutting/packing systems can reduce waste [13].
- Better delivery schedules can reduce fuel emissions [13].
- Simplifying testing in large scale systems for defect predictions [27].
- Efficiently estimating quality of software at different levels of development [3, 74].

Although the Authors have preferred to use meta heuristic method for predictions, there has been significant research in olde classic techniques.

2.2 Classic Old Techniques

This section gives a brief insight on initial developments of optimization of solution space. Readers can get a good foundation on initial techniques used and the reasons for developing complex methods for reducing solutions efficiently.
2.2.1 Linear Programming

Linear programming (LP) is an optimization technique in mathematics that provides an global optimum solution. A set of \(x_1, \ldots, x_n\) real, non-negative values, called decision variables are given as input for LP. A set of constraints subject these decision variables in form of some linear expression. The goal is to maximize this linear expression which in most cases is expressed as linear equation.

That is maximizing the following equation:

\[
\sum_{i=1}^{n} c_i x_i
\]  

(2.1)

where \(c_1, \ldots, c_n\) is a set of problem-specific coefficients, subject to a set of \(m\) constraints. [31]

Linear programming demands clear cut determination of single objective to be optimized. Linear programming serves as a good optimization technique in software engineering if the problem can be formulated in well defined constraints represented by set of linear equations. Harman [31] claims that linear programming is typically applied to problems such as resource and plant allocation.

Linear Programming cannot handle large complex solution spaces [16, 31, 49]. This inspired development of methods to handle solutions that cannot be comprehended into human understandable mathematical equation.

2.3 Heuristic Techniques

This section gives an overview of few heuristic techniques used in optimization of solution space. There are several other methods such as Ant colony optimization [18], Stochastic Search [38], Bayesian Optimization [68], Scatter Search [28], Genetic and Evolutionary algorithms [31, 44] which specialize in dealing with complex models of solution spaces. In this section few of those methods are described and problems of using them are discussed in detail.
2.3.1 Hill Climbing

Hill climbing started as early as 1966 with maximizing quadratic functions in mathematics. Given a point in n-dimensional space corresponding to set of independent variables, most functions compute for a new point at which function is larger. The rationale behind this method is using gradients combined with steepest ascent and replacing the general iterative process of going through all points in space [29].

Hill climbing starts with a randomly chosen solution. With each iteration a set of near neighbors of current solution are determined. The distance computation to determine a neighbor is problem specific, but typically neighbors are only small difference away. Neighbor of current solution that is “better” than current solution is computed using a fitness function. A move is made toward this neighbor and these steps are iterated till there are no neighbors that are “better” than current solution [31, 64, 78]. This is an example of steepest ascent hill climbing which results in finding the local maxima of randomly chosen candidate solution as shown in Figure 2.4.

In 2002, Mitchell et al. [63] used the Bunch algorithm which is a modified version of Hill climbing method to extract design abstractions from source code. Bunch’s hill-climbing clustering algorithms starts by generating a random partition of the MDG (Millennium Development Goals). Modules from this partition are then rearranged systematically in an attempt to find an improved partition with a higher MQ (Modularization Quality). If a better partition is found, the process iterates, using the improved partition as the basis for finding even better partitions. This hill-climbing approach eventually converges when no additional partitions can be found with a higher MQ.

In 2014, Fuchshuber et al. [26] used landscape visualization to improve heuristics in Next Release Problem. The search starts by creating a solution based on a random selection of customers. After the initial selection of customers, the solution fitness is calculated according to Bagnall et al. The solution is stored as the best solution and the main loop of the search follows. The loop tries to find solutions with better fitness by adding or removing a single customer from the current
Figure 2.4: A search space problem with one objective and many local extrema, but only single global extrema. Taken from [44]
solution at a time. Whenever a solution with higher fitness is found, it becomes the new best known solution and the trials are repeated from the first customer. If no movement is able to improve fitness, a random restart is triggered. The search stops after reaching a predefined number of fitness evaluations. The modification proposed, named Visual Hill Climbing, differs from the classic Hill Climbing in two ways. First, a phase of random sampling is executed before the Hill Climbing starts. At this phase, several random solutions with S satisfied customers are generated, S ranging from one to the number of customers in the instance. The solution with the highest fitness becomes the best known solution and S is stored as a lower bound to the Hill Climbing algorithm. After the random sampling is ended, the Hill Climbing is executed with a restriction: it can only visit solutions whose number of satisfied customers is within the range \([S, S+N*a]\), where \(N\) is the number of customers in the instance and \(a \leq 1\) is a parameter to the algorithm. For example, if the best solution found during the random sampling has 15 satisfied customers and the instance has 100 customers, the modified Hill Climbing will consider only solutions whose number of customers is within the range \([15, 15 + 100*a]\). This restriction is used to make the algorithm concentrate in a region where, during the random sampling phase, solutions with high fitness values were found.

There are many cases where Hill climbing is used:

Initial stages of GA and HC development has seen both of them performing competitively. Royal road landscapes were used for comparison between Random mutation Hill Climbing (RMHC) and Idealized Genetic Algorithm (IGA), where latter outperformed significantly. [64]

A Sequential Hill climbing method is used by Kirsopp [40] by only adding features and removing them and by starting with no features instead of random selection of features. This method called to be FSS (Forward Sequential Selectors) proved to better than traditional hill climbing.

Another method which adapted modified Hill climbing is MPPT. Maximum power point tracking (MPPT) must usually be integrated with photo voltaic (PV) power systems so that the photo voltaic arrays are able to deliver maximum available power [80].

Bayesian Networks are frequently used to model decision support system particularly in Medicine. A Min-Max hill climbing algorithm is used by Tsamardinos et al. by directing algorithm towards
max score by addition, deletion or direction reversal [78].

Clearly, the problem with the hill climbing approach is that the hill located by the algorithm may be a local maxima, which may be far poorer than the global maxima in search space [31]. Hill climbing is modified to overcome this problem by restarting hill climbing at multiple random solution points. Despite this modification, hill climbing faces several problems:

1. What hill to climb? In multi-objective optimization this poses as a serious problem when identifying solutions on a pareto optimal front

2. The performance of hill climbing entirely rests on number of restarts, higher number of restarts ensures finding better global maxima; But this would take higher computational power and longer period of time in large landscapes of big data.

For these reasons, in the SE literature, hill climbing is far less used for search-based software engineering than the stochastic method discussed below. Hence in the experiments of this thesis, we benchmark our proposed approach against widely-used stochastic algorithms.

2.3.2 Stochastic Algorithms Early Work: Simulated Annealing

Simulated Annealing (SA) is a classic stochastic search algorithm which can be thought as a variation of hill climbing. It was first described in 1953 [62] and further optimized in 1983 [11, 39]. Simulated annealing is a simulation of metallurgical annealing. Annealing is a process where a highly heated metal is allowed to cool down slowly, thereby increasing its strength. As the temperature decreases the atoms have less freedom of movement and resting themselves in best position possible [31]. Fundamentally, SA is a hill climber: it starts in a random location and travels to higher-scoring locations in the immediate neighborhood. Standard hill climbers are prone to get stuck in local maxima. To avoid this, simulated annealing borrows a heuristic from its namesake, the metallurgy technique “annealing”. Similarly, Simulated Annealing will jump to sub-optimal solutions at a probability determined by the current state of the temperature function. At first, it will rapidly jump around the search space before finding stability [27].
Simulated annealing is designed by modifying hill climbing method to avoid local maxima problem. Simulated annealing starts as hill climbing by selecting solutions in the neighbourhood of candidate solution. Better solutions are always accepted along with worse solutions. The selection of worse solutions is controlled by slowly decreasing the number of worse solutions getting selected. This is analogous to annealing where solutions are allowed to “move” freely in the beginning (analogous to when metal is hot) and then worse solutions are selected in an controlled environment to get the performance (analogous to shape of metal) desired. The idea is to accept short term penalty for long term benefit of getting rid of local maxima and finding a global maxima. [11, 15, 31, 34]

Simulated annealing is general purpose method with algorithm remaining almost similar across several domains. The generic part of algorithm is not changed where as problem specific part of algorithm is modified according to desired constraints of problem.

There are several adaptions of simulated annealing across many domains:

- Detection of flaws in software programs can be automated using test data in controlled annenalling. [77]

- A simulated annealing-based heuristic has been developed to obtain the least-cost design of a looped water distribution network. The results obtained with this approach for networks currently appearing in the literature as case studies in this field (whose solution by other optimization methods was known) have proved the ability of the heuristic to handle this kind of problem [15].

- Simulated Annealing can be used for partitioning system software/hardware level partitioning. Partitioning is performed at the granularity of blocks, loops, subprograms, and processes with the objective of performance optimization with a limited hardware and software cost. One of the methods used is simulated annealing. [21]

- Simulated annealing can also be used in combination with Genetic Algorithms as described by Hwang, et al [34].
Figure 2.5: Demonstrating visual application of Simulated Annealing: A ball with velocity finds global minima. Taken from [44]

Figure 2.5 shows the process of simulated annealing, where a ball traverses through solution space freely to find global maxima and then cools down to find the global maxima. Although simulated annealing overcomes problems of hill climbing, it has its own disadvantages with multiple objectives:

1. How can a worse solution be determined when multiple objectives are defined?
2. Which neighbor should be selected when multiple goals are proposed?
3. Simulated Annealing (SA) doesn’t take advantage of memory available on new computers
4. SA is a stochastic technique and doesn’t provide any insight into data
5. SA produces single point solutions instead of proposing a range of solutions.

With advances in stochastic methods Genetic Algorithms were proposed to take advantage of memory available in new computers, and to find best solution for multi objective optimizations.
For example, we use NSGA-II, one of the most widely used genetic algorithm to benchmark the proposed method in the current thesis.

### 2.3.3 Stochastic Algorithms Later Work: Genetic Algorithms

Genetic algorithms dominated optimization in search space over last couple of years and is most widely applied search technique in SBSE [71]. Genetic algorithms populate samples and use several recombination techniques to generate new samples. An iterative process is executed, initialized by a randomly chosen population. The iterations are called generations and the members of the population are called chromosomes, because of their analogs in natural evolution. The process terminates when a population satisfies some pre-determined condition (or a certain number of generations have been exceeded). On each generation, some members of the population are recombined, crossing over elements of their chromosomes. A fraction of the offspring of this union are mutated and, from the offspring and the original population a selection process is used to determine the new population. Crucially, recombination and selection are guided by the fitness function; fitter chromosomes having a greater chance to be selected and recombined [31].

There are several variations to this process, but core procedure remains fairly same across all the methods. Variations can be in crossovers such as single point crossover, two-point crossover, uniform crossover and arithmetic crossover. Usually crossover is defined in form of a function on sample space defined by a constant specifying how much to crossover. Mutation is also defined in a similar way to mutate population randomly to generate new samples, this random mutation is however controlled by some function specific to each method.

The Genetic algorithms can be shown in an overall framework as depicted in Figure 2.6

**NSGA-II**

According to a recent literature review by Sayyad et al in 2013, [72] the current de facto standard for optimization of any solution space is reserved by NSGA-II. Non-dominated Sorting Genetic Algorithm, proposed by Kalyanmoy Deb et al [17, 73]. is a multiobjective evolutionary algorithm
Set generation number, m:= 0
Choose the initial population of candidate solutions, P(0)
Evaluate the fitness for each individual of P(0), F(Pi(0))

loop
Recombine: P(m) = R(P(m)) Mutate : P(m) = M(P(m)) Evaluate: F(P(m))
Select: P(m+1) = S(P m))

m = m + 1
exit when goal or stopping condition is satisfied
end loop;

Figure 2.6: Generic Genetic Algorithm

that uses non dominated sorting. An initial prototype was released in 1994-1995 which eliminates problems of scalarizing multiobjective problems by weights. User must require domain knowledge to build the weights for objectives and also solutions are highly sensitive to the weight vector. These problems were overcomed by using Goldberg’s notion of non dominated sorting in GA’s along with a niche and speciation method to find multiple Pareto-optimal points simultaneously rather than proposing single point solutions. NSGA-I [73] was one of the first evolutionary algorithms for solving non-convex and non-smooth multiobjective problems. This prototype was improvised into NSGA-II which was designed as a response to criticisms of NSGA-I.

The problems faced by NSGA-I can be summarized into:

- **High computational costs due to an \(O(MN^3)\) complexity (where \(M\) being number of objectives, and \(N\) being population size) with every generation making search very slow.** The complexity involved in every generation adds up to long computation time for large population sizes.

- **Elitism must be adapted to improve speeds** Good solutions must be prevented from loosing by using elitism

- **A sharing parameter necessity** Most mechanisms ensure diversity in the population by sharing. A sharing parameter is used to evaluate the population shared between old and new generations. Though dynamic sharing parameter is an option, parameter-less sharing is highly desired.
NSGA-II overcomes above mentioned disadvantages by computing in $O(MN^2)$ time, elitism is be preserved by considering the parent elements that are better than their child mutants by use of elitism operator, and parameter less crowding-distance function is provided to ensure diversity through density estimation (called niching) of the solutions and replaces the need for a sharing parameter.

The NSGA-II used in this thesis is similar to one developed by Krall [44]. The implementation specifics for this thesis are slightly different and are discussed towards the end of this subsection. The process of NSGA-II algorithm is explained briefly in the Figure 2.7

Initially a population $P_0$ is generated at random. This population is sorted using fast sorting operation into bands $band_0$, $band_1$, $band_2$, ..., $band_i$, where $band_i$ is always better than $band_{i+1}$. A sub selection of samples is selected from population to form $Q_0$ using a tournament selection method along with crossover and mutation. The size of population $Q_0$ does not even be equal to $P_0$. Construction of $Q_0$ is problem specific and can vary with implementation of NSGA-II.

It is assumed that the entire population is duplicated. Standard genetic operators (mutation and crossover) are used in that duplication to form the population $Q_0$. A tournament selection method also assists in drafting as many individuals from the population as desired.

$Q_0$ and $P_0$ are combined to form $R_0$. The fast sort of Figure 2.8 is then run on $R_0$ and the
fast-non-dominated-sort(P)

for each \( p \in P \)

\( S_p = \emptyset \)

\( n_p = 0 \)

for each \( q \in P \)

if \( (p \prec q) \) then

\( S_p = S_p \cup \{q\} \)

else if \( (q \prec p) \) then

\( n_p = n_p + 1 \)

if \( n_p = 0 \) then

\( F_{\text{rank}} = 1 \)

\( F_i = F_1 \cup \{p\} \)

\( i = 1 \)

while \( F_i \neq \emptyset \)

\( Q = \emptyset \)

for each \( p \in F_i \)

for each \( q \in S_p \)

\( n_q = n_q - 1 \)

if \( n_q = 0 \) then

\( g_{\text{rank}} = s + 1 \)

\( Q = Q \cup \{q\} \)

\( i = i + 1 \)

\( F_i = Q \)

If \( p \) dominates \( q \)

Add \( q \) to the set of solutions dominated by \( p \)

Increment the domination counter of \( p \)

\( p \) belongs to the first front

Initialize the front counter

Used to store the members of the next front

\( q \) belongs to the next front

Figure 2.8: The NSGA-II fast sorting process, taken from [17].
best members are used to fill the next population, $P_t$. In every subsequent generation, the same selection+crossover+mutation operations are applied to draft a $Q_t$ from $P_t$. The code of Figure 2.9 describes the process thereafter, beginning by combining $Q_t$ and $P_t$ to make $R_t$ [44].

To build $R_t$, a fast sort as shown in Figure 2.8 is used. A crowding distance operator shown in Figure 2.10 is used to preserve elitism of previous generations and also diversity of future generations represented by $P_{t+1}$. The crowding distance measures the distance between points of same front and is then sorted so that the least crowded and most domination solutions are selected. To end the generation, these selected solutions fill the population of $P_{t+1}$. This process is repeated until the stopping criteria is met indicating the termination of process.
The crossover and mutation operators have parameters are wrapped in an evolutionary framework as follows (Taken directly from [44]):

1. Build initial population, $P_0$. Initialize the generation number: $t = 0$.

2. Repeat until stopping criteria is met:

   (a) Select $R_t = P_t$.

   (b) Perform crossover($cx = 0.9$) and mutation($mx = 0.1, eta = 1.0$) operators on $R_t$.

   (c) Run the fast sort and crowding distance functions of NSGA-II to combing $R_t$ and $P_t$ into $P_{t+1}$.

   (d) Increment generation number $t = t + 1$.

   (e) Collect Stats and evaluate stopping criteria

3. Final reporting mechanisms

   The implementation proposed is no different than original NSGA-II as first proposed, except small modifications mentioned below.

   Crossover works as follows: Every even-indexed element is paired with ever odd-indexed element, and at the chance $cx = 0.9$ (90%), some elements might be swapped. The secondary chance depends on the number of decision attributes of each element, and is defined parameter less as $1.0/D$ where $D$ is the number of decision attributes. In this manner, about half of the attributes are expected to be swapped [44]. Crossover is applied to pairs of solutions.

   Mutation is applied to every solution independently. The implementation of mutation in this thesis is similar to original proposal by Deb [24] The eta parameter mentions how similar mutant will be to its parent, higher the eta means closer resemblance. The parameter $eta$ describes chances by which a solution can be passed through a mutation operator. In this thesis all solutions pass through mutation operator indicating $eta = 1.0$. The $mx = 0.1$ indicates the chances by which a solution’s attribute can be mutated. About 10% of the time, attributes of solutions are modified with code of Figure 2.11.
Figure 2.11: Python Code for Random Mutation (as used by NSGA-II). Taken from [44]

One aspect of evolutionary algorithms are they never stop searching solution spaces. An stopping criteria must be specified for individual evolutionary algorithm depending upon the implementation. The stopping criteria for current thesis is mentioned in section 2.3.4

2.3.4 Stopping criteria

An multiobjective optimization evolutionary algorithm (MOEA) runs forever until a termination criteria is designed to stop it. Typically, a number of evaluations are used (stop after 100000 evaluations are made). In the current thesis, along with number of evaluations, a maximum number of generations are used as outbound limit on stopping criteria.

In [48] and [4], the stopping criteria is specified such that the results are collected after 250 generations whereas in [12] 1000 generations are used. However, while using maximum number of generations as criteria of termination the algorithm burdens with extra cost of computation even after the optimal solutions are found. Also, number of generations is not a pragmatic way of stopping an algorithm.

There are several methods mentioned for appropriate stopping criteria like variance-based approach in [5]. Also, Marti et al. specify a method to measure improvement of solutions after each
generations and stop when improvement is small. These methods require tracking of the best-seen objective scores throughout search. The current thesis uses bstop(\(\lambda\)) which is the same as used by Krall [44]. Bstop(\(\lambda\)) is inspired from works of Bhandari [5] and Marti [51]. The best seen objective scores of all encountered candidates are tracked. When no objectives are improved, the algorithm begins to loose chances(\(\lambda = \lambda - 1\)) to continue.

Searches never loose chances due to very tiny improvements consistently being found. In experiments done by Krall [44], 20 generations is rarely met, and in those rare cases, the improvements found were good, indicating that the stopping criteria was not premature. This novel approach of stopping criteria is coupled with maximum number of generations to enable a hard-stop was proposed by Krall [44] and is also used in this thesis.

### 2.3.5 Time complexity of NSGA-II

This section calculates the time complexity of NSGA-II algorithm. Time complexity of one iteration of entire algorithm as shown in 2.9 can be explained as follows:

1. Non-dominated sorting has a complexity of \(O(M(2N)^2)\), with \(M\) being objective vector size and population size of \(N\). Each domination check requires \(M\) value comparisons, the maximum complexity of this approach is \(O(MN^2)\) [17].

2. Crowding distance assignment does \(M\) independent sorts of at most \(N\) solutions. This accounts to a time complexity of \(O(MN\log N)\).

3. Crowding sort algorithm sorts in \(O(2N\log(2N))\).

As it can be seen, complexity of overall algorithm is \(O(MN^2)\), which is governed by the non-dominated sort.
2.3.6 Why NSGA-II?

This thesis benchmarks proposed method CrossTrees (explained in following chapters) against NSGA-II. Why?

Sayyad et al [71] conducted a literature survey on Search Based Software Engineering papers that used multi-objective search to find Pareto-optimal solutions. NSGA-II was compared with other multiobjective evolutionary algorithms like Multi-Objective Generic Algorithm (MOGA), Pareto Archived Evolution Strategy (PAES), Multi-Objective Cellular algorithm (MOCell), and Strength Pareto Evolutionary Algorithm (SPEA2). It was observed that NSGA-II was used as single algorithm 53% of 36 papers and also in majority of multi objective papers. Figure 2.12 shows the frequency of algorithms used in both single algorithm papers and also multi algorithm papers. Figure 2.13 shows frequency of NSGA-II dominating other algorithms when only one algorithm was used in a paper. Sayyad et al [71] also mentions 42% of time no reason was mentioned why an algorithm is chosen. NSGA-II is therefore established to be one of most widely used multi-objective evolutionary algorithm.
2.4 Summary

While stochastic processing is useful for avoiding the limitations of simplistic hill climbing, the algorithms described above (SA, GA, NSGA-II) all suffer from the same problems:

- **Runtimes**: Undirected mutation can thoroughly explore a space but can waste time exploring unfruitful directions. As shown below, with little data mining, we can build directed mutators in far less time. In case big data, chances of exploring exponentially increases with increase of size of data.

- **Point Solutions**: Algorithms like SA, GA, NSGA-II produce point solutions rather than ranges. Ranges give flexibility to experts on implementing solutions and not constraining with one value for each decision variable.

- **Insight into data**: Since, stochastic algorithms are based on random generations, it is highly complicated for a business user to glance at solution and Figure out decisions needed to be made.

As we shall see below, CrossTrees provides all these three things, while taking less time than standard optimizers such as NSGA-II.
Chapter 3

CrossTree (CT1): Big Trees

This chapter proposes current method “CrossTree”. CrossTrees implementation is explained in detail and also complications of using big trees is introduced. This chapter starts with explaining how initial data is projected, followed by how projected space can be spectrally clustered and ending with explaining how contrast sets are generated from clusters.

3.1 Introduction

In summary, CrossTrees generate decisions to help solutions jump from lower “worse” branch to higher “better” branch. The data set is initially projected on a 2D-plane, which is used to cluster according to how close they are to each other. These clusters are used to build decision trees whose branches help data to cross from worse to better.

Initially, the data set of solutions is represented by \( \{ S_n \} \subset \mathbb{R}^d \) of n points, where d is the number of decisions. So each candidate solution can be represented as d decisions \( D_{1i}, D_{2i}, \ldots, D_{di} \) where i is individual solution in \( S_{1..i..n} \). In order to dive deep into solutions and assess them intricately the huge \( \{ S_n \} \) dataset must be reduced into groups of similar solutions. CrossTree uses a fast spectral learning method WHERE [54] to group similar solutions together which is explained in section 3.3.
3.2 Projecting data on 2D Plane

Spectral learners assist in perceiving behavior of data by reducing dimensions. Principal component analysis (PCA) is one such widely used linear dimension reduction technique which is based on orthogonal transformation of d dimensions into e eigenvectors [75]. Figure 3.1 explains principal component analysis, the solution space can be reduced to orthogonal dimensions without affecting the properties of data [36, 75].

In the current technique, WHERE an adoption of PCA is used. WHERE is a linear time dimensionality reducer which assumes that the dimensions of most interest are the dimensions of greatest variability [54]. WHERE uses FASTMAP heuristic Figure 3.2 to find the eigen vectors or greatest varying dimensions. FASTMAP heuristic plots all solutions using 2 distant points, of which one is chosen at random and the other is farthest solution from it. FASTMAP heuristic projects data on a 2D plane by calculating coordinates for each solution with respect to line between two distant points (poles). One coordinate is where the solutions projects itself on the line and the other coordinate is the distance of solution from that line. These two measures act as x and y coordinate on the 2D plane with \{(X_i, Y_i) \forall S_i \in S_{1..i..n}\}. Refer to Figure 3.2

The distance between 2 rows is calculated using standard euclidean distance between independent decisions as proposed by David Aha; that is, \(dist(S_i, S_j) = \sqrt{\sum_{k=0}^{d} (D_{ki} - D_{kj})^2}\). All the decision values \(D_1, D_2, ..., D_n\) are normalized according to their respective values across all solutions [2]. This distance is correlated to properties of each solution with distance being directly proportional to differences between solutions. In summary, more different the solutions are, more distance there is between them. High correlation between properties of data ensures validity of distance [2].
Figure 3.1: Principal Component Analysis (PCA) uses principal components of data rather than a normal xy-plane. Neither X nor Y plane captures the underlying structure of data [75]. PCA captures essence of data using eigen vectors. Eigen vectors preserve their direction even when data is transformed. Taking advantage of this feature of eigen vectors, PCA preserves properties of data while transforming it into smaller dimensions. In above figure, to capture underlying structure of projected coordinates in the xy-plane, two orthogonal eigen vectors form the new xy-plane. This new plane captures underlying properties and are used for representing data. In summary, Principal component Analysis (PCA) uses eigen vectors to reduce dimensions to orthogonal parameters.

3.3 Clustering 2D Plane

The dataset of solutions \(\{S_n\}\) and their xy coordinates \(\{(X_i, y_i), \; \forall \; i \; \text{in} \; 0..n\}\) are passed to the WHERE4 algorithm which is an adaptation of WHERE algorithm and is more similar to FASTMAP proposed earlier by Menzies et al [37, 54]. WHERE4 also takes \(MIN\) and \(MAX\) values which define the cluster size. As shown in Figure 3.3, the mean of x and y coordinates \(\{\hat{X}, \hat{Y}\}\) of all solutions is calculated. Depending on the \((X,Y)\) coordinate of each solution, they fall into one of \(\{S_{ll}, S_{lh}, S_{hl}, S_{hh}\}\) with \(\{\hat{X}, \hat{Y}\}\) being center of that quadrant according to following condition.

- \(S_{ll}\): \(X_i < \hat{X}\) and \(Y_i < \hat{Y}\)
- \(S_{lh}\): \(X_i < \hat{X}\) and \(Y_i > \hat{Y}\)
- \(S_{hl}\): \(X_i > \hat{X}\) and \(Y_i < \hat{Y}\)
- \(S_{hh}\): \(X_i > \hat{X}\) and \(Y_i > \hat{Y}\)

```python
def FASTMAP(x):
    "Project data on a line between 2 distant points"
    z = random.choose(x)
    east = furthest(z, x)
    west = furthest(east, x)
    x.poles = (west, east)
    c = dist(west, east)
    for one in x.members:
        one.X, one.Y = project(west, east, c, one)
    return x  # data members updated with (X,Y)

def project(west, east, c, x):
    "Project x onto line east to west"
    a = dist(x, west)
    b = dist(x, east)
    X = (a*a + c*c - b*b)/(2*c),  # cosine rule
    Y = (a*a - X)**0.5  # distance to line b/w poles
    return X, Y

def furthest(x, x):
    "What is furthest from x?"
    out, max = x, 0
    for y in x:
        d = dist(x, y)
        if d > max: out, max = y, d
    return out
```

Figure 3.2: Splitting data with FASTMAP
WHERE4 is a recursive algorithm which runs on each of these solution spaces \( \{S_{ll}, S_{lh}, S_{hl}, S_{hh}\} \) to keep yielding clusters out when stopping criteria are met. WHERE4 uses MIN and MAX as stopping criteria by yielding out set of solutions when \( MIN < n < MAX \). After several experiments, \( MIN \) is set to be \( \sqrt{n} \) and \( MAX \) to be double the \( \sqrt{n} \) as they yielded best clusters compared to others. The terms “center” and “quadrant” are used to assist reader understand complex methods described, they may or may not take shape of graph with quadrants as data is non-orthogonally projected. WHERE4 returns \( m \) clusters of size in range \((MIN, MAX)\) represented by \( C_1, C_2, ..., C_m \) which are used to build decision trees to help assist CrossTrees. Figure 3.4 shows the output of WHERE4 algorithm with 500 samples clustered into 24 groups. Each of these clusters are represented with ids from “_1”, “_2”, .., “_24”.

3.4 Decision Tree of clusters and Contrast Sets

Decision trees are used for classification and regression of data to help learn simple decision rules inferred from large solution set. Decision trees can be visualized, they are simple to understand and interpret [42, 69, 70]. They can be used to get ranges of decisions that yield better solution instead of just one decision instance that produces high performing solution. CrossTrees build a simple Classification and Regression Tree based on entropies of decisions.

```python
def WHERE4(Sn, min, max):
    "Recursive cluster quadrants of data"
    if Sn.length > min:
        if Sn.length < max:
            yield Sn
            Xcap, Ycap = mean(Sn.Xs, Sn.Ys)
            for one in Sn:
                Sll += one if one.X < Xcap and one.Y < Ycap
                Slh += one if one.X < Xcap and one.Y > Ycap
                Shl += one if one.X > Xcap and one.Y < Ycap
                Shh += one if one.X > Xcap and one.Y > Ycap
                for S in (Sll, Slh, Shl, Shh): WHERE4(S, min, max)
        else:
            yield Sn
```

Figure 3.3: Recursing Spectrally in quadrants using WHERE4 Algorithm
There are several factors which contribute to the building of a decision tree. The following factors are the ones that affected CrossTrees and should be tuned according to the model before implementing CrossTrees:

1. *Entropy* is used as criteria for building decision trees which proved to be consistent in giving high performance across most models.

2. *Minimum samples per leaf* defines the number of samples allowed for leaves in decision tree.

```
1 500
2 |--147
3 |--|--35
4 |--|--29
5 |--|--29
6 |--|--54
7 |--|--|--13
8 |--|--|--12
9 |--|--|--10
10 |--|--|--19
11 |--100
12 |--|--28
13 |--|--12
14 |--|--35
15 |--|--25
16 |--90
17 |--|--32
18 |--|--24
19 |--|--8
20 |--|--26
21 |--163
22 |--|--60
23 |--|--|--14
24 |--|--|--20
25 |--|--|--15
26 |--|--|--11
27 |--|--|--33
28 |--|--41
29 |--|--29
```

Figure 3.4: Tree showing clustering of 500 samples into 24 groups.
3. Max depth restricts the recursive branching and specifies the number of levels allowed.

CrossTrees use clusters $C_m$ clusters to build decision tree by taking all decisions $D_d$ as data and predict for cluster id. CrossTrees uses ids of obtained clusters as “classes” for building decision tree as represented in Figure 3.5. In the Figure 3.5, nodes of decision tree are $petallength$, $petalwidth$, $sepalwidth$, $petalwidth$, $sepallength$ and leaves of tree are “1”, “2”, “3”, “4”,..., “n” are cluster ids. Number of samples represents samples present in that leaf of tree; and branch id represents different branches of tree.

As shown in Figure 3.6, CrossTree starts with building FASTMAP heuristics to compute $(X_n, Y_n)$ coordinates for all solutions of set $S_n$. These coordinates along with solution set are passed to WHERE4 algorithms to recurse in quadrants to generate clusters having spatially closer solutions. A simple decision tree is built using standard Classification and Regression Tree (CART) based on entropy as explained above which returns branches. The cluster id shown in a branch represents the majority of samples have that cluster id. For example, in Figure 3.5 “4” in branch id 0 shows that class “4” occurs maximum number of times in its 28 samples.

CrossTree prunes solutions here by using following two methods:

- Prune branches with leaves having more than 1 majority cluster.
- Combine branches together with leaves having same majority cluster.

```plaintext
$petallength <= 4.2 samples = 150
| |- $petalwidth <= 0.4 samples = 73
| |- $sepalwidth <= 3.5 samples = 48
| | | $[4\] \# samples = 28 \# branch_id = 0
| | | $[1\] \# samples = 20 \# branch_id = 1
| | $[7\] \# samples = 25 \# branch_id = 2
| $petallength <= 5.1 samples = 77
| |- $sepallength <= 6.1 samples = 43
| | $[8\] \# samples = 21 \# branch_id = 3
| | $[9\] \# samples = 22 \# branch_id = 4
| $[12\] \# samples = 34 \# branch_id = 5
```

Figure 3.5: Sample Decision Tree of Iris Dataset

35
A special SCORE is calculated for each cluster depending on the objectives of model as explained in following section.

### 3.4.1 Scoring

Scoring can be tuned according to the model being used, experts opinion and also according to requirements of end user. In CrossTree, a sample scoring is used to assess the performance. Two Non-parametric statistical tests are used as a part of this scoring process which is explained below.

Before determining the score of each cluster, the objectives of cluster are extracted to check if they are statistically different from each other. Initially, CrossTree uses non parametric A12 statistical test, which is a small non-parametric effect size measure developed by McGraw and Wong [53] and later improved by Vargha and Delaney [79] (also called Vargha and Delaney’s “A” measure). Given a performance measure M seen in “a” measures of A and “b” measures of B, the A12 statistics measures the probability that running algorithm A yields higher M values than running another algorithm B. CrossTree takes A12 measures greater than 0.6 for being statistically enough to be different. CrossTree proceeds with scoring if A12 determines two clusters to be different.

To improve accuracy, Bootstrap, a secondary non parametric statistical measure is used. This test is conducted to test if results of A12 statistical test hold true to determine if two populations are statistically significantly different. Bootstrap [20] is developed by Bradley Efron and is based on running test statistics(comparing mean) on original population and sampled population to determine whether two populations are similar or different. Predefined number of samples are used for resampling of original population while testing with bootstrap in CrossTree. These statistic measures return whether two populations are similar or different.

A classic domination technique is used for scoring after the tests determine that two populations are statistically significantly different. Scoring starts by exhaustively comparing each cluster with every other cluster to calculate “scores”. For multi-objective models, each objective is evaluated individually for scoring. Each objective is individually extracted to build populations and run
through statistical tests. The scoring proceeds when statistical tests determine populations to be different. In case of tests indicating the populations to be same, a “same” measure is used to keep track of nor better nor worse objective populations. Scoring is based on means of populations of each objective across two different clusters. If median of one of the objective of cluster $C_r$ is increasing over mean of other cluster $C_s$ then “better” measure is incremented, else “worse” measure is incremented. After evaluating all objectives, if atleast one objective is getting better with no other objective getting worse, then “score” of that cluster is incremented. The first $\sqrt{m}$ clusters form better set of clusters $B_a$ and rest form worse set of clusters $W_b$, where $m$ is number of clusters and also $\{a + b = m\}$.

3.4.2 Generating Contrast sets

Contrast sets are decision rules that are to be applied to solution sets to improve performance. Every solution in every cluster of Worse set $\{W_b\}$ is iterated according to its position in decision tree to find a nearest branch having better cluster “B” than current cluster “W”. The changes required for current solution to get to better cluster is determined by backtracking along branches of decision tree towards branch having better cluster. Branch having better cluster is determined by visiting neighbor branches on both sides of tree simultaneously to check for any clusters in better $B_a$ group.

After determining better branch a contrast set is generated with differences between decision variables of two branches. Also, $(\text{min, max})$ limits of each decision variable of better cluster are added to contrast set. If there is a conflict between $(\text{min, max})$ limit and previous contrast set, intersection between two set becomes final decision in contrast set.

For each worse solution in clusters $W_1, W_2, \ldots, W_b$ a contrast set $CS_1, CS_2, \ldots$ is calculated which can be applied back to themselves for improving objectives. For example, In Decision Tree shown in Figure 3.5, lets consider _1 is better _12. Then for solutions in _12 cluster to get to better _1, these set of decisions must be applied:
• petallength ≤ 4.2
• petalwidth ≤ 0.4
• septalwidth > 3.5

In this way, CrossTree overcomes single point problem of Stochastic algorithm by using ranges instead of single points. The contrast sets generated in form of ranges helping experts to apply them effectively rather than specifying single point solutions. Also, since the contrast sets are local to cluster, different sets of solutions get different contrast sets supporting the claim of locality [54]. These solution specific decisions can help experts meet end user requirements by solving problems local to section of solution space. The output of CrossTree is therefore a set of decision rules to be applied for each solution \( \forall S_i \in S_n, S_i \in one\{Wb\} \) that can improve its objectives.

Figure 3.6 explains the algorithm in Detail. Initially minimum and maximum size of clusters are determined by the size of input population size. Then FASTMAP algorithm shown in Figure 3.2 gives the X,Y coordinates when projected on a 2D plane as explained before. The WHERE4 algorithm explained in Figure 3.3 divides the solution set \( S_n \) by spectrally recursing in four quadrants returning \( C_m \) clusters, where m is number of clusters. The score algorithm described above computes the sample score of clusters and divides them into better and worse clusters.

Scikit-learn’s [67] CART algorithm is used with criteria of division being entropy and each branch of tree is maintained. Then there are 2 cases for CrossTree to deal with, the clusters in best group and clusters in worse group. New populations are generated for all clusters in both the groups but the constraints that define new population depend on which group cluster belongs to. For best clusters, \((min, max)\) limits are used by Model to generate new population Better\( S_n \). And for the worse clusters, a nearest branch is determined by traversing the CART decision tree in both directions simultaneously. When a branch is found having any of better clusters, that branch is determined to be better branch BB\( \text{Branch} \). A contrast set Contrast\( \text{Set} \) is generated using the branch cluster belongs to W\( \text{Branch} \) and new determined better branch for that cluster BB\( \text{Branch} \). This contrast set is intersected with \((min, max)\) limits of \( B \) which is better cluster in that BB\( \text{Branch} \).
Contrast set acts as constraints for the new population to be generated $ContrastS_n$.

The two newly generated populations are maintained a size proportionally to original population by $size(C) * 100$. The constant 100 is determined using different constants and trading off between time taken to generate new populations. It was observed from experiments of the Authors that this constant has little to no effect on performance of CrossTree. It should be kept in mind that the models generate dynamic data and its not static dataset. This population adheres to the contraints of $Limits$ for better clusters and $ContrastSet$ for worse clusters. Finally the results are

```python
def CrossTree(Sn):
    "Tree of clusters and results from their differences"
    min = Sn.length ** 0.5, max = min + min
    Sn = FASTMAP(Sn)  #Calculate $(X,Y)$ of solutions
    for C in WHERE4(Sn, min, max):  #Accumulate Clusters
        Cm += C
        BetterC, WorseC = score(Cm)  #Score clusters
    #Get Branches from DecisionTree
    Branches = cart(Cm)
    for B in BetterC:
        BetterSn = model(name=[XOMO*, POM3*],
                          PopSize=size(B) * 500,
                          MinMaxLimits(B))
        Results.append(CollectResults(BetterSn))
    for W in WorseC:
        #Branch to which clusters belong
        WBranch = Branches.ID(W)
        #Branch to which better clusters belong to
        B, BBranch = Branches.nearest(WBranch,  # Worse Branch
                                       W,  # Worse Cluster
                                       BetterC)  # Set of Better clusters

        #Find differences b/w worse and better branch
        ContrastSet = Branches.difference(WBranch, BBranch)

        #Add (min,max) limits to contrast set
        ContrastSet = ContrastSet ∩ MinMaxLimits(B)

        #Generate new population from model using
        #Contrast set
        ContrastSn = model(name=[XOMO*, POM3*],
                            PopSize=size(C) * 100,
                            ContrastSet)
        Results.append(CollectResults(ContrastSn))
    return Results
```

Figure 3.6: Generating Deltas between clusters using CrossTree
collected which can be mean or median as explained in experiments section 7.3.

3.5 Time Complexity

Time complexity of an algorithm quantifies time taken by an algorithm in a function of parameters of input. Asymptotic upper bound is most common notation used to denote worst case complexity of an algorithm [41]. It is represented as big oh order of a function, \( O(f(n)) \), where \( f(n) \) is a function with \( n \) being a parameter representing input.

Input to CrossTrees is represented as \((m \times n)\) where \( m \) is number of features or decision variables and \( n \) is number of such decision variables given (population size). To calculate worst case time complexity, CrossTrees can be divided into three modules:

1. **FASTMAP**: FASTMAP reads input data once and projects it on 2D plane. Hence, worst case to read population of size \( n \) is \( f_1(n) = O(n) \). In current thesis, input data is stored as a list of Python language, with time complexity of \( O(n) \).

2. **WHERE4**: WHERE4 reads input data and divides input data into clusters based on medians of \((x,y)\) coordinates of data at each level. Data needs to be sorted to find a median. Complexity to sort accounts to \( O(n\log(n)) \). WHERE4 is then applied on each of 4 parts\((ll, lh, hl, and hh)\) as shown in Figures 3.4 and 3.3. The complexity of WHERE4 is \( f_2(n) = O(m \times n\log(n) \times \log_4(n)) \), where \( m \) is number of times WHERE4 is applied. Since, \( m \) is a constant, time complexity for WHERE4 is \( f_2(n) = O(n\log(n) \times \log_4(n)) \).

3. **Contrast Sets**: Contrast sets between any two clusters \( C_1 \) and \( C_2 \) is a query of binary decision tree. Using theory of computing, time complexity for a query on binary search tree is calculated as \( f_3(n) = O(\log n) \).

Total time complexity of CrossTrees can be calculated as \( O(f(n)) \), where \( f(n) = f_1(n) + f_2(n) + f_3(n) \).
\[ O(f(n)) = O(n) + O(n \log(n) \cdot \log_4 n) + O(\log n) = O(n + n \log(n) \cdot \log_4(n) + \log(n)) = O(n) \]

Therefore, CrossTrees require a worst case time complexity of \( O(n) \), which is a function in linear time.

### 3.6 Summary

CrossTrees is a linear time method that provides insight into data using decision trees. Initial experiments by the Author indicated that size of decision trees increases with increasing population size. This contradicts the motivation behind this thesis to generate readable insight for busy business experts. In the next chapter, an enhanced version of CrossTree is proposed which utilizes entropy based range reduction and information gain to prune data. It is discussed whether smaller trees generated by using those techniques are beneficial to aim of this thesis or contradicts its purpose.
Chapter 4

CrossTree0 (CT0): Smaller Better Trees

This chapter addresses problems associated with using CrossTree (CT1) directly. Chapter starts with identification of problems with using CrossTree and proceeds to propose modifications for those problems. Later, complications involved in proposed modifications are then explained and concerns regarding smaller trees are addressed.

4.1 Problems of using CrossTree (CT1)

After initial implementation of CrossTree, the Author identified few key issues in CrossTree. One major problem identified was the size of decision tree. Since decision tree serves as medium for experts to identify and explore local solution spaces and finalize decisions accordingly. The complexity of understanding huge decision trees pushed initial motivation of designing a readable solution into jeopardy. Figure 4.1 show a sample run of CrossTree on Flight model of XOMO with population of 500 (XOMO model is explained in detail in next chapter).

Computationally it is possible to generate differences between any solutions in clusters of such big trees (Refer chapter 7). But CrossTree faces the same problem of stochastic methods by not providing a readable means for a busy expert who can only spare a glance to understand the solution space. This affects purpose of CrossTree, hence novel techniques described in following
section are used to prune big trees into smaller comprehensible trees.

4.2 Proposed Solution for issue of Big Trees

This section describes two methods used by CrossTree to overcome the issue of big trees. These data mining methods act as directed mutations by pruning irrelevant decision variables that do not contribute towards better solutions.

4.2.1 Feature subset selection using Infogain

Pruning features or decision variables of data helps in reducing solution space in terms of dimensions involved [30, 33, 81]. By directing CrossTree towards better solutions, the number of decisions required can also be reduced significantly. The criteria for pruning decision variables is problem specific [30, 33, 81]. In this thesis, an entropy based feature subset selection is used. Infogain acts as the criteria for pruning decision variables in CrossTree. Infogain is based on entropy of decision variables in data. Higher entropy results in lower infogain and has more probability of getting pruned.

CrossTree reduces datasets to fewer dimensions having higher infogain. Infogain is calculated by normalizing entropy measure for each decision variable. The percentage of decisions pruned using this technique is tuned according to specific model. Several experiments proved that CrossTree was effective after atleast 25 percent of total number of decision variables are be pruned.

In order to further improve initial solution space, data is discretized before CrossTree projects on a 2D plane as explained in following section.

4.2.2 Range reduction by entropy discretization

CrossTree also uses discretization to improve objectives of models and also reduce decision tree size. Before parsing the dataset through CrossTree, data is discretized with an entropy based
discretizer inspired from Fayyad and Irani [22]. This supervised algorithm uses entropy values of objectives to generate bins having least entropy heuristic. Entropy heuristic is defined in following measure [19].

\[ E = \frac{|S_1| \times e_1}{S} + \frac{|S_2| \times e_2}{S} \] (4.1)

CrossTree converts continuous values into ordinal values by using discretized bin identifiers instead of original values. These bins are formed by considering partitions that give least measure of above heuristic in Figure 4.1. A recursive algorithm is applied on individual decision variables to determine best possible bins with minimum entropy. Original values in initial data is then replaced with bin ids which act as ordinal values. Then regular steps of CrossTree are resumed.

### 4.2.3 Readability and Performance

By using above techniques in CrossTree there was significant reduction in size of decision trees. Sample decision tree shown in Figure 4.2

*CrossTree0 (CT0)* represents this version of CrossTree with discretization and infogain and *CrossTree1 (CT1)* represents initially proposed CrossTree with big decision trees.

Figure 4.2 shows that tree is comprehensible for a busy reader. It also provides insight into data with just a glance, supporting the motivation for this thesis. It can be observed that decision variables are contrainted on bin ids instead of actual values. When contrast sets are generated \((min, max)\) limits of actual values present inside bins are used. *In case of a floating point value, a min limit indicates floor value of bin id mentioned and a max value indicates ceil value of bin id mentioned is used.*

CrossTree0 (Small trees) can enhance performance or worsen it, CrossTree1 (Big Trees) on other hand can lead to overfitting by learning more from unfruitful information. To test and benchmark CrossTree1 (CT1) and CrossTree0 (CT0), two models are used: XOMO and POM3, which are explained in detail in following chapters.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Samples</th>
<th>Branch ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{plex}$</td>
<td>16.0</td>
<td>0</td>
</tr>
<tr>
<td>$\text{tool}$</td>
<td>4.0</td>
<td>169</td>
</tr>
<tr>
<td>$\text{time}$</td>
<td>81.5</td>
<td>38</td>
</tr>
<tr>
<td>$\text{doc}$</td>
<td>3.1</td>
<td>97</td>
</tr>
<tr>
<td>$\text{site}$</td>
<td>3.9</td>
<td>59</td>
</tr>
<tr>
<td>$\text{soc}$</td>
<td>0.2</td>
<td>253</td>
</tr>
</tbody>
</table>

**Figure 4.1:** Sample Decision Tree generated from CrossTree. (Sample CT0)

<table>
<thead>
<tr>
<th>Metric</th>
<th>Samples</th>
<th>Branch ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{plex}$</td>
<td>16.0</td>
<td>0</td>
</tr>
<tr>
<td>$\text{tool}$</td>
<td>4.0</td>
<td>169</td>
</tr>
<tr>
<td>$\text{time}$</td>
<td>81.5</td>
<td>38</td>
</tr>
<tr>
<td>$\text{doc}$</td>
<td>3.1</td>
<td>97</td>
</tr>
<tr>
<td>$\text{site}$</td>
<td>3.9</td>
<td>59</td>
</tr>
<tr>
<td>$\text{soc}$</td>
<td>0.2</td>
<td>253</td>
</tr>
</tbody>
</table>

**Figure 4.2:** Sample Decision Tree generated from CrossTree0. (Sample CT0)
Chapter 5

Model: XOMO

This chapter describes XOMO, one of the models used to benchmark CrossTrees. Chapter starts with explaining how objectives are simulated using internal submodels and concludes with identifying problems involved when NSGA-II is implemented on XOMO.

5.1 XOMO

The conventional mining of real empirical data from development teams has passed and new generation of automated models simulating projects have come into practice. The simulations from models are not the same as real time data from software projects, this might arise the question of their validity for testing techniques. This concern has been already addressed several times when monte-carlo simulations were used [14,56–59,61]. The model used in this thesis is XOMO, which produces monte-carlo simulations of software project data. To validate XOMO in context of simulations, an analogy to models built in physics can be considered. Each new model proposed in physics like Thompson model, was wrong since it was superseded by a newer model like quantum mechanical model. But each new model was useful in the sense that it explained more effects than the previous model. The lesson here is that committing to a model of the current best understanding of a phenomenon is good practice, even if that model is not correct in some absolute
sense. And once that new model is generated, it is right and proper that it be exercised, criticized, and improved [60]. XOMO has already been used in several contexts and has been validated for its simulations.

Monte-carlo simulations are run many times in random to obtain a distribution of unknown probabilistic entry. XOMO is a framework of monte-carlo simulation that has been modified for COCOMO-like models. An XOMO user begins by defining a set of “ranges” for model variables. For example, Ranges of Flight software at JPL, NASA are determined to as shown in fig. 5.1. The project defines which software the model relates to. It is followed by all possible values of COCOMO coefficients for Flight software at JPL, NASA specified as discrete values. The minimum and maximum of these values become the range of Flight software XOMO model.

5.2 Models inside XOMO

XOMO uses three internal models to simulate software projects:

1. Boehm et.al.s COCOMO-II (2000) model that computes development effort;

```
@project jplFlight
@discrete ?rely 3 4 5
@discrete ?data 2 3
@discrete ?cplx 3 4 5 6
@discrete ?time 3 4
@discrete ?stor 3 4
@discrete ?acap 3 4 5
@discrete ?pcap 3 4 5
@discrete ?apex 2 3 4 5
@discrete ?plex 1 2 3 4
@discrete ?ltex 1 2 3 4
@discrete ?pmat 2 3
@discrete ?kloc 7 418
@discrete ?tool 2
@discrete ?sced 3
```

Figure 5.1: Sample Flight software ranges
2. Madachys heuristic risk model that computes the risk that schedules will over run;

3. Boehm et.al.s COQUALMO model that estimates the number of defects remaining in delivered code;

5.2.1 The COCOMO Effort Model

The Constructive Cost Model (COCOMO 81), a well-known cost and schedule estimation model, was originally published in the text Software Engineering Economics. The model was defined based on the analysis of 63 completed projects from different domains during the 1970s and the early 1980s. To address the issues emerging from changes in technologies and development processes, the USC Center for Systems and Software Engineering has developed and published COCOMO II, a major extension to COCOMO 81. Among the main upgrades are the introduction of new functional forms that use scale factors, new cost drivers, and a set of parameters values. [66]

In COCOMO 81 (1981) and COCOMO II (2000), the effort can be estimated using the model:

$$ PM = A \times Size^B \times \prod_{i} EM_i $$

(5.1)

Where,

PM = Estimate of effort in person months

A = Multiplicative constant

Size = Size Estimate of the software, represented in KSLOC

B = Exponential constant or Scale Factors

EM = Effort Multipliers

In COCOMO 81, the B term is an exponential constant which is usually greater than 1.0, indicating diseconomies of scale. In COCOMO II, B is defined as a function of scale factors, in form of
\[ B = \beta_0 + \sum_{i=1}^{5} \beta_i SF_i \]  

(5.2)

COCOMO II was developed in 2000 to give more precision to original COCOMO 81. COCOMO II uses variables as shown here:

\[
\text{months} = a \times \left( KLOC^{(b + 0.01 \times \sum_{i=1}^{5} SF_i)} \right) \times \left( \prod_{j=1}^{17} EM_j \right) \]  

(5.3)

where \( a \) and \( b \) are domain specific and \( KLOC \) is calculated directly, or from function point analysis. \( SF \) are Scale Factors like “have we built this before?” or “how does the team interact with each other”. \( EM \) are effort multipliers which act as the drivers of cost. Reliability, Database size, code reusability are examples of effort multipliers. The Scale factors have exponential effect on Effort whereas Scale factors are linear.

XOMO begins by generating random samples (integers) within the ranges specified for each decision variable or COCOMO coefficient. Ideally software models like COCOMO II should be tuned to their local domain. 'Off the shelf' untuned models are said to be 600% inaccurate in their estimates, whereas tuned models are far more accurate [60]. A new approach is used in XOMO taking \( a \) and \( b \) as random variables answering the question “what conclusions hold across the space of possible tunings?”. These randomly generated decision variable integers are mapped back to symbols, with the integers \{1,2,3,4,5\} to symbols \{vl,l,n,h,vh\} respectively. The symbols represent very low, low, nominal, high, very high. The symbols can be used to fine the model by referring to Figure 5.2. These fine tuned values of decision variables help in better estimation of effort.

### 5.2.2 SCED-RISK model

The SCED-RISK model return the chances of schedule over run of a project. It was from an experiment by Madachy explaining the heuristic nature of effort estimation. The valued return can
be perceived as followed:

1. low risk: value of 0-5

2. medium risk: value of 5-15

3. high risk: value of 15-50

4. very high risk: value of 50-100

The model correlates with months/KDSI (thousand lines of delivered code). Internally, the model contains dozens of tables as shown in Figure 5.4.

XOMO calculates risks by using the Madachy SCED-RISK model explained in section 5.2.2. For example, to calculate risk for a model having \( sced = vl \) and \( rely = vh \), the COCOMO mapped coefficients or decision variables after tuning are 1.43 for \( sced = vl \) and 1.26 for \( rely = vh \). The risk factor of coefficients can be determined using the Figure 5.3 which is 2. Hence, a project with those two attribute ranges would contribute \( 1.43 \times 1.26 \times 2 = 3.6036 \) to the schedule risk.
5.2.3 COQUALMO

COQUALMO models process how options add and remove defects to software during requirements, design, and coding. For example, poor documentation leads to more errors since developers lack the guidance required to code the right system. Defect introduction factors of COQUALMO are effects-per-1000 lines of code. A small weighting factor is added to show an increasing number of defects as life cycle progresses. Defects remaining in the software is product of defects introduced times the percentage removed. The removal percentage is calculated using several calculations as shown in Figures 5.6, 5.7, 5.8, 5.11, 5.9. These values represent ratios per 1000 lines of code so the complement shows the remaining defects.

Figures 5.7 and 5.6 shows the scale factors and effort multipliers using which defect is introduced. Figure 5.8 shows how defects are introduced into XOMO and Figure 5.10 shows how defects are added and removed. To calculate the percentage of defects removed Figure 5.11 is used in form of Figure 5.9.

Effort, schedule months, risks and defects are the objective dimensions of XOMO. In the next section our novel technique CrossTree is operated on XOMO. Those dimensions are the objectives of CrossTree, with the decision variables or COCOMO coefficients being independent variables help CrossTree iterate for decisions that are worthwhile.
Total_risk = 
(Schedule_risk + Product_risk + Personnel_risk + 
Process_risk + Platform_risk + Reuse_risk)/3.73

Schedule_risk= 
Sced_Rely_risk + Sced_Time_risk + Sced_Pvol_risk + 
Sced_Tool_risk + Sced_Acap_risk + Sced_Aexp_risk + 
Sced_Pcap_risk + Sced_Plex_risk + Sced_Ltex_risk + 
Sced_Pmat_risk

Product_risk = 
Rely_Acap_risk + Rely_Pcap_risk + Cplx_Acap_risk + 
Cplx_Pcap_risk + Cplx_Tool_risk + Rely_Pmat_risk + 
Sced_Cplx_risk + Sced_Rely_risk + Sced_Time_risk + 
Ruse_Aexp_risk + Ruse_Ltex_risk

Personnel_risk = 
Pmat_Acap_risk + Stor_Acap_risk + Time_Acap_risk + 
Tool_Acap_risk + Tool_Pcap_risk + Ruse_Aexp_risk + 
Ruse_Ltex_risk + Pmat_Pcap_risk + Stor_Pcap_risk + 
Time_Pcap_risk + Ltex_Pcap_risk + Pvol_Plex_risk + 
Sced_Acap_risk + Sced_Aexp_risk + Sced_Pcap_risk + 
Sced_Plex_risk + Sced_Ltex_risk + Rely_Acap_risk + 
Rely_Pcap_risk + Cplx_Acap_risk + Cplx_Pcap_risk + 
Team_Aexp_risk

Process_risk = 
Tool_Pmat_risk + Time_Tool_risk + Tool_Pmat_risk + 
Team_Aexp_risk + Team_Sced_risk + Team_Site_risk + 
Sced_Tool_risk + Sced_Pmat_risk + Cplx_Tool_risk + 
Pmat_Acap_risk + Tool_Acap_risk + Tool_Pcap_risk + 
Pmat_Pcap_risk

Platform_risk = 
Sced_Time_risk + Sced_Pvol_risk + Stor_Acap_risk + 
Time_Acap_risk + Stor_Pcap_risk + Pvol_Plex_risk + 
Time_Tool_risk

Reuse_risk = 
Ruse_Aexp_risk + Ruse_Ltex_risk

Figure 5.4: SCED-RISK: the calculations.
<table>
<thead>
<tr>
<th>sced</th>
<th>vl</th>
<th>l</th>
<th>n</th>
<th>h</th>
<th>vh</th>
<th>xh</th>
</tr>
</thead>
<tbody>
<tr>
<td>rely</td>
<td>vl</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cplx</td>
<td>vl</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>vl</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pvol</td>
<td>vl</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tool</td>
<td>vl</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pexp</td>
<td>vl</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pcap</td>
<td>vl</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aexp</td>
<td>vl</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acap</td>
<td>vl</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ltex</td>
<td>vl</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pmat</td>
<td>vl</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vl</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acap</td>
<td>vl</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.5: SCED-RISK: the details. Taken from [60]
Figure 5.6: COQUALMO: effort multipliers and defect introduction. Taken from [60]

```
function defectsIntroduced() {
    return 10*Ksloc()*defectsIntroduced1("requirements") +
           20*Ksloc()*defectsIntroduced1("design") +
           30*Ksloc()*defectsIntroduced1("coding")
}
```

Figure 5.7: COQUALMO: scale factors and defect introduction. Taken from [60]

```
function defectsIntroduced1(table) {
    # return the product of the Figure 5.6 and
    # and the Figure 5.7 Figures
}
```

Figure 5.8: COQUALMO: defects introduced. Taken from [60]
Figure 5.9: COQUALMO: defect removal

```plaintext
function Total_defects() {
    return defects("requirements", Coqualr) +
            defects("design", Coquad) +
            defects("coding", Coqualc)
}

function defects(what,table) {
    introduced = defectsIntroduced(what,table);
    percentRemoved = defectsRemovedRatio(what);
    return percentRemoved*introduced
}
```

Figure 5.10: COQUALMO: defects added and removed

```plaintext
function defectsRemovedRatio(table, auto, review, tool) {
    return (1 - drf(table,"automated_analysis")) *
            (1 - drf(table,"peer_reviews")) *
            (1 - drf(table,"execution_testing_and_tools"))
}

function drf(table,x ) {
    # return x's value in table from Figure 5.9
}
```

Figure 5.11: COQUALMO: ratio of defects removed
5.3 Implementing NSGA-II with XOMO

In the current thesis, the cutting edge evolutionary algorithm explained in section 2.3.3 is implemented with XOMO. XOMO being a four dimensional model is searched for optimized space using NSGA-II, a multi objective optimizer. NSGA-II is reknown to be one of the most widely used algorithm to search the space of solutions (Refer to section 2.3.6).

5.3.1 Parameters

The parameters used by Krall [44] as explained in section 2.3.3 are used for this implementation of NSGA-II. XOMO model is built on 25 attributes which form the solution space of 25 dimensions to predict 4 objectives. The 4 objectives are:

1. Effort: Effort required for a project measured in Person-months
2. Months: Number of months to complete the project
3. Risk: Chances of schedule overrun
4. Defects: Number of defects

The only parameter that manipulates XOMO in this thesis are the number of generations which vary based on the experiment. A default of 500 is considered in case number of generations are not mentioned (i.e. \( n = 500 \)).

The solution space of size \( n \) is generated from XOMO initially. This solution space is sorted into different bands by the fast sort algorithm of NSGA-II. A new population is merged with this initial population after mutation and crossovers, thereby preserving diversity. The process is exactly as explained in section 2.3.3, with the stopping criteria as explained in section 2.3.4.

To validate XOMO and NSGA-II different datasets were pulled from contacts at JPL, NASA. These datasets were already used by Menzies et al. [10, 44, 60] in applying Case-based reasoning to reduce software development effort. These software is broadly classified into following models:
1. Flight

2. Ground

3. Osp (Orbital Space Program)

4. Osp2

5. All

Flight and Ground are generic data set models whereas Osp and its later version Osp2 are tightly constrained models. All model constitutes of all models clubbed together. Contraints are written based on these datasets and fed into XOMO to generate projects resembling Flight, Ground, Osp, Osp2, and All software projects at JPL, NASA.

5.3.2 Disadvantages of NSGA-II on XOMO

Although, section 2.3.6 states NSGA-II to be one of most-widely used search optimization methods, several problems were discovered when it is implemented with XOMO model. NSGA-II was improved over its initial prototype to handle large populations better by computing faster. Also, the general problems with representing solutions alone on a pareto optimal front persist in using NSGA-II as well. NSGA-II produces single point solutions along pareto-optimal space that are better and but gives no insight on how worse solutions can get better.

To test the performance of NSGA-II on XOMO model, XOMO is integrated with NSGA-II with parameters varying with different parameters to find the best possible performance of NSGA-II. The parameters used in XOMO are same as ones mentioned in Figure 5.2.

NSGA-II performs fairly well but there are a few disadvantages that can be grouped into three categories:
**Time for generations**

Though NSGA-II runs in $O(MN^2)$ time [44], when N is increased considerably, it takes substantial time. XOMO is considerably fast model with computations being linear and not exponential. When NSGA-II runs on XOMO model with constraints specified according to different models of JPL software as specified in parameters section 5.3.1.

Consistently across all models the times were increasing as depicted in Figure 5.12. The X-axis represents population size which was experimentally increased from 100 to 500 with increments of 100. The time seems to increase with larger population size. Also, note that the time is represented in minutes. It can be inferred from the Figure that NSGA-II algorithm running on XOMO Flight model takes less than a minute on population size of 100 which increases to more than 3 minutes when population is increased to 500.

Note that NSGA-II is repeated 20 times to maintain an average case and validate its solution. The time represents 20 repeats of NSGA-II with an average generations varying across each model and population. But the maximum number of generations is fixed at 50 which rarely acts as stopping criteria. In the experiment it was noted to stop with maximum generations only 2 times out of 500 total runs of NSGA-II. Algorithm seems to exhaust its chances as explained in 2.3.4 before reaching maximum generations.

The reason for using population size of up to 500 can be explained using Figure 5.13. It can be seen that the objectives are checked for performance starting with 100 population size. As the population increases, NSGA-II improves on objectives and gets stable around 400. Population 500 seems to stabilize solutions even more, hence it was used as max limit on population size. Also, the Author attempted to benchmark new method doing atleast as good as existing method. Using population size up to 500 will achieve that.
Figure 5.12: Execution time taken by NSGA-II to generate solution for XOMO

Figure 5.13: Population size is restricted from 100 to 500 when NSGA-II is tested on XOMO All model.
Single point solutions

NSGA-II although an multiobjective optimizer produces points across the pareto optimal frame as shown in Figure 5.14.

For multiobjective problems, optimizers search for solutions along pareto optimal front. These solutions might improve few objectives at rate of sacrificing few other objectives. The solutions outside pareto optimal front perform worse than the solutions on pareto optimal front because of conflicting objectives. NSGA-II searches space for these pareto optimal solutions, but reports only single points on pareto optimal front. It is often the case that single point might be surrounded by group of worse performing solutions. So by reporting single point solutions even along the pareto optimal points might not help the user to improve overall performance. And assuming pragmatically, it is ideal to have group of better solutions than one single solution.

Lack of insight

NSGA-II searches for solution space and reports back best solutions to follow to achieve particular goals. But in the process of NSGA-II it is highly complicated and nearly impossible to understand
the complex behavior of algorithm, to figure out why they are better. There is no procedure mentioned by NSGA-II to improve worse solutions or changes in worse solutions that can be done to improve them into better set of solutions. The complex crossover+mutation algorithms improve performance and provide best solutions for a goal but they do not support any method to improve worse solutions to get better.

This thesis is motivated from the above problems of one of the widely used multiobjective optimizer and demanded a need to build CrossTree a method to overcome the above mentioned problems. To further validate CrossTree and confirm the problems of using stochastic methods, CrossTree is tested with one more model, POM3.
Chapter 6

Model: POM3

This chapter describes POM3, a model used to identify problems of Stochastic methods and also to benchmark CrossTree. This chapter starts with history of POM models and explains the motivation of designing better versions of initial model. Chapter concludes with confirming problems identified in last chapter in applying stochastic methods.

6.1 POM3

POM3 is a software estimation model like XOMO for Software Engineering. It is based on Turner and Boehm’s model of agile development. [6, 7]. It compares traditional plan-based approaches to agile-based approaches in requirements prioritization. It describes how a team decides which requirements to implement next. POM3 reveals requirements incrementally in random order, with which developers plan their work assignments. These assignments are further adjusted based on current cost and priority of requirement. POM3 is a realistic model which takes more runtime than standard lab models(2-100ms, not 0.006-0.3ms) [44].
6.2 History

Port, Oklov, Menzies (POM) simulated a fundamental, yet intangible benefits of requirements prioritization strategies. The simulation was based on detailed empirical studies which was later improvised in POM2 and POM3.

6.2.1 POM1

Initial model of POM was a partial implementation of Boehm, and Turner which explored the effects of different prioritization policies while adjusting rate at which new requirements arrive/change. Noor, Rabiser and Grunbacker [65] explored trade-offs in planning agile processes and also criticized POM to explore only one of five scales proposed by Boehm and Turner. POM2 was a result of answering the problems of POM1 to address all 5 scales of Boehm. POM1 assumed small projects with 10 developers per project and maximum of 25 requirements.

6.2.2 POM2

POM2 implements all five scales compared to POM1 which just implements one scale proposed by Boehm and Turner. POM1 implemented one team whereas POM2 implements multiple teams. POM2 has dependents of requirements where child requirements must be satisfied before completing parents requirements. POM2 simulates projects with up to 300 developers and expanded number of requirements to 750.

The following are the inputs that effect the processing within POM2 mentioned in [46]:

- In high critical systems, requirements cost more to develop.
- In high dynamism, the requirements cost and values change frequently.
6.3 POM3 attributes

The catch of POM3 is its highly dynamic so the cost and priority can change after developers have made their work assignments. So POM3 is simulated such that some developers spend some time focusing on wrong requirements as well. It is considered pragmatically that in highly dynamic environments, the agile is better method for reacting to arrival of new requirements and existing requirements changing value. Most of the literature debates agile vs plan based models which can lead to false preconception of practices using interesting combinations of both. A more nuanced view is offered by Boehm and Turner. They define five scales that can characterize the difference between plan-based and agile-based methods:

- 1. Project size
- 2. Project Criticality (higher criticality higher the cost)
- 3. Requirements dynamism (priorities and costs change)
- 4. Personnel (developer skill level)
- 5. Organizational culture (larger the number more conservative the team is)

6.4 POM3

Requirements are represented as a set of trees in POM3. Single node of the tree represents a single requirement. These nodes are combined into trees which actually form the requirements heap. Each node i.e. requirement consists of the following:

- 1. Prioritization value
- 2. Prioritization cost
- 3. Child requirements
4. Dependencies

The prioritization values are in the range of 30 to 500 and cost ranges from 1 to 100. These values are chosen in consultation with Richard Turner which were the same as used in [44]. Some percent of requirements are marked as visible initially leaving the rest to be revealed by POM3 as the project progresses.

In POM3, teams divide the tasks of completing project’s requirements. The number of tasks assigned to a team is relative to the size of the team. The team size is an decision input and remains constant throughout the simulation. POM3’s personnel are divided into three categories based on the programming skill: 1.Alpha 2.Beta 3.Gamma. Alpha programmers are generally the best, most-paid type of programmers while Gamma programmers are the least experienced, least-paid. The ratio of personnel type follows the personnel decision as set out by the Boehm and Turner [6,7] in following table:

<table>
<thead>
<tr>
<th>project size</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>45%</td>
<td>50%</td>
<td>55%</td>
<td>60%</td>
<td>65%</td>
</tr>
<tr>
<td>Beta</td>
<td>40%</td>
<td>30%</td>
<td>20%</td>
<td>10%</td>
<td>0%</td>
</tr>
<tr>
<td>Gamma</td>
<td>15%</td>
<td>20%</td>
<td>25%</td>
<td>30%</td>
<td>35%</td>
</tr>
</tbody>
</table>

Costs are updated after teams are generated and assigned to requirements. Criticality and Criticality modifier play a major role in deciding those cost changes. Criticality affects the cost-affecting nature of the project and ensures how safety-critical a project is. Whereas, criticality modifier indicates a percentage of teams affected by safety critical requirements. In the formula, Cm is a criticality modifier:

\[
\text{cost} = \text{cost} \cdot C^{\text{criticality}}_M
\]  

(6.1)
After generating teams and requirements for those teams to work on, POM3 runs through a shuffling process. This shuffling process is repeated at random for \((1 \leq N \leq 6)\) times, where \(N\) is chosen at random. The shuffling process includes following steps:

- **Collect Available Requirements:** The assigned requirements are searched for all available requirements that can be completed by each individual team. These requirements must be available to complete i.e. without any unsatisfied child requirements or unsatisfied dependencies. The team budget is then updated by calculating the total cost of requirements remaining for the team and dividing by the number of shuffling iterations:

\[
\text{team.budget} = \text{team.budget} + \frac{\text{totalCost}}{\text{numShuffles}}
\]

- **Apply for a Requirements Prioritization Strategy:** The available requirements with high priority should be satisfied first. In order to support that, POM3 sorts the available requirements by some sorting strategy which includes requirement’s cost and values. A plan decision determines this strategy and is implemented inside POM3.

- **Execute Available Requirements:** The team executes the available requirements that are sorted in previous step sequentially. Some requirements may not be executed due to the budget allocations which are executed in the next iteration.

- **Discover New Requirements:** Upon completing few requirements, there is a possibility to open new requirements that depended on these completed requirements. To model the probability of arrival of new requirements, input decision Dynamism is considered to be used in a Poisson distribution. The following formula defines the percentage of new requirements to be added in the heap:

\[
\text{new} = \text{Poisson} \left( \frac{\text{dynamism}}{10} \right) \quad (6.2)
\]
<table>
<thead>
<tr>
<th>Short name</th>
<th>Decision</th>
<th>Description</th>
<th>Controllable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cult</td>
<td>Culture</td>
<td>Number (%) of requirements that change.</td>
<td>yes</td>
</tr>
<tr>
<td>Crit</td>
<td>Criticality</td>
<td>Requirements cost effect for safety critical systems (see Equation 6.1 in the appendix).</td>
<td>yes</td>
</tr>
<tr>
<td>Crit.Mod</td>
<td>Criticality Modifier</td>
<td>Number of (%) teams affected by criticality (see Equation 6.1 in the appendix).</td>
<td>yes</td>
</tr>
<tr>
<td>Init. Kn</td>
<td>Initial Known</td>
<td>Number of (%) initially known requirements.</td>
<td>no</td>
</tr>
<tr>
<td>Inter-D</td>
<td>Inter-Dependency</td>
<td>Number of (%) requirements that have interdependencies. Note that dependencies are requirements within the same tree (of requirements), but interdependencies are requirements that live in different trees.</td>
<td>no</td>
</tr>
<tr>
<td>Dyna</td>
<td>Dynamism</td>
<td>Rate of how often new requirements are made (see Equation 6.2 in the appendix).</td>
<td>yes</td>
</tr>
<tr>
<td>Size</td>
<td>Size</td>
<td>Number of base requirements in the project.</td>
<td>no</td>
</tr>
<tr>
<td>Plan</td>
<td>Plan</td>
<td>Prioritization Strategy (of requirements): one of 0= Cost Ascending; 1= Cost Descending; 2= Value Ascending; 3= Value Descending; 4 = ( \frac{Cost}{Value} ) Ascending.</td>
<td>yes</td>
</tr>
<tr>
<td>T.Size</td>
<td>Team Size</td>
<td>Number of personnel in each team.</td>
<td>yes</td>
</tr>
</tbody>
</table>

Figure 6.1: List of Decisions used in POM3. The optimization task is to find settings for the controllables in the last column. Taken from [44]

- **Adjusting Priorities:** The teams adjust their requirements priorities by making use of input decisions Culture and Dynamism. The following formula helps adjust normal distribution and is scaled by project’s Culture:

\[
\text{value} = \text{value} + \text{maxRequirementValue} \times \text{Normal}(0, D) \times C
\]

Finally, POM3 models score a planning policy by comparing a developer’s incremental decisions to those that might have been made by some omniscient developer. This omniscient developer has access to final cost and priority of requirements.

### 6.5 POM3 submodels

Several submodels of POM3 are defined to give different modes of POM3. In summary, POM3a is an unconstrained version of POM3 with its full decision ranges for all decisions. POM3a represents both classes of large and small projects. POM3b is constrained with decisions for only small projects with high criticality and POM3c is designed for Large projects with high dynamism. There is no claim that this represents all possible kinds of projects. Rather, these three sub-models mark as interesting variety of projects to study. Figure 6.2 defines decisions of these submodels [44] defined below in detail.
A broad space of projects. Highly critical small projects

<table>
<thead>
<tr>
<th>Attribute</th>
<th>POM3a</th>
<th>POM3b Highly critical small projects</th>
<th>POM3c Highly dynamic large projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Culture</td>
<td>0.10 ≤ x ≤ 0.90</td>
<td>0.82 ≤ x ≤ 1.26</td>
<td>0.50 ≤ x ≤ 0.90</td>
</tr>
<tr>
<td>Criticality</td>
<td>0.82 ≤ x ≤ 1.26</td>
<td>0.82 ≤ x ≤ 1.26</td>
<td>0.82 ≤ x ≤ 1.26</td>
</tr>
<tr>
<td>Criticality Modifier</td>
<td>0.02 ≤ x ≤ 0.10</td>
<td>0.80 ≤ x ≤ 0.95</td>
<td>0.02 ≤ x ≤ 0.08</td>
</tr>
<tr>
<td>Initial Known</td>
<td>0.40 ≤ x ≤ 0.70</td>
<td>0.40 ≤ x ≤ 0.70</td>
<td>0.20 ≤ x ≤ 0.50</td>
</tr>
<tr>
<td>Inter-Dependency</td>
<td>0.0 ≤ x ≤ 1.0</td>
<td>0.0 ≤ x ≤ 1.0</td>
<td>0.0 ≤ x ≤ 50.0</td>
</tr>
<tr>
<td>Dynamism</td>
<td>1.0 ≤ x ≤ 50.0</td>
<td>1.0 ≤ x ≤ 50.0</td>
<td>40.0 ≤ x ≤ 50.0</td>
</tr>
<tr>
<td>Size</td>
<td>x ∈ [3, 10, 30, 100, 300]</td>
<td>x ∈ [3, 10, 30]</td>
<td>x ∈ [30, 100, 300]</td>
</tr>
<tr>
<td>Team Size</td>
<td>1.0 ≤ x ≤ 44.0</td>
<td>1.0 ≤ x ≤ 44.0</td>
<td>20.0 ≤ x ≤ 44.0</td>
</tr>
<tr>
<td>Plan</td>
<td>0 ≤ x ≤ 4</td>
<td>0 ≤ x ≤ 4</td>
<td>0 ≤ x ≤ 4</td>
</tr>
</tbody>
</table>

Figure 6.2: Three classes of projects studied using POM3. Taken from [44]

- POM3a: The unconstrained version with full ranges of all decisions
- POM3b: Small Projects, High criticality.
- POM3c: Large Projects, High Dynamism.

### 6.6 Implementing NSGA-II with POM3

The parameters used by NSGA-II are same as ones used by Krall [44] as explained in section 2.3.3. POM3 model is built on nine attributes which act as nine dimensions to predict three objectives. Similar to XOMO parameters explained in section 5.3.1 the only parameter modified in this thesis is the number of generations. A default of 500 is maintained if number of generations are not mentioned (i.e \( n = 500 \)). The objectives being:

- Cost: Cost of project
- Completion: Completion time of project
- Idle: Idle time of project.

Similar, to standard search algorithms, NSGA-II on POM3 starts with generating an initial population \( N \) from POM3. This solution space is explored by NSGA-II by first sorting them by elite
fast sort algorithm as mentioned in section 2.3.3. Then while preserving diversity, mutation and
crossover are performed to generate a new population which is aggregated to existing population
to form new solution space. The process is explained in detail in section 2.3.3.

All the POM3 models mentioned in section 6.5 are tested with NSGA-II and the results are explained in this section. The results form the motivation of current thesis to improve on the existing widely used method.

6.6.1 Disadvantages of NSGA-II

NSGA-II being current widely used method for search based optimization, claims to have few drawbacks when tested with POM models. The parameters used for POM3 models are shown in Figure 6.2. The drawbacks of NSGA-II are consistent across both the explored models. NSGA-II fails to provide any insights on the best solutions it picked and also takes exponential time when population size $N$ is increased.

Time for generations

As explained in section 2.3.3 the time taken by NSGA-II is $O(MN^2)$. It was observed that when $N$ is increasing, time taken by NSGA-II is exponential. The Figure 6.3 represents population size on X-axis and execution time in minutes on Y-axis. We can infer that NSGA-II running on POM3C model takes about 22 min for population size of 100 and 2 and half hours for population size of 500. It was noted by the experiment from the Author that NSGA-II was taking around 2 days to run on population of 800. This indeed is a very long time even when many cores are used for fast processing for applications with Big data as mention in Chapter 1.

It has to be noted that NSGA-II was executed with 20 repeats and maximum number of generations before it stops is maintained to be 50 which is seldom reached (2 times out of 300 in the Author experiment). The stopping criteria with chances seems to be play more crucial role while deciding when NSGA-II should stop compared to maximum number of generations.
Figure 6.3: Execution time taken by NSGA-II to generate solution for POM3

The reason for using population size of up to 500 can be explained using Figure 6.4. It can be seen that the objectives are checked for performance starting with 100 population size. As the population increases, NSGA-II improves on objectives and gets stable around 300. Population 500 seems to stabilize solutions even more, hence it was used as max limit on population size. Also, the Author attempted to benchmark new method doing atleast as good as existing method. Using population size up to 500 will achieve that.

**Single point solutions**

NSGA-II results are single point solutions across pareto optimal frame as shown in Figure 5.14. These pareto optimal points help performance of NSGA-II but in real world application of search algorithm, single points might not enough for experts to implement as explained in Chapter 2.
solutions are given in experiments section and this issue addressed in detail.

**Lack of insight**

The solutions generated from NSGA-II fail to give insight into why they are actually better solutions for experts. A much more efficient method would inform the experts how their worse solutions can improve to get better while improving overall performance. The complex NSGA-II algorithm produces excellent solutions from the model but if the application of these solutions is not easy for experts or if experts ask for insight of why these solutions seem to be better, NSGA-II lacks that provision to convert its inner computations to human understandable language. The Author attempt to design a method which overcomes this drawback of NSGA-II while maintaining the performance which is explained in Chapter 4.

Problems identified in these chapters are overcomed by applying CrossTree method to both models XOMO and POM3. The results of such experiments are explained in next chapter.
Chapter 7

Experiments

This section explains in detail about the experiments performed for this thesis. It starts with a few research questions, and proceeds with mentioning the design and specifications of tools and techniques used. Following which, experiments are described in detail and concludes with how results of those experiments answer research questions.

7.1 Overview

CrossTree algorithm explained in Chapter 3 is experimented in this chapter. To validate the performance of CrossTree, a base case is maintained along with testing alongside one of the most widely used genetic algorithms NSGA-II. The models XOMO and POM3 are used in this process. The adaptions used in experiments are Flight, Ground, Osp, Osp2 and all of XOMO and POM3A, POM3B, POM3C of POM3 series. Both are them are explained in detail in Chapters 5 and 6. The parameters of CrossTree are explained in brief in Chapter 3 and NSGA-II are explained in section 2.3.3. Next section briefly describes the research questions the Authors are trying to address.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Size</td>
<td>$\sqrt{n} &lt; 2 \times \sqrt{n}$</td>
<td>Percentage of features used</td>
</tr>
<tr>
<td>Infogain</td>
<td>75</td>
<td>Decision Tree parameter</td>
</tr>
<tr>
<td>Discretization</td>
<td>CT1: False, CT0: True</td>
<td>Decision Tree parameter</td>
</tr>
<tr>
<td>Minimum Samples per leaf</td>
<td>20</td>
<td>Prune leaves of decision tree with 2 conditions:(1)</td>
</tr>
<tr>
<td>Decision Tree Criteria</td>
<td>Entropy</td>
<td>multiple majority class (2) same majority class in left and right</td>
</tr>
<tr>
<td>Decision Tree Pruning</td>
<td>True</td>
<td>subtree</td>
</tr>
<tr>
<td>Bootstrap population size</td>
<td>250</td>
<td>number of samples generated inside bootstrap test</td>
</tr>
<tr>
<td>Populations</td>
<td>(100, 200, 300, 400, 500)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.1: Default CrossTree specifications of parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of repeats</td>
<td>20</td>
<td>Number of times results are collected</td>
</tr>
<tr>
<td>Max no. of generations</td>
<td>50</td>
<td>Maximum number of generations before NSGA terminates</td>
</tr>
<tr>
<td>Mutation Parameter</td>
<td>0.15</td>
<td>3 chances are given to improve over last generation</td>
</tr>
<tr>
<td>Domination Parameter</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Number of chances(to stop)</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.2: Default specifications of NSGA-II parameters

### 7.2 Design and Specifications

An initial population is generated in each experiment prior to performing any step. The population is generated by the model alone without any external constraints. The only contraints that the model uses is the submodel adaptive contraints specified internally.

By several experiments and varying different parameters across models, the Authors decided to use certain values as parameters while performing experiments on CrossTrees. CrossTrees uses the following specifications as default as mentioned in Figure 7.1

The constraints for experimenting with NSGA-II is same as mentioned by Krall in [44]. Few of them are as shown in Figure 7.2
7.3 How to read output

To collect the results as mentioned in chapter 3, newly generated population of clusters $ConS_n$ and the better clusters $BetterS_n$ are processed through $CollectResults$ module which is built to generate results as mentioned in Figure 9 of [55]. The Figure mentioned serves as an inspiration for concise representation of results.

First column of the sample Figure 7.5 represents techniques CrossTrees is tested with. Baseline and NSGA-II genetic algorithm are two other methods apart from two variations of CrossTree. Baseline represents results collected initial groups or clusters of data. NSGA-II techniques are results collected by implementing NSGA-II algorithm on initial data. The techniques implemented should decrease or increase objectives represented with $-$ for decreasing objectives and $+$ for increasing objectives.

The mqw column represents median, interquartile range(75-25 percent) and worse numbers of respective techniques. For all clusters $C_i$ of size $N_i$ weighted sum is calculated for each of m,q,w by:

$$E(m||q||w) = \sum (Q_i * N_i) / N$$ with i being cluster id.

These E values of each of m, q and w are normalized between 0 and 100 and those values are mentioned in last rows of table.

The tables are also color-coded according to two conditions:

1. **Green**: CrossTrees performing as good as NSGA II If the goal is to decrease an objective, and CT0 has a value less than NSGA-II, the cell is colored green. Cell is colored green if the goal is to increase an objective, and CT0 has value greater than NSGA-II. Using this color we can determine when CT0 performs as good as NSGA-II. A window of 5 percent is taken to color CrossTrees performance when comparing with NSGA II to show the “as good as” measure. This is shown in Figures 7.3 and 7.5
Figure 7.3: Sample results explaining green colored cell. The cell is colored green when CrossTrees variation CT0 is performing better than or as good as NSGA II.

2. **Red**: CT1 performing better than CT0 Cells are color-coded to red when CT1 performs better than CT0. Using this color we can determine the instances when CT0 failed compared to CT1. Since, the aim of this thesis is to find a readable better solutions, more red cells indicate unreadable big trees are performing better than short readable trees. A window of 5 percent is given for CT0 for instances where small trees perform as good as big trees. An example of this color coding is shown in Figures 7.4 and 7.5.

<table>
<thead>
<tr>
<th>Techniques</th>
<th>mqw</th>
<th>-effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT0 xomoo2500</td>
<td>q</td>
<td>0</td>
</tr>
<tr>
<td>NSGA xomoo2500</td>
<td>q</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 7.4: Sample results explaining red colored cell. The cell is colored red when CT1 having bigger decision trees are performing better than CT0 which have smaller decision trees.

For example, in Figure 7.5 the technique CT0 has a median of 2 for objective effort implying that its actual value is 2 percent of max value (3408.55), that is, 0.02 * 3408.55 = 68.171. This is an weighted sum of medians of all clusters for effort. Since different between their values is less than 5 percent, CT0 cell is color-coded with green to indicate that its performance is as good as NSGA II. Similarly, m,q and w measures are evaluated and color-coded.

The techniques mentioned in Figure 7.5 are:

- 1. **Base Line**: Initial data from model clustered and results are retrieved.
7.4 **Experiment 1: CrossTrees on XOMO**

This section provides the results of testing CrossTrees with various submodels of XOMO. The section starts with experimental results followed by discussion on how the experiments on each model supported hypothesis. *Please note that a population size of 500 is maintained and the results shown is consistent with all populations of the models. The tested populations were of sizes 100, 200, 300, 400, 500.*

### Results of XOMO Flight Model

The results of applying CrossTrees to XOMO Flight model is shown in Figure 7.6. CT0 technique slightly dominates this model compared to others. The majority green cells show that CT0 is
performing as good as NSGA II. And also CT1 performs as good as CT0 which is indicated by majority of red cells. In case of months and risk objectives, low m, q and w measures indicate CT0 outperforming NSGA-II. XOMO flight is the only model where CT0 partially fails compared to CT1.

<table>
<thead>
<tr>
<th>Techniques</th>
<th>mqw</th>
<th>-effort</th>
<th>-months</th>
<th>-defects</th>
<th>-risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Line xomof500</td>
<td>m</td>
<td>2</td>
<td>25</td>
<td>1</td>
<td>84</td>
</tr>
<tr>
<td>CT0 xomof500</td>
<td>m</td>
<td>0</td>
<td>31</td>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>CT1 xomof500</td>
<td>m</td>
<td>0</td>
<td>17</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>NSGA xomof500</td>
<td>m</td>
<td>1</td>
<td>36</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Base Line xomof500</td>
<td>q</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>CT0 xomof500</td>
<td>q</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>CT1 xomof500</td>
<td>q</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>NSGA xomof500</td>
<td>q</td>
<td>2</td>
<td>15</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Base Line xomof500</td>
<td>w</td>
<td>16</td>
<td>42</td>
<td>19</td>
<td>98</td>
</tr>
<tr>
<td>CT0 xomof500</td>
<td>w</td>
<td>100</td>
<td>76</td>
<td>100</td>
<td>38</td>
</tr>
<tr>
<td>CT1 xomof500</td>
<td>w</td>
<td>16</td>
<td>38</td>
<td>44</td>
<td>21</td>
</tr>
<tr>
<td>NSGA xomof500</td>
<td>w</td>
<td>29</td>
<td>100</td>
<td>18</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>23134.51</td>
<td>68.97</td>
<td>336598.01</td>
<td>21.4</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>735.9</td>
<td>5.22</td>
<td>5235.99</td>
<td>0.98</td>
</tr>
</tbody>
</table>

Figure 7.6: Results of CrossTrees on XOMO Flight model with population 500

**Results of XOMO Ground Model**

The results of applying CrossTrees to XOMO Ground model is shown in Figure 7.7. The results of XOMO Ground model are different to XOMO Flight model. Majority of green cells indicates CT0 performing as good as NSGA II. Low m,q, and w measure of months and risks indicate CT0 outperforming NSGA-II. No red cells indicate short trees doing as good as big trees. However, NSGA II dominates on objectives: effort and defects.

**Results of XOMO Osp Model**

The results of applying CrossTrees to XOMO Osp model is shown in Figure 7.8. Majority of green cells shows that CT0 performs as good as NSGA-II for this model. However, CT1 has not
Techniques | mqw | -effort | -months | -defects | -risks
---|---|---|---|---|---
Base Line xomogr500 | m | 5 | 22 | 1 | 64
CT0 xomogr500 | m | 0 | 12 | 0 | 13
CT1 xomogr500 | m | 4 | 18 | 6 | 17
NSGA xomogr500 | m | 10 | 38 | 1 | 1
Base Line xomogr500 | q | 3 | 2 | 7 | 11
CT0 xomogr500 | q | 0 | 0 | 2 | 0
CT1 xomogr500 | q | 2 | 1 | 8 | 6
NSGA xomogr500 | q | 12 | 18 | 2 | 4
Base Line xomogr500 | w | 22 | 33 | 34 | 86
CT0 xomogr500 | w | 25 | 24 | 33 | 26
CT1 xomogr500 | w | 34 | 35 | 71 | 42
NSGA xomogr500 | w | 100 | 100 | 34 | 41
100 | 6319.6 | 67.75 | 182979.6 | 23.6
0 | 312.6 | 3.25 | 2393.35 | 1.42

Figure 7.7: Results of CrossTrees on XOMO Ground model with population 500

performed better than CT0. It can be observed that CT0 performs almost as good as NSGA-II and even better in case of effort, months and risks objectives. This poses an interesting alternative when readability of small trees is affecting performance.

**Results of XOMO Osp2 Model**

The results of applying CrossTrees to XOMO Osp2 model is shown in Figure 7.9. CT0 dominates this model by outperforming NSGA-II on all objectives and measures. The q measure shows that CT0 can optimize solutions more accurately than NSGA-II. Results of CT0 on this model is similar to Osp and Ground model.

**Results of XOMO All Model**

The results of applying CrossTrees to XOMO All model is shown in Figure 7.9. This model is developed with a union of constraints of all other models (Refer chapter 5). The contraints hence define the limits of XOMO model in general. Results of this model indicates that CT0 performs as good as NSGA-II on effort and outperforms on months, defects and risks. These results are
<table>
<thead>
<tr>
<th>Techniques</th>
<th>mqw</th>
<th>effort</th>
<th>months</th>
<th>defects</th>
<th>risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Line xomoos500</td>
<td>m</td>
<td>4</td>
<td>20</td>
<td>0</td>
<td>88</td>
</tr>
<tr>
<td>CT0 xomoos500</td>
<td>m</td>
<td>4</td>
<td>18</td>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td>CT1 xomoos500</td>
<td>m</td>
<td>4</td>
<td>13</td>
<td>2</td>
<td>29</td>
</tr>
<tr>
<td>NSGA xomoos500</td>
<td>m</td>
<td>15</td>
<td>42</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Base Line xomoos500</td>
<td>q</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>CT0 xomoos500</td>
<td>q</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>CT1 xomoos500</td>
<td>q</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>NSGA xomoos500</td>
<td>q</td>
<td>16</td>
<td>19</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Base Line xomoos500</td>
<td>w</td>
<td>8</td>
<td>24</td>
<td>6</td>
<td>98</td>
</tr>
<tr>
<td>CT0 xomoos500</td>
<td>w</td>
<td>16</td>
<td>26</td>
<td>16</td>
<td>59</td>
</tr>
<tr>
<td>CT1 xomoos500</td>
<td>w</td>
<td>14</td>
<td>19</td>
<td>18</td>
<td>36</td>
</tr>
<tr>
<td>NSGA xomoos500</td>
<td>w</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>73</td>
</tr>
</tbody>
</table>

Figure 7.8: Results of CrossTrees on XOMO Osp model with population 500

<table>
<thead>
<tr>
<th>Techniques</th>
<th>mqw</th>
<th>effort</th>
<th>months</th>
<th>defects</th>
<th>risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Line xomoos2500</td>
<td>m</td>
<td>6</td>
<td>24</td>
<td>32</td>
<td>75</td>
</tr>
<tr>
<td>CT0 xomoos2500</td>
<td>m</td>
<td>3</td>
<td>14</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CT1 xomoos2500</td>
<td>m</td>
<td>7</td>
<td>16</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>NSGA xomoos2500</td>
<td>m</td>
<td>14</td>
<td>41</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Base Line xomoos2500</td>
<td>q</td>
<td>1</td>
<td>1</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>CT0 xomoos2500</td>
<td>q</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CT1 xomoos2500</td>
<td>q</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>NSGA xomoos2500</td>
<td>q</td>
<td>15</td>
<td>19</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Base Line xomoos2500</td>
<td>w</td>
<td>12</td>
<td>28</td>
<td>66</td>
<td>96</td>
</tr>
<tr>
<td>CT0 xomoos2500</td>
<td>w</td>
<td>11</td>
<td>22</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>CT1 xomoos2500</td>
<td>w</td>
<td>20</td>
<td>21</td>
<td>15</td>
<td>26</td>
</tr>
<tr>
<td>NSGA xomoos2500</td>
<td>w</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 7.9: Results of CrossTrees on XOMO Osp2 model with population 500
consistent with XOMO models: Ground, Osp and Osp2.

<table>
<thead>
<tr>
<th>Techniques</th>
<th>mq</th>
<th>-effort</th>
<th>-months</th>
<th>-defects</th>
<th>-risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Line xomoal500</td>
<td>m</td>
<td>4</td>
<td>22</td>
<td>2</td>
<td>63</td>
</tr>
<tr>
<td>CT0 xomoal500</td>
<td>m</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>CT1 xomoal500</td>
<td>m</td>
<td>4</td>
<td>17</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>NSGA xomoal500</td>
<td>m</td>
<td>9</td>
<td>40</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Techniques</th>
<th>q</th>
<th>-effort</th>
<th>-months</th>
<th>-defects</th>
<th>-risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Line xomoal500</td>
<td>q</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>CT0 xomoal500</td>
<td>q</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>CT1 xomoal500</td>
<td>q</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>NSGA xomoal500</td>
<td>q</td>
<td>10</td>
<td>17</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Techniques</th>
<th>w</th>
<th>-effort</th>
<th>-months</th>
<th>-defects</th>
<th>-risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Line xomoal500</td>
<td>w</td>
<td>24</td>
<td>34</td>
<td>40</td>
<td>87</td>
</tr>
<tr>
<td>CT0 xomoal500</td>
<td>w</td>
<td>46</td>
<td>41</td>
<td>43</td>
<td>39</td>
</tr>
<tr>
<td>CT1 xomoal500</td>
<td>w</td>
<td>46</td>
<td>38</td>
<td>70</td>
<td>44</td>
</tr>
<tr>
<td>NSGA xomoal500</td>
<td>w</td>
<td>100</td>
<td>100</td>
<td>25</td>
<td>33</td>
</tr>
</tbody>
</table>

| 100              | 6414.74 | 65.27  | 229326.0 | 25.7    |
| 0               | 445.34  | 4.59   | 2724.6   | 1.69    |

Figure 7.10: Results of CrossTrees on XOMO all model with population 500

7.5 Experiment 2: CrossTrees on POM3

This section provides the results of testing CrossTrees with various submodels of POM3. The section starts with experimental results followed by discussion on how the experiments on each model supported hypothesis. Please note that a population size of 500 is maintained and the results shown is consistent with all populations of the models. The tested populations were of sizes 100, 200, 300, 400, 500.

Results of POM3A Model

The results of applying CrossTrees to POM3A model is shown in Figure 7.11. Majority of green cells in q measure indicate CrossTrees is performing better than NSGA-II and smaller trees CT0 performs better than CT1 in most cases. Figure 7.11 shows that both variations of CrossTrees can be used to optimize solutions when projects are defined with constraints of POM3A.
Figure 7.11: Cross Tree on pom3A with population 500

Results of POM3B Model

The results of applying CrossTrees to POM3A model is shown in Figure 7.11. For this model CT0 outperforms NSGA-II in almost all measures for objectives cost and idle time. However, for objective completion time CT0 fails to increase.

Results of POM3C Model

The results of applying CrossTrees to POM3A model is shown in Figure 7.11. Majority of red cells indicate bigger trees performing better than smaller trees. CT0 performs as good as NSGA-II on idle time but fails on other two objectives. This model similar to other POM models.

7.6 Summary of Experiments

This section summarizes experiments performed by CrossTrees on both models: XOMO and POM3. By classifying experiments based on aims of this thesis helps in comprehending success or failure of CrossTrees. Two questions can be asked to determine answers for research questions:
### Table: Performance Metrics for Different Techniques

<table>
<thead>
<tr>
<th>Techniques</th>
<th>mqw</th>
<th>-cost</th>
<th>+completion</th>
<th>-idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Line pom3B500</td>
<td>m</td>
<td>25</td>
<td>85</td>
<td>21</td>
</tr>
<tr>
<td>CT0 pom3B500</td>
<td>m</td>
<td>22</td>
<td>83</td>
<td>21</td>
</tr>
<tr>
<td>CT1 pom3B500</td>
<td>m</td>
<td>10</td>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>NSGA pom3B500</td>
<td>m</td>
<td>5</td>
<td>87</td>
<td>22</td>
</tr>
<tr>
<td>Base Line pom3B500</td>
<td>q</td>
<td>18</td>
<td>8</td>
<td>31</td>
</tr>
<tr>
<td>CT0 pom3B500</td>
<td>q</td>
<td>22</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>CT1 pom3B500</td>
<td>q</td>
<td>6</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>NSGA pom3B500</td>
<td>q</td>
<td>0</td>
<td>6</td>
<td>41</td>
</tr>
<tr>
<td>Base Line pom3C500</td>
<td>w</td>
<td>67</td>
<td>99</td>
<td>76</td>
</tr>
<tr>
<td>CT0 pom3C500</td>
<td>w</td>
<td>100</td>
<td>97</td>
<td>96</td>
</tr>
<tr>
<td>CT1 pom3C500</td>
<td>w</td>
<td>34</td>
<td>52</td>
<td>40</td>
</tr>
<tr>
<td>NSGA pom3C500</td>
<td>w</td>
<td>32</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 7.12: Cross Tree on pom3B with population 500

### Table: Performance Metrics for Different Techniques

<table>
<thead>
<tr>
<th>Techniques</th>
<th>mqw</th>
<th>-cost</th>
<th>+completion</th>
<th>-idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Line pom3C500</td>
<td>m</td>
<td>27</td>
<td>92</td>
<td>20</td>
</tr>
<tr>
<td>CT0 pom3C500</td>
<td>m</td>
<td>20</td>
<td>73</td>
<td>6</td>
</tr>
<tr>
<td>CT1 pom3C500</td>
<td>m</td>
<td>13</td>
<td>57</td>
<td>0</td>
</tr>
<tr>
<td>NSGA pom3C500</td>
<td>m</td>
<td>7</td>
<td>91</td>
<td>18</td>
</tr>
<tr>
<td>Base Line pom3C500</td>
<td>q</td>
<td>14</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>CT0 pom3C500</td>
<td>q</td>
<td>15</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>CT1 pom3C500</td>
<td>q</td>
<td>5</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>NSGA pom3C500</td>
<td>q</td>
<td>0</td>
<td>11</td>
<td>24</td>
</tr>
<tr>
<td>Base Line pom3C500</td>
<td>w</td>
<td>63</td>
<td>100</td>
<td>84</td>
</tr>
<tr>
<td>CT0 pom3C500</td>
<td>w</td>
<td>77</td>
<td>79</td>
<td>70</td>
</tr>
<tr>
<td>CT1 pom3C500</td>
<td>w</td>
<td>34</td>
<td>62</td>
<td>46</td>
</tr>
<tr>
<td>NSGA pom3C500</td>
<td>w</td>
<td>59</td>
<td>100</td>
<td>96</td>
</tr>
</tbody>
</table>

Figure 7.13: Cross Tree on pom3C with population 500
1. Does CrossTrees with small trees (CT0) perform as good as NSGA-II?

CrossTrees are considered to perform better when it improves objective “as much as” or more than NSGA-II. To quantify “as good as” measure the Author chose to give a window of 5 percent to CrossTrees when comparing with NSGA-II. Majority of green cells in results table indicates that CT0 dominated NSGA-II. Second column of Figure 7.15 shows the summary of results across all models. A ✓ indicates that a majority of green cells were found and a ✗ indicates otherwise. More ✓ indicates CT1 outperformed NSGA-II.

2. Are smaller trees smarter than bigger trees?

To answer the research question of readability against performance, cells are marked with red when CT1 outperforms CT0. CT1 as described before is a version of CrossTrees having bigger decision trees and CT0 has smaller decision trees. However, huge decision trees fails the purpose of this thesis to develop a readable method of solution for busy businessmen. A ✓ is indicated when CT1 fails to perform as good as CT0 and ✗ is indicated when bigger trees perform better. More ✓ indicates that smaller trees are smarter than bigger trees.
<table>
<thead>
<tr>
<th>Models</th>
<th>Average number of leaves in CT0</th>
<th>Average number of nodes in CT0</th>
<th>Average number of leaves in CT1</th>
<th>Average number of nodes in CT1</th>
<th>% reduction in nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOMO Flight</td>
<td>15</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>39.7</td>
</tr>
<tr>
<td>XOMO Ground</td>
<td>12</td>
<td>12</td>
<td>23</td>
<td>22</td>
<td>48.2</td>
</tr>
<tr>
<td>XOMO Osp</td>
<td>15</td>
<td>14</td>
<td>21</td>
<td>25</td>
<td>44.4</td>
</tr>
<tr>
<td>XOMO Osp2</td>
<td>9</td>
<td>8</td>
<td>20</td>
<td>25</td>
<td>68.8</td>
</tr>
<tr>
<td>XOMO All</td>
<td>14</td>
<td>13</td>
<td>18</td>
<td>21</td>
<td>36.9</td>
</tr>
<tr>
<td>POM 3A</td>
<td>9</td>
<td>8</td>
<td>18</td>
<td>19</td>
<td>59.4</td>
</tr>
<tr>
<td>POM 3B</td>
<td>7</td>
<td>6</td>
<td>17</td>
<td>19</td>
<td>69.8</td>
</tr>
<tr>
<td>POM 3C</td>
<td>7</td>
<td>6</td>
<td>17</td>
<td>19</td>
<td>66.7</td>
</tr>
</tbody>
</table>

Average reduction % in model

<table>
<thead>
<tr>
<th>Models</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOMO</td>
<td>47.6</td>
</tr>
<tr>
<td>POM3</td>
<td>65.3</td>
</tr>
</tbody>
</table>

Figure 7.14: Table shows summary of tree sizes of CrossTree versions CT1 and CT0. It can be noted that CrossTree version CT0 on average has reduced tree size by ≈ 48 percent on XOMO model and ≈ 65 on POM3 model.

Sizes of decision trees in CrossTree versions CT0 and CT1 can be noted in Figure 7.14. Sizes of trees are quantified using number of nodes and number of leaves in decision tree. Figure shows average number of nodes and leaves when CrossTrees are tested on population sizes of 100, 200, 300, 400 and 500 across different models. Last column indicates the percentage reduction in nodes:

\[
\% \text{ reduction} = \frac{(\text{Avg nodes CT1}) - (\text{Avg nodes CT0})}{\text{Avg nodes CT1}} \times 100. \tag{7.1}
\]
Does CT perform better than NSGA-II?  
Are small trees smarter than big trees?  
% reduction in nodes

<table>
<thead>
<tr>
<th>Models</th>
<th>Does CT perform better than NSGA-II?</th>
<th>Are small trees smarter than big trees?</th>
<th>% reduction in nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOMO Flight</td>
<td>✓</td>
<td>✗</td>
<td>39.7</td>
</tr>
<tr>
<td>XOMO Ground</td>
<td>✓</td>
<td>✓</td>
<td>48.2</td>
</tr>
<tr>
<td>XOMO Osp</td>
<td>✓</td>
<td>✓</td>
<td>44.4</td>
</tr>
<tr>
<td>XOMO Osp2</td>
<td>✓</td>
<td>✓</td>
<td>68.8</td>
</tr>
<tr>
<td>XOMO All</td>
<td>✓</td>
<td>✓</td>
<td>36.9</td>
</tr>
<tr>
<td>POM 3A</td>
<td>✓</td>
<td>✓</td>
<td>59.4</td>
</tr>
<tr>
<td>POM 3B</td>
<td>✗</td>
<td>✗</td>
<td>69.8</td>
</tr>
<tr>
<td>POM 3C</td>
<td>✗</td>
<td>✗</td>
<td>66.7</td>
</tr>
<tr>
<td>Overall statistics</td>
<td>6/8</td>
<td>5/8</td>
<td>54.2</td>
</tr>
</tbody>
</table>

Figure 7.15: Table summarizes results of CrossTree. Majority of ✓ in first column indicates CT performed as good as NSGA-II and majority of ✓ on second column shows smaller trees are better compared to bigger trees. Third column shows percentage reduction is sizes of trees. It can be inferred that 6 out of 8 times CT performed as good as NSGA-II and 5 out of 8 times smaller trees are better than bigger trees after ≈ 54 percent of reduction in tree size.

Figure 7.15 summarizes Figures 7.6, 7.7, 7.8, 7.9, and 7.10 of XOMO model and also of Figures 7.11, 7.12, 7.13 of POM3 model. The Figure 7.15 clearly shows that CT performed as good as NSGA-II with 6 out of 8 models having CT0 performed better. Also, with 5 out of 8 models indicating CT0 performing better than CT1, it can be concluded that smaller trees are better than bigger trees which over fit information.

Average reduction in size of trees is also shown in 7.15. By noting the reduction in tree sizes, it can clearly be determined that using techniques like range reduction based on entropy and also feature subset selection based on infogain reduced tree size by almost half. Reduction in tree size has not effected performance of CrossTrees. This supports the Author’s aim of proposing data.
mining method “CrossTrees” as a visualization tool that helps optimize solution space.

7.7 Overcoming problems of NSGA-II

This section discusses about the problems of applying NSGA-II and also how CrossTrees addresses those problems. The problems faced by NSGA-II when applied to models XOMO and POM3 can be summarized as:

- **Time for generations**
  
  Time taken for generations of NSGA-II exponentially increase with population size.

- **Single point solutions**
  
  The solutions prescribed or optimized by NSGA-II are single points rather than a range of choices.

- **Lacks insight**
  
  The solutions prescribed by NSGA-II cannot be directly implemented when experts seek domain knowledge of why those solutions have improved.

These problems are avoided when CrossTrees is designed, after careful experimentation and thorough support of literature. The following sections explain how these problems are overcomed.

7.7.1 Time for generations

Time taken for both techniques NSGA-II and also CrossTrees are shown in Figure 7.16 and Figure 7.17. It can be clearly observed that CrossTrees is definitely twice as fast as NSGA-II. It was also observed by contained experiments by the Author that NSGA-II took about 2 and half days to produce half the number of generations (10) when a population of 800 is given whereas CrossTrees took only 4 hours. It can clearly be explored that NSGA-II takes about exponential time with increasing population unlike CrossTrees which takes quadratic time.
Figure 7.16: Execution time taken by CrossTrees and NSGA-II to generate solutions for XOMO

Figure 7.17: Execution time taken by CrossTrees and NSGA-II to generate solutions for POM3
Techniques | mqw | effort | months | defects | risks |
--- | --- | --- | --- | --- | --- |
NSGA 100 | m | 9 | 45 | 3 | 0 |
NSGA 200 | m | 4 | 34 | 1 | 17 |
NSGA 300 | m | 0 | 15 | 0 | 0 |
NSGA 400 | m | 2 | 21 | 0 | 14 |
NSGA 500 | m | 0 | 15 | 0 | 10 |
NSGA 100 | q | 17 | 34 | 9 | 16 |
NSGA 200 | q | 9 | 15 | 4 | 41 |
NSGA 300 | q | 0 | 2 | 0 | 29 |
NSGA 400 | q | 4 | 6 | 2 | 41 |
NSGA 500 | q | 0 | 0 | 1 | 37 |
NSGA 100 | w | 100 | 100 | 100 | 78 |
NSGA 200 | w | 55 | 97 | 31 | 75 |
NSGA 300 | w | 44 | 72 | 18 | 100 |
NSGA 400 | w | 15 | 78 | 5 | 88 |
NSGA 500 | w | 9 | 66 | 7 | 83 |
100 | 2445.4 | 38.9 | 26807.7 | 7.0 |
0 | 46.4 | 6.3 | 157.78 | 0.54 |

Figure 7.18: NSGA-II performance on XOMO All model for population size from 100 to 500

**Why choose a range of (100, 500)?**

Experiments conducted in current thesis show results for population size of up to 500. The motive behind the Author’s limited use of population size can be explained using following Figures 7.18 and 7.19. Figure 7.18 shows increasing population size from 100 to 500 on XOMO model with ALL constraints. It can clearly be observed that population size gets stable around 500 with negligible improvement in objectives in all three measures m,q and w. Figure 7.19 also supports this claim for POM models. The Figure 7.19 shown is constrained with POM3A model and a similar pattern was noted by the Author when other variations of POM model was used. Since, the improvement gets stable around populations up to 500, the Author tested hypothesis with sizes of 100, 200, 300 400 and 500. However, experiments with larger populations can be extended as future work of current thesis.
<table>
<thead>
<tr>
<th>Techniques</th>
<th>mqw</th>
<th>-cost</th>
<th>+completion</th>
<th>-idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSGA 100</td>
<td>m</td>
<td>13</td>
<td>92</td>
<td>26</td>
</tr>
<tr>
<td>NSGA 200</td>
<td>m</td>
<td>9</td>
<td>91</td>
<td>18</td>
</tr>
<tr>
<td>NSGA 300</td>
<td>m</td>
<td>8</td>
<td>84</td>
<td>10</td>
</tr>
<tr>
<td>NSGA 400</td>
<td>m</td>
<td>7</td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>NSGA 500</td>
<td>m</td>
<td>6</td>
<td>88</td>
<td>25</td>
</tr>
<tr>
<td>NSGA 100</td>
<td>q</td>
<td>10</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>NSGA 200</td>
<td>q</td>
<td>2</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>NSGA 300</td>
<td>q</td>
<td>0</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>NSGA 400</td>
<td>q</td>
<td>1</td>
<td>11</td>
<td>61</td>
</tr>
<tr>
<td>NSGA 500</td>
<td>q</td>
<td>2</td>
<td>11</td>
<td>48</td>
</tr>
<tr>
<td>NSGA 100</td>
<td>w</td>
<td>100</td>
<td>100</td>
<td>74</td>
</tr>
<tr>
<td>NSGA 200</td>
<td>w</td>
<td>45</td>
<td>100</td>
<td>74</td>
</tr>
<tr>
<td>NSGA 300</td>
<td>w</td>
<td>11</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>NSGA 400</td>
<td>w</td>
<td>14</td>
<td>100</td>
<td>87</td>
</tr>
<tr>
<td>NSGA 500</td>
<td>w</td>
<td>18</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>719.1</td>
<td>1.0</td>
<td>0.8</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>27.0</td>
<td>0.1</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Figure 7.19: NSGA-II performance on POM3A model for population size from 100 to 500

7.7.2 Single point solutions

When the process of CrossTrees is observed closely, it utilizes a cluster of solutions to be better or worse than other cluster of solutions. When individual ranges of individual decision variables of these clusters is used, experts can obtain a set of solutions instead of single solution spaces. The ranges of minimum and maximum for individual decision variables can be used by experts as ranges for better solutions instead of constraining themselves along single solutions on pareto frontier.

The result tables generated from the CrossTrees were from the populations generated using the ranges of better clusters along with decisions. The contrast sets included ranges of better clusters which were applied back to models to generate new data which seems to perform as good as NSGA-II. This proves that the ranges of better clusters can be used as cluster of solutions than single point solutions of NSGA-II.
7.7.3 Provides Insight

The decision tree used by CrossTrees acts as helper for worse solutions to get to a better branch. These decisions form the contrast set of CrossTrees. Applying these contrast sets of decisions is tested with NSGA-II to check if the decisions help models to improve solutions. The results shown in this chapter prove that the insight obtained from the decision trees work when applied back to models. For each model and population, CrossTree generates decision trees similar to Figures 4.2. CrossTrees helps busy businessmen to take decisions by finding the cluster current solution belongs to and proposing methods using decision tree to improve that. Hence, CrossTrees provides insight by providing managers and experts with decision trees of data. These decision trees can be studied along with their domain knowledge to understand the how CrossTrees optimized solution set; providing insight of data.
Chapter 8

Threats to Validity

Threats extend validity to results of any research. This section explains the threats involved in validating research. These possible threats to validity might exist within the experiments and conclusions drawn from them as most work is done by software. The possibility of human error or bias is very little as the subject is concerned with computations of software.

The chapter is divided in 4 parts, internal, external, construction and conclusion, each concerning the respective threats involved.

8.1 Internal Threats

The internal validity refers to whether a specific condition makes a difference or not. It also checks whether there is sufficient evidence to support claims that were made. Some of them are:

Selection Bias Selection bias refers to the selection of models for the experiments and how it effects the outcome of research. The models selected for this experiment have been under scrutiny for several years now, by experts in data mining and software engineering. The models have been chosen to reflect the bias of the Author towards experimental software engineering data and also availability of models. Since the motivation for research is software engineering and helping experts visualize software project decisions, these models were selected to test. This theory is
tested with one of MOEAs, performance of CrossTrees against other MOEAs is always a threat.

*Execution Bias* Though measures were taken to reduce external influence in terms of memory and data storage, there is possibility of background processes running which might not be constant throughout the experiments. Controlled experiments are proposed as future work to get more accurate measurements.

*Encoding Bias* This threat deals with programming and software flaws in the code written or used by the Authors. Since models are executed thousands of times, and also all attempts are made to sanitize the code by the Author and also advisor of the Author, possibility of encoding bugs is very minimal. Sanity checks are performed throughout the experiments to avoid any possibility of bugs.

*Analytic Bias* The conclusions drawn from results of experiments can validate the research. It is possible to use inappropriate methods to justify results. To protect the conclusions drawn from results, thesis is clearly defined and justified with the literature.

### 8.2 External Threats

These threats refer to validity of research in terms of generalizing conclusions made. No theory can be tested on all the tools available. And no conclusion can state that it is the best among all others. The algorithms implemented in this thesis are carefully tested against some of very popular and proven algorithms to avoid any favorable bias. As section 2.3.6 explains, NSGA-II has been implemented and tested in several domains and is proved to be one of the most widely used genetic algorithms. Also, base cases are always made available to reviewer to test the hypothesis against. In most cases the NSGA-II performs equally good as the method proposed in this thesis supporting this claim of not all algorithms are bad nor good when generalized across all the domains. Also, strong literature supports validity of this thesis against several other methods which could be tested with as availability of most tools are limited. There is also a possibility of threats in scalability, as this thesis might hold good for only limited number of objectives. With increasing objectives
validity of theory might be questioned. As this thesis is tested against 3 and 4 objective models, validity of thesis can be questioned when more objectives are added which can be tested in the future work of this thesis.
Chapter 9

Conclusion

This thesis has shown that data mining methods can be used to produce succinct visualizations which are readable by busy business user. In several experiments, CrossTrees outperformed or did as good as NSGA-II. The visualizations offered by CrossTrees were tested for validity and was proven that they could be used back on models to improve objectives.

This chapter proceeds with addressing research questions proposed in Chapter 1. The impact of this thesis on landscape visualization and multi-objective optimization is discussed. The latter part of the chapter end with discussing future work of this thesis.

9.1 Addressing Research Questions

The objective of this thesis was to evaluate whether optimizing tasks with data miners provide concise visualizations, can scale to large problems and offer succinct recommendations for busy end user. For that purpose, here are few research questions (Q) and related hypothesis (H):

Q1. Does applying data mining methods provide visualizations on how to improve a system?

H1. Data mining methods can be used to produce visualizations that recommend changes and improvements to busy human experts.

This hypothesis is valid when CT0 is used. Although, trade off between readability and perfor-
formance rarely affects this intuition. By analyzing results it is clear that smaller better trees provide
decisions that perform as good as existing methods. As shown in Figure 7.15 six out of eight times
CrossTrees performs as good as existing method.

Q2. Does data mining methods scale to big problems?

H2. Data mining methods can provide solutions faster than existing stochastic methods and hence can scale big problems more efficiently.

Yes, Compared to existing stochastic methods, CrossTrees is extremely fast and also can be
further optimized to suit the need to handle big data and computations with big CPU. CPU can be
efficiently utilized when the use of multiple cores. CrossTrees can be pre-built with existing data
and with discovery of new data, decision tree can be updated accordingly. Also, multiple cores can
be assigned to each leaf of decision tree to maximize utilization.

Q3. Can data mining methods provide flexibility in prescribed solutions?

H3. Instead of proposing rigid single solution for a problem, a range of solutions can be
proposed using data mining methods.

CrossTrees tests its efficiency with ranges of decision variables. Its performance is evaluated
based on those ranges, hence end user is provided with flexibility of applying solution tuned to
specific problem domain. By avoiding single point solutions data mining methods can obtain
flexibility in prescribed solutions.

Q4. Can concise visualizations affect performance compared to broad visualizations?

H3. Yes, concise readable visualizations does affect performance as they may not capture the
required decisions for improvement. Whereas, broad visualizations may be due to overfitting of
information that is irrelevant to end user.

Initial hypothesis proposed by the Author of readability affecting performance has been proved
by this thesis, in a positive way. CT0 version of CrossTrees produced readable visualizations which
when tested back with model returned performance scores better than NSGA-II most of the times.
This indicates the smaller trees can be better at predicting than big trees. Big decision trees always
face the problem of overfitting data by learning irrelevant information. Conclusions drawn from
this irrelevant data can affect population on whole. Figure 7.15 shows that five out of eight times smaller trees performed better than bigger trees with an average of 54 percent of reduction in tree size.

Therefore, data mining techniques suggest that smaller visualizations can perform better than complex unreadable visualizations. The human experts will have a choice to get an insight into the data in a glance with little compromise on performance. This would enable them to take decisions without spending time on intricacies of data mining or stochastic techniques.

9.2 Impact on landscape visualization and multi-objective optimization

This thesis has proved than data mining on landscape visualization can be a useful technique to optimize conflicting objectives of multi-criteria problems. This thesis tested and claimed CrossTrees to use visualizations to draw conclusions. These conclusions proved to be effective when applied back to initial data models. Further research can be carried on other data mining solutions with vast array of landscape visualization techniques. Since, this thesis is tested on domain of search-based software engineering, it can also be extended to other domains to check validity of this thesis.

9.3 Future work

Future work extends current thesis and exists to validate and reproduce results. This thesis can be extended to test its validity in other domains and also in new popular domain of “Big data”. Apart from extending into broad domain, future work of this thesis can be grouped into following categories:

1. Testing with more models: In this thesis only 2 models XOMO and POM3 were used, which can be extended to cover other complex models. Such experiments extend the validity of this thesis.
2. *Testing with large populations:* This thesis was tested on population size ranging from 100 to 500. Large population sizes can be used to test this theory and confirm the conclusions made in this thesis.

3. *Testing with other stochastic methods:* NSGA-II was used to benchmark CrossTrees which happens to be one of most widely used stochastic methods. There are several other stochastic methods that could be used to test conclusions made in this thesis.

4. *Parameter Tuning:* CrossTrees were tested with several methods involving many parameters. These parameters can be further tuned and tested on models on this thesis and also other models.

5. *Practical testing with end business users:* CrossTrees were tested on models running simulations of projects. This theory can be put in practice on real-time projects or real-time problems to validate and extend this thesis.
Bibliography


software partitioning based on simulated annealing and tabu search. Design automation for

[22] Usama Fayyad and Keki Irani. Multi-interval discretization of continuous-valued attributes

[23] Carlos M Fonseca, Peter J Fleming, et al. Genetic algorithms for multiobjective optimization:


[25] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing
360-degree compared. In Grid Computing Environments Workshop, 2008. GCE’08, pages

[26] Richard Fuchshuber and Márcio de Oliveira Barros. Improving heuristics for the next release
problem through landscape visualization. In Search-Based Software Engineering, pages 222–

the control variables for complex system behavior. Automated Software Engg., 17(4):439–
468, December 2010.


