Sensitivity Analysis in Multi–objective Next Release Problem and Fairness Analysis in Software Requirements Engineering

MSc Project Thesis

August, 2007

Jian Ren

MSc in Computing and Internet System, 2006/07

Department of Computer Science
School of Physical Sciences and Engineering
King's College London

Supervised by Professor Mark Harman

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis.

I am deeply indebted to Professor Mark Harman whose valuable advice, hints and warm inspiration guided me in all the time of this project. His unique way of supervision maximized the learning outcome of this project.

I am bound to thank two of Mark's PhD students Yuanyuan Zhang and Shin Yoo who gave me the initial acknowledge of the project and provided constant support.

I have furthermore to thank Research Assistant Afshin Mansouri and Zheng Li, PhD students Tao Jiang, for their help by offering me lots of suggestions for improvement.

I would like to thank Steffen Christensen from Science and Technology Foresight Directorate of Canada, who gave me useful suggestions on choosing the methods for statistical analysis.

Thanks to my landlord Jonathan Rake, my friends Chen Wang and the other Msc Project Students for all their help, support and interest in all the time of this project.

I would also like to give my special thanks to my parents whose ongoing love and support enabled me to complete this work.

Sensitivity Analysis in Multi–objective Next Release Problem and Fairness Analysis in Software Requirements Engineering

Jian Ren
MSc in Computing and Internet System, 2006/07

Supervised by
Professor Mark Harman

August, 2007

Abstract

This project is concerned with the *Next Release Problem*, a problem in *search-based* requirement engineering. The Next Release Problem is a 0-1 knapsack problem which is NP–hard problem. Previous work only considered using the single-objective algorithm to tackle the problem. Based on those work, both single-objective and multi-objective algorithm are implemented to analysis the problem. This project also develop the method of *Sensitivity Analysis* in multi–objective algorithms. A thorough quantitative and qualitative *Sensitivity Analysis* is also implemented.

In this project, *fairness* is the fist time to be considered in software engineering. It aims to maximize the customer's satisfaction and minimize resource of development at the same time, which is a typical multi–objective optimization problem. The most recent work from Y. Zhang 2007[11], which presents the evidence that NSGA-II is well suited to the *Multi–Objective Next Release Problem*, has been adopted in this project. The project presents a series of solutions for fairness consideration in planing the next release.

Contents

Acknowledgement **iii**

Abstract **v**

1 Introduction **1**
1.1 Multi–objective Next Release Problem 1
1.2 Search-Based Software Engineering 1
1.3 Sensitivity Analysis in NSGA–II 2
1.4 Fairness in Next Release Problem 2
1.5 Roadmap of the Project 2

2 Literature Review **3**
2.1 Search-based Software Engineering 3
2.2 Next Release Problem and MONRP 4
2.3 Sensitivity Analysis 4
2.4 Fairness in Software Requirement Engineering 5

3 Sensitivity Analysis **7**
3.1 Problem Statement of the Next Release Problem 7
3.1.1 Model of NRP using Greedy Algorithm 7
3.1.2 Problem Analysis for MONRP 8
3.2 Implementation, Experimental Setup 11
3.2.1 Greedy Algorithm 11
3.2.2 NSGA–II 11
3.2.3 Data Sensitivity Analysis 11
3.3 Result Analysis 13
3.3.1 Result from Greedy Algorithm 13
3.3.2 Data Sensitivity & Statistical Analysis 13
3.3.3 Result from NSGA–II 14
3.3.4 Data Sensitivity & Statistical Analysis 14

4 Fairness Analysis **21**
4.1 Problem Statement of Fairness Analysis 21
4.2 Two Principle for Objective Functions 22

4.3 Implementation and Experimental Setup 23
4.3.1 NSGA–II and Objective Formulations 23
4.3.1.1 Coverage 23
4.3.1.2 Resource Allocation 24
4.3.1.3 Customer's Satisfaction 24
4.3.2 Experimental Data Sets 25
4.4 Result Analysis 27
4.4.1 Scenario One: Fairness on Coverage 27
4.4.2 Scenario Two: Fairness on Value of Fulfilled Requirements . . 28
4.4.3 Scenario Three: Fairness on Value and Cost 28
4.4.4 Fairness Analysis Result Conclusion 29

5 Conclusion and Future Work **37**

A Source Code **39**
A.1 Code for Sensitivity Analysis 42
A.2 Code for Fairness Analysis 75

Bibliography **112**

List of Figures

2.1 The Optimization Process, adopted from [3] 4

3.1 Sensitivity Analysis Flow Chart 12
3.2 Result of Greedy algorithm comparing to the Expert Ranking. Greedy is much better than human expert. 13
3.3 Hamming Distance matrix, tweak rate = ±25% 14
3.4 Using Pseudo Random Number to stable the NSGA–II 15
3.5 Euclidean Distance from the original front to the 'tweaked' front . . 16
3.6 Scatter plots for tweak rate, cost, value and distance between fronts. 17

4.1 Sparsity Pattern of $Value$ matrix of three data sets 26
4.2 Fairness on Coverage: (Absolute Number of Fulfilled Requirements) 30
4.3 Fairness on Coverage: (Percentage of Fulfilled Requirements) 31
4.4 Fairness on Value: (Absolute Value of Fulfilled Requirements) 32
4.5 Fairness on Value: (Percentage of Value of Fulfilled Requirements) . 33
4.6 Fairness on Value and Cost, Result for Random Data Set 34
4.7 Fairness on Value and Cost, Result for Motorola Data Set 34
4.8 Fairness on Value and Cost, Result for Greer Data Set 35

LIST OF FIGURES

List of Tables

3.1 The Anonymous Feature Data from Motorola Inc. [2] 10
3.2 Spearmans Rank Correlation Coefficient Table for tweak rate and distance: for all the 35 requirements we have the majority of ρ_{rate} which are lager than the critical value 0.5. 18
3.3 Spearmans Rank Correlation Coefficient Table for (cost and distance) and (value and distance) : for all the tweak we have the significant of ρ which are lager than the critical value 19

4.1 The Cost of Randomly Generated Feature Data Set 25
4.2 The Value Matrix of Randomly Generated Feature Data Set 25
4.3 The Value Matrix of Feature Data Set taken from Greer 2004 [5] . . 26

Nomenclature

Acronyms

MOEA Multi-Objectives Optimization Evolutionary Algorithm

MONRP Multi-Objective Next Release Problem

NRP Next Release Problem

SA Sensitivity Analysis

NSGA-II Non-dominated Sorting Genetic Algorithm-II

SBSE Search-Based Software Engineering

Chapter 1

Introduction

This project will focuses on using *Search-based techniques* to tackle these complex optimization problems in the *Next Release Problem (NRP)*.

1.1 Multi–objective Next Release Problem

With or without using strategies and algorithms, people make decisions in their everyday lives. These decisions could be as simple as choosing which clothes to buy or as difficult as designing a space ship. Much less criterias are needed to be considered for making former decision than those needed for the latter one, but both of them could be considered as the problem of *minimizing the cost* while *maximizing the value*. The situation is relatively the same in the NRP. Before the next release of a software, the software engineers have to make the decision on which features should be included in that next version. There are lots of criterias to concern, for example, the budget for the next release, the cost of each feature, the functional ability of each feature and the total degree of satisfying the customers. The question of "how to decide which feature(s) should be added to the next release?" form the main body of the *Next Release Problem*.

1.2 Search-Based Software Engineering

To tackle the NRP, traditional software engineering is to rank the features into one list ordered by their score which is assigned by human experts. However, when the scale of the problem became large humanity became unreliable. The result from expert's ranking could be imprecise, on the other hand, could be easily outperformed by the search techniques.[2] In the field of *Search-Based Software Engineering(SBSE)*, metaheuristic search techniques, such as Greedy, Simulated Annealing and Genetic Algorithms, are introduced to provide a more precise and efficient way to tackle the problems. Features are selected by the algorithms. And the selection procedure completely depends on the numerical. In such a way, the human experts are released from making the selection which the algorithm is good at.

1.3 Sensitivity Analysis in NSGA–II

In the field of SBSE, humanity is replaced by the metaheuristic search techniques, nevertheless most of the numerical data are still estimated by experts. If there must be some human bias get involved in the process of tackling the problem, a *Sensitivity Analysis* could help to build confidence in the model by studying the uncertainties that are often associated with parameters in models. In order to investigate the data sensitivity problem, both Greedy and NSGA–II algorithm are applied to multiple versions of data sets, each containing small changes of costs of one specific feature. The analysis provide a decent reference to identify the data with significant affect on result. Thus, the bias could be minimized by performing a more accurate estimation on those critical inputs.

1.4 Fairness in Next Release Problem

Fairness is a very important concept in modern business world, and it is worthwhile to take it into serious consideration when we are planing the next release of our products. Fairness is a very simple concept at first, but its implementation becomes a complex goal whilst the definition of 'fair' derive various versions, or even conflicting concept. In order to provide fairness service to the customers, the concept of fairness is analyzed from several angles in this thesis.

1.5 Roadmap of the Project

The project first observe the results from the Greedy algorithm in order to understand the search space of the given problem better. Based on the results from the Greedy algorithm and the close observation of the original data, the NSGA-II algorithm is adopted; this algorithm enables us to obtain the globally optimal feature subset from any version of the data without actually going over all the possible solutions. The sensitivity analysis in single–objective search algorithm from Yoo [10] is successfully replicated in this project. And then I develop it to suit the multi–objective evolutionary algorithm. After all, an application of using NSGA–II to tackle the fairness analysis in Next Release Problem is studied.

Chapter 2

Literature Review

2.1 Search-based Software Engineering

'Search–based' techniques, such as *Genetic Algorithms*, *Simulated Annealing*, *Tabu Search*, has been applied successfully in a number of engineering problem[4, 7]. However, search-based software engineering is a recent effort to utilize the existing metaheuristic techniques to the problems of software engineering. In order to reformulate Software Engineering as a search problem, Harman [6] gives the two key ingredients for the application of search-based optimization to software engineering problem:

- The choice of the representation of the problem.
- The definition of the fitness function.

With these two simple ingredients, it becomes possible to implement search algorithm in the area of software engineering. Harman and Jones [7] provided three main steps of in the process of SESE implementation:

- a representation of the problem which is amenable to symbolic manipulation;
- a fitness function (defined in terms of this representation) and
- a set of manipulation operators.

In the optimization cycle, as illustrated in Figure 2.1, the first step is to analysis the problem and provide the search algorithm a appropriate representation of the problem. When the search techniques is applied for the problem, fitness function play the role of directing the search. Therefore we must be able to clearly express the desired property of the solution. The manipulation operator is required in order to move from one solution to another, based on the direction given by the fitness function.

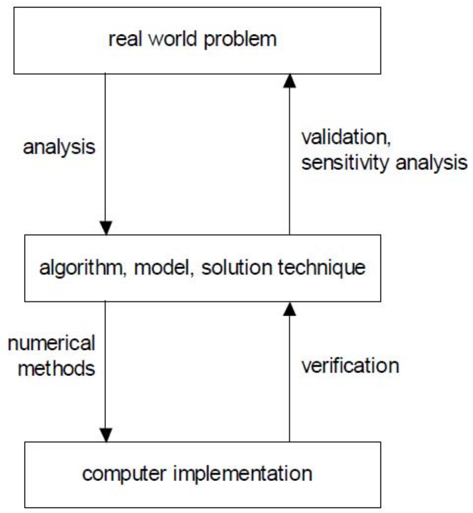


Figure 2.1: The Optimization Process, adopted from [3]

2.2 Next Release Problem and MONRP

The 'Next Release Problem'(NRP) is a problem in software requirement engineering. In the NRP, the goal is to find the ideal set of requirements that balance customer requests, resource constraints, and requirement interdependencies. Bagnall et al. [1] formulated NRP as a search problem and implement *Greedy algorithm*, *Hill Climbing* and *Simulated Annealing*. Recently, Zhang et al. [11] introduced multi–objective algorithm to tackle the NRP for the first time. Zhang et al. applied a variety of techniques, including *NSGA–II*, *Pareto GA*, *single objective GA* and *Random Search* technique to a set of synthetic data created to model features for the next release. The result show that NSGA–II outperform the other algorithm in finding the middling part of the Pareto front in large problems and single objective GA (applied to the unified objective function) performs better in finding extreme regions in some circumstance.

2.3 Sensitivity Analysis

As illustrated in Figure 2.1, sensitivity analysis techniques is usually performed in the last phase of the optimization cycle. It aims to identify how 'sensitive' the mathematical model (or problem model) is to the changes in the value of the input parameters and to the changes in the structure of the model. Yoo [10] performed the parameter sensitivity analysis on the a series of tests in which the input cost of the features was modified 'one–at–a–time'. Yoo's work illustrated how the behavior of Hill climbing search responds to the changes in the input data. Sensitivity analysis is a useful tool in model building as well as in model evaluation.

2.4 Fairness in Software Requirement Engineering

> *"...Fairness is not an economic concept. If you want to talk fairness, you have to leave the department of economics and head over to philosophy ..."*

Like Mankiw said in his paper about fair taxation: "Fair Taxes? Depends What You Mean by 'Fair' " [8], fairness is not a simple engineering concept neither in software requirement engineering. The concept of fairness is very simple at first, which means treating people in a fair, equal and right way. But it ends up with a very complex situation. In the modern taxation system, there is are several principles for fair tax. However, people who have different opinions on what is fairness always argue about they have not been treated fairly. Thus, the definitions of fairness become the most important concept in the implementation of fairness analysis. To the best of our acknowledgment, there is previous work on fairness analysis in software engineering. And multi–objective evolutionary algorithm is good for combining different opinions to optimize the solutions. It worthwhile to have the consideration of fairness in solving the next release problem.

Chapter 3

Sensitivity Analysis

3.1 Problem Statement of the Next Release Problem

3.1.1 Model of NRP using Greedy Algorithm

In the project, a set of real data from Motorola Inc. was used. These software features are free of interference, so that any combination of them can be implemented into the next release. The data contained 35 software features that can be implemented into the future model of a mobile phone. Dependency relations between the these 35 features were very sparse, so the issue of the feature dependency was ignored in current stage of the project.

As shown in Table 3.1 (Page 10), the *i–th* feature f_i $(1 \leq i \leq 35)$ has its own *Cost*: $cost_i$, which represents the effort and resource it takes to implement the feature. The desirability of each feature is represented by two ordinal scale parameters: *Customer Priority*: p_i, and *Expected Revenue*: e_i. Customer priority represents the relative priority given to each customer group which required the particular feature. It ranges from 1 to 4, with 1 being the highest and 4 the lowest. Expected revenue represents the possible revenue that each feature is expected to bring to the company. It ranges from 1 to 3, with 3 being the biggest revenue and 1 the smallest. Harman et al. [2] adopted the following fitness function, $\mathcal{Z}'$, to evaluate the *Fitness Value* for feature f_i:

$$\mathcal{Z}'(f_i) := \frac{e_i}{p_i}$$

Let the set of those features which are chosen for the next release is denoted by F', which is a subset of $F = \{f_1, f_2, \cdots, f_{35}\}$. And vector $\overrightarrow{x} = \{x_1, x_2, \cdots, x_{35}\}$ denote the solution set of which features are chosen in the next release. In this vector, x_i is '1' if the *i–th* feature is selected; and '0' otherwise. Then:

$$F' = \{f_i \in F : x_i = 1\}$$

The overall fitness value for a F' is the sum of fitness value of all the chosen feature:

$$\mathcal{Z}(F') := \sum_{i=1}^{35} \mathcal{Z}'(f_i) \cdot x_i$$

Similarly, the overall cost is:

$$cost(F') := \sum_{i=1}^{35} cost_i \cdot x_i$$

In addition, the experts provide a ranking of all the feature. This help to produce a list of budgets $\overrightarrow{B} = \{b_1, b_2, \cdots, b_{35}\}$, where:

$$b_i = \sum_{n=1}^{i} cost_n$$

In this context, the question of how to tackle this optimization problem could be translated into: How to find out the F' with:

- Maximum overall fitness value:

$$\text{Maximum } \mathcal{Z}(F') = \sum_{i=1}^{35} \mathcal{Z}'(f_i) \cdot x_i$$

- Subject to:

$$cost(F') \leq b_i$$

The implementation of Greedy algorithm is shown in section 3.2.1 and the associated result and analysis is discussed in section 3.3.1

3.1.2 Problem Analysis for MONRP

In order to apply multi-objective algorithm to NRP, we need to reformulate it into a Multi-Objective Optimization Problem. This project adopted the Multi-Objective Next Release Problem(MONPR) model from the recent work of Y. Zhang et al. [11]. In the current stage of the project, the requirement data for MONRP is random generated. For each existing software system, there is a set of customers,

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

where m is the number of customers. Let R denote all the possible requirements[1] from the customers:

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

[1] Regarding to this report, the terms of 'feature' and 'requirement' are interchangeable.

where n is the number of the total requirements. Each requirement $r_i (1 \leq i \leq n)$ has its own $cost_i$ which represents the effort and resource it takes to implement the requirement:

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

In addition, each customer $c_j (1 \leq j \leq m)$ has a specific degree of importance to the company which is denoted by w_j:

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

where $w_j \in [0, 1]$ and $\sum_{j=1}^{n} w_j = 1$. Furthermore, each customer c_j assigns a *value* to every requirement r_i denoted by: $value(r_i, c_j)$, which equals to 1 if the j–th customer needs the i–th requirement; and '0' otherwise. Then the overall $score_i$ for requirement r_i could be calculate by:

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

which represents the overall importance of the i–th requirement. The decision vector $\vec{x} = \{x_1, x_2, \cdots, x_n\} \in \{0, 1\}$ determines which requirements are added in the next release. In this vector, x_i is '1' if the i–th requirement is selected; and '0' otherwise. After all the fitness function, $\mathcal{Z}$,of a decision $\vec{x}$ is defined as:

$$\mathcal{Z}(\overrightarrow{x}) := \sum_{i=1}^{n} score_i \cdot x_i$$

Multiple objective function are needed to formulate the constraints, After all, it is able to adopt two objectives functions to formulate this multi-objective optimization problem:

- Maximize the fitness value:

$$\text{Maximize: } \mathcal{Z}(\overrightarrow{x}) = \sum_{i=1}^{n} score_i \cdot x_i$$

- Minimize the total cost:

$$\text{Minimize: } \sum_{i=1}^{n} cost_i \cdot x_i$$

The implementation of NSGA–II is shown in section 3.2.2, and the result and analysis is discussed in section 3.3.3.

Table 3.1: The Anonymous Feature Data from Motorola Inc. [2]

Feature No.	Cost	Customer	Revenue	Fitness Value	Experts Ranking
1	100	3	3	1	1
2	50	3	3	1	2
3	300	1	3	3	3
4	80	1	3	3	4
5	70	1	3	3	5
6	100	3	3	1	6
7	1000	3	3	1	7
8	40	2	3	1.5	8
9	200	2	3	1.5	9
10	20	2	1	0.5	10
11	1100	2	3	1.5	11
12	10	1	3	3	12
13	500	1	3	3	13
14	10	1	1	1	14
15	10	1	3	3	15
16	10	1	2	2	16
17	20	3	1	0.33333	17
18	200	3	1	0.33333	18
19	1000	3	3	1	19
20	120	2	2	1	20
21	300	2	2	1	21
22	50	2	1	0.5	22
23	10	2	2	1	23
24	30	2	3	1.5	24
25	110	1	2	2	25
26	230	1	2	2	26
27	40	1	1	1	27
28	180	1	2	2	28
29	20	1	2	2	29
30	150	1	2	2	30
31	60	1	3	3	31
32	100	1	1	1	32
33	400	3	3	1	33
34	80	3	1	0.33333	34
35	40	4	1	0.25	35

3.2 Implementation, Experimental Setup

3.2.1 Greedy Algorithm

Several simple greedy algorithms have been developed and applied to the problem set. These algorithms are constructive in nature and start with an empty set of satisfied features. At each iteration, a feature is added to the set until no further additions can be made without exceeding the resource bound. The choice of which customer to select at each iteration is guided by a set of simple metrics.

First of all, all the features shown in Table 3.1 are sorted according their fitness value or any other columns. Then all the features with the highest fitness value will be selected into the solution until the budget bound has been reached. The algorithm of the process of Greedy is described in Algorithm 1. The result will be analysis in Section 3.3.1.

Algorithm 1: Greedy Algorithm

input : N_b:number of budgets; N_f:number of features; cost; budgets
output: *solution*

```
for i ← 1 to N_b do
    for j ← 1 to N_f do
        if actualCost + cost(j) ≤ budget(i) then
            actualCost ⟵ actualCost + cost(j);
            solution(j, i) ⟵ 1;
        end
    end
    solution(N_f + 1, i) ⟵ actualCost;
    actualCost ⟵ 0;
end
```

3.2.2 NSGA–II

The implementation of NSGA–II algorithm is adopted from the recent work of Zhang [11]. Initially, a random parent population P_0 is created. The population size is N. The population is sorted using the non-dominated relations. Each solution is assigned a fitness value equal to its non-domination level. Binary tournament selection, crossover, and mutation operators are used to create a child population Q_0 of size N. Then the NSGA–II procedure goes to the main loop which is described in Table 2. The result is shown in Section 3.3.3.

3.2.3 Data Sensitivity Analysis

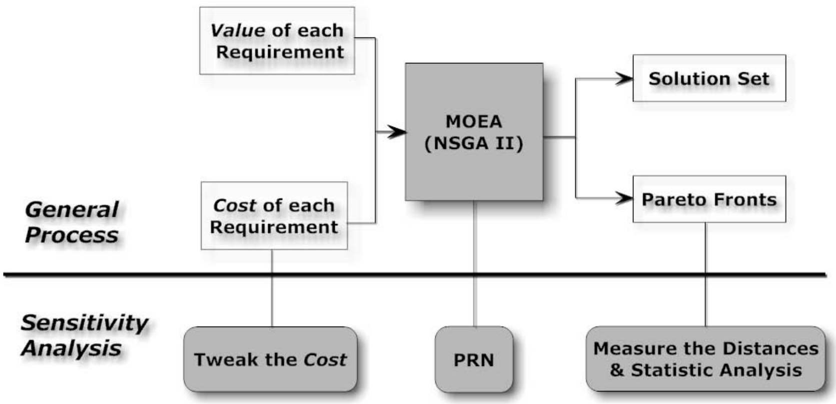
In this project, due to the models applied are relatively small enough to be solved quickly, a *brute force* approach is implemented for sensitivity analysis: simply modify

Algorithm 2: NSGA–II Algorithm

while *not stopping rule* **do**
  Let $R_t = P_t \cup Q_t$;
  Let F = fast-non-dominated-sort(R_t);
  Let $P_{t+1} = \emptyset$ *and* $i = 1$;
  while $|P_{t+1}| + |F_i| \leqslant N$ **do**
    Apply crowding-distance-assignment(F_i);
    Let $P_{t+1} = P_{t+1} \cup F_i$;
    Let $i = i + 1$;
  end
  $Sort(F_i, \prec n)$;
  Let $P_{t+1} = P_{t+1} \cup F_i[1 : (N\text{-}|P_{t+1}|)]$;
  Let Q_{t+1} = make-new-pop(P_{t+1});
  Let t = t + 1;
end

the initial input data and run the algorithm repeatedly to see how the result changes. Figure 3.1 And *Spearman's rank correlation coefficient* is used to analysis how the changes on result relates to the modify of initial data. It is worthwhile to mention here that the indeterminacy of genetic algorithm causes uncertain interference on the performance of sensitivity analysis. Pseudo random number is introduced to NSGA–II to overcome this problem. The result analysis is discussed in Section 3.3.2.

Figure 3.1: Sensitivity Analysis Flow Chart



3.3 Result Analysis

3.3.1 Result from Greedy Algorithm

Figure 3.2 illustrated the fitness value for 35 budgets obtained by the Greedy algorithm. For comparison, there are 8 results are plotted on the same figure, including 7 from greedy algorithms and .the human expert ranking. For most of the budget values, the Greedy algorithm produces results which significantly outperforms the results from the human experts ranking.

3.3.2 Data Sensitivity & Statistical Analysis

Hamming Distance is used to measure the distance between 'tweaked solution' and the 'original solution'. Figure 3.3 is Hamming Distance of the 'tweaked' results and the original solution. It is obvious that any increase in the cost of some feature will result in a decreased fitness value, whereas any decrease in the cost of some feature will bring about the opposite effect. Comparing with the sensitivity analysis result from Yoo [10] the result obtained so far is relatively similar. A detailed result analysis might be generated in next stage of the project.

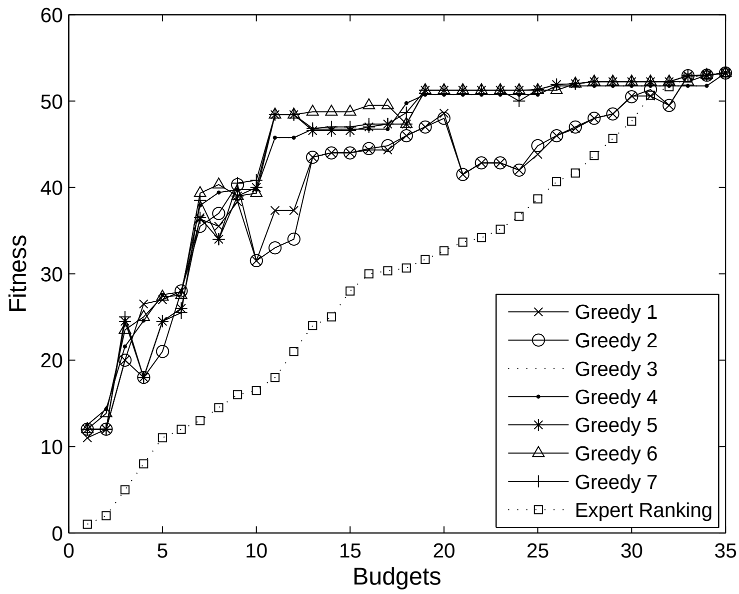
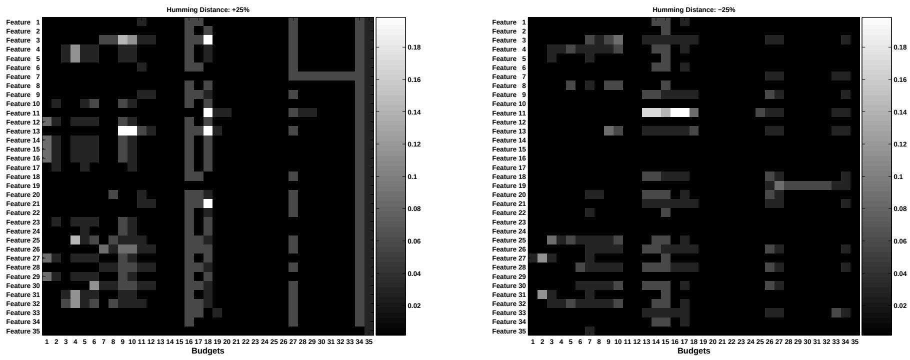


Figure 3.2: Result of Greedy algorithm comparing to the Expert Ranking. Greedy is much better than human expert.



(a) Tweak Rate: +25%

(b) Tweak Rate: −25%

Figure 3.3: Hamming Distance matrix, tweak rate = ±25%

3.3.3 Result from NSGA–II

By adopting the recent work from Zhang [11], NSGA–II is applied to a set of synthetic data which is created to model features for the next release.

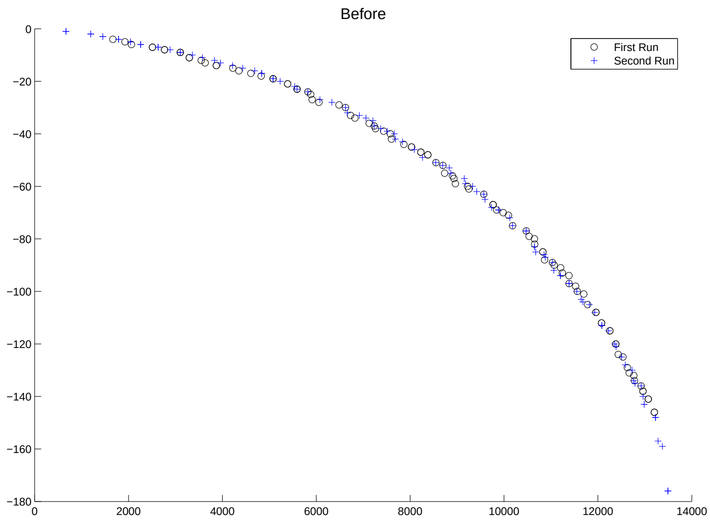
3.3.4 Data Sensitivity & Statistical Analysis

Due to the indeterminacy of the NSGA–II itself and the fact that it can not guarantee obtaining the global optimum, every 'run' of the implementation would provide the different solution result. In order to perform the sensitivity analysis, we need to distinguish the different between the indeterminacy of the algorithm itself and the changed caused by the 'tweak' on the input data. In Figure 3.4,

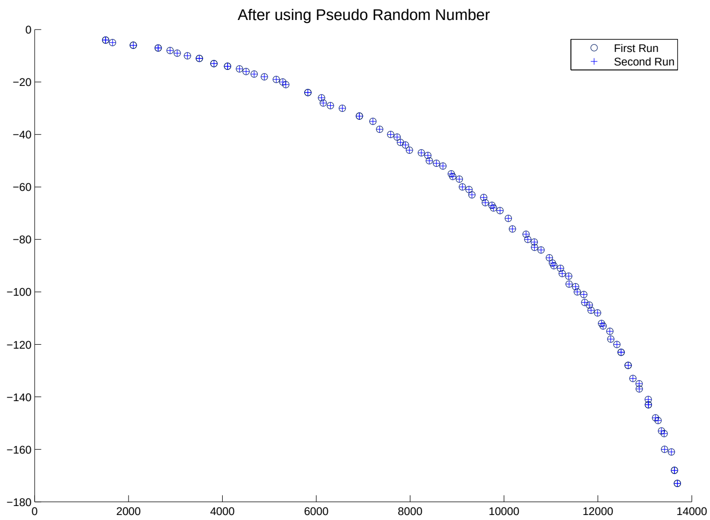
Furthermore, from Greedy to NSGA–II the distance between two solutions is changed to two sets of solutions. In order to measure the distance between two set of solutions, which actually are two Pareto fronts, the Generation Distance is adopted [9]. Figure 3.5 shown the Euclidean distance from the original front to the 'tweaked' front. It is illustrated that the distance will increase when the tweak rate is increased.

In order to observe the relation between the input and output parameters, the results were statistically analyzed by Spearman's rank correlation coefficient. The correlation between of three pairs of parameters are calculated: (Tweak Rate and Distance),(Cost and Distance) and (Value and Distance). As shown in Figure 3.6 the scatter plots do not suggest any strong reason to believe that these correlations are linear, so Spearmans rank correlation coefficient is used to describe the relationship between three pairs of separate variables, without assuming any linear relation between them.

Table 3.3.4 shows a strong correlation between the tweak rate and the distance. This mean the more of tweak rate on the input cost will cause more on distance of



(a) Before using PRN, NSGA–II could provide different fronts in different run.



(b) By using PRN, NSGA–II became stable.

Figure 3.4: Using Pseudo Random Number to stable the NSGA–II

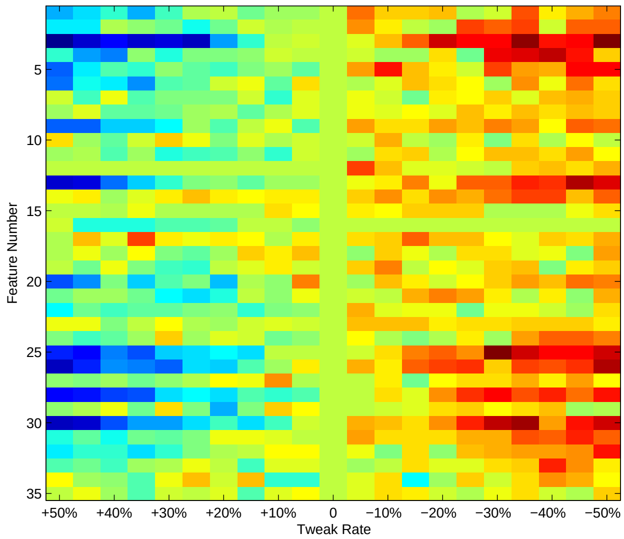
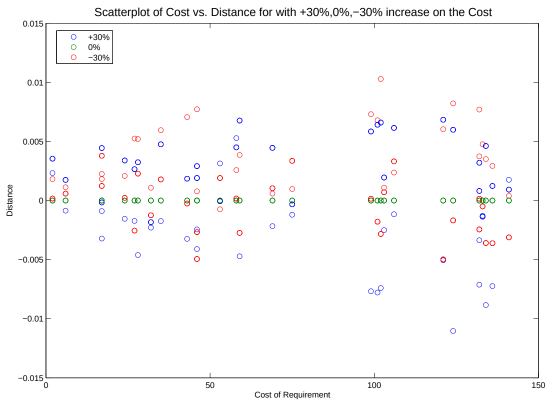


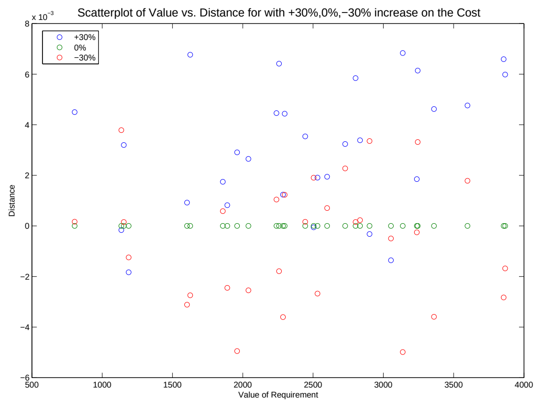
Figure 3.5: Euclidean Distance from the original front to the 'tweaked' front

output solution. On the other hand, as shown in Table 3.3.4, Spearman's rank correlation coefficient does not indicate a strong relation between the two pair parameters (cost, distance) and (value, distance).

(a) Obviously: More of Tweak causes More of Distance



(b) Inconspicuous trend for Cost and Distance



(c) Inconspicuous trend for Value and Distance

Figure 3.6: Scatter plots for tweak rate, cost, value and distance between fronts.

Requirement	ρ_{rate}	p	*Requirement*	ρ_{rate}	p
1	0.6143	0.0031	16	0.1818	0.4302
2	0.9078	0.0000	17	0.5013	0.0206
3	0.9325	0.0000	18	0.8429	0.0000
4	0.7701	0.0000	19	0.3636	0.1051
5	0.8065	0.0000	20	0.4312	0.0510
6	0.9727	0.0000	21	0.4740	0.0299
7	0.6545	0.0013	22	0.4766	0.0289
8	0.7688	0.0000	23	0.7260	0.0002
9	0.9377	0.0000	24	0.3532	0.1162
10	0.9390	0.0000	25	0.0701	0.7626
11	0.9000	0.0000	26	0.1961	0.3942
12	0.7519	0.0001	27	0.0104	0.9643
13	0.9390	0.0000	28	0.1143	0.6218
14	0.8675	0.0000	29	-0.1195	0.6060
15	0.7714	0.0000	30	0.5169	0.0164

Table 3.2: Spearmans Rank Correlation Coefficient Table for tweak rate and distance: for all the 35 requirements we have the majority of ρ_{rate} which are lager than the critical value 0.5.

Rate	ρ_{cost}	p_{cost}	ρ_{value}	p_{value}
+50%	0.2012	0.2863	0.3290	0.0758
+45%	0.0855	0.6531	0.3713	0.0434
+40%	−0.0513	0.7878	−0.1253	0.5096
+35%	0.0513	0.7878	0.4189	0.0212
+30%	0.0973	0.6089	0.3615	0.0497
+25%	0.1287	0.4979	0.5253	0.0029
+20%	0.2215	0.2395	0.0496	0.7946
+15%	0.0373	0.8450	0.0870	0.6476
+10%	0.1145	0.5470	0.4394	0.0151
+5%	0.1082	0.5692	0.1097	0.5640
0%	0.5003	0.0049	0.5000	0.0049
−5%	−0.0408	0.8304	−0.0603	0.7516
−10%	−0.1111	0.5588	−0.2872	0.1238
−15%	−0.2831	0.1296	−0.1862	0.3245
−20%	−0.0021	0.9912	−0.1497	0.4297
−25%	−0.2655	0.1562	−0.1181	0.5341
−30%	−0.4580	0.0109	0.0162	0.9321
−35%	−0.3062	0.0998	−0.3059	0.1002
−40%	−0.2673	0.1533	−0.2917	0.1179
−45%	−0.4284	0.0182	−0.4202	0.0208
−50%	−0.4175	0.0217	−0.2877	0.1232

Table 3.3: Spearmans Rank Correlation Coefficient Table for (cost and distance) and (value and distance) : for all the tweak we have the significant of ρ which are lager than the critical value

Chapter 4

Fairness Analysis

In the previous work, MONRP was tackled from the *software engineer's point of view*. The result could provides the solutions which balance the constrains between the customer's requests and the resources limitations. A new angle of the MONRP is explored: *Fairness Analysis*. The principle motivation of fairness analysis is try to balance the requirements fulfillments *between the customers*. It could provide a convincing reference from the view of *marketing* and help the decision maker to maintain a record of fairness between customers. Firstly, the problem statement of fairness analysis is introduced. Secondly, the procedure of the experimental setup is described. And the result of the experiment is analyzed in the last section of this chapter.

4.1 Problem Statement of Fairness Analysis

The problem model of fairness analysis is slightly different from the one of MONRP, which has been addressed in section 3.1.2. For each existing software system, there is a set of customers,

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

where m is the number of customers. Let R denote all the possible requirements from the customers:

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

where n is the number of the total requirements. Each requirement $r_i (1 \leq i \leq n)$ has its own $cost_i$ which represents the effort and resource it takes to implement the requirement:

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

In addition, each customer $c_j (1 \leq j \leq m)$ assign a value $v_{j,i}$ to each requirement $r_i (1 \leq i \leq n)$ based on the contribution to the business value of the product by that requirement.[1]

$$Value = \begin{Bmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,i} & \cdots & v_{1,n} \\ v_{2,1} & v_{2,2} & \cdots & v_{1,i} & \cdots & v_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ v_{j,1} & v_{j,2} & \cdots & v_{j,i} & \cdots & v_{j,n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{m,1} & v_{m,2} & \cdots & v_{m,i} & \cdots & v_{m,n} \end{Bmatrix}$$

The decision vector $\overrightarrow{x} = \{x_1, x_2, \cdots, x_n\}$ denote the solution set of which features are chosen in the next release. In this vector, x_i is '1' if the *i–th* feature is selected; and '0' otherwise.

4.2 Two Principle for Objective Functions

As a matter of fact, the next release is not supposed to fulfilled all the requirement claimed by every customers. Some of the requirement could be fulfilled and the other must be eliminated. For the purpose of provide the fairness between customers during the selection process, two principles of fairness are proposed to determine whether the customers is been treated fairly: *fair–resources–distribution principle* and *fair–customer–satisfaction principle*.

- *Fair–Resource–Distribution Principle*
  The *fair–resource–distribution principle* holds that the over all resource should be fairly allocated to each customer. One implication of this principle is the measurement of distributed resource, which is the overall cost of the next release in this case. If every customers all have the same amount of fulfilled requirements in terms of the cost, then the fair–resource–distribution principle is satisfied.

- *Fair–Customer–Satisfaction Principle*
  However, suppose that two individual customer have fulfilled requirements which are both worth £1000, but one has claimed for only £1000 requirements then he got 100% satisfied and the other claims for ten thousands pounds and only got 10% satisfied. Obviously, they are not in equal positions. They have not been treated fairly under the fair–customer–satisfaction principle, which states that each customers should have fair degree on satisfaction.

The first principle aims to give everyone a equal absolute amount whilst the second principle aims to give everyone a equal percentage of fulfilled requirement. In an ideal scenario, if all the customer claim for the same amount of requirement, these two principle will share a unique formula. Because if two fractions share a

[1] In Table 3.1, the *Revenue* is the attribute of the requirement itself, which is independent of customers. Here, the *Value of Requirement*'s Value is assigned by customers.

same denominator, asking for fairness (equivalency) on numerator is equal to asking for fairness on the value of the two fractions. As a matter of fact, however, we can not expect every customer claims for the same amount of the requirement. So these two principles compose a pair of conflicting objective. For example, if the company try to fairly satisfy its customer under the standard of the second principle, then every customer has the same percentage of fulfilled requirements, thus it has to be unfair under the standard of the first principle, because different customer ask for different amount of requirements. The actual objective formula will be shown in the next section along with the NSGA–II algorithm which is designed for multiple conflicting objective optimization problem.

4.3 Implementation and Experimental Setup

4.3.1 NSGA–II and Objective Formulations

In the implementation of this project, the two fairness principle are taken into consideration in order to analysis the confliction between them. The overall cost of the next release is also treated as an objective to minimize for the purpose to explore the whole set of Pareto–optimal frontier which is a valuable source for the decision makers to decide which requirements to select at different budget levels.

4.3.1.1 Coverage

The Coverage of the customer's requirement is the first meaningful index for the analysis of customer's satisfaction. Two kinds of coverage are considered in experimental: absolute number of coverage and percentage of coverage.

In formula 4.1, we aim to maximize the fairness by giving every customer the some absolute number of fulfilled requirements, which means minimize the standard deviation of it:

$$\text{Minimize: } f_1(\overrightarrow{x}) = \sigma(\overrightarrow{CVA}) \tag{4.1}$$

where the vector $\overrightarrow{CVA} = \{CVA_1, \cdots, CVA_m\}$ represents the absolute number of fulfilled requirements for each customer.

$$CVA_j = \sum_{i=1}^{n}(x_i \wedge v_{j,i})$$

Similarly in formula 4.2, we aim to maximize the fairness by giving every customer the some percentage of fulfilled requirements, which means minimize the standard deviation of it:

$$\text{Minimize: } f_2(\overrightarrow{x}) = \sigma(\overrightarrow{CVP}) \tag{4.2}$$

where the vector $\overrightarrow{CVP} = \{CVP_1, \cdots, CVP_m\}$ represents the percentage of fulfilled requirement for each customer.

$$CVP_j = \frac{CVA_j}{M_j} \times 100\%$$

In this vector, M_j is the total number of requirements claimed by the j–th customer and CVA_j is the number of fulfilled ones.

4.3.1.2 Resource Allocation

The budget of the Next Release Problem is the main resource to consider. Under the first principle, we aims to provides each customer the same amount of resource which means every customer have the fulfilled requirement with same amount of costs. To translate the objective into formula, we aims to minimize the standard deviation of fulfilled cost between customers in equation 4.3:

$$\text{Minimize: } f_3(\overrightarrow{x}) = \sigma(\overrightarrow{C}) \tag{4.3}$$

where the vector $\overrightarrow{C} = \{C_1, \cdots, C_m\}$[1] represents the costs of fulfilled requirement for each customer. In this vector, $C_j (1 \leq j \leq m)$ is the j–th customer's fulfilled cost:

$$C_j = \sum_{i=1}^{n} (cost(i) \times (x_i \wedge v_{j,i}))$$

The decision vector $\overrightarrow{x} = \{x_1, x_2, \cdots, x_n\}$[2] denote the solution set of features which are chosen in the next release. In this vector, x_i is '1' if the i–th feature is selected; and '0' otherwise. On the other hand, the vector of $(x_i \wedge v_{j,i}$ denote the set of feature which are chosen in the next release and also required by the j–th customer. In this vector, $(x_i \wedge v_{j,i})$ is '1' only if the i–th feature is chosen by the j–th customer and in the next release at the same time, otherwise '0'.

4.3.1.3 Customer's Satisfaction

The value assigned by each customer for each requirement is very important for the evaluation of customer's satisfaction. In objective function 4.4 is considered to maximizing the fairness on giving each customer the requirements with the same amount of value, which means minimizing the standard deviation of the fulfilled values between customers:

$$\text{Minimize: } f_4(\overrightarrow{x}) = \sigma(\overrightarrow{VA}) \tag{4.4}$$

where the vector $\overrightarrow{VA} = \{VA_1, \cdots, VA_m\}$ represents the fulfilled value for each customer. In this vector, similarly, $v_{j,i} (1 \leq j \leq m, 1 \leq i \leq n)$ is the value assigned by the j–th customer to the i–th feature, and $VA_j (1 \leq j \leq m)$ is the j–th customer's fulfilled value:

$$VA_j = \sum_{i=1}^{n} (v_{j,i} \times (x_i \wedge v_{j,i}))$$

[1] m is the number of customers
[2] n is the number of requirements

Table 4.1: The Cost of Randomly Generated Feature Data Set

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
100	50	300	80	70	100	1000	40	200	20
r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
1100	10	500	10	10	10	20	200	1000	120
r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}	r_{29}	r_{30}
300	50	10	30	110	230	40	180	20	150

Table 4.2: The Value Matrix of Randomly Generated Feature Data Set

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}
c_1	1	3	3	5	2	0	1	4	3	2	5	4	2	1	3
c_2	4	1	1	2	3	3	0	1	1	2	2	1	4	1	1
c_3	2	1	1	2	4	3	2	4	4	1	2	2	2	2	1
c_4	3	0	2	0	1	2	1	2	4	0	3	2	2	4	5
c_5	2	0	4	1	3	2	2	5	5	4	1	1	3	0	3
	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}	r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}	r_{29}	r_{30}
c_1	2	2	0	1	4	3	2	4	4	5	0	1	2	2	1
c_2	2	1	3	1	3	0	1	4	5	4	1	0	0	3	2
c_3	1	3	0	5	1	2	2	0	0	2	3	0	0	4	3
c_4	3	3	1	0	5	1	0	0	4	1	0	1	4	2	4
c_5	3	1	1	3	2	3	4	5	3	4	2	0	5	3	4

Similarly, the following objective function (4.5) is consider for maximizing the fairness on giving each customer the requirements with the same percentage out of what they claim for, which means minimizing the standard deviation of the percentage of fulfilled values between customers:

$$\text{Minimize: } f_5(\overrightarrow{x}) = \sigma(\overrightarrow{VP}) \tag{4.5}$$

where the vector $\overrightarrow{VP} = \{VP_1, \cdots, VP_m\}$ represents the percentage of fulfilled value for each customer.

$$VP_j = \frac{VA_j}{\sum_{i=1}^{n} v_{j,i}} \times 100\%$$

4.3.2 Experimental Data Sets

Three sets of data are using to perform the fairness analysis.

The first data set is randomly generate according to the problem model, Table 4.1 describes the *Cost* of each requirement and Table 4.2 describes the *Value* matrix of the data set.

Table 4.3: The Value Matrix of Feature Data Set taken from Greer 2004 [5]

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
c_1	4	2	1	2	5	5	2	4	4	4
c_2	4	4	2	2	4	5	1	4	4	5
c_3	5	3	3	3	4	5	2	4	4	4
c_4	4	5	2	3	3	4	2	4	2	3
c_5	5	4	2	4	5	4	2	4	5	2
	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
c_1	2	3	4	2	4	4	4	1	3	2
c_2	2	3	2	4	4	2	3	2	3	1
c_3	2	4	1	5	4	1	2	3	3	2
c_4	5	2	3	2	4	3	5	4	3	2
c_5	4	5	3	4	4	1	1	2	4	1

The second data set is taken from Motorola Inc.[2] which was displayed in Table 3.1, Section 3.1.2. We consider that if customer c_i claims for requirement r_j, then $v_{j,i}$ is set to '1', $v_{j,i}$ is set to '0'otherwise.

The third data set is taken from Greer 2004 [5], which was shown in Table 4.3. Because Greer's data does not contain the information about cost of each requirement, for the purpose of feeding this useful industrial data into our algorithm, all the cost is set to 1.

For summary of these three data set, in Figure 4.1, it shows the Sparsity Pattern of their $Value$ matrixes. Each spot represents a non-zero value assigned by customer.

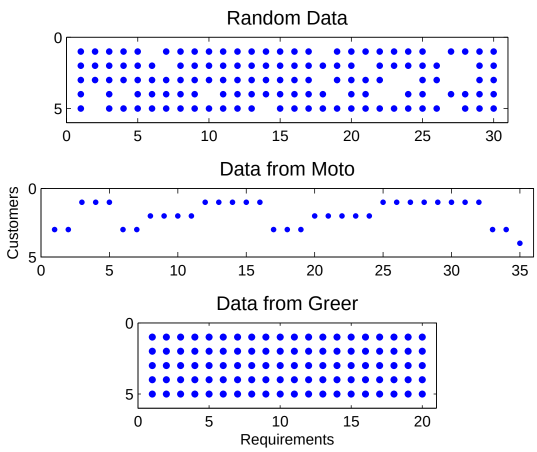


Figure 4.1: Sparsity Pattern of $Value$ matrix of three data sets

4.4 Result Analysis

4.4.1 Scenario One: Fairness on Coverage

In order to analyze the fairness on Coverage, two implementation have been done. The results are shown in Figure 4.2 and 4.3 respectively. In order to demonstrate the progress of NSGA–II finding the optimal solutions, the initial populations, middle populations and the final optimal are plotted in the figures. Each point represents a subset of requirements for the next release, the small '$\bullet$', '$*$' and solid '$\triangle$' is respectively for the representation of the initial populations, middle population and final optimal pareto front in the implementations.

Firstly, two objectives are considered for the optimization:

$$\text{Maximize: } mean(\overrightarrow{CVA})$$

$$and$$

$$\text{Minimize: } \sigma(\overrightarrow{CVA})$$

which aim to maximize the average coverage for all the customers whilst minimize the standard deviation of the absolute number fulfilled requirements for each customer.

In the result shown in Figure 4.2, all the populations are plotted for three data sets. We can observe that the objective function is guiding the population to move towards the optimal front. The regular optimal front are shown in the results for both Random data set and Motorola's data set. On these two fronts, the standard deviation of fulfilled requirements in increasing whilst the overall average number is increasing, which means that the more requirements are fulfilled, the less fairness leave for customer. This is because every customer in these two data set claims for different number of requirements, during the subset of selected requirements is growing, the algorithm is able to adjust the allocations of fulfilled requirement to different customers to obtain a lower standard deviation (more fair). However, eventually every customer has different number of fulfilled requirements when the subset is full.

On the other hand, the result for the Greer's data set shows the standard deviation is stay on the zero level. This is because of the sparsity pattern of $Value$ matrix of this data set which shown in Figure 4.1. In Greer's data set, every customer claims for every requirements, so all the customer have the same number of fulfilled requirements no matter which requirements are selected in the next release.

The differences between the result of Rand data set and could also be explained by the contrast of sparsity pattern of these data set. Comparing the Random data set, in Motorola's, every requirement is required by only one customer, the fact of which causes the standard deviation increase more dramatic than the case in Random data set.

Secondly, we consider another two objective functions in order to analyze the fairness on the percentage of the fulfilled requirements:

$$\text{Maximize: } mean(\overrightarrow{CVP})$$

and

$$\text{Minimize: } \sigma(\overrightarrow{CVP})$$

which aim to maximize the overall average coverage whilst minimize the standard deviation of the percentages of fulfill requirement for customers. The result are shown in Figure 4.3.

From result we can see the optimal front ultimately becomes a single point where all the requirements from all the customers are satisfied. So every customer has 100% satisfaction without any unfairness. There is a very interesting observation in Figure 4.3 (b). The possible solutions are not able to get into the triangle area around 50% fulfillment. The reason for this is the fact that the fourth customer in Motorola's data set only claims for one requirement. Thus, the percentage of fulfilled value for this customer has to be either 0% or 100%. Only considering those solutions on the edge of this triangle, when the overall percentage is growing between 0% and 50%, the fulfilled value for this customer is staying at 0%. This is because the other customer's fulfillment is below 50%, if the fourth customer has 100% fulfillment then the standard deviation will go up and the solution will leave the edge. Thus, on the edge of triangle between before 50% overall fulfillment, the standard deviation will go up if one of the customer's fulfillment is staying at zero and the other customer's fulfillment is increasing. For the same reason the triangle shape appears in subplot (b) of Figure 4.3, subplot (b) of Figure 4.5 and Figure 4.8.

4.4.2 Scenario Two: Fairness on Value of Fulfilled Requirements

In this scenario, the fairness on the value of fulfilled requirements is analyzed. The result for fairness analysis on absolute fulfilled value in shown Figure 4.4 and the result for fairness analysis on percentage of fulfilled value is shown in Figure 4.5. This scenario is implemented similarly with scenario one, but considering the $Value$ matrix of each data set instead of $Cost$. And the observation is also very similar to the previous one.

4.4.3 Scenario Three: Fairness on Value and Cost

For the purpose of obtaining the fairness information on different budget levels, the overall cost of the next release is taken into consideration in this scenario. Four objectives are considered in this scenario:

we aim to minimize the standard deviation of money spend on the customers,

$$\text{Minimize: } \sigma(\overrightarrow{C})$$

minimize the standard deviation of the percentage of fulfilled value for customers,

$$\text{Minimize: } \sigma(\overrightarrow{VP})$$

maximize the overall average fulfillment,

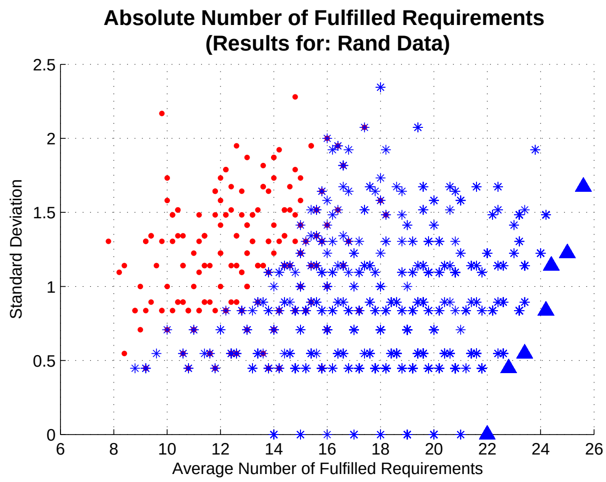
$$\text{Maximize: } mean(\overrightarrow{VP})$$

and finally minimize the overall cost of the next release.

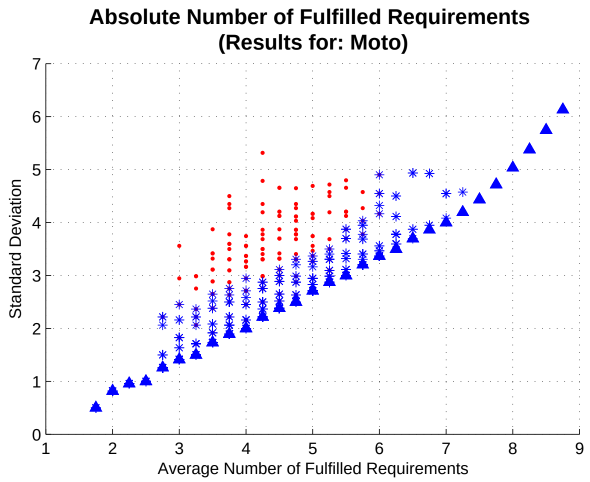
The results are plotted in Figure 4.6, 4.7 and 4.8 respectively for three data sets. From the results of Random data set and Motorola's data set, it is obviously that as the overall fulfillment is increasing the standard deviation of cost spend on the customers is also increasing. This observation is matched with the results from previous scenario.

4.4.4 Fairness Analysis Result Conclusion

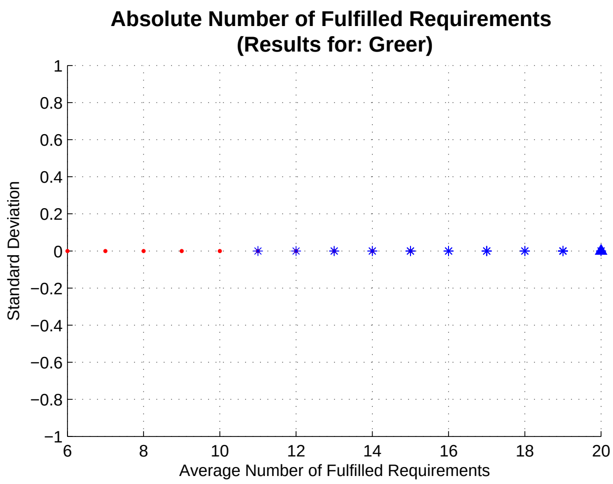
To sum up all the observations above, we have following conclusions for the fairness analysis. The fairness is highly depends on the definition of fair. From the results of fairness analysis on absolute fulfillment of cost or value, we found that more fulfillment will causes less fairness for customer. From the results of fairness analysis on percentage of fulfillment, however, we found that 100% fulfillment will provide a perfect fair solution.



(a) Result for Random Data Set

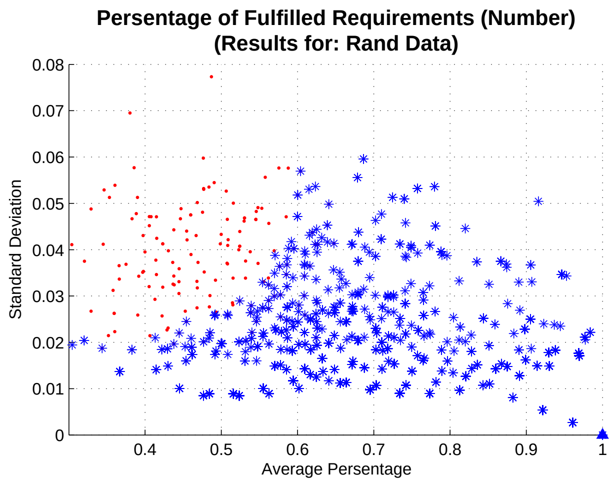


(b) Result for Moto Data Set

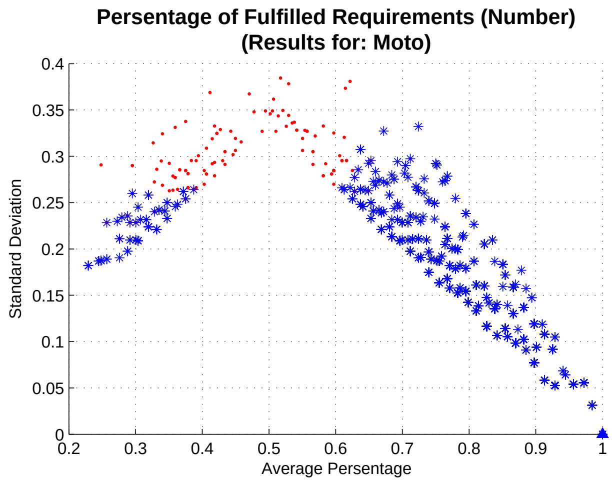


(c) Result for Greer Data Set

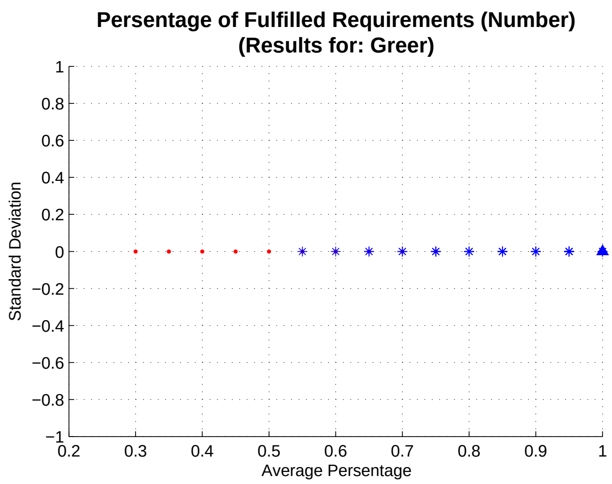
Figure 4.2: Fairness on Coverage: (Absolute Number of Fulfilled Requirements)



(a) Result for Random Data Set

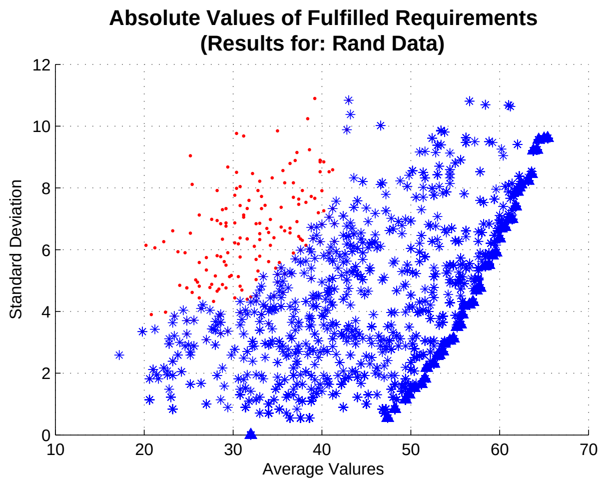


(b) Result for Moto Data Set

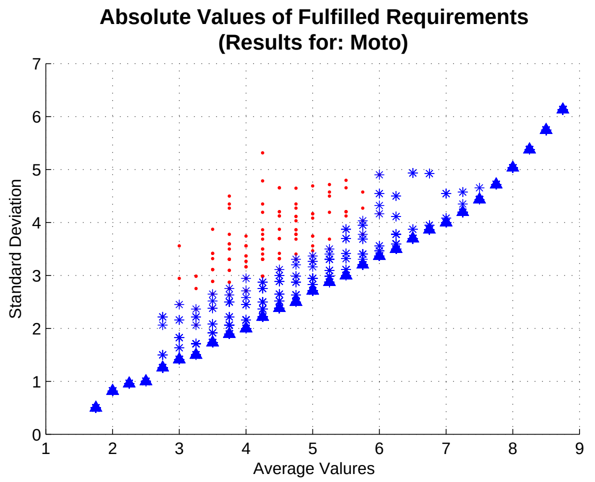


(c) Result for Greer Data Set

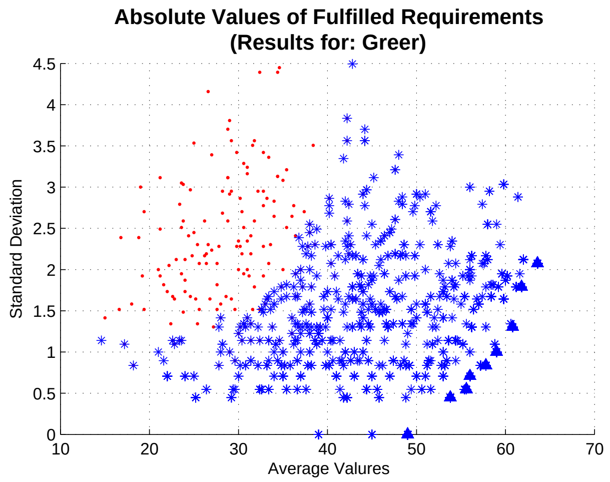
Figure 4.3: Fairness on Coverage: (Percentage of Fulfilled Requirements)



(a) Result for Random Data Set

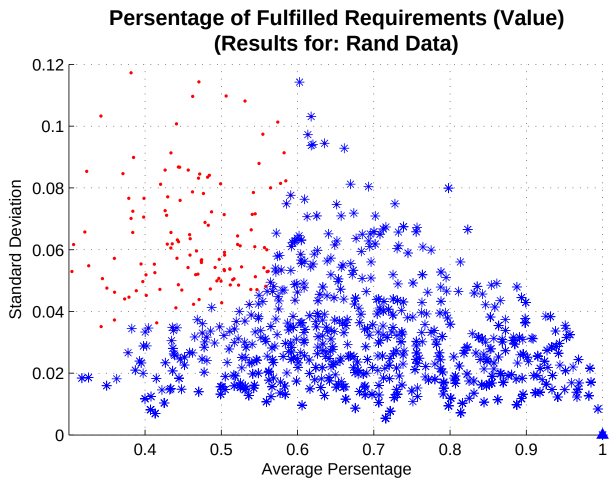


(b) Result for Moto Data Set

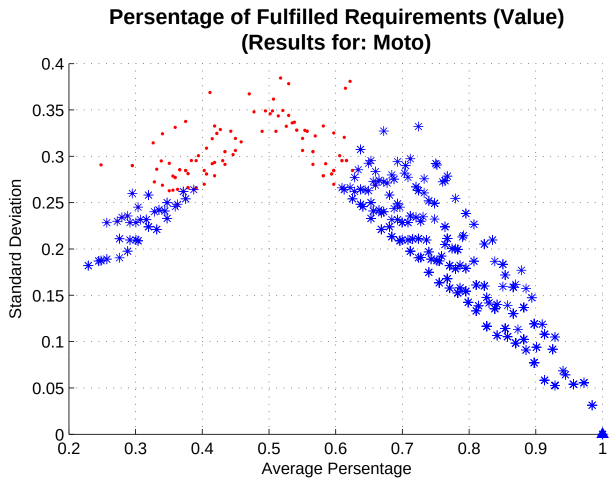


(c) Result for Greer Data Set

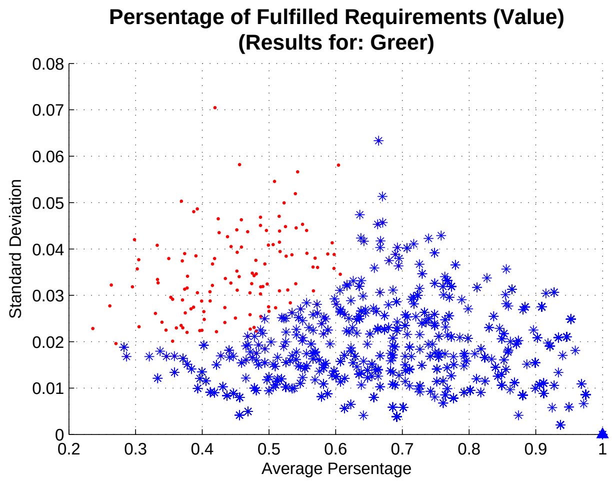
Figure 4.4: Fairness on Value: (Absolute Value of Fulfilled Requirements)



(a) Result for Random Data Set



(b) Result for Moto Data Set



(c) Result for Greer Data Set

Figure 4.5: Fairness on Value: (Percentage of Value of Fulfilled Requirements)

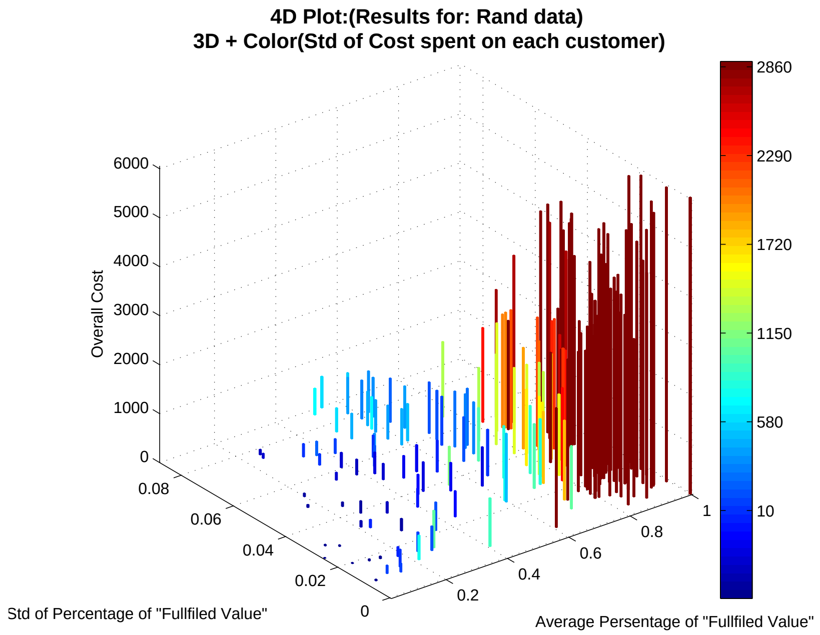


Figure 4.6: Fairness on Value and Cost, Result for Random Data Set

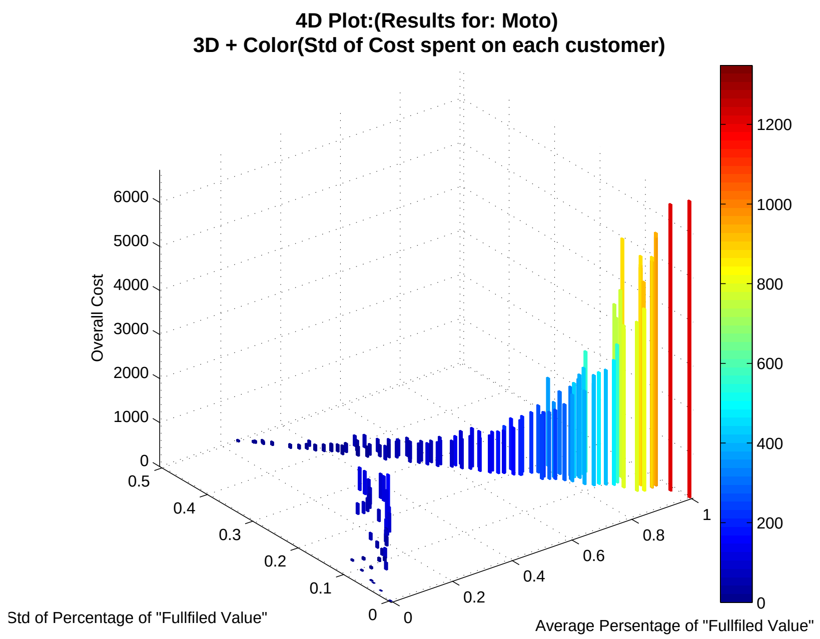


Figure 4.7: Fairness on Value and Cost, Result for Motorola Data Set

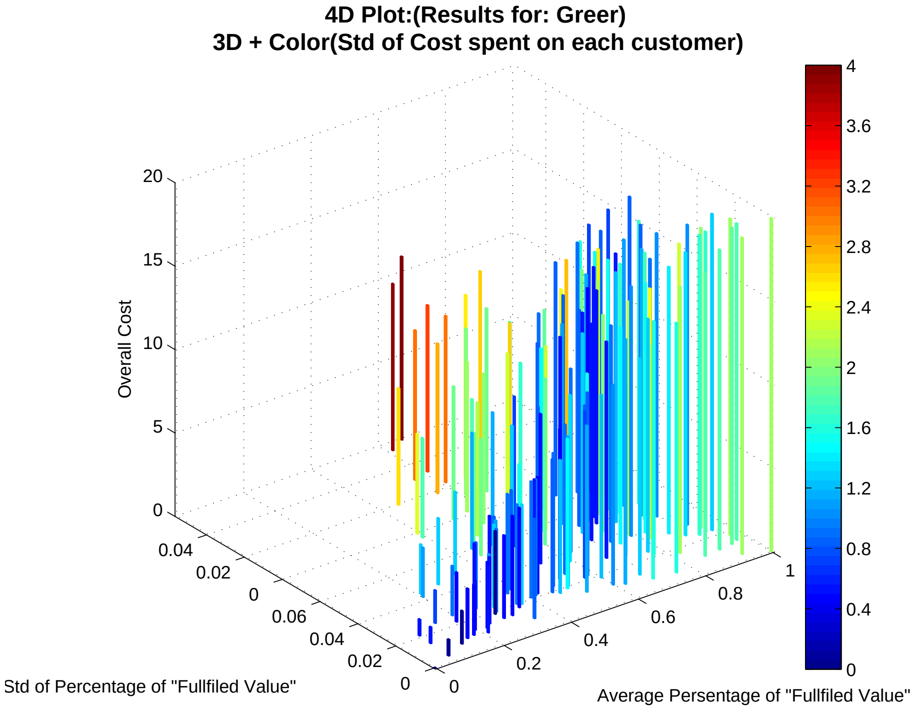


Figure 4.8: Fairness on Value and Cost, Result for Greer Data Set

Chapter 5

Conclusion and Future Work

This project has shown that how the multi–objective evolutionary algorithm solve the complex Next Release Problem, especially when some of the objectives are conflicting and how the search–based software engineering can provide the decision makers with a set of unbiased optimal solutions.

Depending on the previous work of sensitivity analysis in single–objective search algorithm, this project has developed the method to suit multi–objective evolutionary algorithm. The results have shown that the developed method for sensitivity analysis in multi–objective algorithms is successful. For the purpose of developing sensitivity analysis onto industrial level, the interdependence of requirements will be a very interesting problem to be analyze in the future.

The other important contribution that the project has made is the analysis of fairness in the Next Release Problem. It has proved that the definition of 'fair' is the fundamental concept of providing fair solutions for the next release. By adopting several comment definitions of 'fair', the results have shown the tradeoff between fair and for . The future work suggested is to feed the developed problem model with more real world data to testify the balance between maximizing the customer's satisfaction and minimizing the resource for development.

This project has also developed the visualization technique for multi–objective optimization problems. However, it is only developed in four dimensions and call for further developments.

Appendix A

Source Code

Listings

A.1 NSGA–II 42
A.2 evaluate_objective 44
A.3 initialize_variables 45
A.4 non_domination_sort_mod.m 45
A.5 genetic_operator 49
A.6 replace_chromosome 52
A.7 tournament_selection 53
A.8 tweak_analysis 54
A.9 switch_tweak_rate 55
A.10 dis_front_to_fronts 56
A.11 dis_of_2fronts 56
A.12 dis_pt_to_front 58
A.13 b1_21tweak_35R 59
A.14 analysis_fronts 60
A.15 analysis_1kRun 61
A.16 a6_spearman_rank_correlation 62
A.17 a6_analysis 64
A.18 a6_30R_21tweak_1sample 65
A.19 a6_30R_21tweak_2 68
A.20 a6_draw_spear_Scatter 70
A.21 a6_draw_dis 72
A.22 comp_20 72
A.23 nsga_2FA 75
A.24 evaluate_objectiveFA 81
A.25 initialGreer 96
A.26 initialMoto 97
A.27 initialRand 99
A.28 pro101 100
A.29 pro102 101
A.30 pro201 101
A.31 pro202 102
A.32 proCost 102
A.33 proMinStdCost 103

A.34 spyOnC 104
A.35 drawswitch 105

A.1 Code for Sensitivity Analysis

Listing A.1: NSGA–II

```
function f=nsga_2(R, seed4init)

%  clear;
%  clc;
%  load('initialize_problem1 15_40');
% clear;
% clc
% %
%
% load('inDataMoto.mat');
% seed4init = 3;

pop = 150;  % population
gen = 80;  % generation

%M : the number of objectives
% V : the number of requirements
% switch pro
%       case 1

[temp,V] = size(R);
M = 2;

% chromosome = zeros (pop,V+4);
% chromosome matrix
chromosome = initialize_variables(pop,R,seed4init);

chromosome(pop,V+4) = zeros;

chromosome = non_domination_sort_mod(chromosome);


for i = 1 : gen

    pool = round(pop/2);
```

```
    % what is this magic number?
    tour = 5;


    parent_chromosome = tournament_selection...
    (chromosome, pool, tour);


    offspring_chromosome = genetic_operator...
    (parent_chromosome,R);


    [main_pop,temp] = size(chromosome);
    [offspring_pop,temp] = size(offspring_chromosome);
    intermediate_chromosome(1:main_pop,:) = chromosome;
    intermediate_chromosome(main_pop + 1 : ...
    main_pop + offspring_pop,1 : M+V) = ...
        offspring_chromosome;

    %what does this mean?
    intermediate_chromosome = ...
        non_domination_sort_mod(intermediate_chromosome);

    chromosome = replace_chromosome...
    (intermediate_chromosome, pop);

    %generation_=i
    if ~mod(i,10)
        fprintf('%d\n',i);
    end

end

% save(['results/' int2str(month(now)) '_'...
int2str(day(now)) '_' int2str(hour(now)) '_' ...
%       int2str(minute(now)) '_' int2str(second(now))...
'_Moto' ...
%       ], 'chromosome');
% save solution.txt chromosome -ASCII



% figure
% hold on
```

```
% for i = 1 : pop
%       if chromosome(i,V + M + 1)==1
%           plot(chromosome(:,V + 1),chromosome(:,V + 2),'ko');
%       end
% end

% for i = 1 : pop
%       if chromosome(i,V + M + 1)==1
%           f=[chromosome(:,V + 1),chromosome(:,V + 2)];
%       end
% end

f=chromosome;

% xlabel('f(x_1)');
% ylabel('f(x_2)');
```

Listing A.2: evaluate_objective

```
function f = evaluate_objective(x,R)

% Function to evaluate the objective functions for the
  given input vector
% x. x has the decision variables

f = [];

[m,M]=size(R);
x=x(1:M);
%% Objective function one

sum=0;

sum=R(2,:)*x'; % sum1 :

f(1) = sum;

%% Objective function two

sum2=0;

sum2=R(1,:)*x';      % sum2 : cost

%sum2=100-sum2;
```

```
f(2) = -sum2;
```

Listing A.3: initialize_variables

```
function f = initialize_variables(N,R,seed4init)

% min = 0;
% max = 1;

[M,num_of_requirement] = size(R);
M = num_of_requirement;
K = M + 2;
% f = [];
seed_for_rand = seed4init;


% N : population number
% M : requirements number
rand('seed',seed_for_rand);

for i = 1 : N
    for j = 1 : M
        f(i,j) = round(rand(1)); %initial population
    end
    f(i, M+1 : K) = evaluate_objective(f(i,:),R);
end
```

Listing A.4: non_domination_sort_mod.m

```
function f = non_domination_sort_mod(x)

[N,col] = size(x);
M = 2;
V = col-4;

% overwrite variable M
% switch problem
%       case 1
%           M = 2;
%           V = 40;
%       case 2
%           M = 2;
%           V = 80;
%       case 3
```

```
%            M =  2;
%            V =  140;
%       case  4
%            M =  2;
%            V =  15;
%       case  5
%            M =  2;
%            V =  20;
% end


front = 1;

F(front).f = [];
individual = [];
for i = 1 : N

    individual(i).n = 0;

    individual(i).p = [];

    for j = 1 : N
        dom_less = 0;
        dom_equal = 0;
        dom_more = 0;
        %           flag_equal = 0;

        for k = 1 : M
            if (x(i,V + k) < x(j,V + k))
                dom_less = dom_less + 1;
            elseif (x(i,V + k) == x(j,V + k))
                dom_equal = dom_equal + 1;
            else
                dom_more = dom_more + 1;
            end
        end

        if  (dom_less==0||dom_more==0) && ...
        dom_equal ~= M
            % .p .n : index for non_domination

            %if dom_equal ~= M
            if dom_less == 0
```

```
                % .p : positive
                individual(i).p = [individual(i).p j];
                % record the non_dominated individual


            else %if dom_more == 0 && dom_equal ~= M
                % .n : nagitive
                individual(i).n = individual(i).n + 1;
                % record the number of domination

            end
            %end
        end

    end

    if individual(i).n == 0 % individual(i) is non_dominate
        x(i,M + V + 1) = 1;
        F(front).f = [F(front).f i];
    end
end

while ~isempty(F(front).f)
    Q = [];
    for i = 1 : length(F(front).f) ...
    %loop all the front points
        if ~isempty(individual(F(front).f(i)).p)

            for j = 1 : length(individual(F(front).f(i)).p)
                % loop all the non_dominated points

                individual(individual...
                (F(front).f(i)).p(j)).n = ...
                    individual(individual...
                    (F(front).f(i)).p(j)).n - 1;

                if individual(individual...
                (F(front).f(i)).p(j)).n == 0
                    x(individual(...
                    F(front).f(i)).p(j),M + V + 1) = ...
                        front + 1;
                    Q = [Q individual(F(front).f(i)).p(j)];
                end
```

```
                end

            end
        end
        front =  front + 1;
        F(front).f = Q;
end

[temp,index_of_fronts] = sort(x(:,M + V + 1));

for i = 1 : length(index_of_fronts)
    sorted_based_on_front(i,:) = x(index_of_fronts(i),:);
end
current_index = 0;

for front = 1 : (length(F) - 1)
    y = [];
    previous_index = current_index + 1;
    for i = 1 : length(F(front).f)
        y(i,:) = sorted_based_on_front(current_index + i,:);
    end
    current_index = current_index + i;

    %       sorted_based_on_objective = [];

    for i = 1 : M
        [sorted_based_on_objective, index_of_objectives]=...
            sort(y(:,V + i));
        sorted_based_on_objective = [];
        for j = 1 : length(index_of_objectives)
            sorted_based_on_objective(j,:) =...
            y(index_of_objectives(j),:);
        end
        f_max = ...
            sorted_based_on_objective(length...
            (index_of_objectives), V + i);
        f_min = sorted_based_on_objective(1, V + i);
        y(index_of_objectives(length...
        (index_of_objectives)),M + V + 1 + i)...
            = Inf;
        y(index_of_objectives(1),M + V + 1 + i) = Inf;
        for j = 2 : length(index_of_objectives) - 1
```

```
            next_obj   = sorted_based_on_objective...
            (j + 1,V + i);
            previous_obj   = sorted_based_on_objective...
            (j - 1,V + i);
            if (f_max - f_min == 0)
                y(index_of_objectives(j),...
                M + V + 1 + i) = Inf;
            else
                % normalisation
                y(index_of_objectives(j),M +...
                V + 1 + i) = ...
                    (next_obj - previous_obj)...
                    /(f_max - f_min);
            end
        end
    end
    distance = [];
    distance(:,1) = zeros(length(F(front).f),1);
    for i = 1 : M
        distance(:,1) = distance(:,1) + y...
        (:,M + V + 1 + i);
    end
    y(:,M + V + 2) = distance;
    y = y(:,1 : M + V + 2);
    z(previous_index:current_index,:) = y;
end
f = z();
```

Listing A.5: genetic_operator

```
function f  = genetic_operator(parent_chromosome,R)

[N,M] = size(parent_chromosome);
[M,V] = size(R);
M = 2;

% switch pro
%     case 1
%         M = 2;
%         V = 40;
%     case 2
%         M = 2;
%         V = 80;
%     case 3
```

```
%           M = 2;
%           V = 140;
%       case 4
%           M = 2;
%           V = 15;
%       case 5
%           M = 2;
%           V = 20;
% end
p = 1;
 was_crossover = 0;
 was_mutation = 0;

for i = 1 : N
    if rand(1) < 0.8
        child_1 = [];
        child_2 = [];
        parent_1 = round(N*rand(1));
        if parent_1 < 1
            parent_1 = 1;
        end
        parent_2 = round(N*rand(1));
        if parent_2 < 1
            parent_2 = 1;
        end
        while isequal(parent_chromosome(parent_1,:),...
        parent_chromosome(parent_2,:))
            parent_2 = round(N*rand(1));
            if parent_2 < 1
                parent_2 = 1;
            end
        end
        parent_1 = parent_chromosome(parent_1,:);
        parent_2 = parent_chromosome(parent_2,:);


        u=1+round(rand(1)*(V-1));
        for j = 1 : u
            child_1(j) = parent_1(j);
            child_2(j) = parent_2(j);
        end
        for j= u+1 : V
            child_1(j) = parent_2(j);
```

```
            child_2(j) = parent_1(j);
        end

        child_1(:,V + 1: M + V) = ...
            evaluate_objective(child_1,R);
        child_2(:,V + 1: M + V) = ...
            evaluate_objective(child_2,R);
        was_crossover = 1;
        was_mutation = 0;
    end

    if rand(1)< 0.2
        parent_3 = round(N*rand(1));
        if parent_3 < 1
            parent_3 = 1;
        end

        child_3 = parent_chromosome(parent_3,:);

        w = 1+round(rand(1)*(V-1));
        if child_3(w) == 0
            child_3(w) = 1;
        else
            child_3(w) =0;
        end

        child_3(:,V + 1: M + V) = ...
            evaluate_objective(child_3,R);
        was_mutation = 1;
        was_crossover = 0;
    end

    if was_crossover
        child(p,:) = child_1;
        child(p+1,:) = child_2;
        was_cossover = 0;
        p = p + 2;
    elseif was_mutation
        child(p,:) = child_3(1,1 : M + V);
        was_mutation = 0;
        p = p + 1;
    end
end
```

```
f = child;
```

Listing A.6: replace_chromosome

```
function f  = replace_chromosome(intermediate_chromosome,pop)

[N,V] = size(intermediate_chromosome);
V = V-4;
M = 2;
% switch pro
%         case 1
%         M = 2;
%         V = 40;
%     case 2
%         M = 2;
%         V = 80;
%     case 3
%         M = 2;
%         V = 140;
%     case 4
%         M = 2;
%         V = 15;
%     case 5
%         M = 2;
%         V = 20;
% end


[temp,index] = sort(intermediate_chromosome(:,M + V + 1));


for i = 1 : N
    sorted_chromosome(i,:) = ...
     intermediate_chromosome(index(i),:);
end


max_rank = max(intermediate_chromosome(:,M + V + 1));


previous_index = 0;
for i = 1 : max_rank
    current_index = max(find(sorted_chromosome...
     (:,M + V + 1) == i));
```

```
    if current_index > pop
        remaining = pop - previous_index;
        temp_pop = ...
            sorted_chromosome(previous_index + ...
            1 : current_index, :);
        [temp_sort,temp_sort_index] = ...
            sort(temp_pop(:, M + V + 2),'descend');
        for j = 1 : remaining
            f(previous_index + j,:) = ...
            temp_pop(temp_sort_index(j),:);
        end
        return;
    elseif current_index < pop
        f(previous_index + 1 : current_index, :) = ...
            sorted_chromosome...
               (previous_index + 1 : current_index, :);
    else
        f(previous_index + 1 : current_index, :) = ...
            sorted_chromosome...
               (previous_index + 1 : current_index, :);
        return;
    end
    previous_index = current_index;
end
```

Listing A.7: tournament_selection

```
function f = tournament_selection(chromosome,...
   pool_size,tour_size)



[pop,variables] = size(chromosome);
rank = variables - 1;
distance = variables;
candidate = [];
c_obj_rank = [];
c_obj_distance = [];

for i = 1 : pool_size
    for j = 1 : tour_size
        candidate(j) = round(pop*rand(1));
        if candidate(j) == 0
            candidate(j) = 1;
```

```
        end
        if j > 1
            while ~isempty(find(candidate(1 : j - 1)...
            == candidate(j),1))
                %ISEMPTY(FIND(x, 1))
                candidate(j) = round(pop*rand(1));
                if candidate(j) == 0
                    candidate(j) = 1;
                end
            end
        end
    end
    for j = 1 : tour_size
        c_obj_rank(j) = chromosome...
           (candidate(j),rank);
        c_obj_distance(j) = chromosome...
           (candidate(j),distance);
    end
    min_candidate = ...
        find(c_obj_rank == min(c_obj_rank));
    if length(min_candidate) ~= 1
        max_candidate = ...
        find(c_obj_distance(min_candidate) ==...
        max(c_obj_distance(min_candidate)));
        if length(max_candidate) ~= 1
            max_candidate = max_candidate(1);
        end
        f(i,:) = chromosome(candidate...
        (min_candidate(max_candidate)),:);
    else
        f(i,:) = chromosome(candidate...
        (min_candidate(1)),:);
    end
end
```

Listing A.8: tweak_analysis

```
function dis_caused_by_tweak = tweak_analysis...
(original_fronts,tweaked_fronts)

[p,q] = size(tweaked_fronts);
number_of_fronts = q;

% o_index = analysis_fronts(original_fronts);
```

```
% t_index = analysis_fronts(tweaked_fronts);

sum_of_dis = 0;
for i = 1 : number_of_fronts
    sum_of_dis = sum_of_dis + dis_front_to_fronts...
        (tweaked_fronts(i).individual, original_fronts);
end

dis_caused_by_tweak = sum_of_dis / number_of_fronts;
% dis_caused_by_tweak = dis_of_2fronts...
%     (original_fronts(o_index), tweaked_fronts(t_index))
```

Listing A.9: switch_tweak_rate

```
function f=switch_tweak_rate(j)

switch j
    case 1
        tweak_rate = 0;
    case 2
        tweak_rate = -0.05;
    case 3
        tweak_rate = 0.05;
    case 4
        tweak_rate = -0.1;
    case 5
        tweak_rate = 0.1;
    case 6
        tweak_rate = -0.15;
    case 7
        tweak_rate = 0.15;
    case 8
        tweak_rate = -0.2;
    case 9
        tweak_rate = 0.2;
    case 10
        tweak_rate = -0.25;
    case 11
        tweak_rate = 0.25;
    case 12
        tweak_rate = -0.3;
    case 13
        tweak_rate = 0.3;
    case 14
```

```
        tweak_rate = -0.35;
    case 15
        tweak_rate = 0.35;
    case 16
        tweak_rate = -0.4;
    case 17
        tweak_rate = 0.4;
    case 18
        tweak_rate = -0.45;
    case 19
        tweak_rate = 0.45;
    case 20
        tweak_rate = -0.5;
    case 21
        tweak_rate = 0.5;
    otherwise
        fprintf('Unknown j for tweak rate.');
end

f=tweak_rate;
```

Listing A.10: dis_front_to_fronts

```
function dis_of_front_to_fronts = ...
  dis_front_to_fronts(frt_individual,fronts)

[p,q] = size(fronts);
number_of_fronts = q;

sum_of_dis=0;
for j = 1 : number_of_fronts
    sum_of_dis = sum_of_dis + dis_of_2fronts...
        (frt_individual,fronts(j).individual);
end

dis_of_front_to_fronts = sum_of_dis /...
    number_of_fronts;
```

Listing A.11: dis_of_2fronts

```
function [dis_of_2fronts]=...
dis_of_2fronts(front1,front2,dflg)
% front1 & front2 are both p*q matrix
%  p = num_of_population
%  q = num_of_requirement + num_of_objectives + 2
```

```
[pop,q] = size(front1);
% pop = p;     %population number

V = q - 4;      %requirement number
M = 2;       %objective number

%% normalise the fitness space

front1_X = front1(:,V + 1);
front1_Y = front1(:,V + 2);
front2_X = front2(:,V + 1);
front2_Y = front2(:,V + 2);

% max_f1_X = max(front1_X);
% max_f1_Y = max(front1_Y);
% max_f2_X = max(front2_X);
% max_f2_Y = max(front2_Y);
% min_f1_X = min(front1_X);
% min_f1_Y = min(front1_Y);
% min_f2_X = min(front2_X);
% min_f2_Y = min(front2_Y);
%
% f1x = (front1_X - min_f1_X) / (max_f1_X - min_f1_X);
% f1y = (front1_Y - min_f1_Y) / (max_f1_Y - min_f1_Y);
% f2x = (front2_X - min_f2_X) / (max_f2_X - min_f2_X);
% f2y = (front2_Y - min_f2_Y) / (max_f2_Y - min_f2_Y);

f1x = (front1_X) / 55;
f1y = (front1_Y) / -6740;
f2x = (front2_X) / 55;
f2y = (front2_Y) / -6740;


%% calculate the Euclidean Distance between...
   two fronts

for i = 1 : pop
    disP2F_A(i) = dis_pt_to_front([f1x(i),...
    f1y(i)],[f2x,f2y],dflg);
end

d1 = mean(disP2F_A);
```

```
for i = 1 : pop
     disP2F_B(i) = dis_pt_to_front([f2x(i),...
     f2y(i)],[f1x,f1y],dflg);
end

if dflg
     d2 = -mean(disP2F_B);
else
     d2 = mean(disP2F_B);
end

dis_of_2fronts = (d1 + d2)/2;
% dis_of_2fronts=d1;
```

Listing A.12: dis_pt_to_front

```
function dis_of_pt_to_front = ...
   dis_pt_to_front(point, front, dlfg)
% both 'point' and 'front' are normalized.

[pop,d] = size(front);
dis_temp = Inf;
index = 0;

for i = 1 : pop
%      dis = distance (point,front(i,:));
       dis = sqrt(sqrt((point(1) -...
       front(i,1))^2 + (point(2) - front(i,2))^2 ));
% dis = (point(1) - front(i,1))^2 + (point(2) ...
- front(i,2))^2 ;

%       p1=point(1)
%       p2=point(2)
%       f1=front(i,1)
%       f2=front(i,2)
     if ( dis < dis_temp )
         dis_temp = dis;
         index = i;
     end
end

% if ((point(1)-0)^2 + (point(2)-1)^2)<...
```

```
  ((front(index,1)-0)^2+(front(index,2)-1)^2)
%      dis_temp = -dis_temp;
% end

% if (point(1)>front(index,1))&&(point(2)<...
front(index,2))
%      dis_temp
% else

if dlfg
     if (point(1)<front(index,1))&&(point(2)>...
     front(index,2))
          dis_temp = - dis_temp;
     elseif (point(1)>front(index,1))&&(point(2)...
     >front(index,2))...
               &&((point(1)-front(index,1))<=...
               (point(2)-front(index,2)))
          dis_temp = - dis_temp;
     elseif (point(1)<front(index,1))&&(point(2)...
     <front(index,2))...
               &&((front(index,1) - point(1))>=...
               (front(index,2) - point(2)))
          dis_temp = - dis_temp;
     end
end

dis_of_pt_to_front = dis_temp;
```

Listing A.13: b1_21tweak_35R

```
function b1_21tweak_35R(R, seed4init)

% clc;
% clear;

% load ('inDataMoto');

% [M,V] = size(R);

M = 2;
tweak_time = 21;
sample_time = 1;
original_R = R;
[rows, cols] = size(R);
```

```
num_of_requirements = cols;
tweak = [];
tweak.requirement = [];
tweak.requirement.front = [];

for j = 1 : tweak_time

    tweak_rate = switch_tweak_rate(j);

    for k = 1 : num_of_requirements

        R = original_R;
        R(1,k) = ( original_R(1,k) * ( 1 + tweak_rate ) );

        tweak(j).requirement(k).front = nsga_2(R,seed4init);

        fprintf('First Round Processing:\n')
        fprintf(num2str( (j-1) * ...
        num_of_requirements  + k ))
        fprintf(' out of ')
        fprintf(num2str(num_of_requirements ...
        * sample_time * tweak_time))
        fprintf('\nTweak rate = ')
        fprintf(num2str(tweak_rate))
        fprintf('\nRequirement No.')
        fprintf(num2str(k))
        fprintf('\nThere are ')
        fprintf(num2str( 100*((j-1) * ...
        num_of_requirements  + k -1 )/...
            (num_of_requirements * ...
            sample_time * tweak_time)))
        fprintf('%% finished!\n')

    end
end

save(['results/tweak_2nd_' int2str(seed4init)]);

fprintf('All done!\n')
```

Listing A.14: analysis_fronts

```
function index_of_representive = analysis_fronts(fronts)
% analysis fronts within a tweak
```

```
% fronts is a matrix of n fronts

[p,q] = size(fronts);
number_of_fronts = q;

sum_of_dis=[];

for i = 1 : number_of_fronts
    sum_of_dis(i)=0;
    for j = 1 : number_of_fronts
        sum_of_dis(i) = sum_of_dis(i) + dis_of_2fronts...
            (fronts(i).individual,fronts(j).individual);
    end
end

[x,index_of_representive] = min(sum_of_dis);
```

Listing A.15: analysis_1kRun

```
%function analysis_1kRun()
clc;
clear;
load('1000runs_40Reqs_5Tweaks_5Samples.mat');
[x,number_of_requirement] = size(f);
[x,number_of_tweak] = size(f(1).tweak);
[x,number_of_sample] = size(f(1).tweak(1).front);
original_fronts = f(40).tweak(3).front;
o_index=analysis_fronts(original_fronts)
orgn_frt_individual = original_fronts(o_index).individual;

dis1=[];
for k = 1 : number_of_requirement
    for j = 1 : number_of_tweak
        dis1(k,j)=...
            dis_front_to_fronts(orgn_frt_individual,...
            f(k).tweak(j).front);
    end
    k
end




% dis2=[];
% for k = 1 : number_of_requirement
```

```
%       for j = 1 : number_of_tweak
%           dis2(k,j)=...
%               tweak_analysis(f(k).tweak(j).front,...
                original_fronts);
%       end
%       k
% end
```

Listing A.16: a6_spearman_rank_correlation

```
% function a6_spearman_rank_correlation

clc;
clear;
tweak_time = 21;
num_of_requirements = 30;


%% select source data for input X
prob = 1;
switch prob
    case 1
        load('a6_30R_21tweak_3_change_seed');
    case 2
        load('a6_30R_21tweak_2');
end

%% select the source data for output Y
dflg = 1
switch dflg
    case 1
        load('a6_dis_direction');
    case 2
        load('a6_dis_nodirection');
end

%% define
% X (input): 'see each following case'
% Y (output): the distance from original front

pairs = 2
switch pairs
    case 1
        % for one paticular requirement
```

```
        % tweak_rate vs dis

        % == input X == tweak rate inorder
        X = -1 * [1:1:21]'; %decreasing

        % == output Y == dis
        Y = dis1';

        % == calculate the coefficient ==
        [r,t,p] = a6_spear(X,Y);
        plot(X,Y,'ko')

    case 2
        % for one paticular tweak_rate
        % cost vs dis

        % == input X == list of 30 cost of the requirement
        tX = original_R(5,:)';
        for i = 1 : 21
            index = (i-1)*30;
            X(index+(1:30)) = tX';
        end
        X=X';

        % == output Y == dis
        tY = dis1;
        for i = 1 : 21
            index2 = (i-1)*30;
            Y(index2+(1:30)) = (tY(:,i))';
        end
        Y=Y';
        %Y = Y(:,5:17);



        % == calculate the coefficient ==
        [r,t,p] = a6_spear(X,Y);
        plot(X,Y,'ko')

    case 3
        % for one paticular tweak_rate
        % value_of_requirement vs dis
```

```
        % == input X == list of the value of each requirement
        temp_C = original_C(1:4);
        temp_R = original_R(1:4,1:30);
        X = (temp_C * temp_R)';

        % == output Y == dis
        Y = dis1;

        % == calculate the coefficient ==
        [r,t,p] = a6_spear(X,Y);
        plot(X,Y,'ko')

end

%% calculate the correlation coefficient

% [r,t,p] = a6_spear(x,Y);
```

Listing A.17: a6_analysis

```
%   function a6_analysis

% clc;
% clear;

load('results/tweak_2nd_10');


dflg = 1; % direction flag
prob = 1;


[x,number_of_tweak] = size(tweak);
[x,number_of_requirement] = size(tweak(1).requirement);
[x,number_of_sample] = size(tweak(1).requirement(1).front);
% original_fronts = tweak(3).requirement(1).front;
% o_index=analysis_fronts(original_fronts)
% orgn_frt_individual = original_fronts(o_index).individual;
orgn_frt = tweak(1).requirement(1).front;

dis1=[];
for k = 1 : number_of_requirement
    for j = 1 : number_of_tweak
        %         k
```

```
        %           j
        dis1(k,j)=...
            dis_of_2fronts(orgn_frt ,...
            tweak(j).requirement(k).front ,dflg);

        %           if j>6
        %           dis1(k,j)=...
        %               dis_of_2fronts(orgn_frt ,...
                         tweak(j).requirement(k).front);
        %           else
        %           dis1(k,j)=...
        %               dis_of_2fronts(orgn_frt ,...
                         tweak(j).requirement(k).front);
        %           end
    end
    k
end
%%
[row,col]=size(dis1);
dis2=zeros(row,col);

for i=1:10
    dis2(:,11-i)=dis1(:,2*i);
    dis2(:,i+11)=dis1(:,1+2*i);
end
dis2(:,11)=dis1(:,1);

imagesc(dis2); figure(gcf)
```

Listing A.18: a6_30R_21tweak_1sample

```
%  function a6_30R_17tweak_1sample

clc;
clear;

load ('a5_2C_30R');

% [M,V] = size(R);
M = 2;
tweak_time = 21;
sample_time = 1;
original_R = R;
original_C = C;
```

```
[rows, cols] = size(R);
num_of_requirements = cols;
tweak = [];
tweak.requirement = [];
tweak.requirement.front = [];


for j = 1 : tweak_time
    switch j
        case 1
            tweak_rate = 0.50;
        case 2
            tweak_rate = 0.45;
        case 3
            tweak_rate = 0.40;
        case 4
            tweak_rate = 0.35;
        case 5
            tweak_rate = 0.30;
        case 6
            %             continue;
            tweak_rate = 0.25;
        case 7
            %             continue;
            tweak_rate = 0.20;
        case 8
            %             continue;
            tweak_rate = 0.15;
        case 9
            %             continue;
            tweak_rate = 0.10;
        case 10
            %             continue;
            tweak_rate = 0.05;
        case 11
            %             continue;
            tweak_rate = 0.00;
        case 12
            %             continue;
            tweak_rate = -0.05;
        case 13
            %             continue;
            tweak_rate = -0.10;
```

```
        case 14
            %               continue
            tweak_rate = -0.15;
        case 15
            %               continue;
            tweak_rate = -0.20;
        case 16
            %               continue;
            tweak_rate = -0.25;
        case 17
            tweak_rate = -0.30;
        case 18
            tweak_rate = -0.35;
        case 19
            tweak_rate = -0.40;
        case 20
            tweak_rate = -0.45;
        case 21
            tweak_rate = -0.50;
        otherwise
            break;
    end

    for k = 1 : num_of_requirements

        R = original_R;
        R(rows,k)=(original_R(rows,k)*(1+tweak_rate));
        fprintf('First Round Processing:\n')
        fprintf(num2str( (j-1) ...
         * num_of_requirements  + k ))
        fprintf(' out of ')
        fprintf(num2str(num_of_requirements...
         * sample_time * tweak_time))
        fprintf('\nTweak rate = ')
        fprintf(num2str(tweak_rate))
        fprintf('\nRequirement No.')
        fprintf(num2str(k))
        fprintf('\nThere are ')
        fprintf(num2str( 100*((j-1) ...
         * num_of_requirements  + k -1 )/...
            (num_of_requirements * ...
            sample_time * tweak_time)))
        fprintf('%% finished!\n')
```

```
        tweak(j).requirement(k).front = nsga_2(R,C);
    end
end


% save('a6_30R_21tweak_3_change_seed',...
'original_R','original_C','tweak');

a6_30R_21tweak_2;

fprintf('All done!\n')
```

Listing A.19: a6_30R_21tweak_2

```
function a6_30R_21tweak_2;

% clc;
clear;

load ('a5_2C_30R');

% [M,V] = size(R);
M = 2;
tweak_time = 21;
sample_time = 1;
original_R = R;
original_C = C;
[rows,cols] = size(R);
num_of_requirements = cols;
tweak = [];
tweak.requirement = [];
tweak.requirement.front = [];


for j = 1 : tweak_time
    switch j
        case 1
            tweak_rate = 0.50;
        case 2
            tweak_rate = 0.45;
        case 3
            tweak_rate = 0.40;
        case 4
```

```
        tweak_rate = 0.35;
    case 5
        tweak_rate = 0.30;
    case 6
        %           continue;
        tweak_rate = 0.25;
    case 7
        %           continue;
        tweak_rate = 0.20;
    case 8
        %           continue;
        tweak_rate = 0.15;
    case 9
        %           continue;
        tweak_rate = 0.10;
    case 10
        %           continue;
        tweak_rate = 0.05;
    case 11
        %           continue;
        tweak_rate = 0.00;
    case 12
        %           continue;
        tweak_rate = -0.05;
    case 13
        %           continue;
        tweak_rate = -0.10;
    case 14
        %           continue
        tweak_rate = -0.15;
    case 15
        %           continue;
        tweak_rate = -0.20;
    case 16
        %           continue;
        tweak_rate = -0.25;
    case 17
        tweak_rate = -0.30;
    case 18
        tweak_rate = -0.35;
    case 19
        tweak_rate = -0.40;
    case 20
```

```
            tweak_rate = -0.45;
        case 21
            tweak_rate = -0.50;
        otherwise
            break;
    end

    for k = 1 : num_of_requirements

        R = original_R;
        R(rows,k) = ( original_R(rows,k) * ...
          ( 1 + tweak_rate ) );

        fprintf('Second Round Processing:\n')
        fprintf(num2str( (j-1) * num_of_requirements  + k ))
        fprintf(' out of ')
        fprintf(num2str(num_of_requirements...
         * sample_time * tweak_time))
        fprintf('\nTweak rate = ')
        fprintf(num2str(tweak_rate))
        fprintf('\nRequirement No.')
        fprintf(num2str(k))
        fprintf('\nThere are ')
        fprintf(num2str( 100*((j-1) * ...
        num_of_requirements  + k -1 )/...
            (num_of_requirements * ...
            sample_time * tweak_time)))
        fprintf('%% finished!\n')

        tweak(j).requirement(k).front = nsga_2(R,C);
    end
end

% save('a6_30R_21tweak_4_change_seed',...
  'original_R','original_C','tweak');

fprintf('All done!\n')
```

Listing A.20: a6_draw_spear_Scatter

```
% function a6_draw_spear_Scatter

% figure
% X = -X;
```

```
prob = 3;

switch prob
    case 1
        plot(X,Y,'o')
        hold on
        title('Scatterplot of Tweak Rate...
        vs. Distance for 30 Requirements',...
        'FontSize',14);
        xlabel('Tweak Rate');
        ylabel('Distance');
        set(gca,'XTick',1:1:21)
        set(gca,'XLim',[0,22])
        set(gca,'XTickLabel',{...
            '50%','45%','40%','35%','30%',...
            '25%','20%','15%','10%','5%',...
            '0',...
            '-5%','-10%','-15%','-20%','-25%',...
            '-30%','-35%','-40%','-45%','-50%'
            })
        %         set(gca,'YTick',1:1:30)
        %         set(gca,'YTickLabel',1:1:30)
    case 2
        plot(X,Y(:,[5 11 17]),'o')
        hold on
        legend('+30%','0%','-30%',2)
        title('Scatterplot of Cost vs....
        Distance with +30%,0%,-30% increase...
        on the Cost','FontSize',14);
        xlabel('Cost of Requirement');
        ylabel('Distance');
%         set(gca,'XTick',1:1:21)
%         set(gca,'XLim',[0,22])
%         set(gca,'XTickLabel',{...
%             '50%','45%','40%','35%','30%',...
%             '25%','20%','15%','10%','5%',...
%             '0',...
%             '-5%','-10%','-15%','-20%','-25%',...
%             '-30%','-35%','-40%','-45%','-50%'
%             })
%         set(gca,'YTick',1:1:30)
%         set(gca,'YTickLabel',1:1:30)
```

```
    case 3
        plot(X,Y(:,[5 11 17]),'o')
        hold on
        legend('+30%','0%','-30%',2)
        title('Scatterplot of Value vs ...
        Distance with +30%,0%,-30% increase ...
        on the Cost','FontSize',14);
        xlabel('Value of Requirement');
        ylabel('Distance');
end
```

Listing A.21: a6_draw_dis

```
% function a6_draw_dis

figure;
imagesc (dis1);
title('Distance v.s. Tweak');
xlabel('Tweak Rate');
ylabel('Requirement Number');
set(gca,'XTick',1:1:21)
set(gca,'XTickLabel',{...
    '50%','45%','40%','35%','30%' ,...
    '25%','20%','15%','10%','5%' ,...
    '0' ,...
    '-5%','-10%','-15%','-20%','-25%' ,...
    '-30%','-35%','-40%','-45%','-50%'
    })
set(gca,'YTick',1:1:30)
set(gca,'YTickLabel',1:1:30)
```

Listing A.22: comp_20

```
%function comp_20(R,C)
clc;
load ('initialize_problem5_2_20.mat');
V=40;
M=2;
pareto_range=[];
pareto_range(k).tweak(j).individual=[];
sample_time=1;
tweak_time=5;

original_R=R;
original_C=C;
```

```
[rows, cols]=size(R);
num_of_requirements=cols;

% for k=1:num_of_requirements
%     for j=1:tweak_time
%         switch j
%             case 1
%                 tweak_rate = 0.25;
%             case 2
%                 tweak_rate = 0.15;
%             case 3
%                 tweak_rate = 0;
%             case 4
%                 tweak_rate = -0.15;
%             case 5
%                 tweak_rate = -0.25;
%         end
%
%         R = original_R;
%         R(rows,k) = ( original_R(rows,k) * ...
              ( 1 - tweak_rate ) );
%         f(k).tweak(j).front=[];
%         pareto_range(k).tweak(j).individual=[];
%
%         for i=1:sample_time
%             %sample_=i;
%             processing = ( k*sample_time*tweak_time ...
              + i*j*k )...
%                 /( num_of_requirements * ...
                  sample_time * tweak_time)
%             f(k).tweak(j).front(i).individual=nsga_2(R,C);
%             pareto_range(k).tweak(j).individual=...
%                 [pareto_range(k).tweak(j).individual;...
%                 [f(k).tweak(j).front(i).individual...
                                     (:,V+M+1),...
%                 f(k).tweak(j).front(i).individual...
                                     (:,V+M+2)]...
%                 ];
%         end
%     end
% end
```

```
figure(1)
hold on
 for k = 1 : num_of_requirements
% k=5;
    for j = 1 : tweak_time
%            i=2;
        switch j
            case 1
                clr='ro';
                for i = 1 : sample_time
                    if f(k).tweak(j).front(i)....
                    individual(i,V + M + 1)==1
                        plot(f(k).tweak(j).front(i)....
                        individual(:,V + 1),...
                        f(k).tweak(j).front(i)....
                        individual(:,V + 2),clr);
                    end
                end
            case 2
                clr='g+';
            case 3
                clr='bs'
                for i = 1 : sample_time
                    if f(k).tweak(j).front(i)....
                      individual(i,V + M + 1)==1
                        plot(f(k).tweak(j)....
                        front(i).individual(:,V + 1),...
                        f(k).tweak(j).front(i)....
                        individual(:,V + 2),clr);
                    end
                end
            case 4
                clr='k*'
            case 5
                clr='yd'

                for i = 1 : sample_time
                    if f(k).tweak(j).front(i).individual...
                    (i,V + M + 1)==1
                     plot(f(k).tweak(j).front(i).individua...
                     l(:,V + 1),...
                     f(k).tweak(j).front(i).individual...
```

```
                        (:,V + 2),clr);
                      end
                 end
         end
     end
 end

% plot(pareto_range(k).tweak(j).individual(:,1),...
%      pareto_range(k).tweak(j).individual(:,2),'ko',);
```

A.2 Code for Fairness Analysis

Listing A.23: nsga_2FA

```
%  function f=nsga_2(dataFlag,pro)
  clear;
  clc;
  pro = 401;
%%
pop = 200;     % population
gen = 100;      % generation

drawflag=0;
saveflag=1;
sample_rate=1;
CheckFullFlag=0;

  dataFlag = 3;
switch dataFlag
    case 1
        load('initialRand');
        dir       = 'results\1_Rand\';
        dataset   = '1_RandomData_';
        dataname = 'Rand_Data';
    case 2
        load('initialMoto');
        dir       = 'results\2_Moto\';
        dataset   = '2_MotoData___';
        dataname = 'Moto';
    case 3
        load('initialGreer');
        dir       = 'results\3_Greer\';
        dataset   = '3_GreerData__';
```

```
        dataname = 'Greer';
end

[m,n] = size(C);

% pro = 1;
% Max the sum of FRV value for each customer
% pro = 2;
% Max the overall coverage of requirement for ...
  each customer
% pro = 3;
% cost vs score
% pro = 4;
% four objectives: cost,score,expected value of ...
ranking,standard deviation of rank
% pro = 5;     % three objectives: cost,score,...
                 standard deviation of rank
% pro = 6;     % two objectives: expected value,...
                 and standard deviation of
%                (fulfilled_rank) / (total_rank)
% pro = 7;     % two objectives: Min Cost, ...
                 Min standard deviation of
%                (fulfilled_rank) / (total_rank)
%==================================================%
% pro = 8;     % two objectives:
                 Max mean of '4' coverage
%              % Min std
% pro = 9;     % two objectives:
                 Max mean of '3' coverage
%              % Min std
% pro = 10;    % two objectives:
                 Max mean of '2' coverage
%              % Min std
%               there is a point with std=0
% pro = 11;    % two objectives: Max mean
of '1' coverage
%              %                 Min std
% pro = 12;    % two objectives: Max mean of '5' coverage
%              %                 Min std
% pro = 13;    % three objectives: Min Cost
%              %                   Max Mean of average requirements
%              %                   Min Std of fulfilled requirements
%==================================================
```

```
%====== two objectives: NUMBER ====================
%==================================================
% pro = 101;
% %
%==================================================
%========= two objectives: VALUES ================
%==================================================
% pro = 201;
% %
%==================================================
%==================== three objectives ==========
%==================================================
% pro = 301;
% %


switch pro
    case {1,2,3,6,7,8,9,10,11,12,501}
        M = 2;
    case {4,401}
        M = 4;       % M : the number of objectives
    case {5,13}
        M = 3;
    otherwise
        disp('unknown');
end

if pro>100&&pro<299
    M = 2;
end

if pro>300&&pro<400
    M = 3;
end



% chromosome = zeros (pop,V+4);
% chromosome matrix
chromosome = initialize_variables(pop,R,C,M,pro);

chromosome(pop,n+M+2) = zeros;
```

```
chromosome = non_domination_sort_mod(chromosome,M,n);

save ([dir,dataset,'pro_',int2str(pro),'_0gen'],...
       'chromosome')

if drawflag==1
    drawswitch(pro,dataname);
end

if drawflag==1
    switch M
        case 2
            h=plot(chromosome(:,n + 1),-chromosome...
            (:,n + 2),'r*');
        case {4,5}
            h=plot3(chromosome(:,n + 1),-chromosome...
            (:,n + 2),chromosome(:,n + 3),'*');
            hold on
            grid on
            title('Score vs Cost vs Average Rank')
            xlabel('Score')
            ylabel('Cost')
            zlabel('Average Rank')
        case 3
            h=plot3(chromosome(:,n + 1),-chromosome...
            (:,n + 2),chromosome(:,n + 3),'r*');
            hold on
            grid on
%               title('Cost vs Mean and Std of ...
Fulfilled Values')
%               xlabel('Cost')
%               ylabel('Average Fulfilled Values')
%               zlabel('Standard Deviation')
    end
end

for i = 1 : gen
    %       if i==2||i==4||i==6||i==8||i==10||...
    i==15||i==20
    %               save (['front',int2str(pro),'_',...
    int2str(i),'gen'], 'chromosome')
    %       end
```

```
pool = round(pop/2);
i

% what is this magic number?
tour = 5;


parent_chromosome = tournament_selection...
(chromosome, pool, tour);


offspring_chromosome = genetic_operator...
(parent_chromosome,R,C,M,n,pro,CheckFullFlag);


[main_pop,temp] = size(chromosome);
[offspring_pop,temp] = size(offspring_chromosome);
intermediate_chromosome(1:main_pop,:) = chromosome;
intermediate_chromosome(main_pop + 1 : ...
main_pop + offspring_pop,1 : M+n) = ...
    offspring_chromosome;


intermediate_chromosome = ...
    non_domination_sort_mod(intermediate_chromosome,M,n);

chromosome = replace_chromosome(intermediate_chromosome,pop,M,n);

%       if pro==5
%           qq=[chromosome(:,n + 1),-chromosome...
(:,n + 2),chromosome(:,n + 3)];
%           x=sort(qq,1,'descend');
%           if (mod(i,10))==5
%               plot3(x(:,1),x(:,2),x(:,3),'r.-');
%               hold on
%           elseif ~mod(i,10)
%               plot3(x(:,1),x(:,2),x(:,3),'k.-');
%               hold on
%           else
%               plot3(x(:,1),x(:,2),x(:,3),'.-');
%               hold on
%           end
%       end
```

```
    %       X(:,i)=aa(:,1);
    %       Y(:,i)=aa(:,2);
    %       Z(:,i)=aa(:,3);
    %
    %generation_=i

    if drawflag==1
        switch M
            case 2
                %           for i = 1 : pop
                %               if (chromosome(i,n+M+1)==1)
                h=plot(chromosome(:,n + 1),...
                -chromosome(:,n + 2),'*');
                %               end
                %           end
            case {4,5}
                h=plot3(chromosome(:,n + 1),...
                -chromosome(:,n + 2),chromosome(:,n + 3),'*');
                hold on
                grid on
                title('Score vs Cost vs Average Rank')
                xlabel('Score')
                ylabel('Cost')
                zlabel('Average Rank')
            case 3
                h=plot3(chromosome(:,n + 1),...
                -chromosome(:,n + 2),...
                    chromosome(:,n + 3),'*');
        end
    end

    if ~mod(i,sample_rate)
        fprintf('%d\n',i);
        if saveflag==1
            save ([dir,dataset,'pro_',int2str...
            (pro),'_',int2str(i),'gen'], 'chromosome')
        end
    end
end



%save solution.txt chromosome -ASCII
```

Listing A.24: evaluate_objectiveFA

```
function f = evaluate_objective(x,R,C,M,pro)


% Function to evaluate the objective functions for the given input vecto
% x. x has the decision variables

f = [];

%[temp,n]=size(R);
[m,n]=size(C);

switch pro
%% ===================== two objectives: NUMBER ==
%==============================================
    case 101
        % pro = 101; absolute: Max mean, Min std ON
        % %
        f = pro101(x,R,C,m,n);
    case 102
        % pro = 102; persentage: Max mean, Min std ON
        % %
        f = pro102(x,R,C,m,n);

%% ===================== two objectives: VALUES ==
%==============================================
    case 201
        % pro = 201; absolute: Max mean, Min std ON
        % %
        f = pro201(x,R,C,m,n);
    case 202
        % pro = 202; persentage: Max mean, Min std ON
        % %
        f = pro202(x,R,C,m,n);

%% ===================== three objectives ========
%==============================================
    case 301
        % pro = 301; absolute: Max mean, Min std ON
        % %
        f = pro101(x,R,C,m,n);

        sum_cost=0;
```

```
        for i = 1 : n
            sum_cost = sum_cost + R(1,i)*x(i);
        end
        %% Objective function two
        f(3) = -sum_cost;

    case 302
        % pro = 302; persentage: Max mean, Min std ON
        % %

        f = pro102(x,R,C,m,n);

        sum_cost=0;
        for i = 1 : n
            sum_cost = sum_cost + R(1,i)*x(i);
        end
        %% Objective function two
        f(3) = -sum_cost;

    case 303
        % pro = 301; absolute: Max mean, Min std ON
        % %
        f = pro201(x,R,C,m,n);

        sum_cost=0;
        for i = 1 : n
            sum_cost = sum_cost + R(1,i)*x(i);
        end
        %% Objective function two
        f(3) = -sum_cost;

    case 304
        % pro = 301; absolute: Max mean, Min std ON
        % %
        f = pro202(x,R,C,m,n);

        sum_cost=0;
        for i = 1 : n
            sum_cost = sum_cost + R(1,i)*x(i);
        end
        %% Objective function two
        f(3) = -sum_cost;
```

```
%% ====================== four objectives =========
%===================================================
    case 401
        % pro = 301; absolute Value: Max mean, Min std ON
        % %
        f = pro202(x,R,C,m,n);

        sum_cost=0;
        for i = 1 : n
            sum_cost = sum_cost + R(1,i)*x(i);
        end
        %% Objective function two
        f(3) = -sum_cost;

        % Objective function four
        f(4) = proMinStdCost(x,R,C,m,n);
%% Cost: Min Cost, Min Std of Cost_array
% ==================================================
    case 501
        sum_cost=0;
        for i = 1 : n
            sum_cost = sum_cost + R(1,i)*x(i);
        end

        %% Objective function two
        f(1) = -sum_cost;

        % Objective function four
        f(2) = proMinStdCost(x,R,C,m,n);


%% ==================== Others =====================
%===================================================
case 1
        % Customer Rank
        % two objectives:   f1: max expected ...
          value of values from each customer
        %                   f2: min standard ...
          deviation of value for each
        %                       customer
        %           sum_of_rank = 0;
        %           allrank = sum(C);
        %           for i = 1 : n
```

```
        % n : requirement number
        %             sum_of_rank = ...
          sum_of_rank + x(i)*allrank(i);
        %         end


        sumrank_all_customer = zeros(1,m);
        for i = 1 : n
            for j = 1 : m
                sumrank_all_customer(j) = ...
                    sumrank_all_customer(j) + ...
                    x(i)*C(j,i);
            end
        end

        %% Objective function one
        f(1) = mean(sumrank_all_customer);

        %% Objective function two
        f(2) = -std(sumrank_all_customer);

    case 6
        % F/T := (fulfilled_rank) / (total_rank)
        % two objective:    f1: max the ...
          expected value of F/T
        %                   f2: min the ...
          standard deviation of F/T
        %

        sum_of_rank_matrix = zeros(m,n);
        %         coverage_matrix = zeros(m,n);
        %         x = x(1:n);
        for i = 1 : m
            for j = 1 : n
                sum_of_rank_matrix(i,j) = C(i,j)*x(j);
            end
        end
        sum_of_FT_array = zeros(1,m);
        for i = 1 : m
            sum_of_FT_array(i) = ...
            sum(sum_of_rank_matrix(i,:)) ...
                /sum(C(i,:));
        end
```

```
    %% Objective function one
    %total_sum_of_rank = sum(sum...
    (sum_of_FT_matrix));
    %          allones_matrix = ones(m,n);
    % most_possible_coverage_matrix = ...
      C&allones_matrix;

    f(1) = mean(sum_of_FT_array);

    %     individual_sum_of_rank_array = zeros(1,m);
    %     for j = 1 : m
    %          individual_sum_of_rank_array(j) = ...
    %                    sum(sum_of_rank_matrix(j,:));
    %          end
    %
    %% Objective function two
    f(2) = -std(sum_of_FT_array);


case 13
    % Cost vs Mean and Std of
    % three objectives: f1: Min the over all Cost
    %                   f2: Max the average ...
                        of fulfilled requirements
    %                   f3: Min the Std of ...
                        fulfilled requirements

    sum_cost=0;
    for i = 1 : n
        sum_cost = sum_cost + R(1,i)*x(i);
    end
    %% Objective function two
    f(1) = -sum_cost;

    % F/T := (fulfilled_rank) / (total_rank)
    % two objective:    f1: max the expected...
                        value of F/T
    %                   f2: min the standard...
                        deviation of F/T
    %

    sum_of_rank_matrix = zeros(m,n);
    %          coverage_matrix = zeros(m,n);
```

```
        %           x = x(1:n);
        for i = 1 : m
            for j = 1 : n
                sum_of_rank_matrix(i,j) = C(i,j)*x(j);
            end
        end
        sum_of_FT_array = zeros(1,m);
        for i = 1 : m
            sum_of_FT_array(i) =
            sum(sum_of_rank_matrix(i,:)) ...
                /sum(C(i,:));
        end
        %% Objective function one
        %total_sum_of_rank = sum(sum(sum_of_FT_matrix));
        %           allones_matrix = ones(m,n);
        %           most_possible_coverage_matrix...
                     = C&allones_matrix;
        f(2) = mean(sum_of_FT_array);

        %           individual_sum_of_rank_array = zeros(1,m);
        %           for j = 1 : m
        %               individual_sum_of_rank_array(j) = ...
        %                   sum(sum_of_rank_matrix(j,:));
        %           end
        %
        %% Objective function two
        f(3) = -std(sum_of_FT_array);



    case 2
        % Coverage for all
        % two objective:     f1:
        max the average of coverage rate
        %                    f2:
        min the standard deviation of coverage of
        %                    each customers

        sum_of_coverage = 0;
        coverage_matrix = zeros(m,n);
        x = x(1:n);
        for i = 1 : m
            coverage_matrix(i,:) = C(i,:) & x ;
```

```
    end
    %sum_of_coverage = sum(sum(coverage_matrix));
    allones_matrix = ones(m,n);
    most_possible_coverage_matrix = C&allones_matrix;
    customer_coverage_array = zeros(1,m);
    for i = 1 : m
        customer_coverage_array(i) = ...
            sum(coverage_matrix(i,:)) ...
            / sum(most_possible_coverage_matrix(i,:));
    end

    %% Objective function one
    f(1) = mean(customer_coverage_array) ;
    %% Objective function two
    f(2) = -std(customer_coverage_array);

    %           customer_coverage_array = zeros(1,m);
    %           sum_of_requirements = 0;
    %           for j = 1 : m
    %               sum_of_requirements = sum(C(j,:)...
                    & ones(1,n));
    %               customer_coverage_array(j)= ...
    %                   10*(sum(coverage_matrix(j,:))) ...
                        / sum_of_requirements;
    %           end

case 3  % cost vs score
    %sum1 = 0;
    score = 0;
    sum_of_rank = 0;
    for i=1 : n         % M : requirement
        for j=1:m
            sum_of_rank = sum_of_rank + ...
            C(j,i)*x(i); % sum1 : score
        end
        score = score + R(2,i)*sum_of_rank;
    end
    %% Objective function one
    f(1) = score;

    sum_cost=0;
    for i = 1 : n
        sum_cost = sum_cost+R(1,i)*x(i);        ...
```

```
        % sum2 : cost
    end

    %sum2=100-sum2;
    %% Objective function two
    f(2) = -sum_cost;

case 4  % Min cost, Max score,
    % Max expected value of Customer ...
      Ranking, Min standard deviation
    %sum1 = 0;
    score = 0;
    sum_of_rank = 0;
    for i=1 : n          % M : requirement
        for j=1:m
            sum_of_rank = sum_of_rank +...
            C(j,i)*x(i); % sum1 : score
        end
        score = score + R(2,i)*sum_of_rank;
    end
    %% Objective function one
    f(1) = score;

    sum_cost=0;
    for i = 1 : n
        sum_cost = sum_cost+R(1,i)*x(i); ...
        % sum2 : cost
    end

    %sum2=100-sum2;
    %% Objective function two
    f(2) = -sum_cost;

    % Customer Rank
    % two objectives:   f1: max expected ...
    value of rank from each customer
    %                   f2: min standard ...
    deviation of value for each
    %                   customer
    sum_of_rank = 0;
    allrank = sum(C);
    for i = 1 : n          % n : requirement number
        sum_of_rank = sum_of_rank + ...
```

```
            x(i)*allrank(i);
        end
        %% Objective function one
        f(3) = sum_of_rank/(n*m) ;


        sumrank_all_customer = zeros(1,m);
        for i = 1 : n
            for j = 1 : m
                sumrank_all_customer(j) = ...
                    sumrank_all_customer(j) + ...
                    x(i)*C(j,i);     % sum2 :
            end
        end

        %% Objective function two
        f(4) = -std(sumrank_all_customer);

    case 5  % Min cost, Max score,
        % Min standard deviation
        %sum1 = 0;
        score = 0;
        sum_of_rank = 0;
        for i=1 : n          % M : requirement
            for j=1:m
                sum_of_rank = sum_of_rank +...
                C(j,i)*x(i); % sum1 : score
            end
            score = score + R(2,i)*sum_of_rank;
        end
        %% Objective function one
        f(1) = score;

        sum_cost=0;
        for i = 1 : n
            sum_cost = sum_cost+R(1,i)*x(i); ...
            % sum2 : cost
        end

        %sum2=100-sum2;
        %% Objective function two
        f(2) = -sum_cost;
```

```
        % Customer Rank
        % two objectives:    f1: max expected ...
        value of rank from each customer
        %                    f2: min standard ...
        deviation of value for each
        %                    customer

        sumrank_all_customer = zeros(1,m);
        for i = 1 : n
            for j = 1 : m
                sumrank_all_customer(j) = ...
                    sumrank_all_customer(j) + ...
                    x(i)*C(j,i);      % sum2 :
            end
        end

        %% Objective function two
        f(3) = -std(sumrank_all_customer);

    case 7
        % Coverage for all
        % two objective:     f1: max the ...
        expected value of sum_of_rank of
        %                    fulfilled requirement
        %                    f2: min the ...
        standard deviation
        %

        sum_of_rank_matrix = zeros(m,n);
        %          coverage_matrix = zeros(m,n);
        %          x = x(1:n);
        for i = 1 : m
            for j = 1 : n
                sum_of_rank_matrix(i,j) =...
                C(i,j)*x(j);
            end
        end
        sum_of_FT_array = zeros(1,m);
        for i = 1 : m
            sum_of_FT_array(i) = sum...
            (sum_of_rank_matrix(i,:))...
                / sum(C(i,:));
        end
```

```
%% Objective function one
%total_sum_of_rank = sum(sum...
            (sum_of_FT_matrix));
%           allones_matrix = ones(m,n);
%           most_possible_coverage_matrix =...
C&allones_matrix;
%           f(1) = mean(sum_of_FT_array);

%           individual_sum_of_rank_array = zeros(1,m);
%           for j = 1 : m
%               individual_sum_of_rank_array(j) = ...
%                   sum(sum_of_rank_matrix(j,:));
%           end
%
%% Objective function two
f(1) = -std(sum_of_FT_array);

sum_cost=0;
for i = 1 : n
    sum_cost = sum_cost+R(1,i)*x(i);     ...
    % sum2 : cost
end

%sum2=100-sum2;
%% Objective function two
f(2) = -sum_cost;
%=====================================%
case 8
    % Coverage for '4'
    % two objective:    f1: max the average...
    of coverage for '4'
    %                   f2: min the standard...
    deviation of coverage for
    %                   '4' of
    %                   each customers

    sum_of_coverage = 0;
    coverage_matrix = zeros(m,n);
    x = x(1:n);
    objective_matrix = zeros(m,n);
    for i = 1 :m
        for j = 1 : n
            if (C(i,j)==4)
```

```
                    objective_matrix(i,j) = 1;
                end
            end
        end
        for i = 1 : m
            coverage_matrix(i,:) = ...
            objective_matrix(i,:) & x ;
        end
        %sum_of_coverage = sum(sum(coverage_matrix));
        %           allones_matrix = ones(m,n);
        %           most_possible_coverage_matrix ...
        = objective_matrix&allones_matrix;
        customer_coverage_array = zeros(1,m);
        for i = 1 : m
            customer_coverage_array(i) = ...
                sum(coverage_matrix(i,:)) ...
                / sum(objective_matrix(i,:));
        end

        %% Objective function one
        f(1) = mean(customer_coverage_array) ;
        %% Objective function two
        f(2) = -std(customer_coverage_array);

        %           customer_coverage_array = zeros(1,m);
        %           sum_of_requirements = 0;
        %           for j = 1 : m
        %               sum_of_requirements = ...
                        sum(C(j,:) & ones(1,n));
        %               customer_coverage_array(j)= ...
        %                   10*(sum(coverage_matrix(j,:)))...
                            / sum_of_requirements;
        %           end
        %==================================%
    case 9
        % Coverage for '3'
        % two objective:     f1: max the average...
        of coverage for '4'
        %                    f2: min the standard....
        deviation of coverage for
        %                    '4' of
        %                    each customers
```

```
    cover_for = 3;
    sum_of_coverage = 0;
    coverage_matrix = zeros(m,n);
    x = x(1:n);
    objective_matrix = zeros(m,n);
    for i = 1 :m
        for j = 1 : n
            if (C(i,j)==cover_for)
                objective_matrix(i,j) = 1;
            end
        end
    end
    for i = 1 : m
        coverage_matrix(i,:) = ...
        objective_matrix(i,:) & x ;
    end

    customer_coverage_array = zeros(1,m);
    for i = 1 : m
        customer_coverage_array(i) = ...
            sum(coverage_matrix(i,:)) ...
            / sum(objective_matrix(i,:));
    end

    %% Objective function one
    f(1) = mean(customer_coverage_array) ;
    %% Objective function two
    f(2) = -std(customer_coverage_array);
    %==================================%
case 10
    % Coverage for '2'
    % two objective:     f1: max the ...
    average of coverage for '4'
    %                    f2: min the ....
    standard deviation of coverage for
    %                    '4' of
    %                    each customers

    cover_for = 2;
    sum_of_coverage = 0;
    coverage_matrix = zeros(m,n);
    x = x(1:n);
    objective_matrix = zeros(m,n);
```

```
    for i = 1 :m
        for j = 1 : n
            if (C(i,j)==cover_for)
                objective_matrix(i,j) = 1;
            end
        end
    end
    for i = 1 : m
        coverage_matrix(i,:) = ...
        objective_matrix(i,:) & x ;
    end

    customer_coverage_array = zeros(1,m);
    for i = 1 : m
        customer_coverage_array(i) = ...
            sum(coverage_matrix(i,:)) ...
            / sum(objective_matrix(i,:));
    end

    %% Objective function one
    f(1) = mean(customer_coverage_array) ;
    %% Objective function two
    f(2) = -std(customer_coverage_array);
    %=====================================%
case 11
    % Coverage for '1'
    % two objective:     f1: max the ...
    average of coverage for '4'
    %                    f2: min the ...
    standard deviation of coverage for
    %                    '4' of
    %                    each customers

    cover_for = 1;
    sum_of_coverage = 0;
    coverage_matrix = zeros(m,n);
    x = x(1:n);
    objective_matrix = zeros(m,n);
    for i = 1 :m
        for j = 1 : n
            if (C(i,j)==cover_for)
                objective_matrix(i,j) = 1;
            end
```

```
        end
    end
    for i = 1 : m
        coverage_matrix(i,:) = ...
        objective_matrix(i,:) & x ;
    end

    customer_coverage_array = zeros(1,m);
    for i = 1 : m
        if (sum(objective_matrix(i,:))==0)
            customer_coverage_array(i) = 1;
        else
            customer_coverage_array(i) = ...
                sum(coverage_matrix(i,:)) ...
                / sum(objective_matrix(i,:));
        end
    end

    %% Objective function one
    f(1) = mean(customer_coverage_array) ;
    %% Objective function two
    f(2) = -std(customer_coverage_array);
    %==========================================%
case 12
    % Coverage for '5'
    % two objective:    f1: max the ...
    average of coverage for '4'
    %                   f2: min the ...
    standard deviation of coverage for
    %                   '4' of
    %                   each customers

    cover_for = 5;
    sum_of_coverage = 0;
    coverage_matrix = zeros(m,n);
    x = x(1:n);
    objective_matrix = zeros(m,n);
    for i = 1 :m
        for j = 1 : n
            if (C(i,j)==cover_for)
                objective_matrix(i,j) = 1;
            end
        end
```

```
        end
        for i = 1 : m
            coverage_matrix(i,:) = ...
            objective_matrix(i,:) & x ;
        end

        customer_coverage_array = zeros(1,m);
        for i = 1 : m

            customer_coverage_array(i) = ...
                sum(coverage_matrix(i,:)) ...
                / sum(objective_matrix(i,:));

        end

        %% Objective function one
        f(1) = mean(customer_coverage_array) ;
        %% Objective function two
        f(2) = -std(customer_coverage_array);



        %==========================================%
    otherwise
        disp('Unknown Problem!!')
end
```

Listing A.25: initialGreer

```
function initialGreer()
clear;

m=5;    % number of customers
n=20;   % number of requirements

%% Variables Description
% R: requirement matrix
% R(1,:)--cost of each requirement
% R(2,:)--value of each requirement
% C: customers Ranking matrix
% C(i,j)--the i.th customer gives the rank...
  value for the j.th requirement

R = ones(2, n);
```

```
C = zeros(m, n);


% R(1,:)=[...
%      100, 50, 300, 80, 70, 100, 1000, ...
%      40, 200, 20, 1100, 10, 500, 10, ...
%      10, 10, 20, 200, 1000, 120, 300, ...
%      50, 10, 30, 110, 230, 40, 180, ...
%      20, 150, 60, 100, 400, 80, 40];
%
% R(2,:)=[...
%      3, 3, 1, 1, 1, 3, 3, ...
%      2, 2, 2, 2, 1, 1, 1, ...
%      1, 1, 3, 3, 3, 2, 2, ...
%      2, 2, 2, 1, 1, 1, 1, ...
%      1, 1, 1, 1, 3, 3, 4];

C=[...
     4 2 1 2 5  5 2 4 4 4  2 3 4 2 4  4 4 1 3 2 ;...
     4 4 2 2 4  5 1 4 4 5  2 3 2 4 4  2 3 2 3 1 ;...
     5 3 3 3 4  5 2 4 4 4  2 4 1 5 4  1 2 3 3 2 ;...
     4 5 2 3 3  4 2 4 2 3  5 2 3 2 4  3 5 4 3 2 ;...
     5 4 2 4 5  4 2 4 5 2  4 5 3 4 4  1 1 2 4 1 ];

save initialGreer R C;
```

Listing A.26: initialMoto

```
% function initialMoto()
clear;

m=4;    % number of customers
n=35;    % number of requirements

seed_for_rand = 10;
rand('seed',seed_for_rand);

%% Variables Description
% R: requirement matrix
% R(1,:)--cost of each requirement
% R(2,:)--value of each requirement
% C: customers Ranking matrix
% C(i,j)--the i.th customer gives...
  the rank value for the j.th requirement
```

```
R = zeros(2, n);
C = zeros(m, n);


R(1,:)=[...
    100,    50,     300,    80,     70, ...
    100,    1000,   40,     200,    20, ...
    1100,   10,     500,    10,     10, ...
    10,     20,     200,    1000,   120, ...
    300,    50,     10,     30,     110, ...
    230,    40,     180,    20,     150, ...
    60,     100,    400,    80,     40 ...
    ];

R(2,:)=[...
    3, 3, 3, 3, 3, ...
    3, 3, 3, 3, 1, ...
    3, 3, 3, 1, 3, ...
    2, 1, 1, 3, 2, ...
    2, 1, 2, 3, 2, ...
    2, 1, 2, 2, 2, ...
    3, 1, 3, 1, 1  ...
    ];

 C(1,:) = [...
     0, 0, 1, 1, 1, ...
     0, 0, 0, 0, 0, ...
     0, 1, 1, 1, 1, ...
     1, 0, 0, 0, 0, ...
     0, 0, 0, 0, 1, ...
     1, 1, 1, 1, 1, ...
     1, 1, 0, 0, 0, ...
     ];
 C(2,:) = [...
     0, 0, 0, 0, 0, ...
     0, 0, 1, 1, 1, ...
     1, 0, 0, 0, 0, ...
     0, 0, 0, 0, 1, ...
     1, 1, 1, 1, 0, ...
     0, 0, 0, 0, 0, ...
     0, 0, 0, 0, 0, ...
     ];
```

```
C(3,:) = [...
      1, 1, 0, 0, 0, ...
      1, 1, 0, 0, 0, ...
      0, 0, 0, 0, 0, ...
      0, 1, 1, 1, 0, ...
      0, 0, 0, 0, 0, ...
      0, 0, 0, 0, 0, ...
      0, 0, 1, 1, 0, ...
      ];
C(4,35) = 1;

save initialMoto R C;
```

Listing A.27: initialRand

```
function initialRand()
clear;

m=5;    % number of customers
n=30;   % number of requirements

seed_for_rand = 10;
rand('seed',seed_for_rand);

%% Variables Description
% R: requirement matrix
% R(1,:)--cost of each requirement
% R(2,:)--value of each requirement
% C: customers Ranking matrix
% C(i,j)--the i.th customer ...
   gives the rank value for the j.th requirement

R = zeros(2, n);
C = zeros(m, n);


R(1,:)=[...
     100,    50,     300,    80,     70    ,...
     100,    1000,   40,     200,    20    ,...
     1100,   10,     500,    10,     10    ,...
     10,     20,     200,    1000,   120   ,...
     300,    50,     10,     30,     110   ,...
     230,    40,     180,    20,     150   ];%   ,...
%      60,     100,    400,    80            ...%,     40 ...
```

```
%       ];

R(2,:)=[...
     3, 3, 3, 3, 3, ...
     3, 3, 3, 3, 1, ...
     3, 3, 3, 1, 3, ...
     2, 1, 1, 3, 2, ...
     2, 1, 2, 3, 2, ...
     2, 1, 2, 2, 2]; %, ...
%       3, 1, 3, 1        ...%, 1   ...
%       ];

for i=1:m
     for j=1:n
          C(i,j)=round(5*rand(1));
     end
end

save initialRand R C;
```

Listing A.28: pro101

```
function f = pro101(x,R,C,m,n)
% Number Absolute
% two objectives:   f1: max average of number of fulfilled
%                       requirements
%                   f2: min standard deviation
x = x(1:n);
coverage_matrix = zeros(m,n);
for i = 1 : m
     coverage_matrix(i,:) = C(i,:) & x ;
end

num_all_customer = zeros(1,m);
for i = 1 : m
     num_all_customer(i) = sum(coverage_matrix(i,:));
end

%% Objective function one
f(1) = mean(num_all_customer);

%% Objective function two
f(2) = -std(num_all_customer);
```

Listing A.29: pro102

```
function f = pro102(x,R,C,m,n)

        % Coverage for all
        % two objective:     f1: max ...
                the average of coverage rate
        %                    f2: min ...
                the standard deviation of coverage of
        %       each customers

%          sum_of_coverage = 0;
        coverage_matrix = zeros(m,n);
        x = x(1:n);
        for i = 1 : m
            coverage_matrix(i,:) = C(i,:) & x ;
        end
        %sum_of_coverage = sum(sum(coverage_matrix));
        allones_matrix = ones(m,n);
        most_possible_coverage_matrix = C&allones_matrix;
        customer_coverage_array = zeros(1,m);
        for i = 1 : m
            customer_coverage_array(i) = ...
                sum(coverage_matrix(i,:)) ...
                / sum(most_possible_coverage_matrix(i,:));
        end

        %% Objective function one
        f(1) = mean(customer_coverage_array) ;
        %% Objective function two
        f(2) = -std(customer_coverage_array);
```

Listing A.30: pro201

```
function f = pro101(x,R,C,m,n)
% Number Absolute
% two objectives:   f1: max average of number of fulfilled
%                       requirements
%                   f2: min standard deviation
x = x(1:n);
coverage_matrix = zeros(m,n);
for i = 1 : m
    coverage_matrix(i,:) = C(i,:) & x ;
end
```

```
num_all_customer = zeros(1,m);
for i = 1 : m
    num_all_customer(i) = sum(coverage_matrix(i,:));
end

%% Objective function one
f(1) = mean(num_all_customer);

%% Objective function two
f(2) = -std(num_all_customer);
```

Listing A.31: pro202

```
function f = pro102(x,R,C,m,n)

        % Coverage for all
        % two objective:     f1: max ...
                 the average of coverage rate
        %                    f2: min ...
                 the standard deviation of coverage of
        %        each customers

%        sum_of_coverage = 0;
        coverage_matrix = zeros(m,n);
        x = x(1:n);
        for i = 1 : m
            coverage_matrix(i,:) = C(i,:) & x ;
        end
        %sum_of_coverage = sum(sum(coverage_matrix));
        allones_matrix = ones(m,n);
        most_possible_coverage_matrix = C&allones_matrix;
        customer_coverage_array = zeros(1,m);
        for i = 1 : m
            customer_coverage_array(i) = ...
                sum(coverage_matrix(i,:)) ...
                / sum(most_possible_coverage_matrix(i,:));
        end

        %% Objective function one
        f(1) = mean(customer_coverage_array) ;
        %% Objective function two
        f(2) = -std(customer_coverage_array);
```

Listing A.32: proCost

```
function f = proCost(x,R,C,m,n)
% COST
% two objective:      f1: ...
                      min the overall cost
%                     f2:...
                      min the standard
                      deviation of cost spend on each
%                     customer
%

sum_cost=0;
for i = 1 : n
    sum_cost = sum_cost + R(1,i)*x(i);
end
%% Objective function one
f(1) = -sum_cost;



sum_of_cost_matrix = zeros(m,n);
%          coverage_matrix = zeros(m,n);
%          x = x(1:n);
for i = 1 : m
    for j = 1 : n
        sum_of_cost_matrix(i,j)...
        = C(i,j)*x(j);
    end
end
sum_of_FT_array = zeros(1,m);
for i = 1 : m
    sum_of_FT_array(i) = sum(...
    sum_of_rank_matrix(i,:)) ...
        /sum(C(i,:));
end

%% Objective function two
f(2) = -std(sum_of_FT_array);
```

Listing A.33: proMinStdCost

```
function f = proMinStdCost(x,R,C,m,n)
% COST
% two objective:      f1: min the overall cost
%                     f2: min the standard...
```

```
                deviation of cost spend on each
%                       customer
%


sum_of_cost_matrix = zeros(m,n);
%           coverage_matrix = zeros(m,n);
%           x = x(1:n);
for i = 1 : m
    for j = 1 : n
        sum_of_cost_matrix(i,j) = ...
        C(i,j)*R(1,j)*x(j);
    end
end

sum_of_cost_array = zeros(1,m);
for i = 1 : m
    sum_of_cost_array(i) = ...
    sum(sum_of_cost_matrix(i,:));
end

%% Objective function two
f = -std(sum_of_cost_array);
```

Listing A.34: spyOnC

```
% function spyOnC
figure
hold on


load initialRand
subplot(3,1,1); spy(C)
title('Random Data','FontSize',13)
xlabel('')
% ylabel('Customers')


load initialMoto
subplot(3,1,2); spy(C)
title('Data from Moto','FontSize',13)
xlabel('')
ylabel('Customers')

```

```
load initialGreer
subplot(3,1,3); spy(C)
title('Data␣from␣Greer','FontSize',13)
xlabel('Requirements')
% ylabel('Customers')
```

Listing A.35: drawswitch

```
function drawswitch(pro,dataname)
figure
grid on
switch pro
    case 101
        hold on
        title({'Absolute␣Number␣of␣Fulfilled...
␣␣␣␣␣␣␣␣Requirements␣';...
            ['(Results␣for:␣',dataname,')']}...
            ,'FontSize',13,'FontWeight','Demi')
        xlabel('Average␣Number␣of␣Fulfilled␣...
␣␣␣␣␣␣␣␣Requirements')
        ylabel('Standard␣Deviation')
        %               xlim([15 85])
        %               ylim([0 4.5])
    case 102
        hold on
        title({'Persentage␣of␣Fulfilled␣...
␣␣␣␣␣␣␣␣Requirements␣(Number)';...
            ['(Results␣for:␣',dataname,')...
␣␣␣␣␣␣␣␣␣␣␣␣']},'FontSize',13,'FontWeight','Demi')
        xlabel('Average␣Persentage')
        ylabel('Standard␣Deviation')
        %               xlim([15 85])
        %               ylim([0 4.5])
    case 201
        hold on
        title({'Absolute␣Values␣of␣Fulfilled...
␣␣␣␣␣␣␣␣Requirements';...
            ['(Results␣for:␣',dataname,')']},...
            'FontSize',13,'FontWeight','Demi')
        xlabel('Average␣Valures')
        ylabel('Standard␣Deviation')
        %               xlim([15 85])
        %               ylim([0 4.5])
```

```
    case 202
        hold on
        title({'Persentage␣of␣Fulfilled␣...
␣␣␣␣␣␣␣␣Requirements␣(Value)';...
            ['(Results␣for:␣',dataname,')']},...
            'FontSize',13,'FontWeight','Demi')
        xlabel('Average␣Persentage')
        ylabel('Standard␣Deviation')
        %               xlim([15 85])
        %               ylim([0 4.5])

    case 301
        hold on
        title({'Absolute␣Number␣of␣Fulfilled␣...
␣␣␣␣␣␣␣␣Requirements␣(Number)␣vs␣Cost';...
            ['(Results␣for:␣',dataname,')']},...
            'FontSize',13,'FontWeight','Demi')
        xlabel('Average␣Number␣of␣Requirements')
        ylabel('Standard␣Deviation')
        zlabel('Cost')
        %               xlim([15 85])
        %               ylim([0 4.5])
    case 302
        hold on
        title({'Persentage␣of␣Fulfilled␣...
␣␣␣␣␣␣␣␣Requirements␣(Number)␣vs␣Cost';...
            ['(Results␣for:␣',dataname,')']},...
            'FontSize',13,'FontWeight','Demi')
        xlabel('Average␣Persentage␣(Number)')
        ylabel('Standard␣Deviation')
        zlabel('Cost')
        %               xlim([15 85])
        %               ylim([0 4.5])
    case 303
        hold on
        title({'Absolute␣Value␣of␣Fulfilled␣...
␣␣␣␣␣␣␣␣Requirements␣vs␣Cost';...
            ['(Results␣for:␣',dataname,')']},...
            'FontSize',13,'FontWeight','Demi')
        xlabel('Average␣Value')
        ylabel('Standard␣Deviation')
        zlabel('Cost')
        %               xlim([15 85])
```

```
        %               ylim([0  4.5])
    case 304
        hold on
        title({'Persentage␣of␣Fulfilled␣...
␣␣␣␣␣␣␣␣Requirements␣(Value)␣vs␣Cost';...
            ['(Results␣for:␣',dataname,')']},...
            'FontSize',13,'FontWeight','Demi')
        xlabel('Average␣Persentage␣(Value)')
        ylabel('Standard␣Deviation')
        zlabel('Cost')
        %               xlim([15  85])
        %               ylim([0  4.5])

    case 30
    case 1
        hold on
        title('Fairness␣for␣each␣customer...
␣␣␣␣␣␣␣␣on␣their␣Fulfilled␣Value')
        xlabel('Expected␣Value␣of␣Fulfilled...
␣␣␣␣␣␣␣␣Values')
        ylabel('Standard␣Deviation␣of␣...
␣␣␣␣␣␣␣␣Fulfilled␣Values')
        %               xlim([15  85])
        %               ylim([0  4.5])
    case 2
        hold on
        title('Fairness␣on␣Coverage␣%')
        xlabel('Expected␣Value')
        ylabel('Standard␣Deviation')
        %               xlim([30  100])
        %               ylim([0.1  0.3])
    case 8
        hold on
        title('Fairness␣on␣Coverage␣of␣"4"')
        xlabel('Expected␣Value')
        ylabel('Standard␣Deviation')
        %               xlim([0.4  1])
        %               ylim([0  0.2])
    case 9
        hold on
        title('Fairness␣on␣Coverage␣of␣"3"')
        xlabel('Expected␣Value')
        ylabel('Standard␣Deviation')
```

```
        %               xlim([0.4  1])
        %               ylim([0  0.2])
    case 10
        hold on
        title('Fairness on Coverage of "2"')
        xlabel('Expected Value')
        ylabel('Standard Deviation')
        %               xlim([0.4  1])
        %               ylim([0  0.2])
    case 11
        hold on
        title('Fairness on Coverage of "1"')
        xlabel('Expected Value')
        ylabel('Standard Deviation')
        %               xlim([0.4  1])
        %               ylim([0  0.2])
    case 12
        hold on
        title('Fairness on Coverage of "5"')
        xlabel('Expected Value')
        ylabel('Standard Deviation')
        %               xlim([0.4  1])
        %               ylim([0  0.2])
    case 3
        hold on
        title('Cost vs Score')
        xlabel('Cost')
        ylabel('Score')
        xlim([0.4*10^5  1.8*10^5])
        ylim([0  7000])
    case 7
        hold on
        title('Cost vs Std')
        xlabel('Std of ')
        ylabel('Cost')
        xlim([-0.065  -0.02])
        ylim([0  6000])
    case 6
        hold on
        title('Fairness on fulfilled ranking')
        xlabel('Expected Value')
        ylabel('Standard Deviation')
        %               xlim([30  100])
```

```
        %                   ylim([0.1  0.3])
    case {4,5}
        hold on
    case 13
        hold on
    otherwise
        disp('Unknown Problem')
end
return;
```

Bibliography

[1] Bagnall, A., Rayward-Smith, V., and Whittley, I. The Next Release Problem. *IEE Proceedings - Software 43(14)* (Dec 2001), 883–890. 4

[2] Baker, P., Harman, M., Steinhöfel, K., and Skaliotis, A. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In *22nd International Conference on Software Maintenance (ICSM 06)* (Philadelphia, Pennsylvania, USA, September 24-27 2006), pp. 176–185. xi, 1, 7, 10, 26

[3] Chinneck, J. W. Practical Optimization: A Gentle Introduction. Lecture Notes, Systems and Computer Engineering, Carleton University, Ottawa, Canada, 2006. ix, 4

[4] Clark, J., Dolado, J. J., Harman, M., Hierons, R., Jones, B., Lumkin, M., Mitchell, B., Spiros Mancoridis, K. R., Roper, M., and Martin Shepperd, s. Reformulating Software Engineering as a Search Problem. *IEE Proceedings - Software 150(3)* (2003), 161–175. 3

[5] Greer, D., and Ruhe, G. Software Release Planning: An Evolutionary and Iterative Approach. University of Calgary, Canada, 2004. xi, 26

[6] Harman, M. The Current State and Future of Search Based Software Engineering. In *29th Int. Conference on Software Engineering (ICSE 2007), Future of Software Engineering (FoSE)*. 3

[7] Harman, M., and Jones, B. Search Based Software Engineering. *Journal of Information and Software Technology 43(14)* (2001), 833–839. 3

[8] Mankiw, N. G. Fair Taxes? Depends What You Mean by 'Fair'. *Economic View, New York Times* (Published: July 15, 2007). 5

[9] Veldhuizen, D. A. V. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovation.* PhD thesis, DS/ENG, Air Force Institute of Technology, Air University, USA, 1999. 14

[10] Yoo, S. The Use of a Novel Semi-Exhaustive Search Algorithm for the Analysis of Data Sensitivity in a Feature Subset Selection Problem. Master's thesis, DCS/PSE, King's College London, London, England, 2006. 2, 4, 13

[11] Zhang, Y., Harman, M., and Mansouri, A. The Multi-Objective Next Release Problem. In *ACM Genetic and Evolutionary Computation Conference (GECCO 2007). (London, England, 7 - 11 July 2007) To appear.* (2007). v, 4, 8, 11, 14