Multi-Objective Search-based Requirements Selection and Optimisation

by
Yuanyuan Zhang

Submitted in fulfilment of the requirements

for the degree of Doctor of Philosophy in Computing

King's College London

University of London

Strand, London WC2R 2LS, UK

February 2010

Abstract

Most software product developments are iterative and incremental processes that are seldom completed in a single release. It is critical but challenging to select the requirements from a large number of candidates for the achievement of overall business goal and the goals of multiple stakeholders, each of whom may have competing and often conflicting priorities.

This thesis argues that search-based techniques can be applied to the optimisation problem during the requirements selection and analysis phase for release planning problem. Search-based techniques offer significant advantages; they can be used to seek robust, scalable solutions, to investigate trade-offs, to yield insight and to provide feedback explaining choices to the decision maker.

In the thesis, a Search-based Requirements Selection and Optimisation Framework is proposed that includes search spaces, representations, solution processes and empirical studies. This framework formulates the requirements selection problem as an optimisation problem and allows multi-objective search-based techniques to be used in order to provide optimal or near optimal solutions and to find a suitable balance between priorities in different contexts.

The thesis reports the results of experiments using different multi-objective evolutionary optimisation algorithms with real world data sets as well as synthetic data sets in three studies of the applications of this framework: Value/Cost Trade-off in Requirements Selection, Requirements Interaction Management and Multi-Stakeholder Requirements Analysis and Optimisation. Empirical validation includes a statistical analysis of the performance of the algorithms as well as simple graphical methods to visualise the discovered solutions in the multi-dimensional solution space. Though these visualisations are not novel in themselves, the thesis is the first to use them for visualisation of requirements optimisation spaces.

Acknowledgements

I would like to thank all those who made this thesis possible. First of all, I must express my deepest respect and gratitude to my supervisor, Professor Mark Harman, for his excellent guidance, inspiration and supervision. His encouragement and assistance gave me so much confidence and helped me endure some difficult times. He has always been a great support throughout my research work. I would also like to thank my second supervisor Dr. Nicolas Gold for his suggestions during my study.

I am very grateful to Professor Anthony Finkelstein for his guidance and very valuable contributions on this work. I am really thankful to Dr. Afshin Mansouri and Jian Ren for their helpful discussions and tireless assistance.

My sincere thanks and appreciation go to Telelogic (IBM) and Sue Ward for kindly providing the *IBM Rational Focal Point* tool as well as discussion on RE. Sincere thanks also go to both Paul Baker from Motorola Inc. and Sigrid Eldh from Ericsson for providing the real world data sets to support my research work. Many thanks to my examiners Prof. Neil Maiden and Dr. Emmanuel Letier for their suggested comments on improving the quality of my thesis. Also, I am thankful to Lorna Anderson for her careful proofreading of this thesis.

I gratefully acknowledge the KC Wong Postgraduate Scholarship Programme for my financial support during my three-year study.

I particularly express my appreciation to my parents who, though thousands of miles away, have always stood by me at all times, and my boyfriend, Yue, for his constant encouragement, support and patience throughout.

Declaration

The work presented in this thesis is original work undertaken between September 2006 and April 2009 at King's College, University of London. Some of the work presented in this thesis has previously been published and submitted by the author in the following papers:

Published

- Yuanyuan Zhang, Mark Harman and S. Afshin Mansouri. The Multi-Objective Next Release Problem 9^{th} *International Conference on Genetic and Evolutionary Computation (GECCO '07)*, Vol. 1, pages 1129–1136, London, UK, 7–11 July 2007, ACM.

- Yuanyuan Zhang, Anthony Finkelstein and Mark Harman. Search Based Requirements Optimisation: Existing Work & Challenges 14^{th} *International Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ '08)*, LNCS Vol. 5025/2008, pages 88–94, Montpellier, France, 16–17 June 2008, Springer.

- Anthony Finkelstein, Mark Harman, S. Afshin Mansouri, Jian Ren and Yuanyuan Zhang. "Fairness Analysis" in Requirements Assignments 16^{th} *IEEE International Requirements Engineering Conference (RE '08)*, pages 115–124, Barcelona, Spain, 8-12 September 2008, IEEE Computer Society.

- Anthony Finkelstein, Mark Harman, S. Afshin Mansouri, Jian Ren and Yuanyuan Zhang. A Search Based Approach to Fairness Analysis in Requirements Assignments to Aid Negotiation, Mediation & Decision Making *Requirements Engineering Journal*, 14(4):231–245, December, 2009.

Additionally, while undertaking this programme of research, the author also published and submitted the following papers, which are related to the thesis, but from which no material is presented herein:

Published

- Juan J. Durillo, Yuanyuan Zhang, Enrique Alba and A. J. Nebro. A Study of the Multi-Objective Next Release Problem 1^{st} *International Symposium on Search Based Software Engineering (SSBSE '09)*, pages 49–58, Windsor, UK, 13-15 May 2009, IEEE Computer Society.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Requirements Engineering (RE) is an important branch of Systems Engineering (SE) which addresses a broad range of problems. It includes the identification of the objectives to be fulfilled, the acquisition and analysis of the requirements to be satisfied, the documentation of such requirements as the specifications, the validation and verification of the stakeholders' needs to be met, as well as the management and support of the requirements processes and activities by using many notations, methodologies and techniques.

One of the important tasks is requirements selection and optimisation. The goal is to identify optimal choices from all possible requirements and to explore trade-off decision-making to satisfy the demands of stakeholders, while at the same time making sure that there are sufficient resources to undertake the selected task.

This thesis presents a solution framework for Requirements Selection and Optimisation by using search-based techniques, in order to better support requirements engineers in their RE decision-making activities.

This chapter mainly focuses on the introduction of requirements selection and op-

timisation and search-based software engineering. The motivation, objectives and contributions of the thesis are then described. Finally, this chapter presents an overview of the thesis structure.

1.1 Search-based Software Engineering

Search-based requirements selection and optimisation can be viewed as an application area of Search-based Software Engineering (SBSE), a term which was coined by Harman and Jones in 2001 [78]. SBSE uses metaheuristic optimisation techniques such as Genetic Algorithms (GA) [86], Simulated Annealing (SA) [105] and Hill Climbing (HC) [95] that explore and solve complex, multi-objective, highly constrained problems in Software Engineering [76].

There are three aspects for formulating software engineering problems as search-based optimisation problems:

1. **Problem Representation**

   Choosing a proper problem representation is essential for reconstructing the specific problem tackled into a search-based optimisation problem. Encoding an individual can use discrete values, such as binary, alphabet, integer or real-value for discrete binary or non-binary problems [82]; An individual also can be represented as a permutation for ordering and sequencing problems [106]; Tree-based representation is typically used to express the functions and computer programs [172].

2. **Fitness Function Definition**

   At the heart of SBSE is the fitness function which guides the search to capture the properties and insights of problems. The evaluation of fitness values

makes one solution preferable to another. In software engineering applications, fitness functions can be thought of as metrics [77]. These metrics translate objectives such as quality objectives (usability, reliability), organisational objectives (scalability), and environmental objectives (security, privacy) into some measurable attributes of a candidate solution.

3. **Choice of Search-based Technique**

   SBSE is usually involved with experimental results from empirical studies by applying search-based approaches. Once the nature of the problem is better understood empirically, it is important to generalise these results and to augment them with theoretical analysis of search landscape characteristics. This will support a more formal and rigorous analysis of potential algorithmic complexity, thereby motivating the choice of algorithm to apply.

1.2 Search-based Requirements Selection and Optimisation

Analysis of software requirements is important because mistakes and misunderstandings as this early stage in the software development lifecycle can be extremely costly. In the field of RE, once an initial set of requirements has been gathered by requirements elicitation, there is a business-level analysis problem: decisions have to be made to identify optimal choices and trade-offs for decision makers. For example, one important goal is to select near optimal subsets from all possible requirements to satisfy the demands of stakeholders, at the same time making sure that there are sufficient resources to undertake the selected tasks.

To illustrate, Figure 1.1 demonstrates a possible spread of equally optimal require-

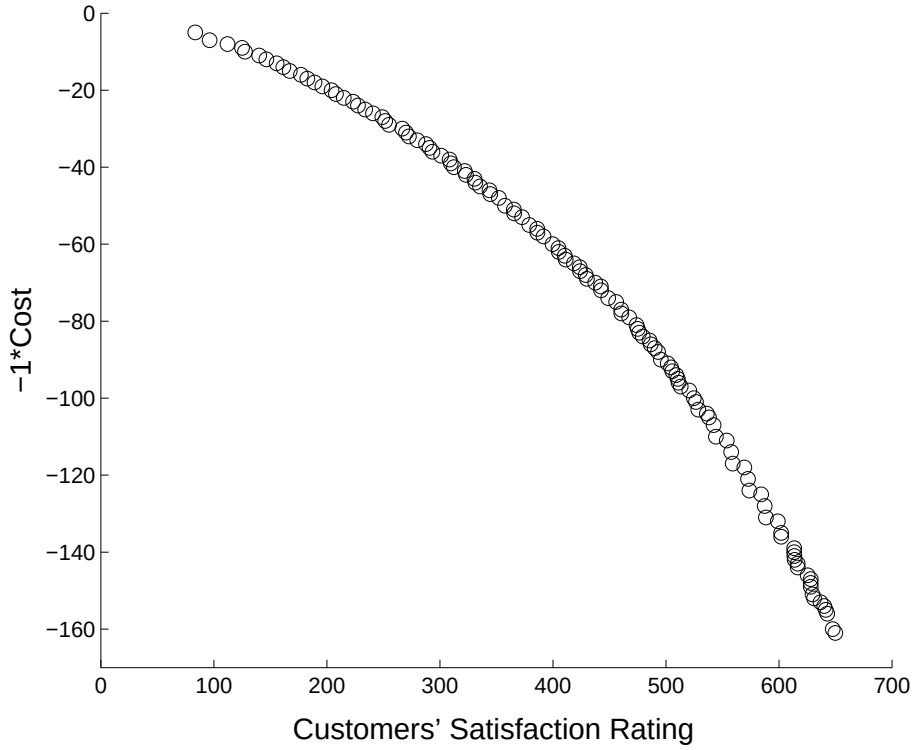


Figure 1.1: Fictitious Data: 15 stakeholders; 40 requirements. Each circle represents an equally optimal candidate solution that balances the objective of minimising supplier cost against the objective of maximising stakeholder satisfaction.

ments optimisation results. Two competing objectives are considered: *i.* cost to the provider and *ii.* estimated satisfaction rating achieved by a solution. Each circle in the figure represents an equally optimal solution. That is, each circle denotes a solution for which no better solution (subset of requirements) can be found that offers better stakeholder satisfaction without increasing cost. The set of possible solutions forms what is known as a Pareto front. Pareto fronts show a solution space of candidate solutions, from which the decision maker can select. As will be seen later, Pareto fronts also yield insights into the structure of the problem.

This requirement selection problem is one example of the way in which requirements decisions can be formulated as optimisation problems. Other examples include reordering requirements to achieve earliest satisfaction, balancing each stakeholder's

needs against the others and balancing tensions between system and user requirements.

Such problems with their large space of possible solution choices are inherently complex optimisation problems that seek to balance many competing and conflicting concerns, so it would be natural to seek algorithms for decision support. Sadly is often infeasible to apply precise analytic algorithms, because the problems are typically NP hard.

1.3 Motivation of the Thesis

The overall aim of the thesis is to propose a Search-based Requirements Selection and Optimisation framework in order to widen and explore the scope of the release planning as a problem for SBSE.

As discussed in Section 1.2, we argue that SBSE techniques can be applied to optimisation problems during the requirements analysis phase. Search-based techniques offer significant advantages: they can be used to seek robust, scalable solutions, to perform sensitivity analysis, to yield insight, to provide requirements prioritisation and fairness analysis, finally, to give the decision maker feedback and an explanation of the solutions.

RE problems are typically 'messy' problems in which the available information is often incomplete, sometimes vague and almost always subject to a high degree of change (including unforeseen change). Requirements change frequently, and small changes in the initial stages often lead to large changes to the solutions, affecting the solution complexity and making the results of these initial stages potentially fragile.

An important contribution of SBSE techniques is the way in which they can take

changing factors and constraints into account in solution construction. They can, for example, provide near optimal solutions in the search space which remain near-optimal under change, rather than seeking optimal but fragile solutions [81]. This better matches the reality of most software projects, in which robustness under change is often as valuable as any other objective.

Moreover, in Requirements Optimisation, problems arise not merely because of the number of requirements, stakeholders and other participating factors, but also because of complexity arising from constraints and dependencies. Currently, the Requirements Optimisation process, where it is practiced at all, is a highly labour-intensive activity. Search-based techniques have the potential to handle large scale problems because they are naturally parallelisable [37, 178].

It may be helpful to explore the extent to which the obtainable solutions can be said to be fair. Of course, fairness can come in different forms: should we spend the same amount on each stakeholder or give each the same number or value of fulfilled requirements? Each notion of fairness can also be treated as a separate objective, for which a Pareto optimal search seeks non-dominated solutions [63, 64]. In this way it becomes possible to automatically investigate the extent to which several notions of fairness can simultaneously be satisfied.

Current decision making techniques such as the Analytic Hierarchy Process (AHP) [165] or the Weighted Score Method (WSM)/Weighted Average Sum (WAS) evaluate the candidate solutions according to only a single fitness score and give only single so called “optimal” solution. This might easily lead the search in the wrong direction. Moreover, objectives or criteria considered may be conflicting, so it would be difficult to combine them into a single objective function. By contrast, multi-objective decision support suggests a series of optimal or near-optimal solutions without bias, from which decision makers can choose a reasonable one, best adapted

to the specific context they have in mind.

Multi-objective decision making support not only provides solutions themselves, but may also yield interesting insights. There is another dimension to this problem, that is requirements optimisation problem instances have structure. The data have implicit characteristics that the decision maker needs to expose in order to inform decision making. However, for any non-trivial problem, the number of requirements, stakeholders, their interactions and dependencies make these implicit properties far from obvious. No human could be expected to simply look and see all the implications and important features of a problem instance. However, the search problem may, for example, make it easier to see that satisfaction of one stakeholder tends to lead to dissatisfaction of another or that requirement R_i is always in generated solutions in which R_j is present.

In order to show how SBSE can yield insight in Requirements Optimisation, we applied the cost-satisfaction formulation of Zhang et al. [202] to real requirements data from Motorola. The results are shown in Figure 1.2. These results have been anonymised to prevent disclosure of sensitive information.

Compare the real world results of Figure 1.2 with the smooth Pareto front in Figure 1.1. There is an 'elbow point' in Figure 1.2's Pareto front which reveals a potential for optimisation: The stakeholders' satisfaction can be increased from 5 to approximately 58 at cost 2,000. This would be more attractive than the increase in satisfaction from 60 to 80, which would cost almost 3 times as much. The search has revealed a very attractive elbow point which occurs at approximately cost 2,000. This kind of insight is very hard to achieve without automated optimisation, like that provided by such a search-based approach.

Besides the issues described above, there are many interesting aspects that can be

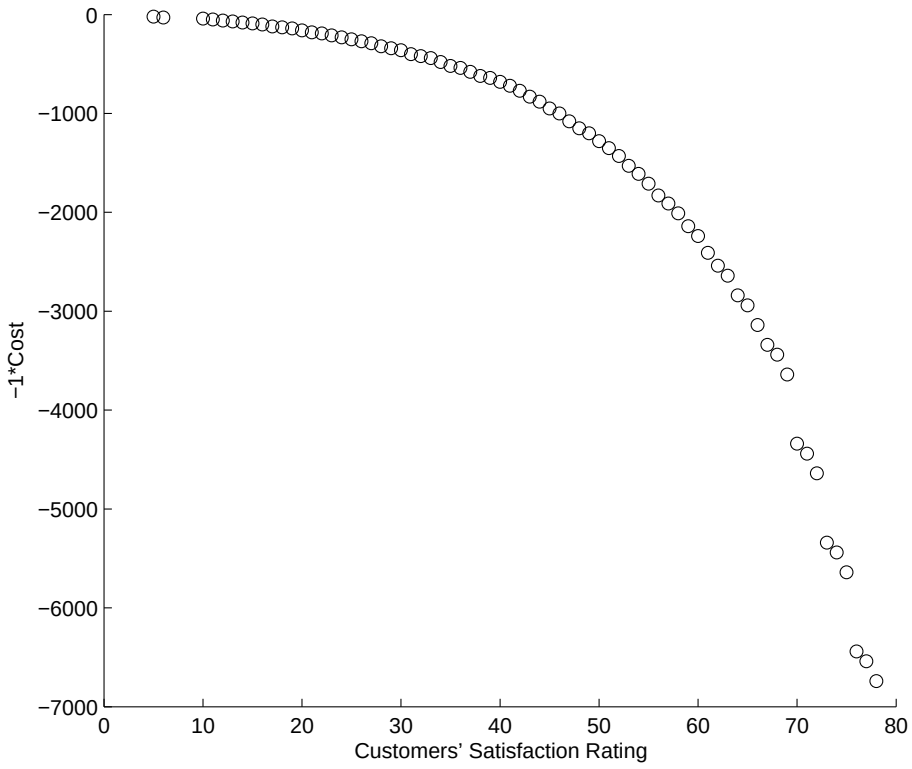


Figure 1.2: Motorola mobile device requirements: 4 stakeholders; 35 requirements. The optimisation produces a Pareto front of candidate solutions which reveals an important elbow point at cost 2,000.

addressed by the search-based requirements selection and analysis framework.

- In the requirements optimisation process, the decision maker not only selects the optimal or near optimal subset of requirements, but also the priority ordering of requirements. This method offers the potential for risk reduction. That is, circumstances vary and resources may become insufficient to fulfil all requirements. Prioritisation ensures that the most important requirements will be released first so that the maximum attainable benefits of the new system are gained earliest.

- In SBSE, human effort is partly replaced by metaheuristic search. Neverthe-

less, the numerical data upon which the automated part of the process depends come from expert domain knowledge. In the case of requirements engineering, the decision maker is forced to rely on estimates of these crucial inputs to the requirements optimisation process. Sensitivity analysis [79] helps the developer build confidence in the model by studying the uncertainties that are often associated with parameters in models. It aims to identify how 'sensitive' the model is to change. This allows the decision maker to pay additional attention to estimates for which the model is particularly sensitive.

- An additional problem arises when solutions are found: how do developers explain the solution to the stakeholder? Of course, the stakeholder expects to get the highest interest from the solution and they are likely to want to know, not merely the results, but also why a certain set of features were chosen or why some excluded requirements were those for which they had a particular care. Feedback to the stakeholder should form a part of the solution obtained by the optimisation process. This will maximise each stakeholder's satisfaction and make explicit their participation in the optimisation process. In some cases involving politically sensitive choices, solution justification and explanation may be as important as the solution itself.

1.4 Objectives of the Thesis

The objectives of the search-based requirements selection and optimisation are:

1. To investigate the feasibility and applicability of search-based requirements selection and optimisation for release planning problem.

2. To propose a search-based requirements selection and optimisation framework with four formulations for requirements optimisation.

3. To justify the flexibility of the framework for different applications via empirical studies.

4. To adapt and compare the performance of different search-based techniques for the different problems.

1.5 Contributions of the Thesis

The contributions of the thesis are as follows:

1. A Search-based Requirements Selection and Optimisation Framework is introduced. This framework formulates requirements analysis as a set of optimisation problems and allows search-based techniques to be used to explore the trade-offs and to provide the best solutions for the decision maker.

2. A "Pareto optimal" approach is provided that allows the decision maker to specify their preferences in different contexts. In addition, it not only provides solutions themselves, but also may yield interesting insights into the nature of the problem.

3. A formulation is proposed that treats *value* for stakeholder and *cost* of implementation as the two separate objectives by applying multi-objective search techniques to requirements analysis and optimisation areas for the first time.

4. A Today/Future Importance Analysis (T/FIA) formulation is presented based on a three-objective formulation aiming to select optimal subsets of requirements both for today and future scenarios.

5. A process to handle Requirements Interaction Management (RIM) is presented. The process supports the five most common types of requirement dependencies.

6. A formulation is proposed that treats each stakeholder's set of requirements as a separate objective to calculate and investigate the extent of satisfaction for each stakeholder. This single-objective-per-stakeholder formulation is the first to be introduced that treats each stakeholder in this way.

7. The thesis caters for the tensioning of the relationship between the stakeholders and resources available as well as the natural tensioning among the stakeholders themselves. These scenarios let the decision maker explore and optimise the trade-offs allowed by the instantiation of the problem with which they are faced.

8. Simple methods to visualise the discovered solutions in the multi-dimensional solution space are presented. Though these visualisations are not novel in themselves, this thesis is the first to use them for visualisation of requirement optimisation spaces.

1.6 Organisation of the Thesis

The rest of the thesis is organised as follows:

- Chapter 2 **Literature Review** briefly surveys the state of the art of RE process in general, the requirements selection and prioritisation process and search-based requirements analysis in detail. In addition, the search-based techniques adopted in the research work are also described.

- Chapter 3 **The Framework for Search-based Requirements Selection and Optimisation** provides the Search-based Requirements Selection and

Optimisation framework including the explicit context, the problem search space, representation, solving process and performance metrics for search-based techniques.

- Chapter 4 **Value/Cost Trade-off in Requirements Selection** presents the Multi-Objective Next Release Problem (MONRP), treating the 'cost' constraint as an objective and combining it with the 'value' objective. Four empirical studies are provided, that address the scale problem, 'search vs greedy' problem, boundary problem and handling requirements change, aiming to provide an approach to the provision of automated requirements selection process.

- Chapter 5 **Requirements Interaction Management** focuses on five types of requirements dependencies: And, Or, Precedence, Value-related and Cost-related. A set of empirical studies were carried out to investigate the likely impact of requirements dependencies on the automated requirements selection process. Also the performance of the search techniques is studied as the problems scale up in size.

- Chapter 6 **Multi-Stakeholder Requirements Analysis and Optimisation** provides novel approaches to problems in the multi-stakeholder analysis. One aspect of this work is the automation of the analysis of multi-stakeholder satisfaction trade-offs to explore internal tensioning among the stakeholders. The other is fairness analysis in requirements assignments to understand "what is a reasonable way to measure fairness?" and "To what extent can a solution be shown (to the stakeholders) to be a fair allocation of requirements".

- Chapter 7 **Conclusions and Future Work** concludes the thesis with a summary, the tools used in the thesis, discussion of threats to validity and possible future research directions.

Chapter 2

Literature Review

2.1 Introduction

This chapter presents an overview of Requirements Engineering (RE). First, a brief introduction and notions of RE are provided. Second, the RE process and its general activities are described. In particular, requirements analysis and optimisation related activities are discussed in detail. Next, the survey focuses on the work that applies search-based methods to this area. Finally, search-based optimisation techniques are introduced.

2.1.1 Context and Definition

2.1.1.1 Requirements

A vital notion in RE is the requirement. Requirements work is to answer the question: "What do you want to achieve in a product or service?", which is easy to say, but not necessarily so easy to do.

The importance of requirements can be expressed in a RE proverb:

> If you don't know where you're going, you're unlikely to end up there.
>
> – Forrest Gump

The "where you're going" means to find and understand what people need; the purpose and constraints of the system. The term, *requirement*, is defined in different ways in terms of emphasising different aspects. For example, the IEEE Standard 610 (1990) defined requirements in the strictly formal manner:

1. A condition or capacity needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
3. A documented representation of a condition or capability as in 1 or 2.

Kotonya and Sommerville [112] defined a requirement to be "a description of how the system should behave, application domain information, constraints on the system's operation, or specifications of a system property or attribute." In addition, Alexander and Beus-Dukic [9] provided more flexible, extensible definition: a network of related nine "requirements elements" and five "discovery contexts", including goals, rationale and measurements, combining both aspects to suggest approaches for different types of projects.

As described above, various *requirements* definitions provide the same information, that is "where you're going", which is prerequisite and foundation for success in the software development process.

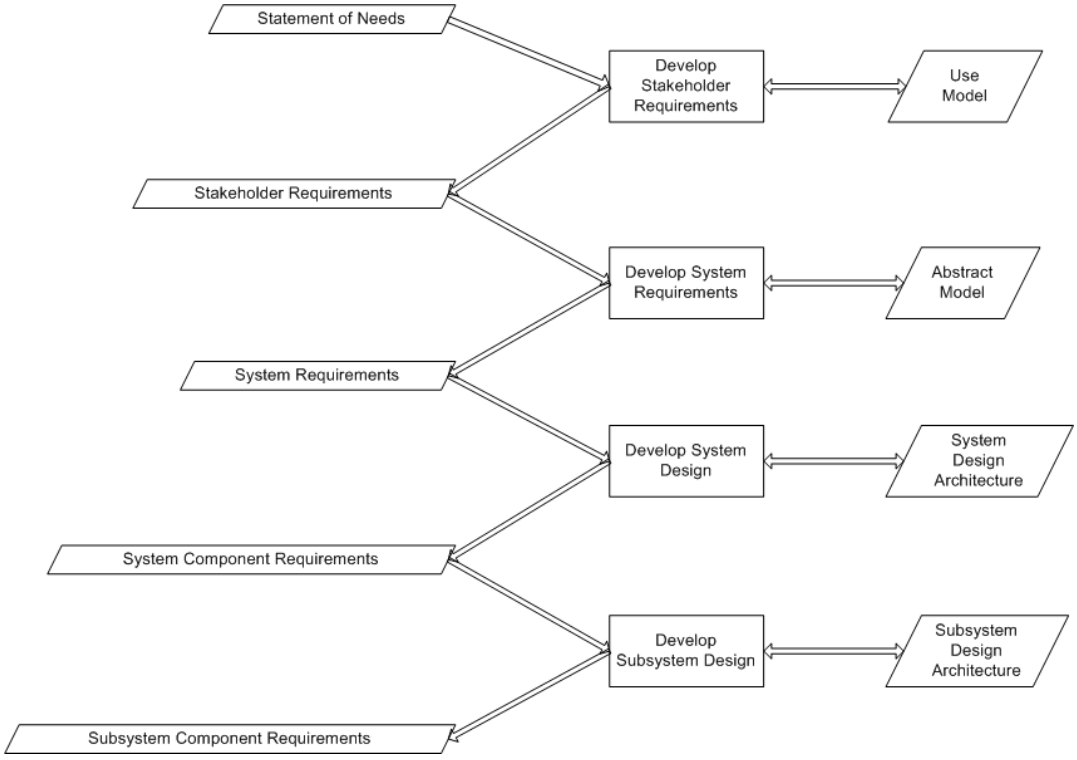


Figure 2.1: Requirements Levels in the System Development Process [89]

Requirements can be categorised in numerous ways from different perspectives. For example, according to previous work [9, 89], there are different types of requirements at each of the levels in the system development process, illustrated in Figure 2.1 [89]. Initially, a product or service may be expressed and collected as a list of *goals* from the point of view of stakeholders; the statements of needs are then transformed into the *stakeholder requirements*, also known as *business requirements* to better "define the problem"; in order to "provide the solution", next, taking the nature of systems into account, *system requirements* are produced to determine what characteristics the system must have and to abstractly describe what the system has to do; then *system component requirements* can be provided based on the knowledge of the given design architecture; if the system is large, it can be broken down into several subsystems. Each of them is specified its own *subsystem component requirements* in the similar way.

Requirements also can be divided into two major categories: mandatory and optional requirements based on their abilities to satisfy the stakeholders. Another common way to classify requirements is functional requirements and non-functional requirements (NFRs). Functional requirements define the functionality and the behaviour of a software system; NFRs provide both quality guarantee (such as performance, security, usability, scalability, robustness and reliability) and constraints (limits on cost, size, regulations). NFRs are also called softgoals [32, 112] or quality requirements in literature [93, 177].

In practice, there is no clear distinction among the requirements classifications. One requirement can possible belong to several categories in terms of properties. Such as, a requirement is considered as a functional requirement at the same time it may also be a NFR, even it contains several non-functional aspects. Requirements classification is a useful way to investigate completeness and reduce the missing requirements through the different points of view.

2.1.1.2 Requirements Engineering

By now we highlight the concept of RE to better understand the problem domain and to clearly define what the system should provide:

> Requirements Engineering (RE) is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the contexts in which it will be used. Hence, RE acts as the bridge between the real-world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies [52].

A software product or service is designed for one or more purposes which are found in human activities. As indicated in the definition [52], RE links people with the system in order to identify the problem (e.g. stakeholders, goals, boundaries, context, interfaces, scenarios, constraints etc.).

RE is not just a single and first phase in the systems engineering lifecycle, which is carried out at each development process level [61]. Systems engineering "is an engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder's needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire life cycle" – International Council on Systems Engineering (INCOSE) [92]. By contrast, software engineering only focuses on the system of the software. Systems engineering address the whole system (hardware, software, people, rules and environments).

2.1.2 Soft and Hard

Requirements engineering contains both "soft" and "hard" systems work, as illustrated in Figure 2.2 [9].

The "soft" process in RE concentrates more on social, psychological and human factors rather than on product of design. In the early requirements phase, available information is incomplete or contradictory and the environment is uncertain or always under change. Different stakeholders' perspectives also need to be included in order to capture the requirements and define the scope. These kind of complex problems are often considered as "wicked problems" [149], characterised as follows. Wicked problems:

- have no unified formulations of the problems;

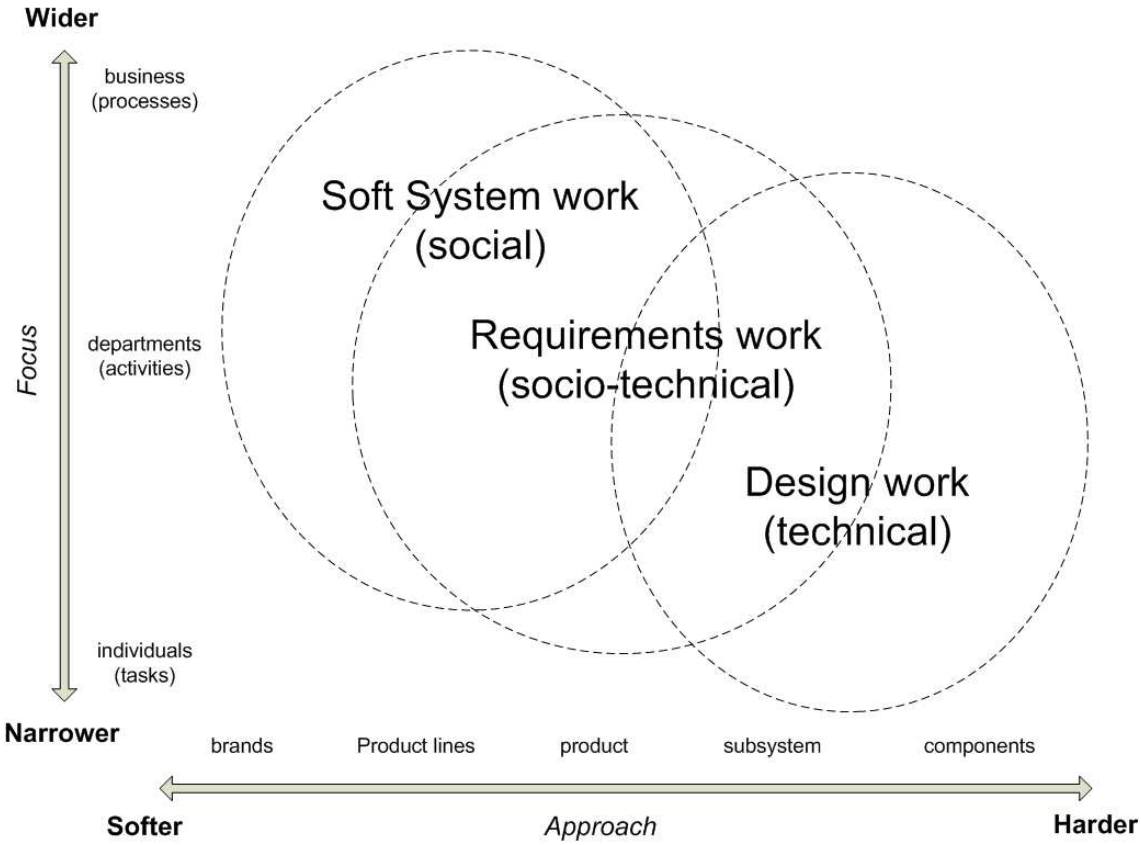


Figure 2.2: "Soft" and "Hard" process in Requirements Engineering [9]

- have no clear stopping rules;
- have no right or wrong solution, only better, worse or good enough solution to choose from;
- involve people who cannot necessarily agree on the problem to be addressed; etc.

There are a number of "soft" approaches, such as, Soft Systems Methodology (SSM) [26, 27, 146] which enable debate among conflicting concerned stakeholders and put several human activities into seven philosophical stages to arrive at reasonable compromise (definition of problem).

"Hard" aspects focus exclusively on technical design work which include modelling,

algorithms, decomposition, requirements analysis and trade-offs to provide abstract and specific design (solution of problem).

This thesis focuses on technical based "hard-oriented" requirements selection and analysis rather than social based soft systems work.

2.2 Overview of Requirements Engineering Process

In this section, we summarise the state-of-the-art in RE process which is described as a series of activities including Elicitation, Modelling & Analysis, Specification, Validation & Verification and Evolution & Management.

- ***Elicitation.*** Collecting and gathering the information of the systems, extracting and eliciting the needs of stakeholders.
- ***Modelling & Analysis.*** Rendering the more complete and consistent requirements data through creating and analysing models of requirements.
- ***Specification.*** Transferring the requirements to the precise documentation of the expected behaviour of a system.
- ***Validation & Verification.*** Checking and inspecting the consistency, completeness, security, and quality in the set of requirements.
- ***Evolution & Management.*** Managing and evaluating the stakeholder requirements change and systems operate change during the software life cycle.

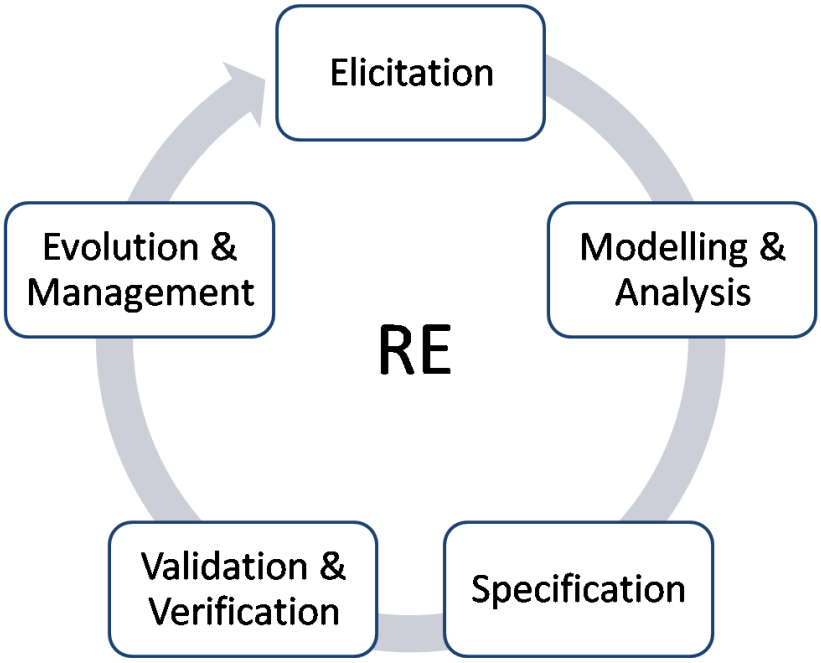


Figure 2.3: The iterative RE process with its phases

In the RE community, there are some variations in the definitions of RE phases [29, 135, 143, 171, 200]. In fact, these slightly different labels are attached to phases which comprise the similar RE activities.

In terms of relationship among these phases in RE, Hickey and Davis have pointed out that "the majority of existing models of the requirements process show it as an ordered sequence of activities. In reality, requirements activities are not performed sequentially, but relatively in parallel" [84]. Moreover, the RE process is highly iterative, illustrated in Figure 2.3. (Not all the phases are necessary to be involved in each iteration.) Some work [59, 188] inclined to the view that the activities involved in the RE process have intertwined with each other, and there is a strong correlation among them.

2.2.1 Elicitation

Requirements elicitation, also called acquisition, as the first step in the RE process involves many knowledge-intensive activities that help identify the project stakeholders, understand the problem context and interfaces, establish the boundary and identify the scope of the system. It is all about determining the needs of stakeholders, which is a crucially important issue in the software development process.

Stakeholder Identification

Stakeholders are always involved in the system development process. Stakeholder identification is a critical preliminary step, which provides a basic understanding of the context. A well-defined stakeholder networking contains more detailed and complete information of the system. As such, the deeper the analysis of the relations between stakeholders, the better the understanding of their capacities, duties and responsibilities.

Alexander [10] constructed an 'onion model' to illustrate the relationship among stakeholders and to identify their roles from the system itself to the wider environment. In terms of characteristics, stakeholders have been defined into several generic categories: operational roles, beneficiaries, interfacing roles, surrogate roles, hybrid roles and negative roles [9]. For example, surrogate roles means the representation of one on behalf of another role. Developers who design, establish and maintain the system belong to surrogates. Customers purchase the product or service. Users are both operator and beneficiary, and thus have hybrid roles. The Volere stakeholder analysis template [5] provided a list of possible stakeholder classifications which is relatively comprehensive in the literature. The list also can be used to help discover specific system qualities and constraints from different groups of stakeholders.

Information Gathering

Collecting and gathering information to understand a specific system's context and interfaces is an essential and difficult task to undertake in requirements elicitation. The scope of the system needs to be defined early in the project. That is, determining what should be included within the scope and where the boundary of the system lies. In many cases, stakeholders may have poor or incomplete understanding of their needs or unnecessary information may be given or requirements may keep changing. As a result the scope of the system might be ill-defined.

A wide range of techniques and methods have been developed to address these problems. According to previous work [29, 84, 135, 203], there are different ways of communication for information gathering: general conversational techniques such as interview [112], brainstorming [139] and RAD/JAD workshops [20]. Cognitive and knowledge based techniques include task analysis [169], card sorting [157], laddering [38], repertory grids [179], protocol analysis [73] and requirements reuse [42]; Synthetic based techniques include scenarios analysis [126, 194], rapid prototyping [48] and storyboards [135]. Contextual techniques include observation, social analysis [190], ethnographic study [173] and contextual inquiry [19].

Goal-modelling techniques, such as Knowledge Acquisition in Automated Specification (KAOS) approach [45, 46] and i^* approach [197], are also widely used in the requirements elicitation process.

Maiden and Rugg [124] proposed an integrated framework called ACquisition of REquirements (ACRE) which offered twelve techniques used for eliciting requirements. The framework evaluated and compared the strengths and weaknesses of each technique to guide elicitation technique selection in different contexts. Hickey and Davis [84] extended this selection work and provided a unified model.

Maiden et al. [94, 122] proposed the RESCUE (Requirements Engineering with Sce-

narios for User-Centred Engineering) process, combining four modelling techniques: human activity modelling, system goal modelling, use case modelling and requirements management. In the RESCUE process, the i^* approach (supported by the REDEPEND tool [120, 131]) was used and extended to model system goals, boundaries and actor dependencies. In addition, Creativity Workshops [127] were introduced in the RESCUE process to generate and capture stakeholders' requirements for use case specification. The ART-SCENE (Analysing Requirements Trade-offs: Scenario Evaluations) [198] is a scenario-driven based environment which is applied to discover and elicit requirements in traditional workshop settings or in the workplace [125].

Communication and Negotiation

The stakeholders in a system may have different needs and goals. Nuseibeh et al. [136] considered the problems associated with multiple stakeholders with completing and conflicting view points. Boehm et al. [21] proposed the WinWin model to help stakeholders negotiation processes based on a Multi-Criteria preference analysis. Another approach to resolve stakeholder conflicts is the ViewPoint approach [53, 54] which separates the different opinions among the stakeholders and can detect conflicts automatically. van Lamsweerde et al. [189] proposed goal-oriented KAOS methodology to manage conflicts and inconsistencies in RE process. In the stakeholder analysis problem, Robinson et al. [152, 155] worked on the requirements negotiation model which provided automated support to generate requirements resolutions.

2.2.2 Modelling & Analysis

Requirements Modelling and Analysis activities and requirements elicitation activities are highly iterative [7]. Requirements models systematically clarify the requirements of a system in an accurate, succinct and clear manner. The process of creating precise models is the focal point of requirements analysis activities. Requirements Analysis is essentially a process of progressive and continuous completion and improvement of software modelling.

There are many different types and levels in the modelling categories. Each modelling level corresponds to a specific requirement level. According to Nuseibeh and Easterbrook's general categories of Requirements Engineering modelling approaches [135], the framework the modelling lists as below:

- Enterprise Modelling

  Enterprise modelling is also called goal/usage modelling, often adopted to capture the purpose of a system according to stakeholders' statements of need, which are high-level organisational objectives or goals and associated tasks and resources [197]. Enterprise modelling help to understand the organisation's structure and business rules more precisely and completely.

- Domain Modelling

  Developing domain description in large information systems requires a specific domain model. Information on the domain gathered provides an envisioned system in a detailed manner. "A domain model is an object model of the domain that incorporates both behavior and data" [67]. Much previous work on Object-Oriented analysis [33] was focused on domain modelling to specify a system.

- Behavioural Modelling

  Converting statements of need to requirements, the corresponding modelling method is also changed from goal modelling to behavioural modelling. The application of behavioural modelling is used to describe, understand and simulate the required behaviours of stakeholders in a system. It includes data processing models such as Data Flow Diagrams (DFDs) [89] and state machine models like statecharts [89].

- Data Modelling

  There is a large amount of semi-structured and unstructured data that needs to be managed in the RE process. In order to define, structure and organise the data, data modelling techniques are adopted to add constraints or limitations on data within the structures in a system. Several techniques have been developed for the design of data models, such as Entity-Relationship model [28] and RM/T [34].

2.2.3 Specification

Generally speaking when the requirements elicitation and modelling process are complete, all the information obtained from the previous phases needs to be documented. The documents usually play two roles: RE activity descriptions and requirements specification. The former contains domain and problem descriptions, the latter describes behaviours of the solution system [22].

The requirements can be written in natural language (which may be more convenient for communication among stakeholders), or specified in the Software Requirement Specification (SRS). The SRS is a comprehensive expression of the system behaviour in a formal language. It is also a collection of properties at some level of abstraction

[187] which is the most valuable documentation during the software development. The standard IEEE 830-1998 [91] describes possible structures, desirable contents, and qualities of a hybrid SRS.

Turing, Floyd, Hoare and Naur were pioneers of the field of formal specification [66, 85, 130, 182]. According to the particular specification type [187], formal specification techniques can be divided into the five types: history-based specification [113], state-based specification [144], transition-based specification [83], functional specification [69] and operational specification [116, 199]. However, formal specification is not widely adopted in practice.

2.2.4 Validation & Verification

Verification & Validation (V&V) is a vital component within the software development process. The terms validation and verification relate very closely, but they are not the same.

Validation is the more general process of ensuring that the requirements and models elicited satisfy the expectations of the stakeholders as well as resolving inconsistencies and conflicts [189]. It corresponds to the question "Are we doing the right thing?". Therefore, stakeholders need to be involved in this activity directly and closely.

The literature describes a number of methods. Perhaps the most straightforward technique is manual inspections: stakeholders check and review the requirement documents, then provide the feedback and update the system information. More formal methods can be used to examine formal requirements specification, like the quasi-classical (QC) logic approach [90]. Prototyping [180] combine with scenario [176] can simulate the behaviour of the system to be developed for validation.

Requirements Verification involves checking that the requirements conform to their specification. It attempts to answer "Are we doing the thing right?". For example, research on this area focuses on customer acceptance testing [88], consistency management [189, 62] and model checking [13, 53].

2.2.5 Evolution & Management

Requirements management and requirements change management are also very important issues of RE. REQuirements Management (REQM) is "the activity concerned with the effective control of information related to system requirements and in particular the preservation of the integrity of that information for the life of the system and with respect to changes in the system and its environment" [60]. It should be carried on throughout the whole RE process.

Requirements change management is a part of REQM that affects software development activities, including the requirements' evolution over time and across product families. The task is to identify and document traceability links among requirements and between requirements and following SE activities in both a forwards and backwards direction [74]. Requirements traceability is crucial for the success of the system. It enables detection of the conflicting requirements and reduction of missing requirements. Further, it can also track the progress of a project, assess the impact of various changes and provide complete information from requirements, through design to testing.

There are many ways to represent traceability links. The traceability matrix [147] and cross references [74] are both regarded as good practice which have been widely used in industry. In addition, a large number of requirements tools support traceability management [74]. One of the most famous tools is DOORS (Dynamic Object

Oriented Requirements System) [1].

Other tasks, for example, requirements quality assurance, impact analysis and reuse of requirements also need to be undertaken. So the management of requirements is an essential activity in the development lifecycle for ensuring overall system success.

2.3 Requirements Analysis and Optimisation

2.3.1 Context and Foundation

In this section, the research study focuses on selection and prioritisation of requirements. Discovered requirements may differ in their value to stakeholders; and they may not be practical and cost-effective in the context of the chosen design [9]. Given limited budgetary resources and timescale, the project manager needs to know which requirements are critical and should be implemented in the system as a part of the next release and which can be dropped or deferred. These activities are concerned with selecting and prioritising a subset of requirements that the software developers decide to meet in order to satisfy the demands of stakeholders, at the same time, scaling to ensure that they have sufficient resources to undertake the task.

Requirements selection and analysis activities also depend on the project context. There are various project types including in-house development, custom development, market-driven development and Commercial-Off-The-Shelf (COTS) [110]. In-house means the system is developed by an organisation internally and requirements are from the people in the organisation. Custom development is designed for a single, specific client from whom the business requirements are provided. It is usually a contract application for particular users which is similar to an in-house system.

In general, these two project types are customer-specific developments, also called bespoke system development [8].

There has been much recent interest in market-driven development [102, 138, 148], compared to development of customer-specific systems. Market-driven RE is different from the latter in several ways: In a market-driven environment, the developing organisations are the primary stakeholders; requirements are acquired from the marketing activities or by the developers and specialists. Furthermore, because software is not for a specific customer, but rather for the mass market, the pressure on short time-to-market and risks are much higher than for bespoke systems.

2.3.2 Selection of COTS Component

Although not dealt with in the thesis, COTS-based software component selection problems and related processes are relevant to requirements selection and prioritisation. COTS selection activities are involved in requirements, design, implementation and maintenance phases of a software lifecycle. In the requirements phase, certain components are selected for integration in order to meet certain functional and non-functional requirements. The employment of COTS products for the market offers significant reduction in system development time, managing cost and maintenance. Typically, 'COTS' is a term that refers to software components or packages that are ready-made and are available for reuse in the new contexts [110]. There is much work discussing the motivations, advantages and challenges of COTS [65, 110, 111, 114, 158].

Selecting appropriate COTS to best meet the software requirements is the vital process in the success of system development. The *progressive filtering* evaluation strategy based COTS selection process was described by several authors [11, 110,

114, 118, 119, 128, 133]. What follows is the author's paraphrase of this process, as found in the citation.

- **Preparation**

  Review functional and non-functional requirements;
  Search and identify potential COTS component candidates on a large scale;
  Define evaluation criteria according to requirements and constraints.

- **Filter**

  Apply initial COTS filter based on the mandatory requirements;
  Further identify COTS component candidates in the smaller scale.

- **Evaluation and Selection**

  Evaluate COTS component candidates with the predefined criteria;
  Select optimal ones with good fitness in a context by using decision making techniques;
  Integrate the selected COTS components to the architecture.

- **Supporting**

  Document and assess results data, review experiences and planning for future selection process.

Besides *progressive filtering*, *puzzle assembly* and *keystone identification* COTS evaluation strategies also can be used in the process. The details refer to the work [137].

In terms of COTS selection approaches, there are more than twenty types of approach provided in the literature [114, 128, 158]. We enumerate some of those implementations to briefly outline the general situation. The OTSO (Off-The-Shelf Option) [110] was one of the first methods proposed in the literature in 1995. The

method defined hierarchical evaluation criteria, including functional requirements, quality characteristics, strategic concerns and compatibility, to estimate *cost* and *value* of COTS alternatives. In 1997, the PRISM (Portable, Reusable, Integrated, Software Modules) [118] approach provided a five-phase COTS selection process: product identification, screening, stand-alone test, integration test and final analysis. After that, Ncube and Maiden presented the PORE (Procurement-Oriented Requirements Engineering) [132, 133] approach for component-based systems engineering development in 1999. It proposed an iterative process between requirements acquisition and product selection by using different guidelines for each. PORE also integrated a number of techniques to help the decision maker with the selection of COTS activities. The CEP (Comparative Evaluation Process) [142] offered a process which is made up of five top-level activities in 2002. It suggested the credibility factor (CF) for criteria estimation and then a weighted-averages-based decision model was applied to evaluate COTS alternatives. The SCARLET (Selecting Components Against Requirements) Process Advisor [121] was proposed in 2003. It allowed to closely integrate different processes and techniques to support software component selection which was one of the essential parts in BANKSEC's component-based system development process [123]. In 2004, Chung and Cooper defined CARE/SA (COTS-Aware Requirements Engineering and Software Architecting) framework [30, 31] for matching, ranking and selecting COTS products in an iterative manner. Both functional, non-functional requirements and their architecture were considered in the COTS products. My work in this thesis might be applied to COTS, but this is not the specific emphasis of this work.

2.3.3 Selection and Prioritisation for Release Planning

As mentioned earlier, requirements can be roughly categorised into two groups according to their ability to satisfy the stakeholders' needs and wishes. These two group are: mandatory and optional requirements. Mandatory requirements are non-negotiable and provide the core functionality that must be fulfilled regardless of resources and time; all the other requirements are optional requirements that might be met to some extent when there are leftover resources in the project, otherwise some of them may be temporarily suspended or even be discarded. Therefore, a precise selection of which optional requirements the software provider decides to meet, and not to meet, is crucial to perform at an early stage of the project [98].

In market-driven software development, if all of the requirements cannot be fully satisfied under existing constraints, how and when do the stakeholders deal with the remainder of the requirements? In order to achieve a successful project outcome, two of the most important and critical activities are requirements selection and prioritisation for evolving the system based on incremental Release Planning (RP) [12, 14, 75, 159, 163, 164, 184], which can help the project manager resolve the conflicts and make sensible decisions.

RP is very complex as well as important for the success of a software product [23, 24]. RP for incremental software development selects and prioritises a collection of requirements for realisation in a certain release. It allows stakeholders to receive an increment that is a complete system as soon as possible and ensures that the most important requirements are delivered first. In addition, new requirements or changes to requirements can be introduced during each increment according to stakeholders' feedback.

2.3.3.1 Process

First, the stakeholders' intent and goals need to be correctly understood. Our task is trying to compare goals to find out which ones are more important, even if the stakeholders always want to satisfy "all of them". According to the initial selection and priorities based on requirements' value to stakeholders as well as the accept/reject threshold, the ones that are not worth working on are excluded from the set of candidates.

Then, taking resource constraints (budget, time, design issues etc.) into account, we need to translate the requirements into measurable criteria (type, value, cost, dependencies, risk etc.); identify and discover the trade-offs inherent in meeting conflicting objectives; evaluate and provide best solutions to arrive at a set of requirements to be implemented.

Last but not least, according to knowledge of the design, the decision maker selects the best solutions in different circumstances and according to their priorities. This is an iterative optimisation process that involves selecting and prioritising requirements in the evolving systems.

2.3.3.2 Approaches

From the industry point of view, many companies feel that they cannot control the release planning challenge, because many of them may only rely on the product or project manager to investigate the implicit characteristics of requirements and study the competing interests of the stakeholders.

In the literature, Yeh and Ng [196] argued that a target system benefited directly from ranking and prioritising requirements in 1990. Karlsson [97] adopted two types

of techniques for selecting and prioritising software requirements: Quality Function Deployment (QFD) [175] and Analytical Hierarchy Process (AHP) [165] in 1996. In QFD the stakeholders prioritise the requirements on an ordinal scale (using a numerical assignment). The drawback of this is that there is no clear and obvious definition of the distinction among the absolute number assigned to each requirement. Moreover, relationships between requirements are not supported by QFD. The most serious drawback is that QFD cannot manage functional requirements, because there is no degree of fulfilment for functional requirements.

In 1997, Karlsson and Ryan [100] proposed a cost-value approach, using AHP, applying it to compare all the candidate requirements in order to determine which of the two is of higher priority and to what extent its priority is higher. Moreover, in 1998, they evaluated six different methods for selecting and prioritising requirements [101] and found that AHP is the most promising method. However, the disadvantage of using a pairwise comparison technique is the huge number of required pairwise comparisons. The method becomes laborious and inefficient as the scale of the project increases. In addition, this prioritising process has a lack of support for requirement interdependencies.

Aiming to reduce the complexity of applying AHP, in 1998, Jung [96] adopted linear programming techniques as a two-step process: firstly, maximising the sum of the requirements' values within cost budgets; secondly, determining which requirements can be fulfilled to minimise the sum of the costs within maintaining the maximum value of the first step. The process was based on single objective formulation with a cost factor as constraint. In 1999, Wiegers [192] presented a semi-quantitative approach for requirements prioritisation, which combined value, cost and risk criteria using weighting factors to evaluate requirements. This approach was limited by the ability to attach the weights for each objective. Robertson and Robertson

[150, 151] presented the Volere Requirements Specification Template [4] for assessing requirements. The template provided the simple and effective starting point for quantifying requirements, but it did not refer to the requirements selection issue.

There are some other techniques and methods for requirements prioritisation, such as the Binary Search Tree [98], the Minimal Spanning Tree [98], Hierarchical Cumulative Voting (HCV) [17, 166], the Priority Group [101], Top 10 Requirements [115], Outranking [156], the Planning Game [16], the Weighting Method [167], Multi-Attribute Utility Theory [167], Quantitative Win-Win [160], Wiegers' method [193], Planguage based impact validation [70] and Kano analysis [181].

Some of these strategies arrange the requirements in a hierarchy; some cluster the requirements into several groups by different priority levels using a verbal ranking; some rely on relative values by pairwise comparison using a numerical ranking; some use discrete values, the others use the continuous scale. One of the advantages of a numerical ranking is that it can be sorted. These priority-based approaches usually assign a rank order or level to each requirement from the 'best' to the 'worst'. In this thesis we address the related selection problem of choosing a subset of requirements for release planning.

Compared with these priority-based methods, we can provide more than one (usually many) optimal alternative solutions within a certain criterion (such as under specific project budget). As such, the requirements engineer has the opportunity to observe the impact of including or excluding certain requirements, and can use this to choose the best from the different alternatives, without affecting the quality of solutions.

2.3.4 Requirements Interdependencies

According to Carlshamre et al.,

> "The task of finding an optimal selection of requirements for the next release of a software system is difficult as requirements may depend on each other in complex ways" [25].

Some requirements might have technical, structural or functional correlations that need to be fulfilled together or separately, or one requirement might be the prerequisite of another one. The analysis and management of dependencies among requirements is called Requirements Interaction Management (RIM) which is defined as

> "the set of activities directed towards the discovery, management, and disposition of critical relationships among sets of requirements" [153].

Robinson et al. [153] gave the definition of requirements interaction:

> "Two requirements R_1 and R_2 is said to interact if (and only if) the satisfaction of one requirement affects the satisfaction of the other."

Dependence analysis is a part of the overall traceability problem for requirements engineering. Pohl [143] proposed the *traceability meta model* to establish a traceability structure, which included dependence models aiming to describe the relations between trace objects. Karlsson et al. [99] opened up the discussion on supporting requirements dependencies in the requirements selection process. Carlshamre and Regnell [24] described a two-dimensional (*scope* and *explicitness*) representation to investigate different types of dependencies. Subsequently Carlshamre et al. [25] extended their work and carried out an industrial survey of requirements interdependencies in software product release planning. A functional and value-related dependence classification scheme was proposed in detail. The survey also tried

to find the possible relationship between the dependence types and development contexts. Dahlstedt and Persson [43, 44] provided an overview of research work about comparing and validating the different requirements dependencies classification frameworks.

There was some work which suggested that proper treatment of RIM should take account of different types of requirements interactions [25, 43, 75, 99, 154]. Here we present the most common interaction types that will be used in the thesis in Chapter 5:

And Given requirement R_1 is selected, then requirement R_2 has to be chosen.

Or Requirements R_1 and R_2 are conflicting to each other, only one of R_1, R_2 can be selected (Exclusive OR).

Precedence Given requirement R_1 has to be implemented before requirement R_2.

Value-related Given requirement R_1 is selected, then this selection affects the value of requirement R_2 to the stakeholder;

Cost-related Given requirement R_1 is selected, then this selection affects the cost of implementing requirement R_2.

2.3.5 Requirements Tools

RE process involves a series of activities and a large amount of information. Challenges and problems are bound to arise as the size and complexity of the system increases, such as lack of appropriate skills and knowledge required to gather requirements, poor communication of details relations to changes, lack of suitable

requirements traces. So it is difficult to fulfil requirements management tasks in a robust, efficient manner.

Requirements tools offer automatic assistance for RE activities. These provide an underlying platform upon which to store and adjust requirements attributes in a database, manage changes and versions, track and trace the status of requirements, communicate with stakeholders, and enable distributed development with other software development tools [191].

There are some related investigations covered in the requirements tools area: Wiegers [191] briefly discussed the major characteristics of four commercial requirements tools, which showed that all the tools compared had good operability for requirements management. Finkelstein and Emmerich [61] described the potential developmental tendencies of requirements tools in the short, medium and long-term from industry's point of view, which provided a useful and practical guideline in the area. Beuche at el. [18] gathered and prioritised requirements tools, visualised them using radar maps. In terms of tool categorisation, Alenljung [7] argued that the term "RE tools" should be used, instead of "requirements management tools". The reason is that all the RE activities need support, rather than only in the management of requirements. Some organisations have also conducted comprehensive surveys of requirements tools. For example, INCOSE [92] has maintained and updated a survey of evaluations of 50 requirements management tools. There are 70 survey questions with an answer to each question being provided by tool vendors. INCOSE shares "survey responses" with the public after reviewing and correcting unobjective information. They also suggested a hierarchical requirements tool classification scheme. Volere provided a requirements tools list which covered a broad selection including more than 60 tools in total.

We briefly describe one of those requirements tools – the *IBM Rational Focal Point*

tool [2] since it is the previous work most closely related to the work set out in the remainder of the thesis. *Focal Point* is the product of the IBM Corporation after IBM's acquisition of Telelogic AB in April 2008. Through its acquisition of Telelogic, IBM supplies development and analysis tools, solutions and products for advanced systems and software development. Related products include *DOORS* [1], *Focal Point* [2] and *Tau* [3].

Karlsson [97, 98, 100] proposed the Analytic Hierarchy Process (AHP) for assigning priorities to requirements and developing strategies for selecting an optimal set of requirements for implementation. The *Focal Point* tool is based on this work. It is a fully web-based decision support tool that helps to guide the RE development process. It can identify, structure, pairwise-compare and prioritise the requirements and then visualise, discuss and focus on the final results, based on business objectives, stakeholder satisfaction and resources availability.

At the time of writing, the current stable version is Focal Point 6.3. Telelogic kindly provided an academic licence for 6 months, allowing the author to gain first hand experience with the tool. For example, the Motorola data set mentioned in Chapter 1 was analysed using the tool, to investigate its methods of supporting requirements selection. Figure 2.4 illustrates the result in a stacked bar chart format.

Cost and *Value* are two criteria considered in the requirements selection in this case. They are listed on the upper right hand 'key' to the figure. Each horizontal bar represents a single requirement. The length of a bar indicates the sum of numeric values of *Value* and *Cost*. The bars are split into two parts by the vertical line: the length of each bar on the right side represents the *Value* for a single requirement and the length of each bar on the left side represents the *Cost* for a single requirement. *Value* and *Cost* were converted to numeric data by Motorola without units of measurement in order to protect company sensitive data. However, for our

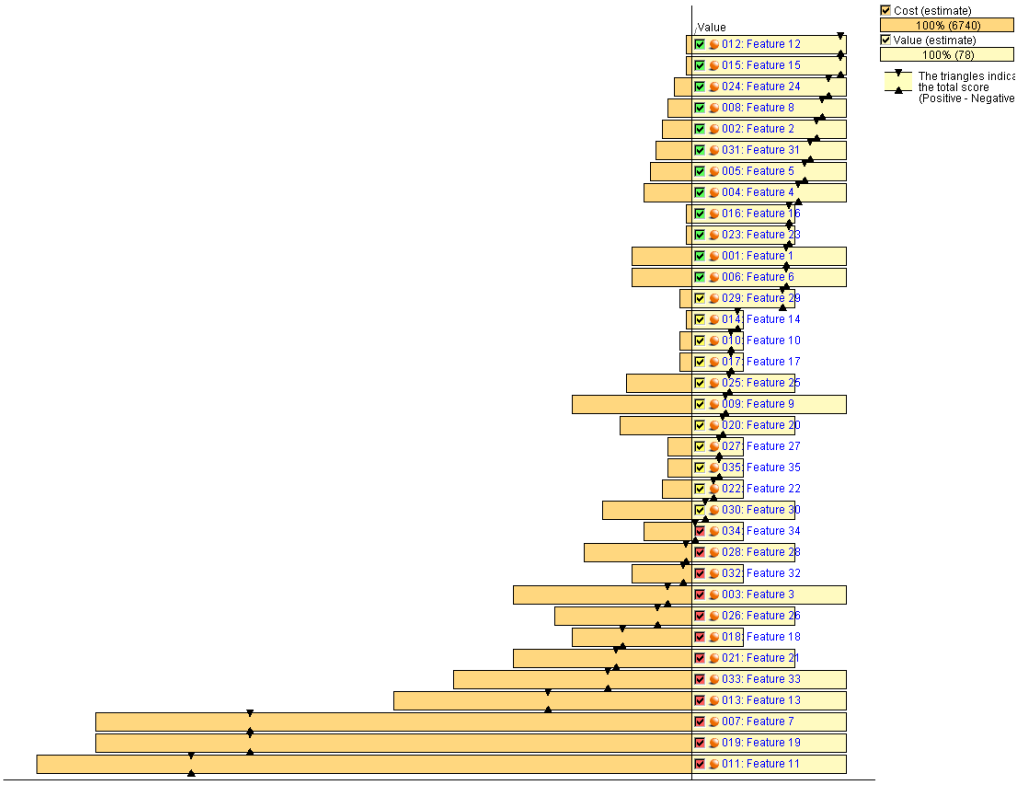


Figure 2.4: Focal Point Stacked Bar Chart

purpose, this is sufficient. There are 35 requirements in total in the data set and each is represented by a pair of *Cost/Value* bar in the figure. The triangles indicate the *Score* for each requirement.

The *Score* is calculated simply:

$$Score = Value - Cost$$

Focal Point tool evaluates each requirement according to the *Score* calculated and prioritises them from the highest *Score* to the lowest. As illustrated in the figure, from the top to the bottom, the triangles on the bars move from the leftmost to the rightmost. The bar at the top is the best and the one at the bottom is the worst considering both criteria together.

Currently, both criteria are equally important and have the same weight. The tool allows the user to change the criteria weight when one criterion is more important than the others. Weight based decision support combines multiple objectives into single objective using weighting factors. However, such a combination is difficult in many situations, for example, for conflicting objectives, it may not be possible to assign weights to each. Furthermore, the criteria might measure fundamentally different properties (so combining them would be to seek to 'compare apples and oranges'), with the consequence that weights cannot be adequately determined.

For a set of optional requirements, Focal Point provides a single prioritised requirements list from the 'best' to the 'worst'. In this thesis we address the related selection problem of choosing a subset of requirements. We extend previous work by considering Pareto optimal formulations of multiple objectives. This overcomes the 'apples and oranges' problem. It also avoids the need to make arbitrary choices of weights. Instead, our approach is used to reveal interesting potential 'sweet spots' at which there is a potentially large shift in the trade-off between objectives.

2.4 Search-based Requirements Analysis and Optimisation

Within the Search-based Software Engineering (SBSE) community, a recent trend has emerged in which search-based optimisation techniques have been used to solve requirements selection and optimisation problems. This would seem to be a natural and realistic extension of the initial work on SBSE. This section reviews that work on search-based requirements optimisation.

Bagnall et al. [14] suggested the term *Next Release Problem* (NRP) for require-

ments release planning and described various metaheuristic optimisation algorithms, including greedy algorithms, branch and bound, simulated annealing and hill climbing. The authors did not give any *value* property to each requirement. They only used an associated *cost*. The task of the work was to find a subset of stakeholders, whose requirements are to be satisfied. The objective was to maximise the cumulative measure of the stakeholder's importance to the company under resource constraints. This single-objective formulation based Next Release Problem was the first attempt on SBSE for requirements.

Feather and Menzies [58] built an iterative model to seek the near-optimal attainment of requirements. The authors proposed a Defect Detection and Prevention (DDP) process based on a real-world instance: a NASA pilot study. The DDP combined the requirements interaction model with the summarisation tool to provide and navigate the near-optimal solutions in the risk mitigation/cost trade-off space. The paper was one of the first to use Pareto optimality in SBSE for requirements, though, unlike the work in this thesis, the Pareto fronts were not produced using multi-objective optimisation techniques (as with more recent work), but were produced using the iterative application of a weighting based single objective formulation by applying simulated annealing. Also, with relevance to Pareto optimal formulations, Feather et al. [56, 57] summarised the visualisation techniques used to present requirements status, including Pareto front plotted by Simulated Annealing.

Ruhe et al. [75, 161, 162] proposed the genetic algorithm based approaches known as the EVOLVE family which aimed to maximise the benefits of delivering requirements in an incremental software release planning process. Their approaches balance the required and available resources; assessing and optimising the extent to which the ordering conflicts with stakeholder priorities. They also took requirement changes and two types of requirements interaction relationship into account and provided

candidate solutions for the next release in an iterative manner. As with previous work, this piece of work still adopted a single objective formulation, taking the resource budget as a constraint.

Moreover, Carlshamre [23] take requirements interdependencies into consideration by using Linear Programming techniques. Ruhe and Saliu [163] also presented an Integer Linear Programming (ILP) based method which combined computational intelligence and human negotiation to resolve their conflicting objectives. Van den Akker et al. [117, 183, 185, 186] further extended the technique and developed an optimisation tool based on integer linear programming, integrating the requirements selection and scheduling for the release planning to find the optimal set of requirements with the maximum revenue against budgetary constraints.

Using search-based techniques in order to choose components to include in different releases of a system was studied by Harman et al. [15, 80]. The work of AlBourae et al. [6] was focused more on the requirements change handling. That is, re-planning of the product release. A greedy replan algorithm was adopted to reduce risks and increase the number of requirements achieved in the search space under change.

In addition, Cortellessa et al. [39, 40, 41] described an optimisation framework to provide decision support for COTS and in-house components selection. The Integer Linear Programming (LINGO model solver) based optimisation models (CODER, DEER) were proposed to automatically satisfy the requirements while minimising the cost.

The aforementioned work on this problem has tended to treat the requirements selection and optimisation as a single objective problem formulation, in which the various constraints and objectives that characterize the requirements analysis problem are combined into a single objective fitness function. Single objective formulations have

the drawback that the maximisation of one concern may be achieved at the expense of the potential maximisation of another resulting in a bias guiding the search to a certain part of the solution space.

More recently, there has been work on multi-objective formulations of the problem, some of which has been conducted by the author of this thesis. Zhang et al. [202] aimed to obtain the near-optimal set of requirements that balance the stakeholder requests and the resources, which considered value and cost as two separate criteria in the Multi-Objective Next Release Problem (MONRP) formulation. They consider an integrated value function, comprising the values associated with each stakeholder using search-based techniques. Also, the scalability of the approach (in terms of for e.g. number of requirements and number of stakeholder) was discussed in the paper. The authors compared the performance of the techniques and the impact of requirement and stakeholder set size on the different scale data sets. Saliu and Ruhe [168] showed how multi-objective search-based optimisation can balance the tension between user-level and system-level requirements and track dependencies from user requirements into their impact on system components. Zhang et al. [201] summarised existing achievements and described future challenges for Search-based Requirements Optimisation in recent years, with a focus on the need for multi-objective techniques.

Finkelstein et al. [63, 64] considered the problem of fairness analysis in requirements optimisation. This was the first paper to introduce techniques for analysis of the trade-offs between different stakeholders' notions of fairness in requirements allocation, where there are multiple stakeholders with potentially conflicting requirement priorities and also possibly different views of what would constitute fair and equitable solution.

In this work, like others on multi-objective solutions, each of the objectives to be

optimised is treated as a separate goal in its own right; multiple objectives are not combined into a single (weighted) objective function. This allows the optimisation algorithm to explore the Pareto front of non-dominated solutions. Each of these non-dominated solutions denotes a possible assignment of requirements that maximises all objectives without compromising on the maximisation of the others. Using Pareto optimal search it becomes possible to explore precisely the *extent* to which it is possible to satisfy "all of the people all of the time". Of course, this is unlikely to be completely achievable. However, the algorithm attempts to produce a set of non-dominated solutions that are as close as the stakeholders' prioritisations will allow to this ideal situation.

2.5 Search-based Techniques

This section provides a brief overview of search-based optimisation techniques, such as those that will be used in the remainder of the thesis.

2.5.1 Introduction

The techniques adopted in the thesis are search-based algorithms which are the metaheuristics (also refered to as *optimisation*, *evolutionary* or *learning* techniques). These techniques explore and navigate in a search space which is the set of all possible solutions, therefore they are called search-based techniques.

The term "Heuristic" comes from *Greek* root means "to find" which is the information, strategy or procedure used to guide and help solve problems. Here it refers specifically to algorithms that are able to (hopefully rapidly) come to an acceptable and reasonable solution to a problem. There is no guarantee that these algorithms

will provide an optimal solution. "Meta" is a prefix also derived from *Greek* which is the higher level structure containing operators later tailored to problem.

"Metaheuristic" is the term used for a structured optimisation algorithm composed of different operators. It is used to iteratively solve complex problems in procedures which knowledge can be transferred between different domains.

Some well-known metaheuristics are Genetic Algorithms (GAs) [86], Genetic Programming (GP) [172], Evolution Strategies (ES) [170], Simulated Annealing (SA) [105], Hill Climbing (HC) [95], Tabu Search (TS) [72], Artificial Immune Systems (AIS) [55], Ant Colony Optimisation (ACO) [51], Particle Swarm Optimisation (PSO) [104] and Estimation of Distribution Algorithms (EDA) [129].

Some of those have a nature-inspired origin, such as GAs, ACO and PSO, while others are pure abstract artefacts, like EDA. In terms of search behaviour, search algorithms can be categorised as two types: trajectory based (SA, TS etc.) which tracks one solution and follows a path (or trajectory) to find optimal (local or global); and population based (GA, ACO, PSO etc.) which scatter the population over the search space to perform a global search. Also, all Evolutionary Computation approaches (GA, GP, ES) are typically grouped together under the heading of Evolutionary Algorithms (EAs).

2.5.2 Genetic Algorithms

This thesis concentrates on GAs. Single and multi-objective GAs are applied to requirements optimisation problem. Holland [86] first introduced the term "genetic algorithm" in his book *Adaptation in Natural and Artificial Systems* published in 1975. The first International Conference on Genetic Algorithms (ICGA 1985) was held at The University of Illinois. In the late 1980's, GAs became popular and were

widely applied to solve problems in a wide variety of fields. This popularity has continued to grow since then.

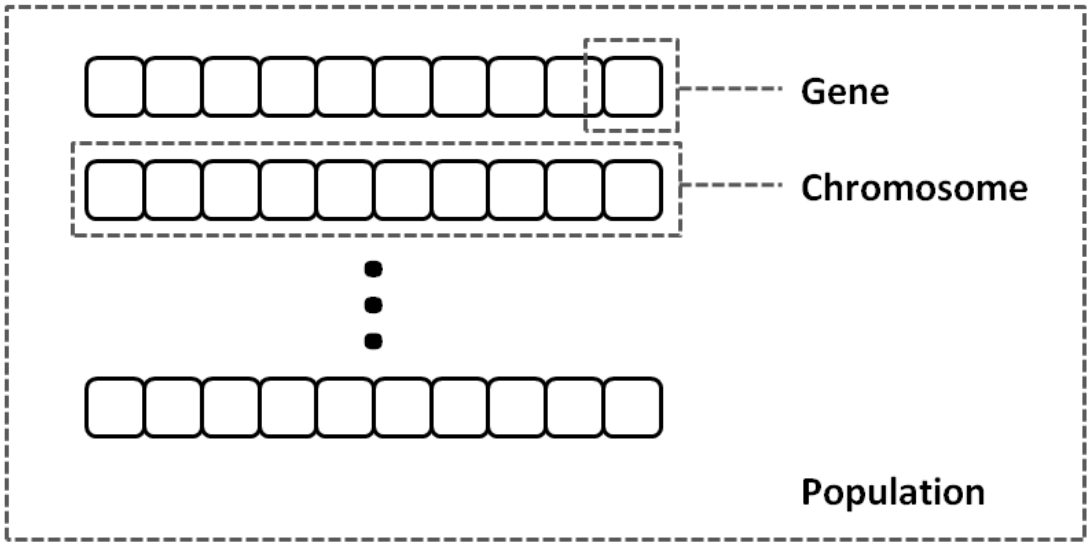


Figure 2.5: Gene, Chromosome and Population in GAs

GAs are inspired by Darwin's theory of *natural selection* [47] and simulate the process of "Survival of the fittest" in the biological evolution for solving computational optimisation problems. In GAs, a chromosome (also called genome) is often a finite length string representation of a possible solution to a problem, but other data structures also can be used. A collection of genes constitutes a chromosome in which each gene is single or subset of digits (binary, alpha or numerical encoding) at specific position in a chromosome. A population of n chromosomes is illustrated in Figure 2.5.

A basic GA works as shown in Algorithm 1. Once an initial population is randomly generated, a fitness value is evaluated and assorted to each chromosome according to the defined fitness functions. The algorithm runs through the three operators in a number of generations:

Algorithm 1: Basic Genetic Algorithm

```
Set t = 0
Initialise the population P_0
Evaluate initial population P_0
while t <= MAX_GENERATION do
    t = t + 1
    Select P_t from P_{t-1}
    Crossover P_t
    Mutate P_t
    Evaluate P_t
end
```

- Selection

  Choose chromosomes (parents) from the current generation for breeding the next generation (offsprings) based on fitness value. The better the fitness, the greater the chance of being selected. The most widely used selection schemes are roulette-wheel selection and tournament selection.

  In this thesis, the tournament selection method is adopted to choose individual chromosomes. Firstly, the tournament size k (values range from 1 to the size of the population) needs to be determined. Then, k chromosomes are chosen from the population randomly in each tournament. The one with the best fitness is selected from each tournament as parent. Conceptually, the k individuals fight a hypothetical 'tournament' to see which one wins, hence the name for this process of selection.

- Crossover

  Combine two chromosomes (parents) to generate the new chromosomes (offspring). The operator picks crossover point(s) randomly within a chromosome, then interchanges these two chromosomes at this point (Single-Point crossover), as shown in Figure 2.6, or between two (Two-Point crossover) or multiple points (Multi-Point crossover). The crossover operates according to

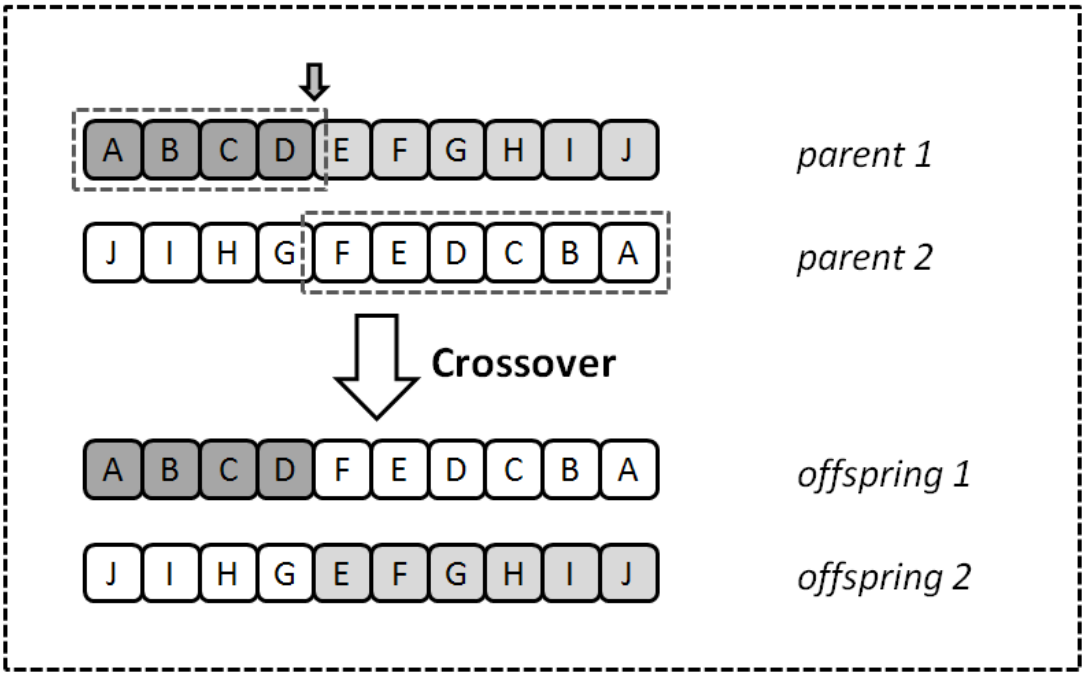


Figure 2.6: Crossover Operator in GAs

the predefined crossover probability.

- Mutation

  Change one or more gene values in a chromosome randomly in order to maintain the diversity of population and avoid local extrema (maxima or minima). As illustrated in Figure 2.7, mutation causes local modification comparing with crossover operator. Usually the mutation probability is set to a very low level. With regard to mutation scheme, Bitflip can be used for binary genes and Gaussian, Uniform or Boundary types for integer or float genes.

The emphasis in the following is to outline the multi-objective optimisation concepts and to introduce multi-objective genetic algorithms used in the thesis.

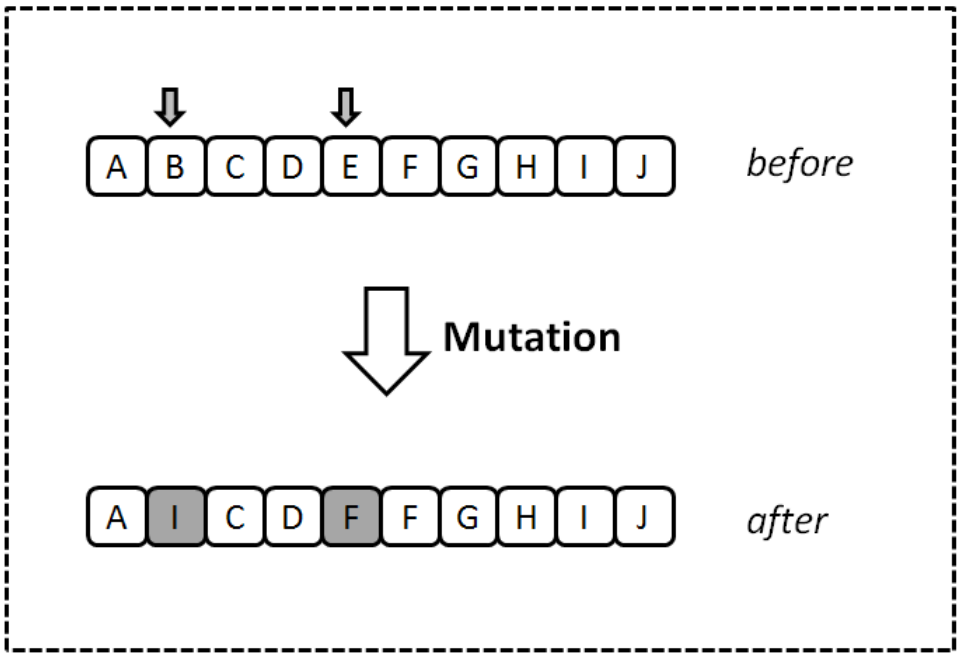


Figure 2.7: Mutation Operator in GAs

2.5.3 Pareto-Optimal Front

The multi-objective search algorithms used in the thesis are based on the concept of dominance to solve Multi-Objective Optimisation Problems (MOOPs). In the algorithms, two solutions are compared on the basis of whether one dominates the other solution or not [49]. We describe the domination concept below:

In a MOOP, each solution here is defined as a vector of decision variables $\overrightarrow{x}$. It is assumed that there are M objective functions $f_i(\overrightarrow{x})$ where $i = 1, 2, \ldots, M$. The objective functions are a mathematical description of performance criteria. Often these criteria are in conflict with each other [140].

We wish to find a set of values for the decision variables that optimises a set of objective functions. A decision vector $\overrightarrow{x}$ is said to dominate a decision vector $\overrightarrow{y}$ (also written as $\overrightarrow{x} \succ \overrightarrow{y}$) iff:

$$f_i(\overrightarrow{x}) \geq f_i(\overrightarrow{y}) \;\; \forall \; i \in \{1, 2, \ldots, M\};$$

and

$$\exists \; i \in \{1, 2, \ldots, M\} \mid f_i(\overrightarrow{x}) > f_i(\overrightarrow{y}).$$

All decision vectors that are not dominated by any other decision vector are called *non-dominated* or Pareto-Optimal and constitute the Pareto-Optimal Front. These are solutions for which no objective can be improved without detracting from at least one other objective.

The goal of the MOOP is to find an ideal vector of decision variables $\overrightarrow{x}$, which optimises a vector of M objective functions $f_i(\overrightarrow{x})$ subject to inequality constraints $g_j(\overrightarrow{x}) \geq 0$ and equality constraints $h_k(\overrightarrow{x}) = 0$ where $j = 1, 2, \ldots, J$ and $k = 1, 2, \ldots, K$.

Without loss of generality, a MOOP can be defined as follows:

$$\text{Maximise } \{f_1(\overrightarrow{x}), f_2(\overrightarrow{x}), \ldots, f_M(\overrightarrow{x})\}$$

subject to:

$$g_j(\overrightarrow{x}) \geq 0; \;\; j = 1, 2, \ldots, J$$

and

$$h_k(\overrightarrow{x}) = 0; \;\; k = 1, 2, \ldots, K.$$

where $\overrightarrow{x}$ is vector of decision variables; $f_i(\overrightarrow{x})$ is the i^{th} objective function; and $g(\overrightarrow{x})$ and $h(\overrightarrow{x})$ are constraint vectors.

These objective functions constitute a multi-dimensional space which is called the objective space, Z. For each solution $\overrightarrow{x}$ in the decision variable space, there exists

a point z^* in the objective space:

$$\exists\ z^* \in Z$$

$$z^* = \overrightarrow{f}(\overrightarrow{x}) = (f_1, f_2, \ldots, f_M)^T$$

Using Pareto optimality, it is not possible to measure 'how much' better one solution is than another, merely to determine whether one solution is better than another. When searching for solutions to a problem using Pareto optimality, the search yields a set of solutions that are non-dominated. That is, each of the non-dominated set is no worse than any of the others in the set, but also cannot be said to be better. Any set of non-dominated solutions forms a Pareto front.

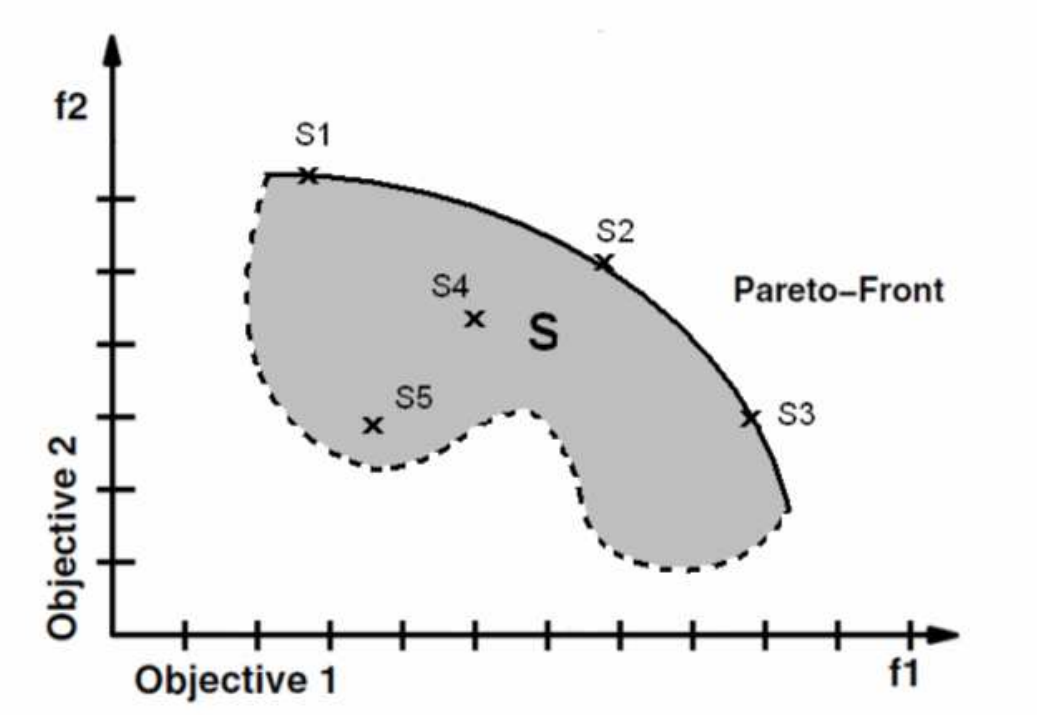


Figure 2.8: Pareto-Optimal Front

Consider Figure 2.8, which depicts the computation of Pareto optimality for two imaginary fitness functions (Objective 1 and Objective 2). The longer the search

algorithm is run the better the approximation becomes to the real Pareto front. In the figure, points $S1$, $S2$ and $S3$ lie on the Pareto front, while $S4$ and $S5$ are dominated.

2.5.4 Multi-Objective Optimisation Algorithms

In the research, we have established and applied several multi-objective optimisation formulations to requirements anlysis and optimisation problems. There are various general approaches to solve MOOPs. Among the most widely adopted techniques are sequential optimisation, the ϵ-constraint method, the weighting method, goal programming, goal attainment, the distance based method and the direction based method. For a comprehensive study of these approaches, the reader is referred to Szidarovsky et al. [178] and Collette and Siarry [37].

Among metaheuristics, Evolutionary Algorithms (EAs) are particularly desirable to solve MOOPs, primarily because of their population-based nature. This enables them to capture the dominance relations in the population as a vehicle to guide the search towards the Pareto-optimal front. They deal simultaneously with a set of possible solutions (the so-called population) which unlike traditional mathematical programming techniques, can find good approximations to the Pareto-optimal set in a single run. Additionally, EAs are less susceptible to the shape or continuity of the Pareto-optimal front [35], whereas these two issues pose a barrier to classical mathematical programming techniques.

EAs usually contain several parameters that need to be 'tuned' for each particular application, which is, in many cases, highly time consuming. In addition, since the EAs are stochastic optimisers, different runs tend to produce different results. Therefore, multiple runs of the same algorithm on a given problem are needed to

statistically describe their performance on that problem. For a more detailed discussion of applications of EAs to multi-objective optimisation see Coello et al. [36] and Deb [49].

2.5.5 Techniques Used in the Thesis

This section describes the search algorithms used in the thesis. The research presents results mainly from two search techniques namely: the NSGA-II algorithm and the Two-Archive algorithm. These techniques are also compared with the Pareto GA, the Single-Objective GA and Random Search to verify viability of applying computationally expensive search techniques for the Multi-Objective Formulation.

2.5.5.1 NSGA-II

The Non-dominated Sorting Genetic Algorithm-II (NSGA-II), introduced by Deb et al. [50] is an extension to an earlier Multi-Objective EA called NSGA developed by Srinivas and Deb [174]. NSGA-II incorporates elitism to maintain the solutions of the best front found. The rank of each individual is based on the level of non-domination. NSGA-II is a computationally efficient algorithm whose complexity is $O(mN^2)$, compared to NSGA with the complexity $O(mN^3)$, where m is the number of objectives and N is the population size.

The population is sorted using the non-domination relation into several fronts. The process is described in Algorithm 2. Each solution is assigned a fitness value according to its non-domination level. In this way, the solutions in better fronts are given higher fitness values. The NSGA-II algorithm uses a measure of crowding distance to provide an estimation of the density of solutions belonging to the same

Algorithm 2: NSGA-II (fast-non-dominated-sort) [50] Deb (2001)

input : R_t
output: F

for *each individual(p)*$\in R_t$ **do**
  Set $S_p = \varnothing$ // the set of individuals dominated by individual p
  Set $n_p = 0$ // the number of individuals that dominate individual p
  for *each individual(q)*$\in R_t$ **do**
    if *individual(p) dominates individual(q)* **then**
      Let $S_p = S_p \cup \{\text{individual}(q)\}$
    end
    else if *individual(q) dominates individual(p)* **then**
      Let $n_p = n_p + 1$
    end
  end
  if $n_p == 0$ **then**
    Set $P_{rank} = 1$
    Let $F_1 = F_1 \cup \{\text{individual}(p)\}$
  end
end
Set $i = 1$
while $F_i \neq \varnothing$ **do**
  Set $Q = \varnothing$
  for *each individual(p)*$\in F_i$ **do**
    for *each individual(q)*$\in S_p$ **do**
      Let $n_q = n_q - 1$
      if $n_q == 0$ **then**
        Let $q_{rank} = i + 1$
        Let $Q = Q \cup \{\text{individual } q\}$
      end
    end
  end
  Let $i = i + 1$
  Let $F_i = Q$
end

front, which is described in Algorithm 3. This parameter is used to promote diversity within the population. Solutions with higher crowding distance are assigned a higher fitness compared to those with lower crowding distance, thereby avoiding the use of any form of 'fitness sharing factor' with its associated computational cost [87].

Algorithm 3: NSGA-II (crowding-distance-assignment) [50] Deb (2001)

input : F
output: F

Set $l = |F|$
for *each individual(p)*$\in F$ **do**
  Set $D_p = 0$
end
for *each objective* m **do**
  Let $F = sort(F, m)$
  Let $D_1 = D_l = \infty$
  for $p = 2$ *to* $(l-1)$ **do**
    Let $D_p = D_p + (D_{p+1}.m - D_{p-1}.m)/(f_m^{max} - f_m^{min})$
  end
  // $D_p.m$ is the value of the mth objective function for individual p
  // f_m^{max} and f_m^{max} are the maximum and minimum values of the mth objective function.
end

The algorithm progresses as follows. Initially, a random population P_0 with size N is created. Tournament selection, single-point crossover, and bitflip mutation operators are used to create a child population Q_0 of size N [50]. The main loop is described in Algorithm 4.

2.5.5.2 Two-Archive

The Two-Archive algorithm was introduced by Praditwong and Yao [145]. It substitutes the new dominated solutions for the existing dominated solutions introduced by PESA [107]. Truncation is performed at the end of the process of archiving non-dominated individuals, as is in NSGA-II [50] and SPEA2 [204].

Algorithm 4: NSGA-II (main loop) [50] Deb (2001)

input : $t = 0$; P_0 and Q_0
output: F_{t-1}

while $t <= MAX\_GENERATION$ **do**
  Let $R_t = P_t \cup Q_t$
  Let F = fast-non-dominated-sort (R_t) $//F = (F_1, F_2, ..., F_i, ...)$
  Let $P_{t+1} = \varnothing$ *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 $P_{t+1} = P_{t+1} \cup F_i[1 : (N - |P_{t+1}|)]$
  Let Q_{t+1} = make-new-pop(P_{t+1})
  Let $t = t + 1$
end

In the algorithm there are two archives, used to collect and retain the candidate solutions in the population: the convergence archive (CA) and the diversity archive (DA). If a non-dominated solution, selected from the population, dominates some other member in either archive, it enters the CA and the dominated members are deleted, otherwise it enters the DA without deleting any members in either archive.

The total combined size of CA and DA is fixed but the size of each archive may vary. When the number of members in the archives exceeds the capacity of archives, the members of DA with the shortest distance to some member in CA are deleted until the total size falls below a predefined threshold.

The details of the Two-Archive algorithm are shown in Algorithm 5 and Algorithm 6.

2.5.5.3 Pareto GA

The Pareto GA algorithm used in this research is a variation of the simple genetic algorithm. A simple GA maintains the populations of candidate solutions that are

Algorithm 5: The Two-Archive Algorithm (the main loop) [145] Praditwong and Yao (2006)

```
Initialise the population
Initialise archives to the empty set
Evaluate initial population
Set t=0
repeat
    Collect non-dominated individuals in archives
    Select parents from archives
    Apply genetic operators to generate a new population
    Evaluate the new population
    t=t+1
until t == MAX_GENERATION ;
```

evaluated according to a single fitness value. In the MONRP there are at least two fitness values for each solution which are used in tournament selection. The algorithm uses Pareto dominance relations between solutions to select candidates for the *mating pool*. The new population is created by recombining the selected solutions using crossover and mutation operators. The top level procedure of the Pareto GA is described in Algorithm 7.

2.5.5.4 Greedy

The Greedy algorithm makes the optimal choices depending on the information at hand without considering the effect in the future. As such, Greedy is for local optimisation with the hope of leading to global optimisation. It is simple, efficient and easy to implement, so it is commonly used in practice. However, many problems cannot be solved correctly by Greedy. In the thesis, two simple Greedy algorithms were applied to the problem sets and were compared with other techniques.

Algorithm 6: Collect Non-Dominated Individuals in archives [145] Praditwong and Yao (2006)

```
for i=1 to popsize do
    if individual(i) is non-dominated solution then
        if no member in both archives can dominate an individual(i) then
            Set individual(i).Dflag=0
            if individual(i) dominates any member in both archives then
                Set individual(i).Dflag=1
                Delete dominated member
            end
            if member(i).Dflag==1 then
                Add individual(i) to Convergence Archive(CA)
            else
                Add individual(i) to Diversity Archive(DA)
            end
        end
    end
end
// Remove Strategy
if sizeof(CA)+sizeof(DA)>limit then
    for i=1 to sizeof(DA) do
        DA(i).length=maxreal
        for j=1 to sizeof(CA) do
            if DA(i).length>Dist(CA(j),DA(i)) then
                DA(i).length=Dist(CA(j),DA(i))
            end
        end
    end
    repeat
        Delete the member of DA with the shortest length
    until sizeof(DA)+sizeof(CA)==limit ;
end
```

Algorithm 7: Pareto GA

```
t = 0;
Generate population P_0;
while not stopping rule do
    Evaluate objective functions for ∀i ∈ P_t;
    Let t = t + 1;
    Let i = 0;
    while i < N do
        Consider two solutions x and y ∈ P_t at random;
        Let selected solution = ∅;
        if x ≻ y then
            Let selected solution = x;
            Let i = i + 1;
        end
        else if y ≻ x then
            Let selected solution = y;
            Let i = i + 1;
        end
        Add selected solution to mating pool;
    end
    Apply genetic operators (crossover and mutation) to P_t;
end
```

2.5.5.5 Single-Objective GA

To apply a Single-Objective GA to MONRP, we use a weighting method that combines the objective functions into a single objective using a weight factor $\omega(0 \leq \omega \leq 1)$. The fitness value of a given solution $\overrightarrow{x}$ in the Single-Objective GA is calculated as follows:

$$F(\overrightarrow{x}) = (1 - \omega) \cdot f_1(\overrightarrow{x}) + \omega \cdot f_2(\overrightarrow{x})$$

By changing the weight factor $\omega \in [0, 1]$, one can explore various regions of the Pareto-optimal front. This approach is employed to compare the performance of the Single-Objective GA with the performance of other search techniques.

2.5.5.6 Random Search

We also applied the Random Search technique to the release planning problem. This is merely a 'sanity check'; all metaheuristic algorithms should be capable of comfortably outperforming Random Search for a well-formulated optimisation problem. Thus the Random Search technique was given the same number of fitness evaluations as the other algorithms to provide a useful lower bound benchmark for measuring the other algorithms' performance.

2.6 Summary

This chapter described a review of RE and its process, and introduced search-based techniques. First of all, we presented the conceptions of requirements, RE and related contexts. We then provided the state of the art of the research in each requirements activities: elicitation, modelling and analysis, specification, validation and verification, evolution and management. We next focused on the requirements analysis and optimisation activities and discussed search-based requirements analysis in particular. Finally search-based optimisation techniques from single objective to multi-objective formulations were introduced. The rest of this thesis concerns the applications of these search-based optimisation techniques to novel formulations of RE problems, their evaluations and visualisations as a decision support for the RE process.

Chapter 3

The Framework for Search-based Requirements Selection and Optimisation

This chapter introduces the Search-based Requirements Selection and Optimisation framework that will be used in this thesis. It includes the explicit context, problem search space, representation, solving process and performance metrics for search-based techniques.

The framework lays a foundation for search-based requirements selection and optimisation. It provides quantified representation, process and metrics to formulate release planning as a search-based optimisation problem.

3.1 Context

The framework proposed in the thesis is working in the context of software release planning. "Release planning is known to be a cognitively and computationally difficult problem" [134]. It combines soft factors like human and social constraints with the formalised models. The emphasis of the framework is to provide a systematic method which combines the search-based optimisation approach and multi-objective decision support to solve the computational and cognitive complexity of the release planning problem.

We assume that an initial set of requirements has been collected and the stakeholders have been identified using a requirements elicitation process. We also assume that each one of the stakeholders or optional requirements has its own attributes which are expressed numerically. The task of quantifying requirements is usually regarded to be a challenging and hard problem in itself [181]. Nevertheless, there are still several feasible approaches in previous work that address this problem [4, 70, 100], and so we consider the elicitation process to be beyond the scope of this thesis. The benefits of using quantified requirements include better support for budget estimates and feedback, and improved communication of the requirements [71].

3.2 Search Space

Search-based requirements selection and optimisation lies within the general SBSE framework [76]. SBSE is concerned with the application of search-based algorithms to software engineering topics such as RE. The purpose of all search algorithms is to explore the optimal, near-optimal or "good enough" solutions among a number of possible solutions in a search space.

In the experiments studied here, the search space is the space of all possible subsets of requirements. Therefore, for an existing system with n requirements, there will be $C_n^1 + C_n^2 + \cdots + C_n^n$ kinds of possible solutions to take (if there is no account of the requirements dependencies). Clearly, enumeration will not be possible since n can be arbitrarily large. The solution problem is an $\mathcal{NP}$-hard (nondeterministic polynomial-time hard) problem, which motivates the search-based software engineering approach advocated in this thesis.

3.3 Representation

It is assumed that for an existing software system, there is a set of stakeholders,

$$C = \{c_1, \ldots, c_m\}$$

whose requirements are to be considered in the development of the next release of the software.

Each stakeholder may have a degree of importance for the company that can be reflected by a weight factor. The set of relative weights associated with each stakeholder c_j $(1 \leq j \leq m)$ is denoted by a weight set:

$$Weight = \{w_1, \ldots, w_m\}$$

$w_1 = w_2 = \ldots = w_m$ if all the stakeholders are treated as equal, that is, the stakeholder weight factor can be ignored.

The set of possible software requirements is denoted by:

$$\Re = \{r_1, \ldots, r_n\}$$

As described in Section 2.1.1.1, requirement granularity can be decomposed hierarchically. At the top level (goals) are statements of stakeholder needs; the lower levels (refined subgoals) include stakeholder requirements, system requirements, system component requirements and even subsystem component requirements. The lower the level, the more detailed will be the requirement expression. The requirement r_i is represented in the thesis at a similar level of abstraction. They should not be stated in too much detail nor on too high a level of abstraction. If the level differs among the requirements, there might be difficulty in selecting the correct subset of requirements for release planning.

In order to satisfy each requirement, some resources need to be allocated. The resources needed to implement a particular requirement can be transformed into cost terms and considered to be the associated cost to fulfil the requirement. Typically, these cost values are estimated, which is the case with the real world case studies presented in this thesis. The resultant cost vector for the set of requirements r_i $(1 \leq i \leq n)$ is denoted by:

$$Cost = \{cost_1, \ldots, cost_n\}$$

Usually, different stakeholders have different needs and perspectives. It is assumed that not all requirements are equally important for a given stakeholder. The level of satisfaction for a given stakeholder depends on the requirements that are satisfied in the next release of the software, which provides a *value* to the stakeholders'

organisations. Each stakeholder c_j $(1 \leq j \leq m)$ assigns a *value* to requirement r_i $(1 \leq i \leq n)$ denoted by: $v(r_i, c_j)$ where $v(r_i, c_j) > 0$ if stakeholder c_j desires implementation of the requirement r_i and 0 otherwise.

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

Besides the *value* and *cost* attributes to each requirement, there are five common interaction types between the requirements, that is, *And*, *Or*, *Precedence*, *Value-related* and *Cost-related*.

Each stakeholder c_j has, therefore, a subset of requirements that they expect to be satisfied denoted by R_j.

$$\text{such that} \quad R_j \subseteq \Re, \qquad \forall\, r \in R_j \quad v(r, c_j) > 0$$

The overall *score* or importance of a given requirement r_i $(1 \leq i \leq n)$ can be calculated by:

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

The '*score*' of a given requirement is represented as its overall '*value*' for the com-

pany.

The decision vector $\overrightarrow{x} = \{x_1, \ldots, x_n\} \in \{0, 1\}$ determines the requirements that are to be satisfied in the next release. In this vector, x_i is 1 if requirement i is selected and 0 otherwise. This vector denotes the solution to the problem.

3.4 Process

Requirements selection and optimisation are extremely difficult because of their complexity, ambiguity and uncertainty. The framework is designed as an iterative procedure. There are four phases at each iteration described as below:

1. **Representation:** transfer the requirement attributes to measurable criteria and define the decision variables as well as their constraints and dependencies (including stakeholders' evaluation of the requirements).

2. **Fitness:** define the "goodness" of a solution in terms of the problem model, namely, the fitness function. In general, a fitness function is a set of mathematical expressions that automatically qualifies the optimality of a solution.

3. **Exploration:** Apply search-based optimisation techniques to explore the search space and generate optimal or near-optimal solutions as well as expose the implicit characteristics of the problem.

4. **Analysis and Feedback:** Based on the knowledge of design, a decision maker can investigate the insight achieved and the solutions provided in order to fully understand the problem and arrive at the practical solutions. In addition, changes to the system can also be involved in this phase and be evolved for the next iteration.

3.5 Performance Metrics for Search-based Techniques

A number of performance metrics suggested in the literature [49, 50, 145] show that the quality of a set of non-dominated solutions produced can be measured in terms of convergence and diversity. The approximated solutions must obtain good convergence to the *real* Pareto front, and good diversity along the front in population members.

It is usually necessary to know the location of the real Pareto front. In the case of requirements selection and optimisation problems, the real Pareto front is unknown. Therefore, a reference Pareto front was constructed and used to compare the Pareto fronts produced by different algorithms. Consisting of the best solutions of each technique, the reference Pareto front denotes the best available approximation to the *real* Pareto front. The Pareto fronts generated by the different search techniques may partly contribute to the reference Pareto front.

In the thesis, three performance metrics were applied to measure the quality of the Pareto fronts produced by the search-based optimisation algorithms:

3.5.1 Metrics for Convergence

1. The Euclidean distance C between approximated solutions and reference Pareto front measures the extent of convergence to a known set of solutions.

$$C = \frac{\sum_{i=1}^{N} d_i}{N}$$

where N is the number of solutions obtained by an algorithm. d_i is the smallest

Euclidean distance of each solution i to the reference Pareto front. The smaller the calculated value of C, the better the convergence. This metric $C = 0$ if each obtained solutions are exactly on each of the reference solutions.

2. The percentage P of the solutions on the reference Pareto front, namely the solutions that are not dominated by the reference Pareto front, provides a quantitative comparison between algorithms. The larger the number, the better the convergence.

$$P = \frac{num}{NUM}$$

where num is the number of solutions that are obtained by an algorithm and are on the reference Pareto front and NUM is the total number of solutions on the reference Pareto front.

3.5.2 Metric for Diversity

The metric Δ [49] measures the extent of distribution in the obtained solutions and spread achieved between approximated solutions and the reference Pareto front.

$$\Delta = \frac{\sum_{k=1}^{M} d_k + \sum_{j=1}^{N-1} |d_i - \overline{d}|}{\sum_{k=1}^{M} d_k + (N-1)\overline{d}}$$

where $k(1 \leq k \leq M)$ is the number of objectives for a multi-objective algorithm. d_k is the Euclidean distance to the extreme solutions of the reference Pareto front in the objective space. N denotes the number of solutions obtained by an algorithm. $d_j(1 \leq j \leq N-1)$ is the Euclidean distance between consecutive solutions, $\overline{d}$ is the average of all the distance d_j.

The smaller the value of Δ, the better the diversity. This metric $\Delta = 0$ if all the obtained solutions have perfect distribution ($d_j = \overline{d}$) and also include all the extreme solutions on the reference front ($d_k = 0$).

Chapter 4

Value/Cost Trade-off in Requirements Selection

This chapter is concerned with the Value/Cost based Multi-Objective Next Release Problem (MONRP). Software companies may have more than one objective and constraint in market-driven development. To deliver a quality product in releases, there are trade-offs to the different aspects. Almost all previous work has considered only single objective formulations in release planning problem. In the multi-objective formulation, there are at least two (possibly conflicting) objectives that the software engineer wishes to optimise. It is argued that the multi-objective formulation is more realistic, since requirements engineering is characterised by the presence of many complex and conflicting demands, for which the software engineer must find a suitable balance.

In this Chapter, firstly, a basic Value/Cost multi-objective requirements selection formulation will be introduced in detail including defined fitness function, experimental setup and the results of the analysis of three related empirical studies. Secondly, we will describe our extended formulation – the Today/Future Importance

Analysis which will be applied to a real world data set from Ericsson.

4.1 Introduction

We address the Multi-Objective Next Release Problem (MONRP), in which a set of stakeholders with varying requirements are targeted for the next release of an existing software system. Satisfying each requirement entails spending a certain amount of resources which can in turn be translated into *cost* terms. In addition, satisfying each requirement provides some *value* to the software development company. The problem is to select the set of requirements that maximises the total value and minimises the required cost. These objectives are conflicting and there is no clear way to 'trade' one for another, so it would be difficult to assign weights to each. This makes it inappropriate to combine the two objectives into a single fitness function. Rather, a multi-objective formulation is better suited to this problem.

The single objective problem to maximise the value, subject to a limited and fixed amount of resources (a fixed budget constraint), is an instance of a *Knapsack* Problem which is $\mathcal{NP}$-hard [141]. As a result, the multi-objective problem stated above is also $\mathcal{NP}$-hard, and therefore cannot be solved using exact optimisation techniques for large scale problem instances.

Single objective formulations of the Next Release Problem (NRP) have been considered in the literature. Also, approaches have been considered, where the multiple objectives are combined into a single objective using weight. However, the multi-objective version of the problem, in which there are two or more competing objectives (which may conflict) has not been considered. The multi-objective formulation is important because, in practice, a software engineer is more likely to have many conflicting objectives to address when determining the set of requirements to

include in the next release of the software. As such, the MONRP is more likely to be appropriate than the single objective NRP.

Metaheuristic search techniques are applied to find approximations to the Pareto-optimal set (or front) for the MONRP. This allows the decision maker to select the preferred solution from the Pareto-optimal set, according to their priorities. The Pareto front can also provide valuable insights into the outcome of the selected set of requirements to the software development company, because it captures the trade-offs between the two competing objectives.

4.2 Basic Value/Cost Trade-off Analysis *

4.2.1 Fitness Function

In this section, four types of fitness functions will be introduced that were used in the value/cost trade-off analysis.

4.2.1.1 Multi-Objective based Fitness Function

In the formulation of the MONRP, two objectives are taken into consideration in order to maximise stakeholder satisfaction (or total *value* for the company) and minimise required cost. We treat cost in the current research as an objective instead of a constraint. The reason is to explore the whole set of points on the Pareto-optimal front; this is a valuable source of information for the decision maker to understand the trade-offs inherent in meeting the conflicting objectives and inherent in balancing cost and value.

*This chapter is based on the work published by the author at GECCO 2007 [202].

The following objective function is considered for maximising total value:

$$\text{Maximise } \sum_{i=1}^{n} score_i \cdot x_i$$

The problem is to select a subset of the stakeholders' requirements which results in the maximum value for the company.

The second objective function is defined as follows to minimise total cost required for the satisfaction of stakeholder requirements:

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

In order to convert the second objective to a maximisation problem in the MONRP, the total cost is multiplied by -1. Therefore, the MONRP model consisting of fitness functions can be represented as follows:

$$\text{Maximise } f_1(\overrightarrow{x}) = \sum_{i=1}^{n} score_i \cdot x_i$$

$$\text{Maximise } f_2(\overrightarrow{x}) = -\sum_{i=1}^{n} cost_i \cdot x_i$$

4.2.1.2 Single Objective based Fitness Function

1. Fitness Function for Single Objective GA

   A weight factor $\omega(0 \leq \omega \leq 1)$ was used to combine the two objectives into a single fitness function. It is likely that different objectives are within the different ranges. For example, the *cost* of the requirements may vary between

50 and 1000, whereas the *value* attribute may vary between 0 and 5. When such objectives are weighted, normalisation of objectives is needed to scale them so that each objective has the same magnitude and they are treated equally important.

$$f_1(\overrightarrow{x})_{normalised} = \frac{\sum_{i=1}^{n} score_i \cdot x_i}{\sum_{i=1}^{n} score_i}$$

$$f_2(\overrightarrow{x})_{normalised} = -\frac{\sum_{i=1}^{n} cost_i \cdot x_i}{\sum_{i=1}^{n} cost_i}$$

After the objectives are normalised, the fitness function is represented as follows:

$$\text{Maximise } f(\overrightarrow{x}) = (1-\omega) \cdot f_1(\overrightarrow{x})_{normalised} + \omega \cdot f_2(\overrightarrow{x})_{normalised}$$

By changing the weight factor $\omega \in [0, 1]$ of every execution, one can explore various regions of the Pareto-optimal front.

2. Fitness Function for Greedy (Value/Cost Ratio)

   All the requirements r_i are ranked according to their ratios $score_i/cost_i$ from the highest to the lowest. The fitness function is represented as:

$$\text{Maximise} \qquad \sum_{i=1}^{p} \frac{score_i}{cost_i}$$

   where p $(1 \leq p \leq n)$ is the number of requirements that can be selected in one solution.

3. Fitness Function for Greedy (Highest Value)

All the requirements r_i are ranked according to their $score_i$ from the highest to the lowest. The fitness function is represented as:

$$\text{Maximise} \qquad \sum_{i=1}^{p} score_i$$

where p $(1 \leq p \leq n)$ is the number of requirements that can be selected in one solution.

4.2.2 Experimental Set Up

In this section, we describe the test problems used, the development environment and the algorithmic tuning to compare the performance of NSGA-II with Pareto GA, Random Search, Single-Objective GA and Greedy algorithms.

4.2.2.1 Test Problems

The test problems were created by assigning random choices for value and cost. The range of costs were from 1 through to 9 inclusive (zero cost is not permitted). The range of values were from 0 to 5 inclusive (zero value is permitted, indicating that the stakeholder places no value on it, i.e. does not want this requirement). The five algorithms adapted and applied to the MONRP in this thesis were applied to two test problem sets for three separate empirical study cases: Empirical Study 1 (ES1), Empirical Study 2 (ES2) and Empirical Study 3 (ES3).

In ES1 we report results concerning the performance of the first four algorithms for what might be considered 'typical' cases, with the number of stakeholders ranging from 15 to 100 and the number of requirements ranging from 40 to 140. In ES2, we investigate and compare the search-based algorithms against a greedy algorithm

where requirements are ordered according to the cumulative value or value/cost ratio. In ES3, we are concerned with bounding the problem below, to determine the size of the problem for which the search is not appropriate; the point at which the problem becomes too small. In order to do this, we seek to find the point at which the metaheuristic techniques can (only just) outperform a Random Search, since we deem this to indicate that the problem is sufficiently large for it to be worth considering the application of metaheuristic search techniques.

4.2.2.2 Environment

All the experiments were performed using the MATLAB system (*R2007a*). The main programming language is Matlab script and all the data sets were formatted into .mat or .dat files. The system was installed on an Intel Core 2 Duo processor 2.26 GHz with 4Gb RAM.

4.2.2.3 Algorithmic Tuning

All search-based approaches were run for a maximum of 50,000 fitness function evaluations. The initial population was set to 500. We used a simple binary GA encoding, with one bit to code for each decision variable (the inclusion or exclusion of a requirement). The length of a chromosome is thus equivalent to the number of requirements. Each experimental execution of each algorithm was terminated when the generation number reached 101 (i.e after 50,000 evaluations). The NSGA-II algorithm was implemented based on Aravind Seshadri's Matlab package. All genetic approaches used tournament selection (the tournament size is 5), single-point crossover and bitwise mutation for binary-coded GAs. The crossover probability was set to $P_c = 0.8$ and mutation probability to $P_m = 1/n$ (where n is the string length

for binary-coded GAs) were used.

In the Single-Objective GA, there are nine different weight coefficients ω for each objective ranging from 0.0 to 1.0. The step size of weight is 0.1. The algorithm was executed 11 times for different choices of weight coefficients to obtain different solutions within a single experiment.

The two greedy algorithms adopted in the thesis were executed n (the number of requirements) times. For each execution, the two algorithms attempt to find the best solution (with a fixed subset size) according to two fitness functions described in Section 4.2.1.2 separately. The subset size increases from 1 to n in the process of n-time executions. For example, the ith execution is able to include the best i requirements combination in one solution.

4.2.3 Empirical Study 1 – Scale Problem

4.2.3.1 Motivation

In ES1 we investigate the relative performance of the four approaches to the MONRP for cases that we consider typical. We consider three 'scales' of problem that we hope are characteristic of some of the problems to which the approach may be applied.

4.2.3.2 Data sets

The number of stakeholders and requirements for each scale problem is listed in Table 4.1:

Table 4.1: Scale Data Sets

Scale	Stakeholder	Requirement
S1	15	40
S2	50	80
S3	100	140

4.2.3.3 Results and Analysis

The Pareto front in this problem is orientated towards the upper right. The closer to the upper right the better the solution. All solutions on the Pareto front are optimal. The '$\circ$', '$\triangle$' and '$\square$' symbols plotted in the figures denote the final non-dominated solutions obtained by NSGA-II, Pareto GA and Single-Objective GA separately. The '$\cdot$' symbols plotted the all solutions generated by Random Search. Each symbol represents one or more solutions that have the same fitness values. Each solution includes a subset of requirements selected.

It is possible to allow the software engineers to use Pareto front evaluation as a comparative tool for a number of objectives. First of all, the analyst can pick up the best solutions on the Pareto front in different circumstances based on their priorities. For example, at a specific budget level for a project, one or more optimal solutions might be found when moving along the Pareto front. Meanwhile, the stakeholders' satisfaction can also be concerned. Each satisfaction level has its own corresponding cost (resource allocation, spending) according to the Pareto front. This can help the analyst make rough estimates and adjustments for a project budget.

Moreover, the Pareto front not only gives solutions themselves, but also may yield interesting insights into the nature of the problem. The shape of the Pareto front (concave, convex, discontinuous, knee point, nadir point, etc.) reflects the structure

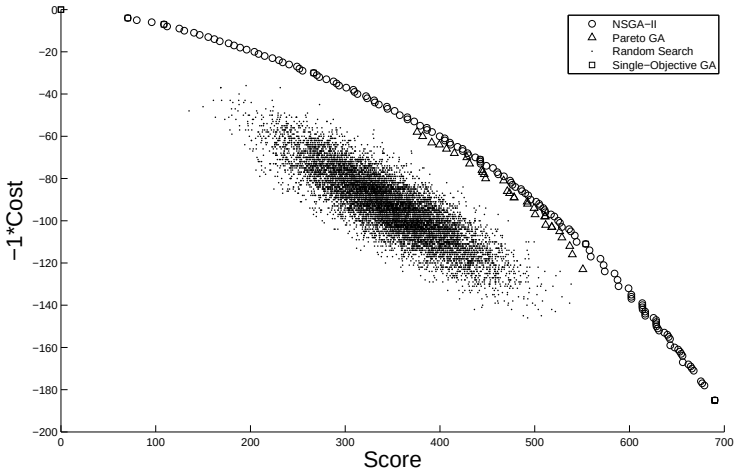
of data in an intuitive way, and provides the analyst with very valuable information about the trade-off among the different objectives and helps fully understand the problem and reach practical solutions. In addition, the Pareto front is useful when communicating with others about the results.

The results of S1, S2 and S3 are shown in Figure 4.1 (a-c). The figure shows typical results from the 30 runs of each algorithm. In each picture, the results show the non-dominated front obtained after 100 generations with NSGA-II, Pareto GA, Single-Objective GA, and all the solutions obtained by Random Search.

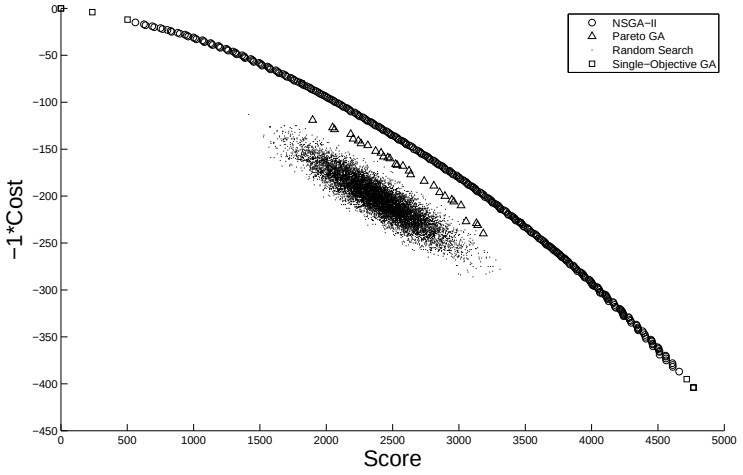
The results lend evidence to support the following claims:

1. The NSGA-II algorithm performs the best in all three scale problems. Figures 4.1 (a), (b) and (c) show a smooth, non-interrupted Pareto-optimal front generated by NSGA-II. Clearly, it is able to find a better diversity of solution distribution and it also converges on results that are better than those for any of the other algorithms, because the front line dominates those produced by the other approaches. This provides empirical evidence that NSGA-II is effective in solving the MONRP and that it may outperform the Pareto GA and the Single-Objective GA (as well, of course, as random).

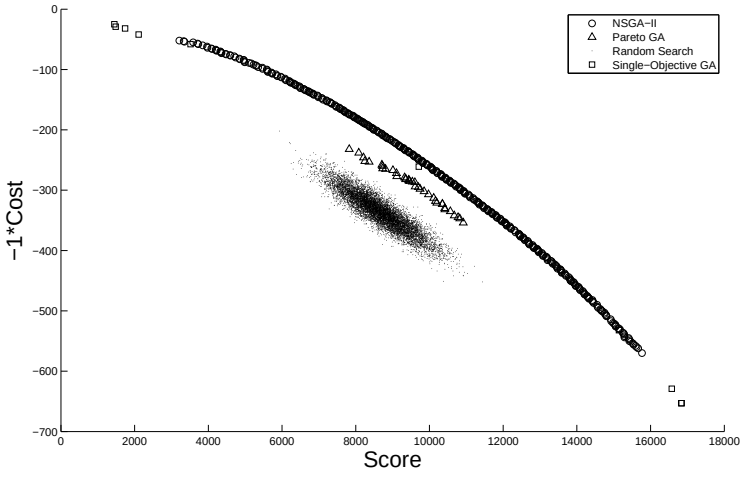
2. The solutions obtained using the weight-based Single-Objective GA where ω was close to 0.0 or 1.0 drive the search towards the extreme ends of the Pareto front. These extreme parts of the Pareto front do not appear to be explored by NSGA-II, even though it does produce a good spread of results. In such extreme solutions, one objective is maximised to the almost total exclusion of the other. Although the Single-Objective GA may have difficulties in finding a good spread of solutions near the Pareto-optimal front, these extreme non-dominated solutions which the Single-Objective GA managed to



(**a**) 15 stakeholders; 40 requirements



(**b**) 50 stakeholders; 80 requirements



(**c**) 100 stakeholders; 140 requirements

Figure 4.1: In (a), (b) and (c) metaheuristic search techniques have outperformed Random Search. NSGA-II performed better or equal to others for a large part of the Pareto front while Single-Objective performed better in the extreme regions. The gap between search techniques and Random Search became larger as the problem size increased.

find provide a necessary supplement to the NSGA-II solutions. They may be useful in real world scenarios because they allow the software engineer to explore the extremes of the Pareto front in order to discover where one objective dominates.

3. The trend we observed from the three experiments is that the larger the scale of the test problem, the wider the gap in performance, both between meta-heuristic techniques and Random Search and between the leader (NSGA-II) and the runner up (Pareto GA). This indicates that as the problem scales, the performance improvement offered by the use of NSGA-II also scales, making it an increasingly preferred choice of solution algorithm.

Another observation of results is the different execution time on three scales. We measured the execution time of NSGA-II in data set levels listed in Table 4.2. The unit of time measured is CPU time.

Table 4.2: CPU Time of Scale Data Sets

Data Sets	CPU Time
15 – 40	62.69
50 – 80	225.69
100 – 140	714.79

4.2.4 Empirical Study 2 – Search vs. Greedy

4.2.4.1 Motivation

Currently, the most common approach for solving value-cost analysis is to use a greedy strategy where requirements are ordered according to the cumulative highest

value or a value/cost ratio. In ES2, the aim is to investigate and compare the performance between search-based algorithms and greedy algorithms, and to evaluate and analyse the quality of the results produced by the algorithms.

4.2.4.2 Data sets

There are two data sets used in the experiments. The first data set is the same as the one adopted in Section 4.2.3.2, namely three synthetic sub data sets: S1, S2 and S3. The second data set was provided by Motorola [15]. The Motorola data set concerns a set of 35 requirements for hand held communication devices. The stakeholders are four mobile telephony service providers, each of which has a different set of priorities with respect to the features that they believe ought to be included in each handset. Motorola also maintain cost data in the form of the estimated cost of implementation of each requirement.

4.2.4.3 Results and Analysis

The results of S1, S2 and S3 are illustrated in Figures 4.2, 4.3 and 4.4 separately; and the results of Motorola data set is shown in Figure 4.5. The '$\circ$', '$+$' and '$\times$' symbols plotted in the figures denote the final non-dominated solutions found by the NSGA-II algorithm, Greedy (Value/Cost Ratio) and Greedy (Highest Value) separately. They are also marked in different shades of grey to distinguish between them.

In Figure 4.2, we observe that most of the '$+$' solutions obtained by Greedy (Value/Cost Ratio) lie exactly on the '$\circ$' Pareto front produced by the NSGA-II algorithm. Greedy (Highest Value) has the worst convergence in the three algorithms. In terms of number of solutions found, Greedy (Value/Cost Ratio) and Greedy

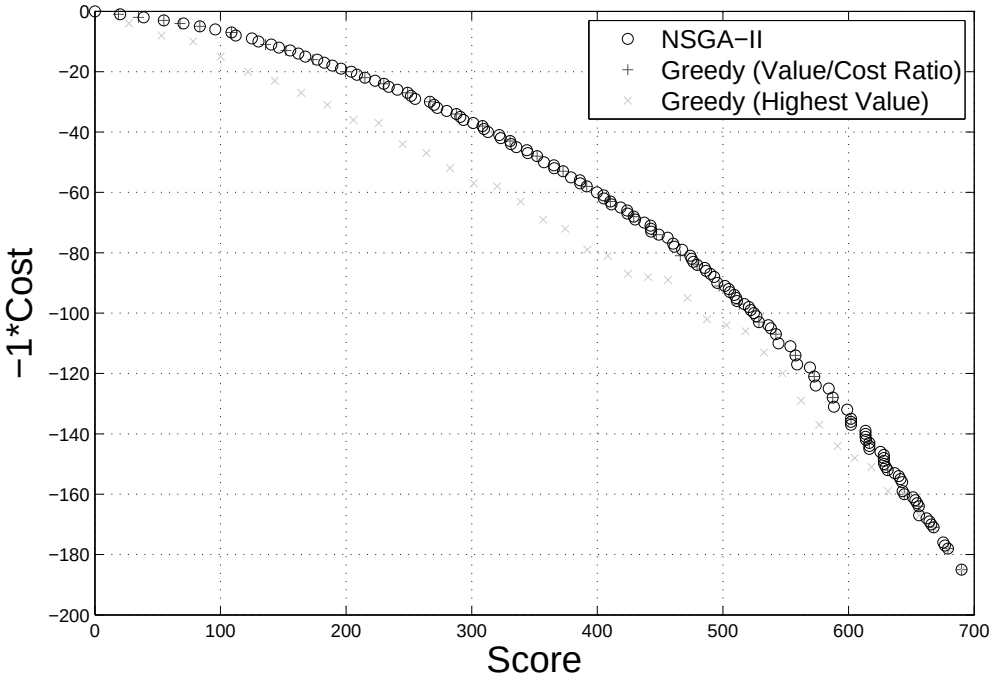


Figure 4.2: Search vs Greedy: 15 stakeholders; 40 requirements

(Highest Value) are the same, which are equivalent to the total number of requirements. That is, each '+' and '×' represent only one solution. NSGA-II yielded the relatively larger number of solutions. In addition, each '∘' represents one or more solutions that have the same fitness values. These solutions form a non-interrupted and smoother Pareto front than the Greedy algorithm's front.

We illustrate the results in Figures 4.3 and 4.4 for scenarios in which the size of data sets scales up. Firstly, the two Greedy strategies have better diversity than NSGA-II. This can be seen from the graphs: Greedy yielded some extreme non-dominated solutions which were not found by NSGA-II. Secondly, in terms of the convergence, the gap between the solutions of the three types becomes wide as the scale increases. Also, most of the solutions obtained by NSGA-II outperformed the others, with only a few slightly worse than Greedy (Value/Cost Ratio).

The above three figures are the results obtained from synthetic data sets. Figure 4.5

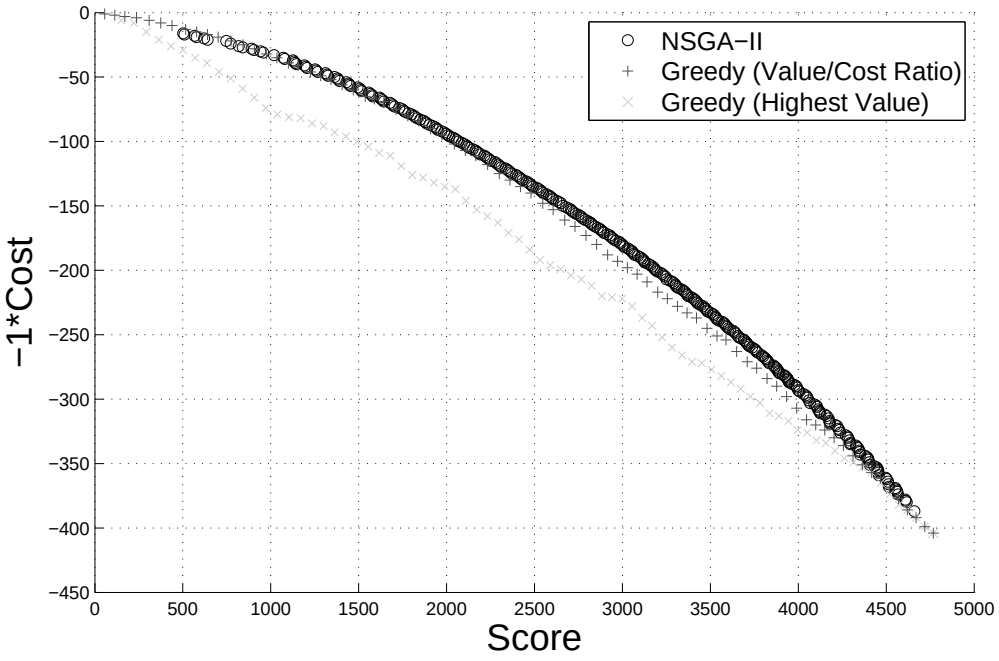


Figure 4.3: Search vs Greedy: 50 stakeholders; 80 requirements

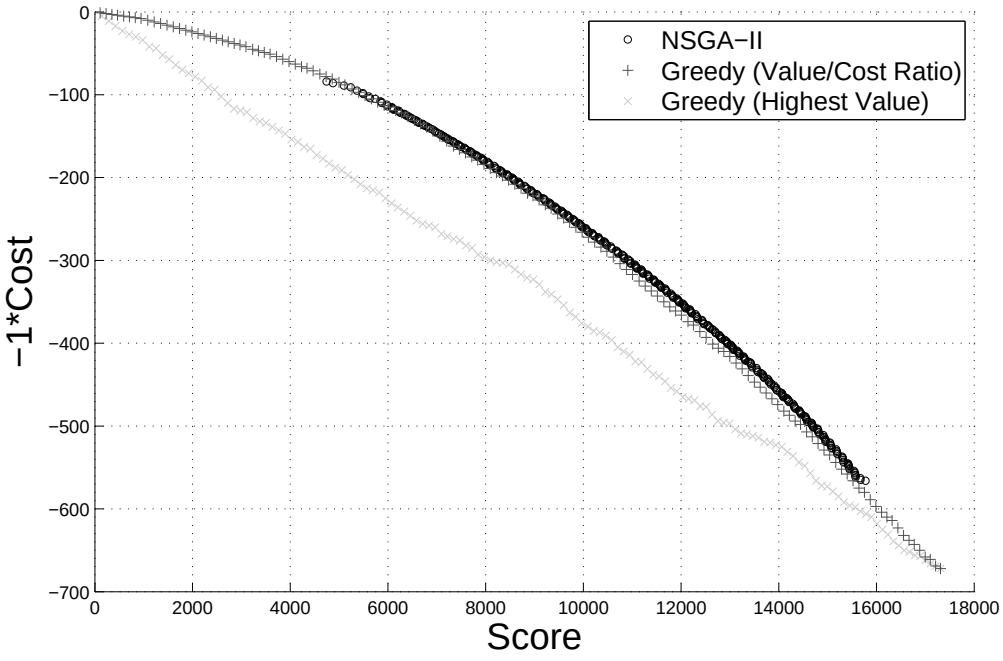


Figure 4.4: Search vs Greedy: 100 stakeholders; 140 requirements

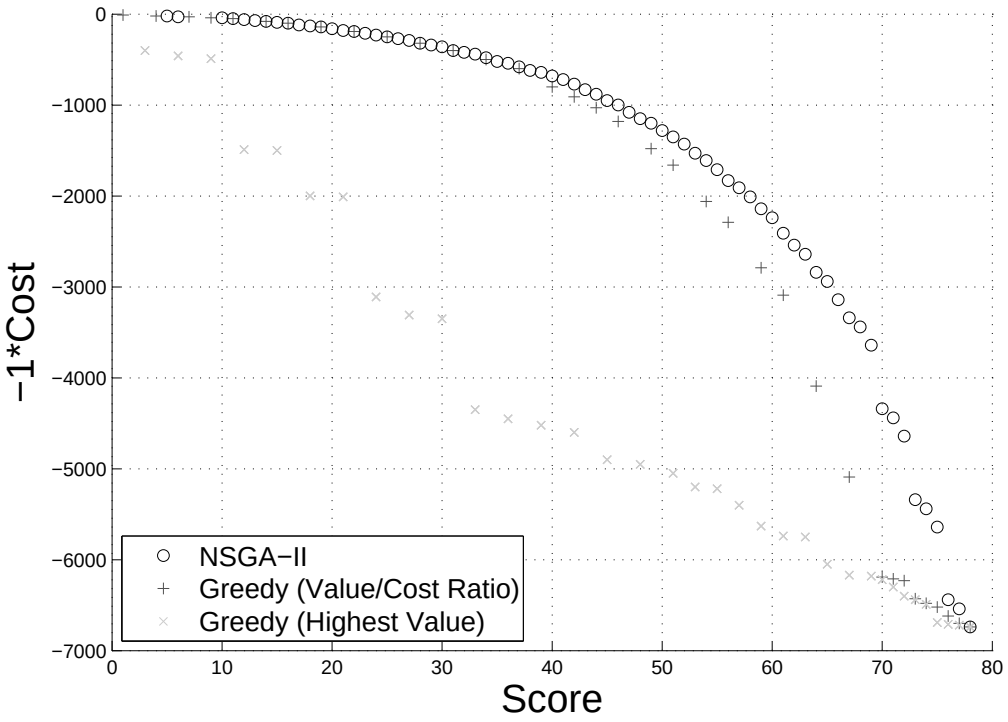


Figure 4.5: Search vs Greedy: 4 stakeholders; 35 requirements

illustrates the results produced from Motorola data set. Although the scale of this real world data set is the smallest in the empirical studies, the results shown in the figure are very interesting. NSGA-II performed the best in both convergence and diversity. Greedy could only produce a discontinuous Pareto front with a limited number of solutions. Moreover, Greedy (Value/Cost Ratio) was worse than NSGA-II on the convergence performance.

Compared to the results generated from synthetic and real world data sets, we can clearly find that NSGA-II had a consistently high performance on the all data sets, but this is not the case for Greedy. Greedy (Value/Cost Ratio) could obtain relatively good solutions from three synthetic data sets, but performed badly on the Motorola data set. The reason for this poor performance is that the greedy algorithm highly depends on the character of a data set.

For S1, S2 and S3, these three synthetic data sets were generated randomly following a uniform distribution. So the average *Values* for the requirements assigned by m stakeholders could be flattened out. That is, all the requirements might have very similar values (if m is large). In this case, there might be actually only one objective that can be compared for requirements – *Cost*. Therefore, Greedy (Value/Cost Ratio) has the ability to closely converge to the Pareto front generated by NSGA-II, with good diversity but a limited number of solutions.

For the real world data set, unlike synthetic ones, all the *Value* and *Cost* related data were maintained by Motorola. When two objectives have competing and often conflicting priorities, NSGA-II showed superiority in performance over Greedy, even on a small-scale data set. Like the Single-Objective GA discussed in ES1, especially when the data sets scale up, Greedy (Value/Cost Ratio) explored the extremes of the Pareto front and, thereby, provided a good supplement to the NSGA-II solutions.

4.2.5 Empirical Study 3 – Boundary Problem

4.2.5.1 Motivation

In ES3, we want to explore the boundaries of MONRP. For the lower boundary, there are two extreme situations, namely many stakeholders with very few requirements and few stakeholders with many requirements. In this section, we are interested in obtaining the 'critical mass' of both stakeholders and requirements. Once the critical mass exceeds a predetermined critical point, it becomes worthwhile applying metaheuristic techniques to the MONRP; below this point, the problem is too trivial to merit a metaheuristic solution.

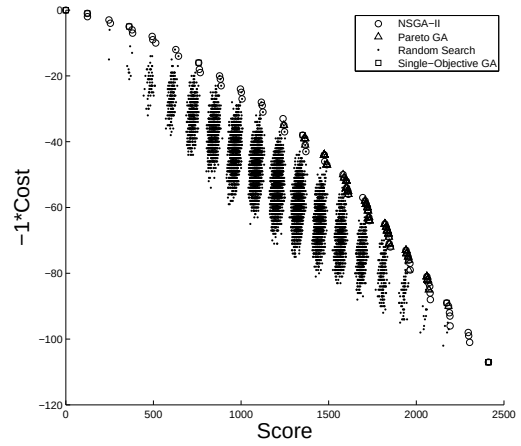
4.2.5.2 Data sets

In the first situation, there are many stakeholders and few requirements. We set the number of stakeholders to 100, with the aim of finding the smallest set of requirements to meet the critical point. In the second situation, there are few stakeholders and many requirements. We assume that the smallest number of stakeholders is 2.

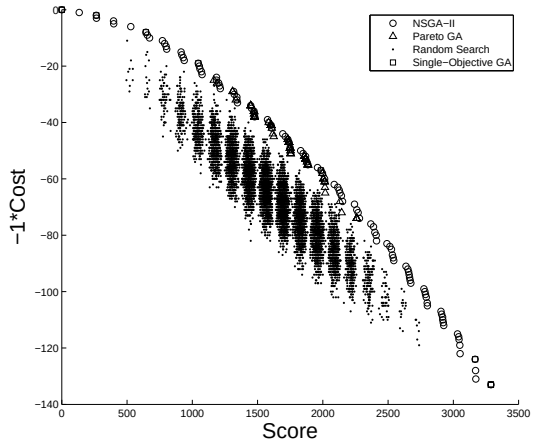
4.2.5.3 Results and Analysis

Figure 4.6 (a) illustrates this critical point. From the figure we can see that there is no gap between the solutions generated by the Random Search and the Pareto-optimal front produced by NSGA-II. The number of requirements is set to 20 in Figure 4.6 (a). NSGA-II and the Random Search solutions share several common points. However, when we increase the number of requirements to 25, as shown in Figure 4.6 (b), we can see that there is a clear gap between the solutions produced by Random Search and NSGA-II. If we increase the number of requirements further, the gap continues to widen. Another observation is that the shapes of Pareto fronts generated in the first situation are not continuous because of comparatively small number of requirements in the data sets.

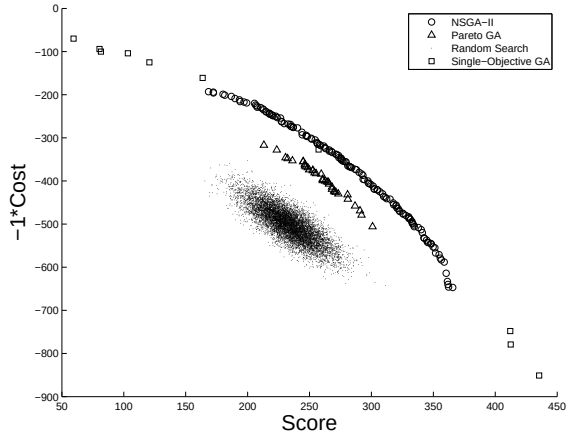
In the second situation, even with only 2 stakeholders, with sufficient requirements, there is a noticeable difference in the performance of the metaheuristic search algorithms and Random Search. This is illustrated in Figure 4.6 (c), where the number of requirements is 200. In this figure, NSGA-II clearly outperforms the other algorithms.



(**a**) 100 stakeholders; 20 requirements



(**b**) 100 stakeholders; 25 requirements



(**c**) 2 stakeholders; 200 requirements

Figure 4.6: (a) shows the boundary case concerning the number of requirements beyond which Random Search fails to produce comparable results with metaheuristic search techniques. (b) shows 25% increase in the number of requirements, the gap became significant. The NSGA-II performed the best, and Pareto GA shared part of the front with NSGA-II. (c) shows that the gap was obviously large. The NSGA-II has outperformed Pareto GA but not the Single-Objective GA in the extreme ends of the front.

Table 4.3: CPU Time of Boundary Data Sets

Data Sets	CPU Time
100 – 20	119.42
100 – 25	137.30
2 – 200	156.23

4.3 Today/Future Importance Analysis

4.3.1 Motivation

This section further extends the basic Value/Cost trade-off model by taking requirements change into consideration. It has been widely observed [60, 74] that requirements are the subject of many changes and revisions as the process of development progresses. This tendency for requirements to change throughout the development process has been regarded as one of the reasons why software development is so difficult and expensive. The change might be either because the stakeholder's need itself changes or because incomplete, uncertain and/or vague information in the earlier stage causes change. So we need a flexible and robust approach to requirements change management.

One specific situation is discussed in the thesis. As described in Chapter 3, Section 3.3, not all requirements are of equal value to the stakeholder. When asking a stakeholder "How important is this requirement to you?", the requirements are graded by the respondents to low, medium or high. But this degree of importance – the *value* of the requirement might vary over time for a specific stakeholder. For example, one requirement might currently have a low value but it may become very important in future, or vice verse. Under these circumstances, we introduce *Today/-Future Importance Analysis* (T/FIA). In T/FIA, the search technique is adapted to

seek to find robust solutions, both for today and for the future. The approach seeks to find a balance between what might be termed 'immediate requirement need' and 'future requirement stability'.

There are two types of requirements change: 1. predictable and 2. unpredictable. 1. Unpredictable requirements change is very hard to deal with; it would require one to 'expect the unexpected'. This section considers the comparatively easier problem of release planning with respect to a set of 'known likely requirements changes'. 2. Predictable requirements change applies when it is known that the requirements for a system will change over time and the changes required can be predicated to some extent.

Though the requirements changes may be known to be likely at the start of the development or selection process, it will be advantageous to seek a balance between those requirements for meeting today's needs and those for the future. For instance, where a system is to be acquired from a known set of possible candidate choices or a configuration of a system is to be chosen that is to be standardized across an entire organisation, there will be a set of choices. These choices can be thought of as a set of requirements to be selected.

In selecting the requirements, there is a need to balance the immediate needs of the organisation against those of the future. Since there may be many stakeholders with competing and conflicting ideas about the set of requirements to be chosen, simply balancing the needs of each stakeholder with regard to today's needs is, in itself, a non-trivial SBSE problem [14, 75, 98, 202]. However, the decision maker should also attempt to take account of likely known changes in requirements over time, in making the choice of system. This raises the issue of the independent (but also potentially competing) objective of selecting a set of requirements that is not only optimal for today, but also for the future.

4.3.2 Data Sets

A real world data set is taken from Ericsson to evaluate our approach to T/FIA. It includes questionnaire forms for test management tools which were completed by 14 groups of Ericsson software test engineers, each drawn from different parts of the organisation. The test management tool is software used for generating, organising and executing the tests (manual or automatic), allowing for requirements tracking, defect tracking and test result reporting.

This questionnaire included 124 requirements for a possible test management tool which was to be selected. The requirements were divided into three major aspects: general, test management and technical requirements. The details are listed in the Table 4.4. The questionnaire design and collection of data was performed by Ericsson, entirely independently of their involvement with the author of this thesis.

To complete the questionnaires, the 14 stakeholders measured how important each requirement is to them in two ways. One is to evaluate the degree of importance for today, the other is the importance for the future. This approach was adopted by Ericsson and not suggested by the author. However, we realised that the information could be useful as a source of analysis for requirements change (from a novel perspective), and this suggested the T/FIA approach, using a multi-objective approach, advocated also elsewhere in this thesis.

Each measurement was graded in four levels: *ignore*, *low*, *medium* or *high*. For instance, "The tool shall support project-specific test case parameters, such as test case priority, test environment" was assigned as *medium* for today and *high* for the future by one of the stakeholders, which is a requirement in the "Test Analysis and Design" section of the questionnaire.

Each stakeholder $c_j (1 \leq j \leq m)$ assigns two types of *value* to each requirement

Table 4.4: Requirements for Test Management Tools

1.	General Requirement
1.1	Application
1.2	Usability
1.3	System Environment and Installation
1.4	Service and Support
1.5	The Tool Supplier
2.	Test Management Requirement
2.1	Test Planning and Management
2.2	Test Analysis and Design
2.3	Test Artefacts Handling
2.4	TR or Defect Handling
2.5	Requirement Handling
2.6	Test Reporting (actual and trends graphs)
2.7	Configuration Management
2.8	Interfaces
3.	Technical Requirement
3.1	Capacity
3.2	Reliability, Stability and Scalability
3.3	Security

$r_i (1 \leq i \leq n)$ denoted by *value for today*: $v_{today}(r_i, c_j)$ and *value for the future*: $v_{future}(r_i, c_j)$, where $v(r_i, c_j) \in \{low, medium, high\}$ if stakeholder c_j desires implementation of the requirement r_i and $v(r_i, c_j) = ignore$ otherwise. The four levels *ignore*, *low*, *medium* and *high* were quantified by assigning numerical value of 0, 3, 6 and 9 respectively.

Accordingly, the *score* of a given requirement r_i can be represented as:

$$score\ for\ today: \qquad score_{(i,today)} = \sum_{j=1}^{m} w_j \cdot v_{today}(r_i, c_j)$$

$$\textit{score for the future}: \qquad score_{(i,future)} = \sum_{j=1}^{m} w_j \cdot v_{future}(r_i, c_j)$$

where stakeholder *weight* $w_1 = w_2 = \ldots = w_m$, that is, all the stakeholders are treated as equals in this study.

4.3.3 Fitness Function

The purpose of T/FIA is to provide robust solutions not only in the context of present conditions but also in response to those future changes that can be anticipated. Therefore, three objectives are taken into consideration in order to maximise stakeholder satisfaction for today, for the future and to minimise the cost of implementation.

The following two fitness functions are considered for maximising total value for today and future respectively:

$$\text{Maximise } f_1(\overrightarrow{x}) = \sum_{i=1}^{n} score_{i,today} \cdot x_i$$

$$\text{Maximise } f_2(\overrightarrow{x}) = \sum_{i=1}^{n} score_{i,future} \cdot x_i$$

The problem is to select a subset of the stakeholders' requirements which results in the maximum value for the company.

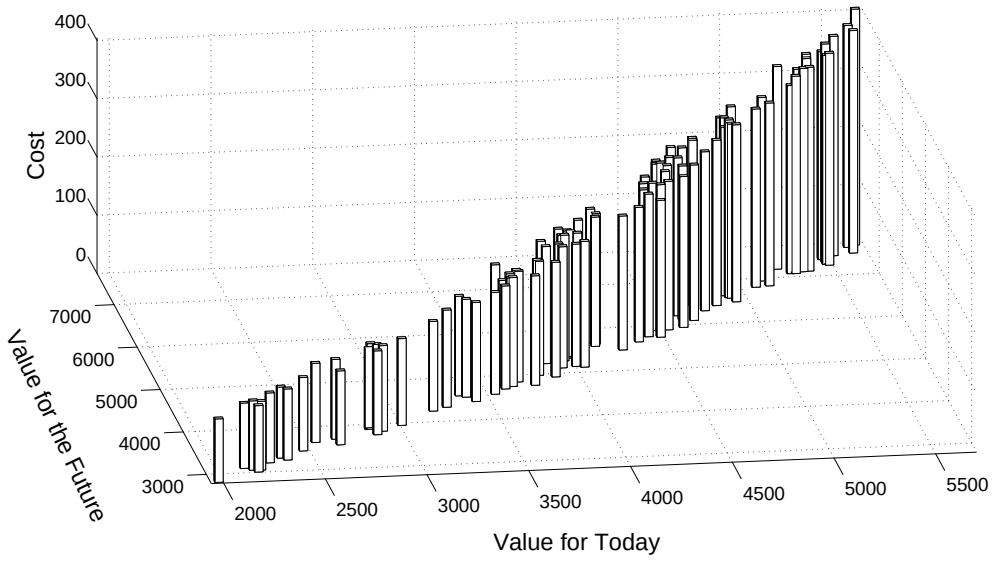


Figure 4.7: Results from Ericsson Data Set: 124 Requirements, 14 Stakeholders

The third fitness function is defined as follows to minimise total cost required for the satisfaction of stakeholder requirements.

$$\text{Minimise } f_3(\overrightarrow{x}) = \sum_{i=1}^{n} cost_i \cdot x_i$$

Since we cannot say whether it is better to be a good solution for today (at the expense of the future) or vice versa, it seems natural to treat the objectives as incomparable and to use a Pareto-optimal approach, as we have advocated elsewhere in this thesis.

4.3.4 Results and Analysis

The NSGA-II algorithm was also chosen to this empirical study. The algorithm tuning parameters used are the same as described in Section 4.2.2.3.

The results are plotted in Figure 4.7. In the figure, it is a three-dimensional solution

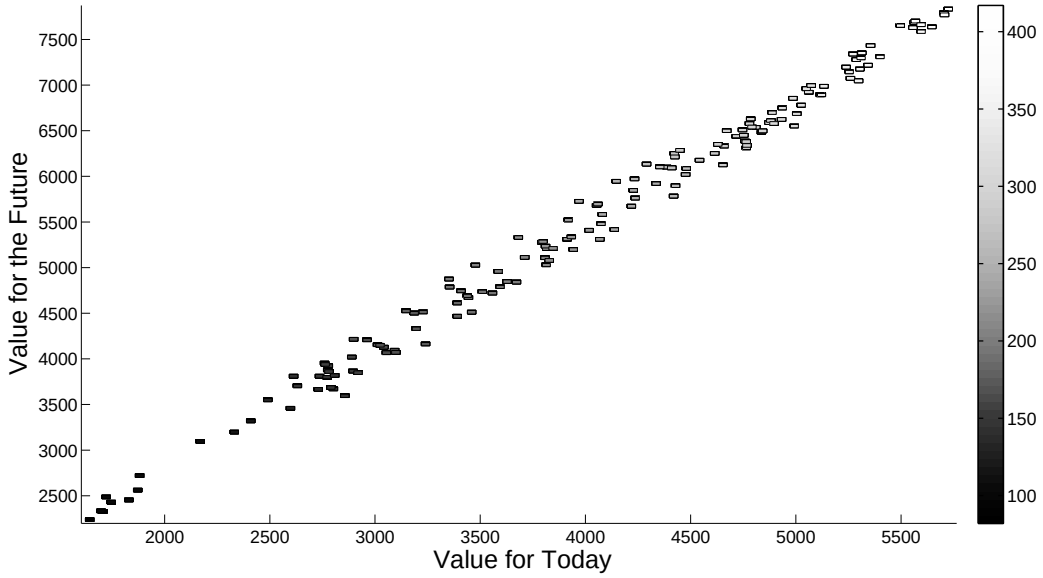


Figure 4.8: Projection onto the X-Y Plane (Results from Ericsson Data Set)

space. The three axes represent the three objectives as *Value for Today*, *Value for the Future* and *Cost*. Each bar denotes an optimal solution on the Pareto front. The location of each bar in the horizontal plane shows the fitness values of the first two fitness functions for all the stakeholders. The height of each bar presents the overall cost for each optimal solution. The average execution (CPU) time is 210.57.

It can be seen that the overall *Value* for both today and future increase with no bias along the *Cost* axis in the graph. The algorithm produced a Pareto front with a good spread. We provide another view of the results onto the X-Y plane, shown in Figure 4.8. The cost of solutions is represented as the different grey scales. If one slices the graph according to different cost levels, corresponding solutions can be found on the Pareto front. The decision maker can choose the optimal one in different contexts from alternative solutions.

In addition, we can analyse the character of the data set based on the results. The shape of Pareto front in Figure 4.7 looks like a "thin hedge" which bisects the

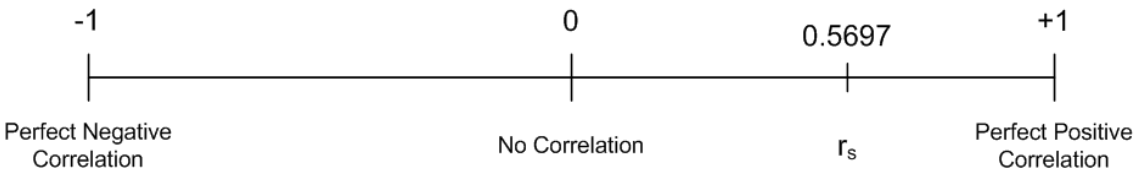


Figure 4.9: Spearman's Rank Correlation Coefficient r_s

solution space. The same observation can be seen from the projected view on the X-Y plane in Figure 4.8. The solutions are scattered in narrowly on the solution space.

Therefore, from the graph we predict that the requirements' values for today and for the future graded by the stakeholders have a strong correlation. This provides some degree of security that the risk is low compared to a solution in which value for today differs greatly from value for the future. The Spearman's Rank Correlation [103] statistical analysis test was carried out on the Ericsson data set to access whether the two *measurement variables* are indeed correlated. That is, as one variable changes, the other is inclined to change too (a rank correlation). The test generated a Spearman's Rank Correlation Coefficient, r_s, can take value between -1 and $+1$. As illustrated in Figure 4.9, $r_s = -1$ means two variables have a perfect negative correlation (as one increases, the other decreases); Accordingly, $r_s = +1$ means two variables have a perfect positive correlation (as one increases, so does the other); $r_s = 0$ means two variables are entirely independent; there is no correlation between them.

In our test, $r_s = 0.5697$ which is larger than the critical value of r_s at the 95% significance level. This indicates a positive correlation between the *value for today* and *value for the future*. Our expectation from the optimisation is also borne out by the statistical analysis. This is a straightforward illustration that the search-based requirements selection not only generates solutions themselves but can also provide

the insights into the data set.

4.4 Summary

This chapter described the investigation of a Value/Cost trade-off analysis for software release planning based on multi-objective optimisation techniques. In the first part of the chapter, the basic two-objective formulation was presented. In the second part, the Today/Future Importance Analysis (T/FIA) was introduced based on the three-objective formulation aiming to select an optimal subset of requirements both for today and future scenarios.

The work formulated the release planning as multi-objective optimisation problem, treating the 'cost' constraint as an objective and combining it with the 'value' objective. In Empirical studies 1, 2 and 3, results were presented for the relative performance of different search techniques for solving the problem. The results showed that:

1. The weighted Single-Objective GA, Pareto GA, Greedy (Value/Cost Ratio) and NSGA-II can all outperform Random Search (passing the 'sanity check');

2. NSGA-II outperforms the Pareto GA, both in terms of diversity of results and in terms of quality of the results (which dominate all results from the Pareto GA);

3. NSGA-II outperforms the Greedy(Value/Cost Ratio) in terms of convergence in both synthetic and real world data sets;

4. The weighted Single-Objective GA and Greedy (Value/Cost Ratio) can be helpful in finding extreme points on the Pareto front;

5. The ES3 aimed at determining the size of problem for which the MONRP becomes non-trivial. This reveals that the number of requirements is the primary determinant.

The T/FIA addressed the problem of balancing the requirements of today with those of the future. We extended the work on release planning by considering the problem of finding a suitable set of requirements that balances the 'needs for today' against the 'needs for the future'. A multi-objective formulation of the problem was implemented using multi-objective Pareto optimal evolutionary algorithms. Therefore, the decision maker may have the ability to take account of likely known changes in requirements over time and to make the optimal choices of system.

The author reported on experimentation with this formulation to a real world data set from Ericsson. The NSGA-II algorithm has produced the Pareto fronts with a good spread. According to different cost levels, a number of corresponding solutions can be found on the Pareto front. The software engineer can choose alternative ones in different contexts. Moreover, the T/FIA not only provide optimal solutions themselves, but also yield interesting insight of the data sets. The shape of Pareto fronts reflect the correlation between *value for today* and *value for the future*, which is also supported by the statistical analysis carried out.

The traditional method to decide which requirement will be in a product release is prioritisation and ranking of requirements. Each optional requirement is assigned a unique order in a group and then the Greedy strategy is used to prioritise and select the requirements from the best to the worst.

Comparing the search-based Value/Cost trade-off approach to this traditional method, we first focus on the solutions themselves. The Greedy algorithms produced only one solution per run. By contrast, Value/Cost trade-off analysis generated a number

of high quality solutions in one run. Moreover, these two approaches offered different solutions in terms of the quality. The results of empirical studies illustrate that search-based techniques can provide better quality approximations to the Pareto front than the Greedy strategy for the release planning problem.

When asked about what makes a "good" release selection, the requirements engineer may think of more than one objective. This is why we use a multi-objective approach to fit into the context of the problem. Three objectives were introduced in T/FIA to provide flexible and robust solutions both for today and for the future. It also avoids the need to make arbitrary choices of weights to combine several objectives into one fitness function. Our approach is used to capture the trade-offs between the two competing objectives and reveal interesting 'knee points' at which there is a potentially large shift in the trade-off between objectives.

Search-based requirements selection can yield insight. However, these implicit properties are very hard to achieve just by relying on human intuition, experience and communication. This revealed information allows the requirements engineer to be more concerned about these interesting areas and select one or a set of preferred solutions from the Pareto front.

Last but not least, the trade-off analysis results can be used in 'what if?' analyses, for instance:

> "What would we gain if we could securely assign 10% more resources to the project?"

or

> "What would we lose by reducing the project's budget by 20%?"

The requirements engineer can also perform a 'what if?' analysis by building several budgets that each assumes a certain level of stakeholders' satisfaction. The requirements engineer can also specify the result of a certain objective that the project needs, and then discover what range can be achieved by other objectives.

In such situations, a Pareto front is more useful in exploring the outcome of these changes to the scenario, because it captures the entire range of trade-off decisions, rather than fixing on a single point. The traditional method or single objective formulation of the problem lacks the ability to provide such decision aids.

Chapter 5

Requirements Interaction Management

5.1 Introduction

In the release planning for software development, the requirements interdependency relationship is an important element which reflects how requirements interact with each other in a software system. Furthermore, it also directly affects requirements selection activity as well as requirements traceability management, reuse and evolution process. As described in Chapter 2, the RIM consists of a series of activities related to requirements dependencies which are complex and challenging tasks.

In this chapter, RIM will be taken into consideration in the automated requirements selection process for release planning. The study is based on the assumption that the dependence identification activity has been completed. In this chapter, we present the motivation for the study of this problem, redefined fitness functions, experimental studies and results for dependence aware requirements optimisation.

5.2 Motivation

Few previous authors [14, 68, 75] focus on the role of requirements dependencies in the solution space. However, dependencies can have very strong impact on the development process in a typical real world project. Bagnall et al. [14] only considered the *Precedence* dependency type, representing the relationship as a directed, acyclic graph. Its vertices are denoted as individual requirements and its edges directed from one vertex to another are denoted as the *Precedence* dependency between the requirements. Greer and Ruhe extended the work by adding the *And* dependency type together with *Precedence* as the constraints in their EVOLVE model. Franch and Maiden applied the *i** approach to model dependencies for COTS component selection.

In this thesis, the Search-based Requirements Selection Optimisation framework allows for the five most common types of requirements dependencies for the first time. The objectives are to investigate the influences of requirements dependencies on the automated requirements selection process and to validate the ability of the proposed framework to find the optimal balance in the solution space for release planning under different circumstances.

5.3 Fitness Function

In the context of Value/Cost based requirements assignments analysis, the fitness functions used in this chapter are similar to those defined in Section 4.2.1. These dependency factors need to be introduced within the fitness function.

Assume that the set of possible software requirements is denoted by:

$$\Re = \{r_1, \ldots, r_n\}$$

The requirements array R is defined by:

$$R = \begin{Bmatrix} r(1,1) & r(1,2) & \cdots & r(1,i) & \cdots & r(1,n) \\ r(2,1) & r(2,2) & \cdots & r(2,i) & \cdots & r(2,n) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ r(j,1) & r(j,2) & \cdots & r(j,i) & \cdots & r(j,n) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ r(n,1) & r(n,2) & \cdots & r(n,i) & \cdots & r(n,n) \end{Bmatrix}$$

More formally, the dependence constraints that this thesis will consider will be defined, as follows:

And Define an equivalence relation ξ on the requirements array R such that $r(i,j) \in \xi$ means that given requirement r_i is selected, then requirement r_j has to be chosen.

Or Define an equivalence relation φ on the requirements array R such that $r(i,j) \in \varphi$ and $r(j,i) \in \varphi$ means that requirements r_i and r_j are in conflict; only one of r_i, r_j can be selected.

Precedence Define a partial order χ on the requirements array R such that $r(i,j) \in \chi$ means that requirement r_i has to be implemented before requirement r_j.

Value-related Define a partial order ψ on the requirements array R such that $r(i,j) \in \psi$ means that if the requirement r_i is selected, then its inclusion affects the value of requirement r_j for the stakeholder.

Cost-related Define a partial order ω on the requirements array R such that $r(i,j) \in \omega$ means that if the requirement r_i is selected, then its inclusion affects the cost of implementing requirement r_j.

In addition, the relations ξ, φ and χ should satisfy

$$\xi \bigcap \varphi = \varnothing \quad \wedge \quad \xi \bigcap \chi = \varnothing$$

in order to guarantee consistency in the requirements dependency relationship.

Therefore, the fitness function with dependency constraints can be expressed as below:

$$\text{Maximise } f_1(\overrightarrow{x}) = \sum_{i=1}^{n} score_i \cdot x_i$$

$$\text{Maximise } f_2(\overrightarrow{x}) = -\sum_{i=1}^{n} cost_i \cdot x_i$$

subject to

$$x_i = x_j \text{ for all pairs } r(i,j) \in \xi \text{ (And constraints)}$$

$$x_i \neq x_j \quad \vee \quad x_i = x_j = 0 \text{ for all pairs } r(i,j) \in \varphi \text{ (Or constraints)}$$

$$x_i = 1 \wedge x_j = 1 \quad \vee \quad x_i = 1 \wedge x_j = 0 \quad \vee \quad x_i = x_j = 0$$

$$\text{for all pairs } r(i,j) \in \chi \text{ (Precedence constraints)}$$

In terms of Value-related and Cost-related requirements dependencies, they cannot

be transformed from dependencies into constraints. However, the fitness values of a solution could be changed directly, if there exists a Value-related or Cost-related dependency, indicated as follows:

If $x_i = x_j = 1$ for all pairs $r(i,j) \in \psi \Rightarrow$ Update Fitness Value of $f_1(\overrightarrow{x})$

If $x_i = x_j = 1$ for all pairs $r(i,j) \in \omega \Rightarrow$ Update Fitness Value of $f_2(\overrightarrow{x})$

5.4 Experimental Set Up

To assess the likely impact of requirements dependencies on the automated requirements selection process, a set of empirical studies were carried out. This section describes the test data sets used and the search-based algorithm applied to the requirements interaction management.

5.4.1 Data Sets

First of all, the "27 combination random data sets" will be introduced. These are the basis of the data sets we will use in the empirical studies. The "27-random" data sets were generated randomly according to the problem representation. The synthetic test problems were created by assigning random choices for value and cost. The range of costs were from 1 through to 9 inclusive (zero cost is not permitted). The range of values were from 0 to 5 inclusive (zero value is permitted, indicating that the stakeholder places no value on this requirement).

This simulates the situation where a stakeholder ranks the choice of requirements (for value) and the cost is estimated to fall in a range, very low, low, medium,

Table 5.1: 27 combination random data sets

	R_{small}	R_{medium}	R_{large}
C_{small}	$C_s\ R_s\ D_{low}$ $C_s\ R_s\ D_m$ $C_s\ R_s\ D_h$	$C_s\ R_m\ D_{low}$ $C_s\ R_m\ D_m$ $C_s\ R_m\ D_h$	$C_s\ R_l\ D_{low}$ $C_s\ R_l\ D_m$ $C_s\ R_l\ D_h$
C_{median}	$C_m\ R_s\ D_{low}$ $C_m\ R_s\ D_m$ $C_m\ R_s\ D_h$	$C_m\ R_m\ D_{low}$ $C_m\ R_m\ D_m$ $C_m\ R_m\ D_h$	$C_m\ R_l\ D_{low}$ $C_m\ R_l\ D_m$ $C_m\ R_l\ D_h$
C_{large}	$C_l\ R_s\ D_{low}$ $C_l\ R_s\ D_m$ $C_l\ R_s\ D_h$	$C_l\ R_m\ D_{low}$ $C_l\ R_m\ D_m$ $C_l\ R_m\ D_h$	$C_l\ R_l\ D_{low}$ $C_l\ R_l\ D_m$ $C_l\ R_l\ D_h$

Table 5.2: Scale Range of '27-random' data set

	Small	Medium	Large
No. of Stakeholders	2-5	6-20	21-50
No. of Requirements	1-100	101-250	251-600
	Low	Medium	High
Density of Matrix	0.01-0.33	0.34-0.66	0.67-1.00

high, very high. The number of stakeholders and the number of requirements are divided into three situations, namely, small scale, medium scale and large scale; the density of the stakeholder-requirement matrix is defined as low level, medium and high level. Table 5.1 lists the combination of all cases schematically. As can be seen in Table 5.2, the data set divides the range of a variable into a finite number of non-overlapping intervals of unequal width.

Any randomly generated, isolated data set clearly cannot reflect real-life scenarios. We do not seek to use our pseudo random generation of synthetic data as a substitute

for real world data. Rather, we seek to generate synthetic data in order to explore the behaviour of our algorithms in certain well defined scenarios. The use of synthetic data allows us to do this within a laboratory controlled environment. Specifically, we are interested in exploring the way the search responds when the data exhibits a presence or absence of correlation in the data.

As well as helping us to better understand the performance and behaviour of our approach in a controlled manner, this also allows us to shed light on the real world data, comparing results with the synthetic data.

However, there is still missing information in the "27 combination random data sets" model. For example, only uniform distribution is adopted in the generation of random numbers. In the real world, the data structure might be followed by other probability distributions. In addition, the different types of requirement relationships may also have different dependency density levels. All of these factors are likely to impact on the results obtained.

In the empirical studies, we generated four data sets in different scales and densities using the approach to data set generation depicted in Table 5.1 and Table 5.2. We call A, B, C and D data set separately to distinguish four of them.

In the A data set, the scale of requirements number and stakeholders number were chosen to be "medium" and the density of stakeholder-requirement matrix was chosen to be "medium". The parameters of data set were randomly generated during the given scale intervals. That is, the number of requirements is 230, the number of stakeholders is 11 and the density of matrix is 0.53.

Following the same principle, the B, C and D data sets were generated. The scales and densities chosen and the specific parameters created are listed in Table 5.3 and Table 5.4.

Table 5.3: Scale of A, B, C and D data sets

	R_{small}	R_{medium}	R_{large}
C_{small}			**C:** C_s R_l D_m
C_{median}		**A:** C_m R_m D_m	
C_{large}	**B:** C_l R_s D_m		**D:** C_l R_l D_h

Table 5.4: A, B, C and D Data Sets

Data Set	No. of Stakeholders	No. of Requirements	Density of Matrix
A	11	230	0.53
B	34	50	0.39
C	4	258	0.51
D	21	412	0.98

In the four data sets that were generated, all the requirements were initially created to be independent. To introduce the requirements dependency issue, first of all, the introduced relationships among requirements need to be added to in the data set structures. The five two-dimensional arrays $And(i,j)$, $Or(i,j)$, $Pre(i,j)$, $Val(i,j)$ and $Cos(i,j)$ ($1 \leq i \leq n$ and $1 \leq j \leq n$) are defined to present five requirements dependency types respectively:

$$And(i,j), Or(i,j) \; and \; Pre(i,j) \in \{0,1\}$$

$And(i,j) = 1$ && $And(j,i) = 1$ if requirement r_i and r_j have **And** dependency and 0 otherwise; $Or(i,j) = 1$ && $Or(j,i) = 1$ if requirement r_i and r_j have **Or** dependency and 0 otherwise; $Pre(i,j) = 1$ && $Pre(j,i) = 0$ if requirement r_i and r_j have **Precedence** dependency.

The above three dependency arrays are bit arrays which compactly store individual boolean values, as the flags to indicate the relationship among requirements. As each random relationship is created, we check to ensure that the **And**, **Or** and **Precedence** dependence constraints are respected, thereby guaranteeing the generation of a valid instance.

In the $Val(i,j)$ and $Cos(i,j)$ arrays, the values are not 0 or 1, but rather the extent of impact of the *Value* or *Cost* which are expressed as a numerical percentage. $Val(i,j) \neq 0$ if requirements r_i and r_j have a **Value-related** dependency; $Cos(i,j) \neq 0$ if requirements r_i and r_j have a **Cost-related** dependency.

5.4.2 Algorithms

The search algorithms used in this work are NSGA-II [50] and a variation. The results are presented to compare the performance of two algorithms. The variation of NSGA-II here refers to an archive strategy based NSGA-II algorithm, that is, all the non-dominated solutions produced in each generation are collected and archived. Usually, the solutions on the Pareto front in the final generation are considered to be the best.

Keeping the final set of non-dominated solutions is good enough for the general multi-objective work. However, when we take requirements dependencies into account, the selected optimal non-dominated solutions might not respect the dependency constraints and so some solutions might have to be eliminated. This may mean rejecting otherwise 'optimal' solutions in favour of previously considered and otherwise less optimal solutions.

In order to preserve these potential candidate solutions, an archive based variation of the NSGA-II algorithm is proposed to optimise the solutions (maintaining diversity and quantity of the solutions) based on constraints. This will be discussed in detail later on.

The parameter tuning options of the NSGA-II algorithm we used here are the same as those described as in Chapter 4 Section 4.2.2.3. In the archive based NSGA-II algorithm, the total capacity of the archives was set to 500.

5.5 Empirical Studies and Results

This section presents the experiments carried out, which were designed to investigate the results under the influence of requirement dependencies and to compare the

performance of two search algorithms.

There are two types of empirical studies: a Dependency Impact Study (DIS) and a Scale Study (SS). Data set A is used for DIS and data sets B, C and D are used for SS.

In DIS, the experiment is designed for the purpose of evaluating the impacts of five different dependency types on the requirements selection process. Data set A is used throughout the DIS experiment in order to set up a uniform baseline for comparison. Three experiments were conducted in the DIS, described as follows:

1. Applying the NSGA-II algorithm to data set A with and without dependencies separately, in order to carry out the comparison of the results of each dependency type (five types individually).

2. Applying the NSGA-II and archive based NSGA-II to data set A aiming to compare the performances of two algorithms under the dependency constraints (five types individually).

3. Considering dependence relationship as a whole to seek to investigate the difference among the solutions generated by the two algorithms.

In the SS, we report results concerning the performance of the two algorithms as the data sets increase in size. There are three data sets B, C and D with the number of stakeholders ranging from 4 to 34 and the number of requirements ranging from 50 to 412.

In both studies, the five dependency types can be divided into two categories: fitness-invariant dependency (And, Or and Precedence) and fitness-affecting dependency (Value-related and Cost-related). Therefore, we will discuss the two scenarios separately in each study. In addition, the same dependency density levels were used

for all five dependencies. That is, we assume that they are equally common in the requirements correlations.

5.5.1 Dependency Impact Study

5.5.1.1 Aims

Three goals need to be achieved in DIS listed as follows:

1. The Pareto front should cover the maximum number of different situations and provide a set of well distributed solutions.
2. The solutions contained in the Pareto front should be as close as possible to the optimal Pareto front of the problem.
3. The solutions are required to pass the evaluation without failure to meet constraints.

5.5.1.2 And, Or and Precedence

In the first part of the section, we present the results of applying the NSGA-II and the archive based NSGA-II algorithms to handle **And**, **Or** and **Precedence** requirements dependencies. The results generated by the standard NSGA-II algorithm are shown in Figures 5.1, 5.2 and 5.3; and the results from the archive based NSGA-II algorithm are shown in Figures 5.4, 5.5 and 5.6 separately.

The '+', '◯' and '∗' symbols plotted in the figures denote the final non-dominated solutions found. Each solution represents a subset of requirements selected. The '+' symbol represents the solutions found without regard to requirement dependencies.

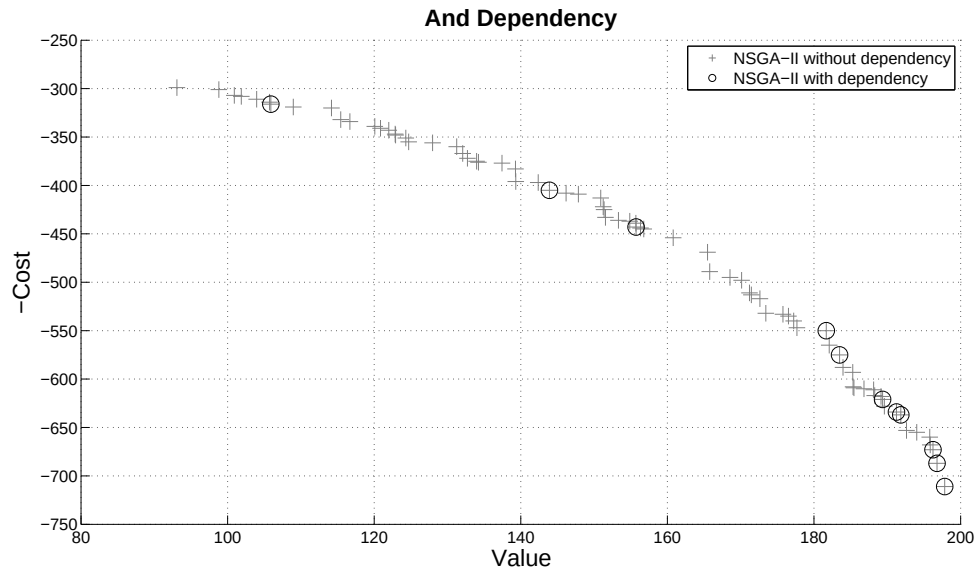


Figure 5.1: Results Comparison: with and without *And* dependency

They are also marked in grey colour to distinguish them from the others (which do take account of dependencies).

In Figures 5.1, 5.2 and 5.3, we observe that the shapes of Pareto fronts, consisting of a number of grey '+' symbols, are the same. These are solutions generated by the NSGA-II algorithm without consideration of requirements dependency relationship and so they are expected to be identical. They are used as the baseline to explore the impact of three types of dependencies on the requirements selection results.

We illustrate the results in Figure 5.1 when the *And* dependency relationships exists among the requirements, which are depicted by '◯' symbols. It can be seen from the graph, all the '◯' solutions still fall on the Pareto front composed of grey the '+' symbols. However, there is a large decrease in the number of '◯' solutions compared to the number of '+' solutions. In other words, a few solutions survived and the rest were eliminated (from the selection) because of the failure to meet dependency constraints. Another obvious observation drawn from this graph is that the distribution of '◯' solutions is not as smooth and uniform as the '+'

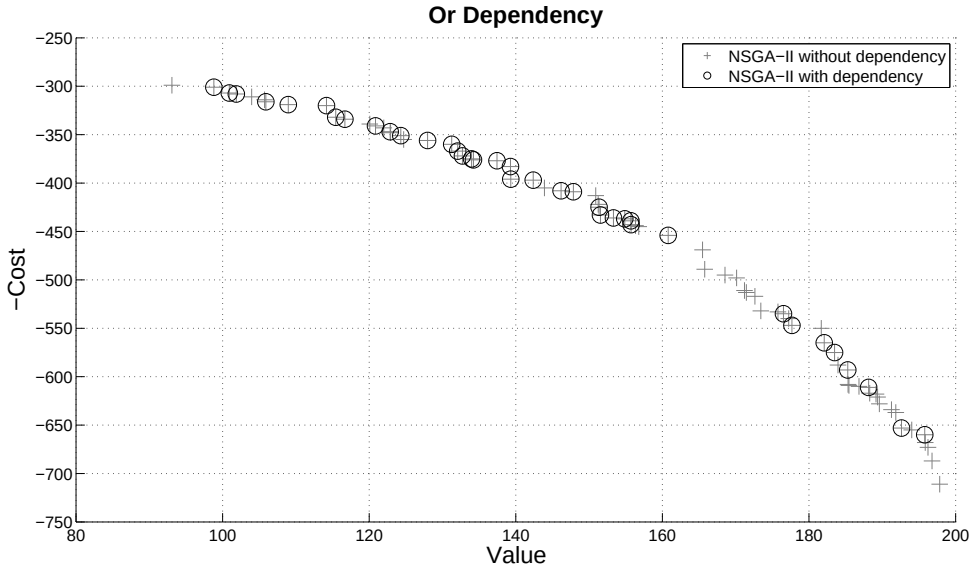


Figure 5.2: Results Comparison: with and without *Or* dependency

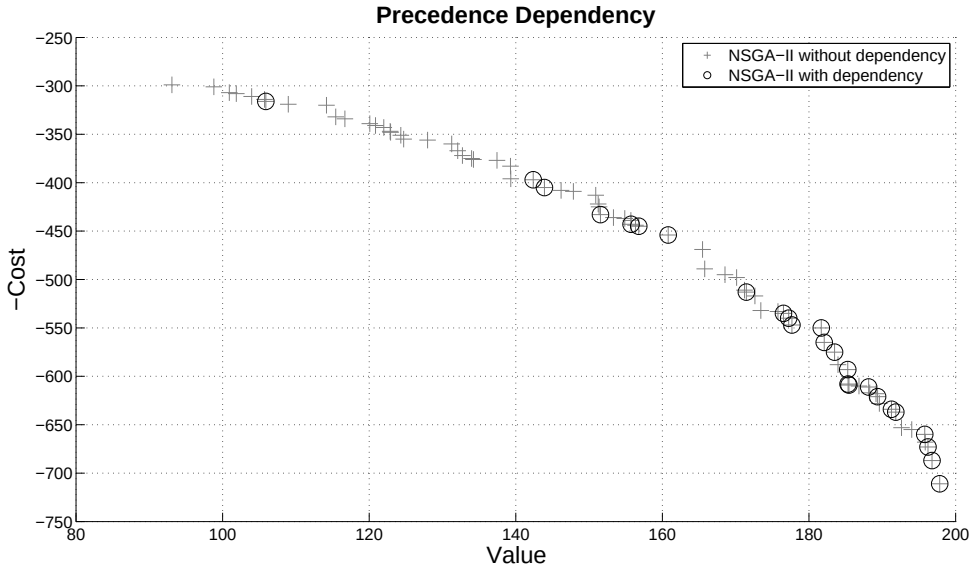


Figure 5.3: Results Comparison: with and without *Precedence* dependency

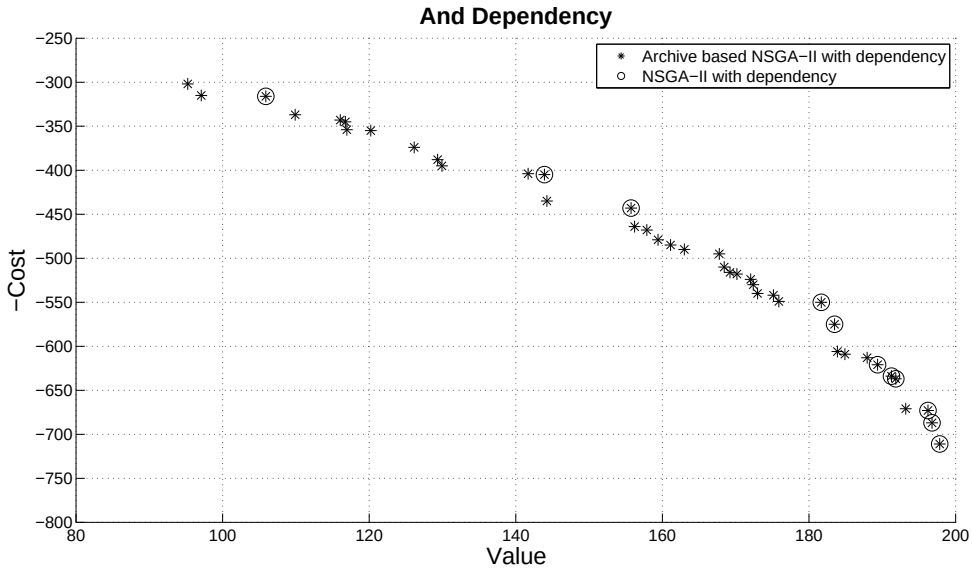


Figure 5.4: Results Comparison: Original and Degenerated Pareto fronts with *And* dependency

solutions. That is, a certain number of big or small gaps exist among them. These two observations indicate that the algorithm can neither provide good solutions in quantity nor maintain a good diversity (quality) in the Pareto front under *And* dependency constraints.

Compared to the results of *Or* and *Precedence* dependencies in Figures 5.2 and 5.3, the downward trends in the number of '◯' solutions are roughly the same, but the extent is different. Similar observations can be made from the two figures that the results show a slight decrease in the number of the solutions. Moreover, the distribution is more continuous, exhibiting a few small gaps among the solutions.

In conclusion, the shapes of Pareto fronts (results) are affected by the different dependency constraints to a different extent. The *And* dependency problem appears to denote a tighter constraint than the *Or* and *Precedence* dependencies for search-based requirements optimisation. The latter two denote problems for which it is relatively easy to find the solutions that satisfy the constraints.

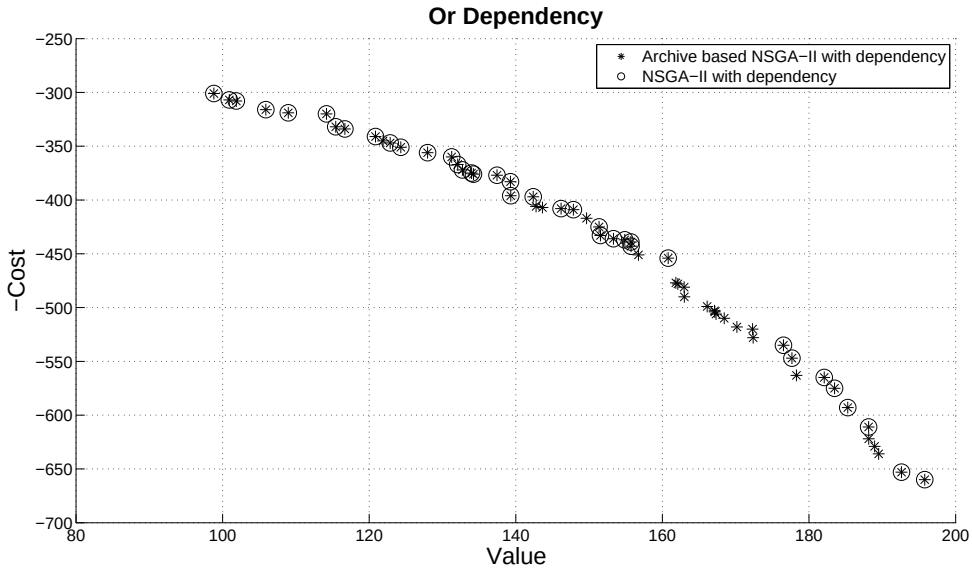


Figure 5.5: Results Comparison: Original and Degenerated Pareto fronts with *Or* dependency

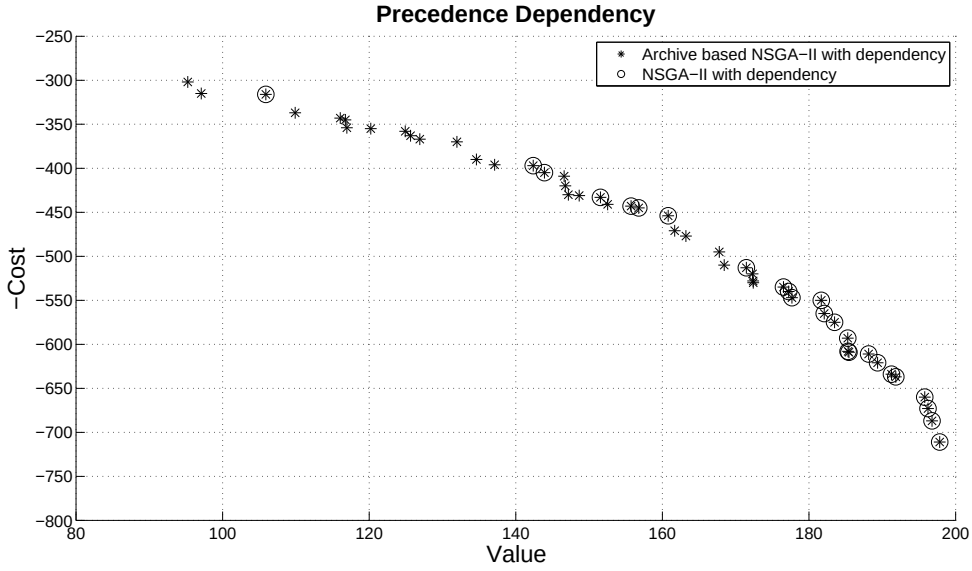


Figure 5.6: Results Comparison: Original and Degenerated Pareto fronts with *Precedence* dependency

To explore these findings in more detail, we designed a more robust adaptive algorithm for both tight and loose constraints, that is the archive based NSGA-II algorithm. The results for three types of constraints are shown in Figures 5.4, 5.5 and 5.6 respectively. The '$*$' denotes the solution generated by the archive based NSGA-II algorithm and the '$\bigcirc$' by NSGA-II.

From the Figure 5.4 we can see, the archive based '$*$' solutions actually reach all the points on the previous '$\bigcirc$' Pareto front, sharing all the common points generated by NSGA-II. The Pareto front in this problem is orientated towards the upper right. The improved algorithm provided a *degenerated* '$*$' Pareto front.

The *degenerated* Pareto front means that the '$*$' front generated seems to become worse when compared to the grey '$+$' front, but it discovers a larger number of good solutions to fill the gaps while meeting the constraints. That is, the diversity of solutions is significantly improved and the number of solutions on the Pareto front is also increased. The algorithm generated similar results when dealing with *Or* and *Precedence* dependency constraints, as illustrated in Figures 5.5 and 5.6.

In this way, the new variation – the archive based NSGA-II search technique – can provide stable and fruitful solutions, which are not merely 'good enough' but also 'robust enough' under the strict constraints that characterise the problem.

Finally, all three dependencies were taken into consideration to access their overall combined impact. In the Figure 5.7, the '$\bigcirc$' solutions denote the final results that satisfy all the dependencies constraints. When put *And*, *Or* and *Precedence* dependencies together, the constraints in this case become much tighter. It is easy to see that very few '$\bigcirc$' solutions are left on the Pareto front based on the NSGA-II algorithm. By contrast, the Figure 5.8 shows a smooth, relatively non-interrupted Pareto front, consisting of '$*$' solutions, generated by archive based NSGA-II.

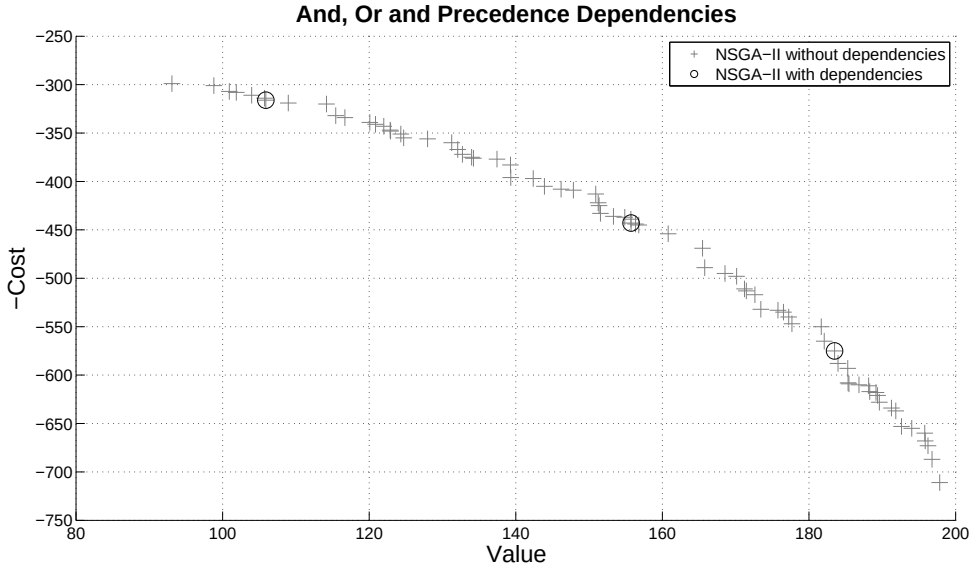


Figure 5.7: Results Comparison: with and without *And*, *Or* and *Precedence* dependencies

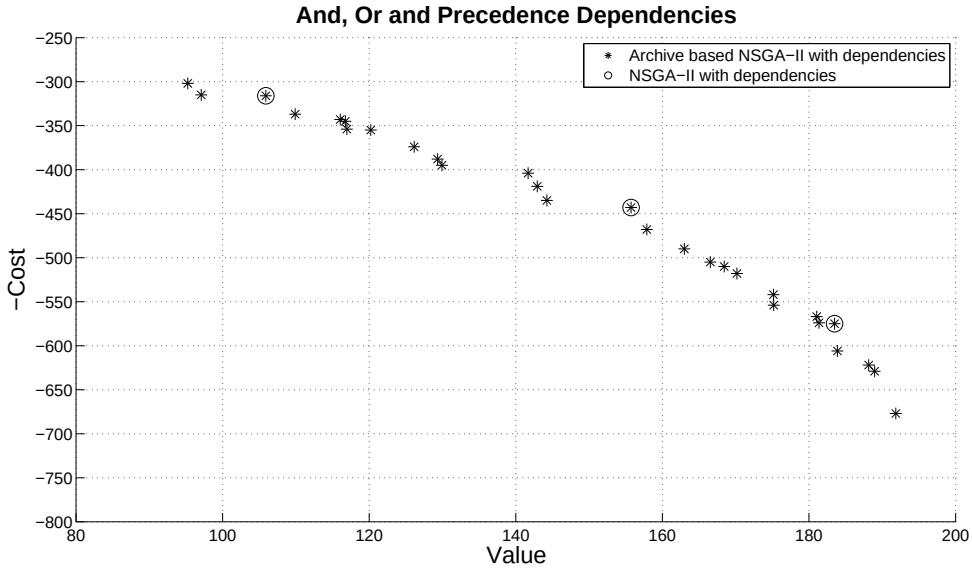


Figure 5.8: Results Comparison: Original and Degenerated Pareto fronts with *And*, *Or* and *Precedence* dependencies

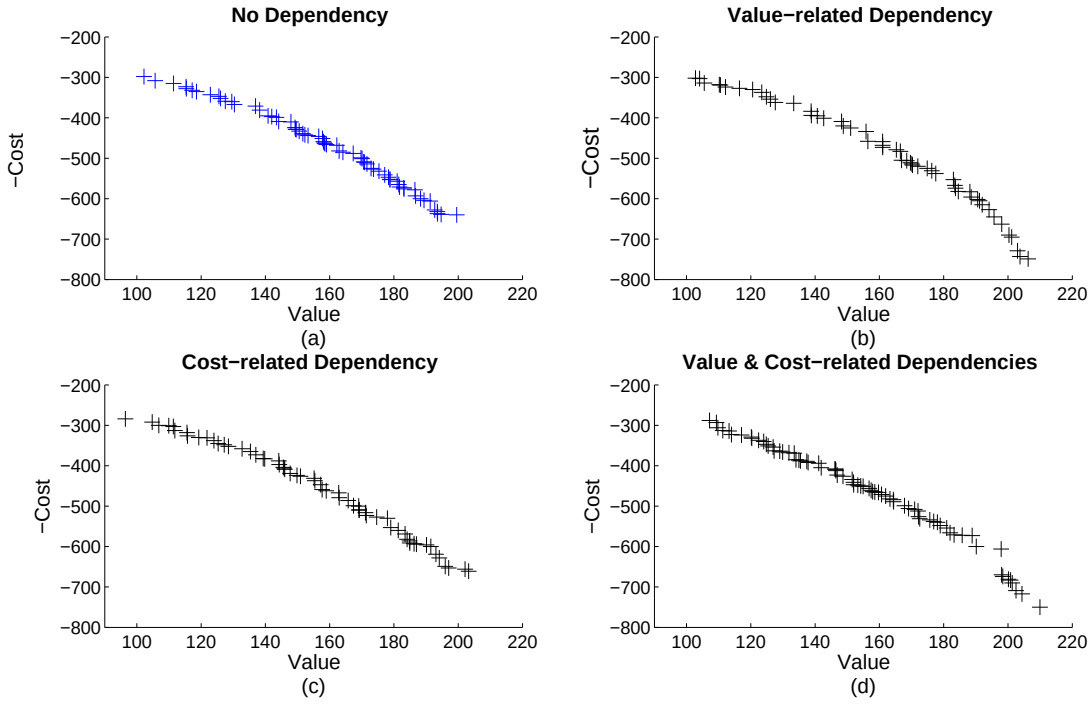


Figure 5.9: Results Comparison: Original and Changed Pareto front with Value-related and Cost-related dependencies

5.5.1.3 Value-related and Cost-related

In this section we focus on the last two types of requirement dependencies: *Value-related* and *Cost-related*. These two impose no constraint on the fitness function but have direct influence on the fitness value.

The results are illustrated in Figure 5.9. There are four subgraphs in the figure: (a) is the original Pareto front without dependency generated by NSGA-II; (b) and (c) show the results under *Value-related* and *Cost-related* dependencies respectively; (d) presents the changed Pareto front when combining these two dependencies together.

We observe that the shape of the four Pareto fronts produced are different. They are not like the previous results of the first three dependencies, such as eliminating the solutions on the original Pareto fronts or degenerate the optimal fronts to explore

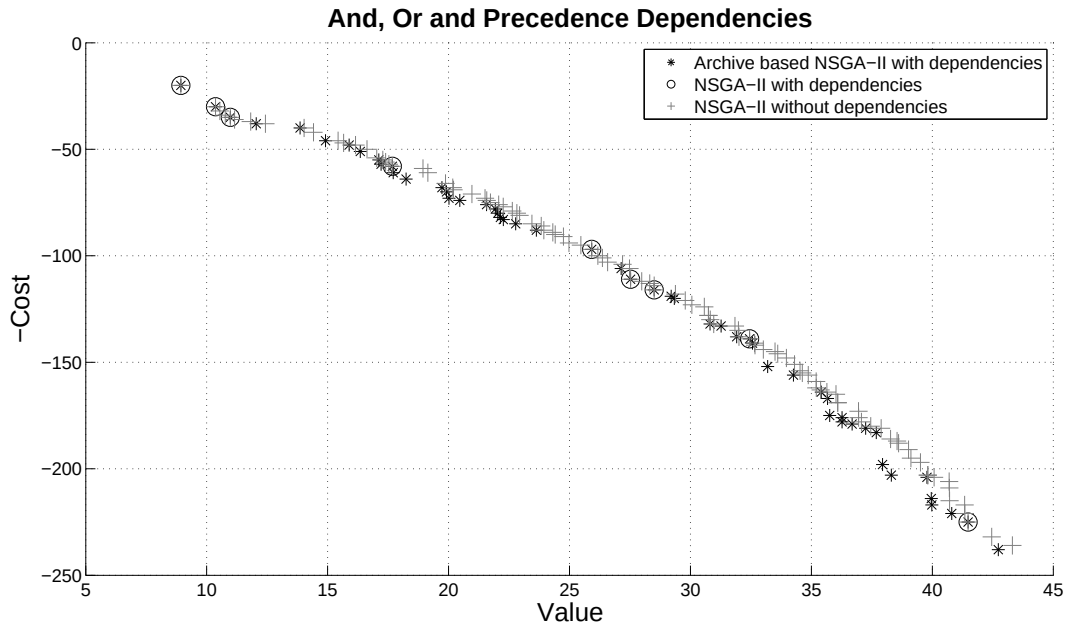


Figure 5.10: Results for Data Set B: 34 Stakeholders, 50 Requirements

the compromise solutions. These kinds of relationship among the requirements can directly contribute to an increase or a decrease in the fitness values obtained for a selected solution. In this way, the shape of the Pareto front is changed more than once without dependency.

5.5.2 Scale Study

In this section, we report on the second empirical study – the Scale Study. The results are presented in the Figures 5.10, 5.11 and 5.12. As described at the beginning of Section 5.5, the techniques were applied to three data sets B, C and D generated from the smaller scale to a relatively larger one in terms of the number of stakeholders involved and the number of requirements fulfilled. The details are listed in Table 5.3 and Table 5.4.

In this study, all three dependency constraints are considered together. The results

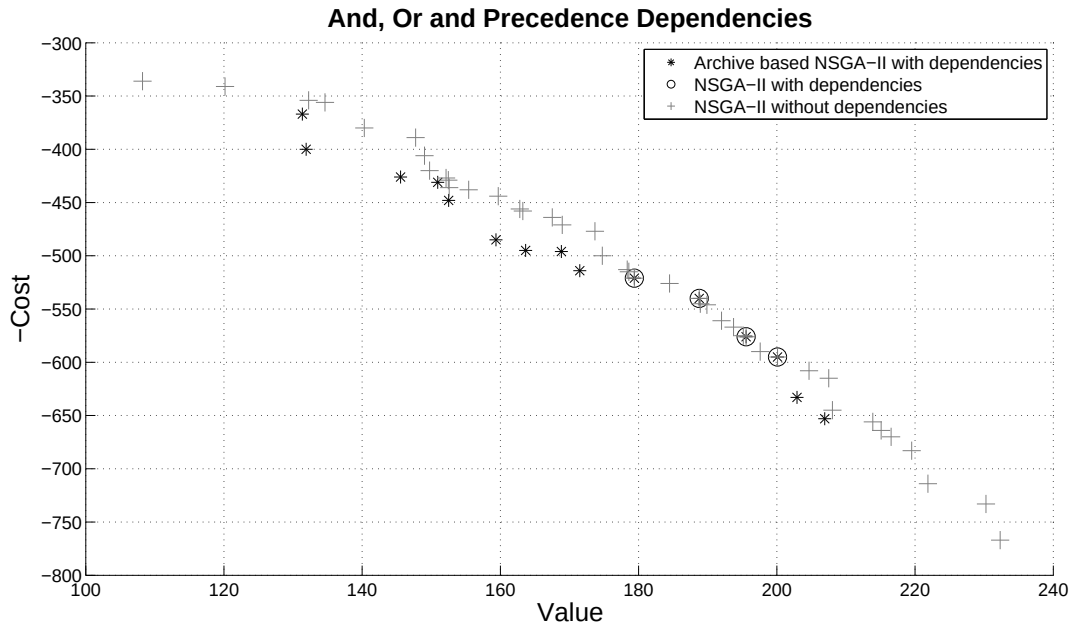


Figure 5.11: Results for Data Set C: 4 Stakeholders, 258 Requirements

are plotted in one graph for each data set. In the figures, the grey Pareto front, consisting of a number of '+' solutions, denotes the results without handling dependencies generated by the NSGA-II algorithm; the '◯' solutions are the survivors after selection for meeting the constraint; the '∗' solutions which are produced by the archive based NSGA-II algorithm constitute the *degenerated* Pareto front.

When the problem is gradually scaled up, from the graphs we can see that the number of the '◯' solutions consistently and rapidly decreases. As illustrated in Figure 5.12, the '◯' solutions have a poor spread over the Pareto front. By contrast, the '∗' solutions still fill the gaps among the '◯' solutions and produce a relatively smooth and continuous Pareto front.

These three data sets B, C and D are considered using the same proportion of possible dependencies (6% of the number of requirements). Another observation from the three figures is that the distance between the original '+' Pareto front and the *degenerated* '∗' Pareto front is wider in Figure 5.12 than in Figure 5.10. The

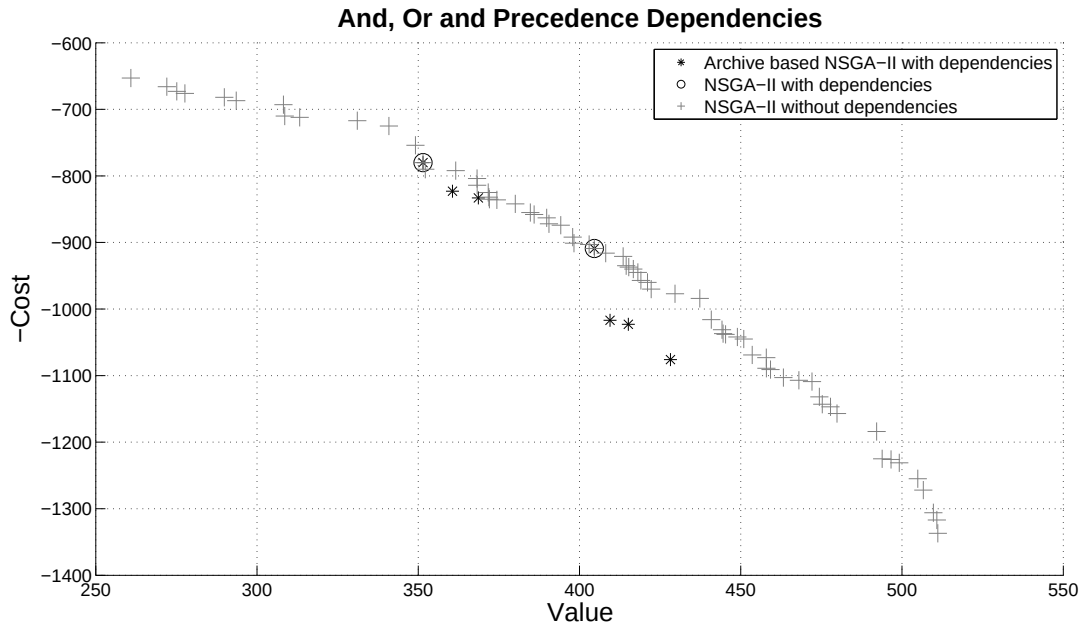


Figure 5.12: Results for Data Set D: 21 Stakeholders, 412 Requirements

Pareto fronts move towards the lower left part of the solution space, in order to find near-optimal solutions that have a good spread as well as having (more than) enough candidate solutions.

5.6 Summary

This chapter presented Requirements Interaction Management (RIM) and has taken RIM into consideration in the automated requirements selection process for the release planning problem. Five basic requirement dependencies were introduced. The first three types were considered to be constraints within the fitness functions; the latter two directly involved in the performance.

A "27 combination random data sets" model was generated to develop a procedure in order to better approximate real world situations. Two variable factors were considered in the data generation model, one is the different levels of data set scales

which are related to the number of requirements and the number of stakeholders, the other is the density of the data sets.

To simulate the release planning selection process under requirements dependencies, two empirical studies were carried out, that is, the Dependency Impact Study (DIS) which is designed to investigate the influences of five different dependency types and the Scale Study (SS) which concerns the performance of the two search techniques when the data sets scale up.

The same data set was used throughout the DIS experiment aiming to set up a uniform baseline for influence comparison. The results of the empirical studies illustrated that the *And* dependency appears to denote a tighter constraint than the *Or* and *Precedence* dependencies for search-based requirements optimisation. When all three dependencies were taken into consideration to access their overall combined impact, the constraints in this case became much tighter. For *Value-related* and *Cost-related* dependencies, they directly contributed to an increase or a decrease in the fitness values and further changed the shape of the Pareto front.

In SS, three data sets from the smaller scale to relatively larger one were applied. The results showed that Archive based NSGA-II could produce a smooth, relatively non-interrupted Pareto front comparing with NSGA-II. When the data set was gradually scaled up, the number of solutions generated by the latter consistently and rapidly decreases. Instead, Archive based NSGA-II could still find better solutions both in diversity and quantity.

RIM is of vital importance from a software release planning point of view. For instance, the certain optional requirements can be put into one release or be separated to several releases according to their dependency relationships in order to save implementation cost and increase revenue.

Aided by search-based automated RIM, the requirements engineer can faster and more easily address this problem. For any non-trivial problem, many factors need to be considered in the requirements selection process. It is always important to look at the requirements from different perspectives. Unlike human-based search, automated search techniques carry with them no bias. They automatically scour the search space for solutions that best fit the (stated) human assumptions in the fitness function.

Chapter 6

Multi-Stakeholder Requirements Analysis and Optimisation

6.1 Introduction

Requirements Engineering for multiple stakeholders, each of whom have competing and often conflicting priorities, suggest a natural, alternative formation, in which each stakeholder's needs are treated as a separate objective. This chapter introduces the search-based techniques for automated analysis of the problem of balancing requirements assignments where there are several stakeholders, each with their own view of the importance of different requirement sets, in order to aid negotiation and mediation.

There are two main aspects to the research. The first is concerned with a tensioning analysis to maximise requirements satisfaction for each stakeholder. The second is a fairness analysis to seek a reasonable way to measure and balance requirement fulfilment between the stakeholders.

6.2 Tensioning Analysis

This section presents the concept of internal tensioning among multiple stakeholders in requirements analysis and optimisation, treating each stakeholder as a separate objective. The section reports on the experiments using two different multi-objective evolutionary optimisation algorithms with real world data sets as well as synthetic data sets. This empirical validation includes a statistical analysis of the performance of the two algorithms. This section also shows how (animated and traditional) Kiviat diagrams can be used to visualise the tensions between the stakeholders' competing requirements, thereby presenting the results in an intuitive manner to the decision maker.

6.2.1 Motivation

The multi-stakeholder scenario is increasingly important. Often the requirements engineer has to find a set of requirements that reflect the needs of several different stakeholders, while remaining within budget. In many development organisations, there may be many stakeholders with their own varied even conflicting views on the sets of requirements to be prioritised.

The problem of balancing such competing and conflicting concerns is beyond the realm and reach of automation, since such problems inherently rest upon and produce subjective, vague and even ill-defined human assessments. This study aims to demonstrate that there is an important and valuable role to be played by Search Based Software Engineering in such complex scenarios, in which multiple competing and conflicting human objectives have to be balanced by a non-technical decision maker.

It is important that the presentation of the results of optimisation are presented to the decision maker in a form that is intuitive and free from technical detail. Many decision makers are not, themselves, technical experts. Rather, they are often managers and business analysts, who may rely upon automated decision making support, but cannot be expected to familiarise themselves with algorithmic details. Though we use sophisticated multi-objective evolutionary algorithms to search for near optimal solutions among the enormous sets of potential solutions, our approach requires no familiarity with these techniques on the part of the decision maker.

Sometimes, satisfying one stakeholder can only be achieved at the expense of failing to satisfy another. We call this the 'tensioning' between these stakeholders and visualise it using the Kiviat diagram. In this diagram, visually, we may consider one stakeholder 'pulling' the closed polygon solution towards their choice of requirements, creating this 'tension'.

While these are sensible choices, there has been no work on the most natural choice of objectives; the one in which each stakeholder's requirements are considered as an objective in their own right. In this formulation, there may be many stakeholders, each with their own requirements and therefore an arbitrary number of objectives.

Using Pareto optimal search it becomes possible to explore precisely the *extent* to which it is possible to satisfy "all of the people all of the time". Of course, this is unlikely to be completely achievable. The algorithm, however, attempts to produce a set of non-dominated solutions that are as close as the stakeholders' prioritisation will allow to this ideal situation.

6.2.2 Fitness Function

In the tensioning analysis among multiple stakeholders, each stakeholder is taken into consideration as a separate objective in order to maximise each stakeholder's satisfaction. That is, the number of objectives in the problem is the same as the number of stakeholders. We treat *cost* as a budgetary constraint. This allows us to:

1. Explore the relationship between stakeholders and resources. We can understand the extent to which requirement assignments meet stakeholder expectations under varying budget allocation conditions.

2. Analyse the internal tensions between the stakeholders; which stakeholder can easily be satisfied? Whose requirements are hard to fulfil? When does pleasing one stakeholder entail displeasing another?

Evolutionary multi-criteria optimisation has traditionally concentrated on problems comprised of two or three objectives. Our formulation comprises of a relatively large number of objectives. Such problems pose new challenges for algorithm design, visualisation and implementation. In multi-objective evolutionary search, the populations are likely to be largely composed of non-dominated solutions.

The following objective function is used for maximising the total value to each stakeholder, expressed as a percentage. For each stakeholder c_j, we seek to maximise their expected satisfaction subject to the available budget. More formally,

$$\text{Maximise} \qquad \frac{\sum_{i=1}^{n} value(r_i, c_j) \cdot x_i}{\sum_{r \in R_j} value(r, c_j)}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} cost_i \leq B, \qquad B > 0$$

In the rest of the thesis we refer to this term to be maximised as "each stakeholder's satisfaction".

In the fitness function, all the stakeholders are treated as equals, that is, the stakeholder weight factor $w_1 = w_2 = \ldots = w_m$.

The problem is to select a subset of the stakeholders' requirements which results in the maximum percentage of the requirements' *value* that each stakeholder expects, while falling within the budgetary constraint. The fitness value of the non-dominated solutions on the Pareto front can be visualised by the Kiviat diagrams [108, 109].

6.2.3 Experimental Set Up

This section describes the test data sets used to compare the performance of the NSGA-II algorithm with the Two-Archive and Random Search algorithms applied in this study.

6.2.3.1 Data Sets Used

There are three data sets used in the experiments to perform multi-stakeholder analysis. The first data set is the "27 combination random data sets" which has been described in Chapter 5, Section 5.4.1. The details can be found in Table 5.1 and Table 5.2. In the empirical studies, we generated 27 variations (data sets) in different scales and densities followed by the rules described in "27-random".

The second data set is provided by Motorola [15] as we used in Chapter 4, Section 4.2.4. The data set concerns a set of 35 requirements for hand held communication devices and 4 stakeholders who are four mobile telephony service providers.

The third data set is taken from Greer and Ruhe [75]. The Greer and Ruhe data set has 5 stakeholders and 20 requirements. Each requirement was assigned a specific value by a stakeholder from 1 to 5, listed in Table 6.1. The Greer and Ruhe data set does not contain information about the cost of each requirement. For the purpose of feeding this useful industrial data into our algorithm, the cost of the requirements were generated randomly within the range from 10 to 1100, following a Gaussian distribution.

Table 6.1: Data Set taken from Greer and Ruhe [75]

	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

6.2.3.2 Algorithmic Tuning

Each algorithm was executed 20 times for each data set to allow for the application of statistical tests for significance of algorithmic differences. In addition, there are 27 variations in the first data set, so each algorithm was run 20 times for each variation

and the data subset was regenerated randomly to produce a unique data set for each run. In total 1.7×10^7 fitness evaluations were computed. Each of these denotes a possible assignment of requirements.

The parameter tuning options of the NSGA-II algorithm we used here are the same as those described in Chapter 4, Section 4.2.2.3. In the Two-Archive algorithm, the total capacity of the archives was set to 500. The algorithm selects parents from both archives to the mating pool. An archive was chosen with a probability that was set to 0.8. The Random Search technique was given the same number of fitness evaluations as the other algorithms in order to provide a lower bound benchmark for measuring the other algorithms' performance.

6.2.4 Results and Analysis

This section reports results of the experiments from synthetic and real data sets described in Section 6.2.3.

6.2.4.1 Results from Random Data Sets

In the '27-random' data set, the number of stakeholders, the number of requirements and the density of the stakeholder-requirement matrix are the three determining factors used to generate each subset of random data. Table 5.2 shows the variation range of each factor. The information contained in the data set from Motorola was used as a reference and the maximum number of requirements was set to 600.

The results are shown in Table 6.2 and Table 6.3. For the '27-random' data set, 540 (27×20) runs were executed in the empirical study. The distances from the Pareto front generated by each algorithm to the reference Pareto front, namely, the value

Table 6.2: Rank Order for Convergence

	winner	runner up	loser
Random Search	0%	0%	100%
Two-Archive	95.19%	4.81%	0%
NSGA-II	7.04%	92.96%	0%

Table 6.3: Solutions on the Reference front

Random Search	Two-Archive	NSGA-II
2.68%	94.57%	38.25%

of convergence were calculated. The smaller the value of the distance, the better the convergence of the algorithm. The three algorithms were ordered by the value of the convergence in each run. The cumulative total number of orderings for each algorithm was recorded. In other words, we sought to understand how many times a specific algorithm was ranked in a specific position in the 540 runs respectively.

Table 6.2 shows the proportion of each of the three algorithms in the rank positions: winner, runner up and loser. The Two-Archive algorithm was in the first rank in more than 95% of the cases. It performs the best in the three algorithms overall.

At the same time, the number of the solutions on the Pareto front of each algorithm and the number of the solutions on the reference Pareto front were also recorded to calculate the proportion of the non-dominated solutions on the reference front. As shown in the Table 6.3, a substantial proportion of the solutions on the reference front are from the results of the Two-Archive algorithm.

6.2.4.2 Results from Real Data Sets

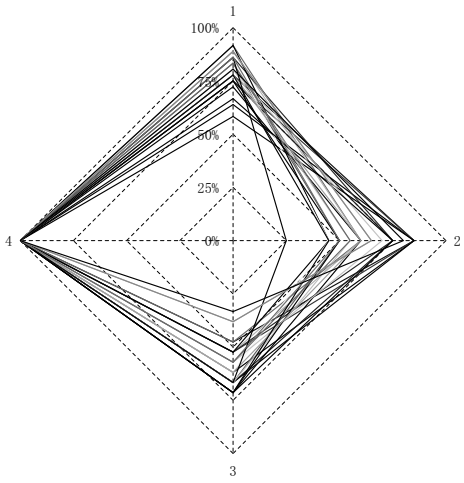
The aim of the empirical studies is to compare the performance of the algorithms, and to explore the effect of different resource limitations on conflicting objectives. We considered budgets increasing in steps of 5% from 0% to 100%, where a budget of x% means x% of the cost of all requirements are provided. For statistical investigation of the performance of the algorithms we used three budget levels: 30%, 50% and 70%.

Figures 6.1 and 6.2 present Kiviat diagrams that address the two research questions proposed in Section 6.2.2. From the Kiviat diagrams depicted in the figures, the number of axes indicates the number of the stakeholders. For example, in Figure 6.1, there are four stakeholders, so four axes. This is also the number of the objectives in the Multi-Stakeholder Objective Formulation. Each axis also, therefore, represents an objective (the degree to which the corresponding stakeholder is satisfied as a percentage). The intersection points of a closed solid polygon to the axis in the Kiviat diagrams denote the fitness values of the non-dominated solutions.

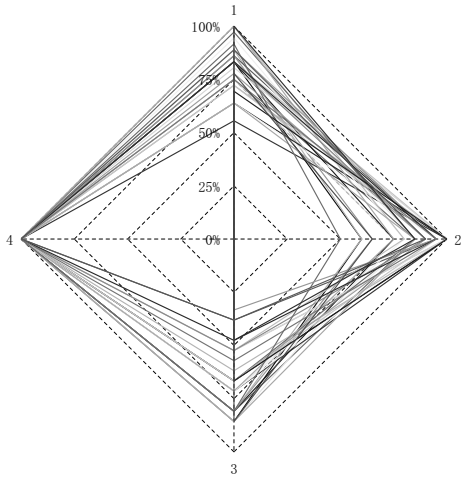
Animated versions of these Kiviat diagrams show the evolution of stakeholders' satisfaction as the budget allowance increases from 0 to 100%. The animations provide a vivid dynamic view of the way in which stakeholder satisfaction improves unevenly for each stakeholder as the budget constraints become more relaxed.

In the thesis we can only show 'snap shots' of the animations at 3 illustrative choices for budget. Figure 6.1 (a), (b) and (c) show the results of the data set from Motorola with 30%, 50% and 70% of resource limitations respectively. The Figure 6.2 (a), (b) and (c) show the results of the Greer and Ruhe data set.

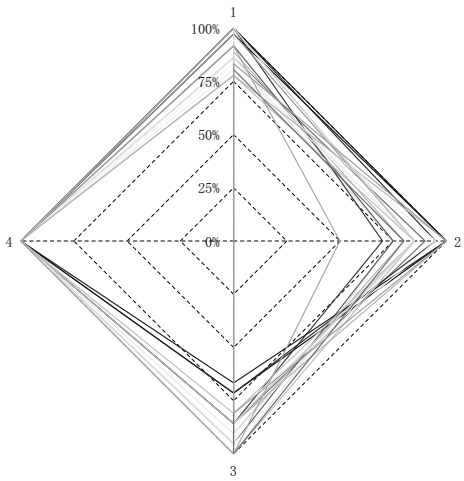
The internal tensioning between the stakeholders is shown as some stakeholders' demands are easy to satisfy. For example, stakeholder 4 in the Motorola data set,



(**a**) Motorola Data Set: 4 stakeholders; 35 requirements
30% resource limitation

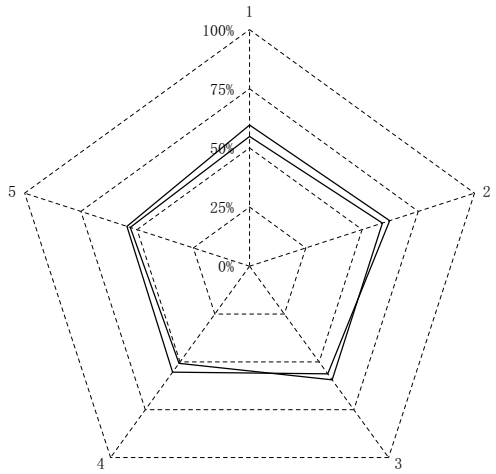


(**b**) Motorola Data Set: 4 stakeholders; 35 requirements
50% resource limitation

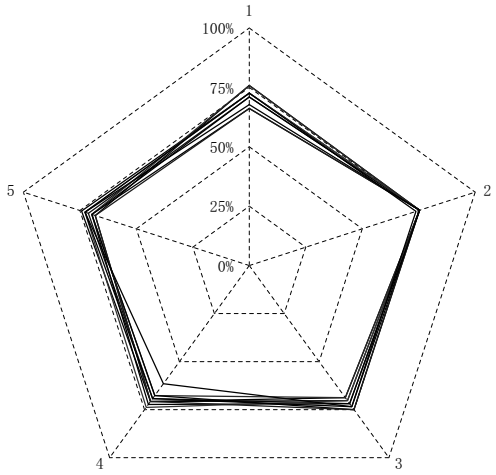


(**c**) Motorola Data Set: 4 stakeholders; 35 requirements
70% resource limitation

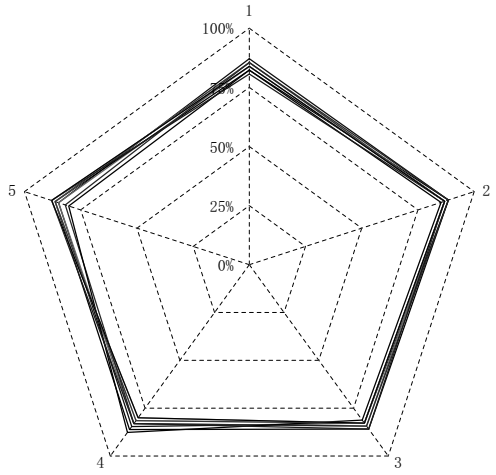
Figure 6.1: Kiviat diagrams for illustrative budget values for Motorola Data Set



(**a**) Greer and Ruhe Data Set: 5 stakeholders; 20 requirements
30% resource limitation



(**b**) Greer and Ruhe Data Set: 5 stakeholders; 20 requirements
50% resource limitation



(**c**) Greer and Ruhe Data Set: 5 stakeholders; 20 requirements
70% resource limitation

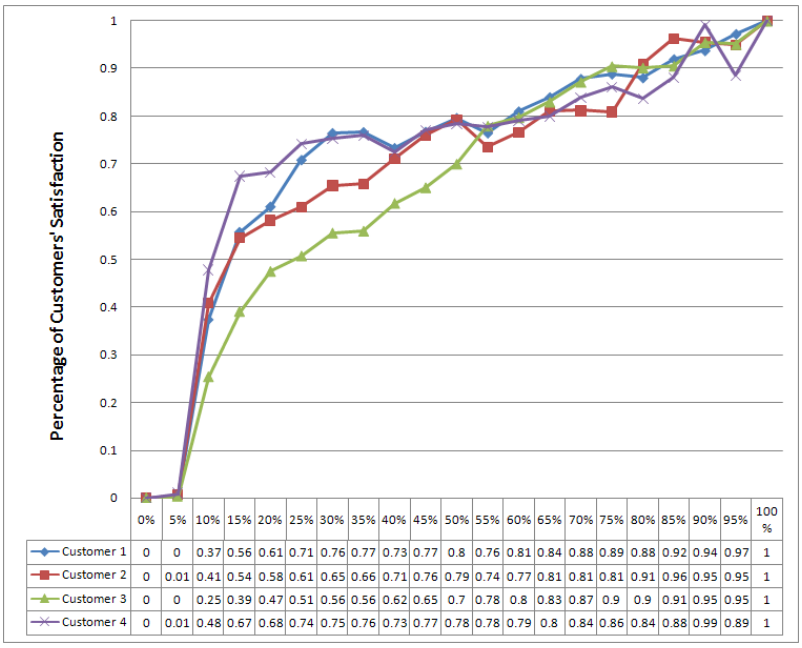
Figure 6.2: Kiviat diagrams for illustrative budget values for Greer and Ruhe Data Set

whose fitness values can quickly converge to 100% regardless of the satisfaction of the other stakeholders. By contrast, stakeholder 3 is relatively hard to satisfy, while maintaining the other stakeholders' satisfaction. The results from the Greer and Ruhe data set show that there is comparatively little tension between those stakeholders.

Figure 6.3 (a) and (b) provide the plots of the growth in stakeholder satisfaction with increasing budgets, which are the results of the data sets from Motorola and Greer and Ruhe. The budgetary resource has been shown from 0% to 100% and the step value is 5%. So there are 21 quantified budgetary levels in total. Every stakeholder's average satisfaction in 20 runs was calculated in each budgetary resource level.

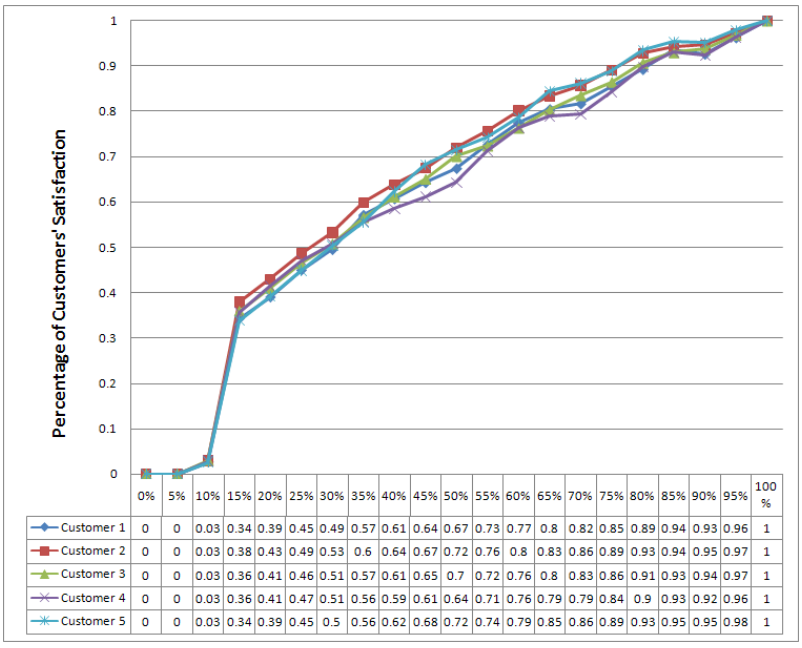
From the figures we can see, the stakeholders' satisfaction began with 0% where there were no available resources and finally all of them converged to 100% when the budgetary resource level reached 100%. There is a remarkable difference in the growth trend of stakeholders' satisfaction between Figure 6.3 (a) and (b). The four stakeholders illustrated in Figure 6.3 (a) have very different satisfaction degrees for each resource level. Such as, the percentages of satisfaction of stakeholder 3 (represented by ▲) are the lowest when the budgetary resource levels are between 5% and 55%. The other three stakeholders also have greater variation in satisfaction degree compared to the results for the stakeholders in the Greer and Ruhe data set in Figure 6.3 (b). The five stakeholders' satisfactions show a relatively consistent and stable growth trend.

The ANOVA statistical analysis technique was used to analyse the diversity and the convergence of the results for the two sets statistically. The null hypothesis is that all three algorithms have the same performance. Rejection of the null hypothesis by ANOVA analysis can, however, only tell us whether the three algorithms' performance were significantly different to one another. To explore further, to see



	0%	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100 %
Customer 1	0	0	0.37	0.56	0.61	0.71	0.76	0.77	0.73	0.77	0.8	0.76	0.81	0.84	0.88	0.89	0.88	0.92	0.94	0.97	1
Customer 2	0	0.01	0.41	0.54	0.58	0.61	0.65	0.66	0.71	0.76	0.79	0.74	0.77	0.81	0.81	0.81	0.91	0.96	0.95	0.95	1
Customer 3	0	0	0.25	0.39	0.47	0.51	0.56	0.56	0.62	0.65	0.7	0.78	0.8	0.83	0.87	0.9	0.9	0.91	0.95	0.95	1
Customer 4	0	0.01	0.48	0.67	0.68	0.74	0.75	0.76	0.73	0.77	0.78	0.78	0.79	0.8	0.84	0.86	0.84	0.88	0.99	0.89	1

(**a**) Motorola Data Set



	0%	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100 %
Customer 1	0	0	0.03	0.34	0.39	0.45	0.49	0.57	0.61	0.64	0.67	0.73	0.77	0.8	0.82	0.85	0.89	0.94	0.93	0.96	1
Customer 2	0	0	0.03	0.38	0.43	0.49	0.53	0.6	0.64	0.67	0.72	0.76	0.8	0.83	0.86	0.89	0.93	0.94	0.95	0.97	1
Customer 3	0	0	0.03	0.36	0.41	0.46	0.51	0.57	0.61	0.65	0.7	0.72	0.76	0.8	0.83	0.86	0.91	0.93	0.94	0.97	1
Customer 4	0	0	0.03	0.36	0.41	0.47	0.51	0.56	0.59	0.61	0.64	0.71	0.76	0.79	0.79	0.84	0.9	0.93	0.92	0.96	1
Customer 5	0	0	0.03	0.34	0.39	0.45	0.5	0.56	0.62	0.68	0.72	0.74	0.79	0.85	0.86	0.89	0.93	0.95	0.95	0.98	1

(**b**) Greer and Ruhe Data Set

Figure 6.3: Tensions between the Stakeholders' Satisfaction for Different Budgetary Resource Constraints

Table 6.4: ANOVA analysis by multiple comparisons (Least Significant Difference) for Greer and Ruhe data set

		Diversity		Convergence	
Algorithm (x)	Algorithm (y)	Diff ($x-y$)	Sig.	Diff ($x-y$)	Sig.
5 stakeholders, 20 requirements, 30% resources					
Random	Two-Archive	.8567(*)	.000	.1370(*)	.000
	NSGA-II	.0567	.458	.1370(*)	.000
Two-Archive	Random	-.8567(*)	.000	-.1370(*)	.000
	NSGA-II	-.8000(*)	.000	.0000	1.000
NSGA-II	Random	-.0567	.458	-.1370(*)	.000
	Two-Archive	.8000(*)	.000	.0000	1.000
5 stakeholders, 20 requirements, 50% resources					
Random	Two-Archive	.0071	.324	.0751(*)	.000
	NSGA-II	-.0072	.317	.0751(*)	.000
Two-Archive	Random	-.0071	.324	-.0751(*)	.000
	NSGA-II	-.0143(*)	.049	-.0000	.999
NSGA-II	Random	.0072	.317	-.0751(*)	.000
	Two-Archive	.0143(*)	.049	.0000	.999
5 stakeholders, 20 requirements, 70% resources					
Random	Two-Archive	.2983(*)	.000	.0540(*)	.000
	NSGA-II	.1382	.073	.0537(*)	.000
Two-Archive	Random	-.2983(*)	.000	-.0540(*)	.000
	NSGA-II	-.1600(*)	.039	-.0003	.907
NSGA-II	Random	-.1382	.073	-.0537(*)	.000
	Two-Archive	.1600(*)	.039	.0003	.907
(*) The mean difference is significant at the .05 level.					

where the differences lie, a multiple–comparison procedure was performed. The LSD (Least Significant Difference) method was employed in a multiple comparison to compare the three algorithms, pairwise.

Table 6.4 and Table 6.5 present the results of this analysis for the Greer and Ruhe data set and the Motorola data set respectively. The table reports the pairwise comparisons for diversity and convergence respectively. For each pair of algorithms, the mean difference between the results of applying the algorithms is presented. If

Table 6.5: ANOVA analysis by multiple comparisons (Least Significant Difference) for Motorola data set

		Diversity		Convergence	
Algorithm (x)	Algorithm (y)	Mean Diff ($x-y$)	Sig.	Mean Diff ($x-y$)	Sig.
4 stakeholders, 35 requirements, 30% resources					
Random	Two-Archive	.1466	.000	.1819(*)	.000
	NSGA-II	.0371	.197	.1678(*)	.000
Two-Archive	Random	-.1466	.000	-.1819(*)	.000
	NSGA-II	-.1095	.000	-.0142(*)	.001
NSGA-II	Random	-.0371	.197	-.1678(*)	.000
	Two-Archive	.1095	.000	.0142(*)	.001
4 stakeholders, 35 requirements, 50% resources					
Random	Two-Archive	.0655(*)	.005	.1676(*)	.000
	NSGA-II	0.0426	.066	.1481(*)	.000
Two-Archive	Random	-.0655(*)	.005	-.1676(*)	.000
	NSGA-II	-.0229	.316	-.0194(*)	.000
NSGA-II	Random	-.0426	.066	-.1481(*)	.000
	Two-Archive	.0229	.316	.0194(*)	.000
4 stakeholders, 35 requirements, 70% resources					
Random	Two-Archive	.0596	.088	.1949(*)	.000
	NSGA-II	-.0758(*)	.031	.1820(*)	.000
Two-Archive	Random	-.0596	.088	-.1949(*)	.000
	NSGA-II	-.1354(*)	.000	-.0129(*)	.000
NSGA-II	Random	.0758(*)	.031	-.1820(*)	.000
	Two-Archive	.1354(*)	.000	.0129(*)	.000
(*) The mean difference is significant at the .05 level.					

the significance is smaller than .05, the difference between the algorithms is statistically significant at the 95% α level. The results show that the diversity of the Two-archive algorithm is significant in most cases. The convergence performance among Two-archive, NSGA-II and Random Search is significant. The Two-archive and NSGA-II algorithms always have a better convergence than the Random Search in the all three situations. The Two-archive and NSGA-II algorithms have almost identical performance in terms of convergence.

The analysis results for the Motorola data set presented in Table 6.5 shows, however, that the diversity of the three algorithms is significant in some cases. Their performance in convergence are significant, not only between metaheuristic techniques and Random Search but also between Two-Archive and NSGA-II. The Two-Archive algorithm outperforms NSGA-II and Random Search in the three instances of resource limitations. If we increase the number of objectives further, the significance increases.

6.3 Fairness Analysis *

This section uses multi-objective optimisation approaches to support investigation of the trade-offs in various notions of fairness between multiple stakeholders. Results are presented to validate the approach using two real-world data sets and also using data sets created specifically to stress test the approach. The study reports on experiments to determine the most suitable algorithm for this problem, comparing the results of the NSGA-II algorithm, a widely used multi-objective evolutionary algorithm, and the Two-Archive evolutionary algorithm, a recently proposed alternative, both of which are described in Chapter 2, Section 2.5.5.

6.3.1 Motivation

This study is concerned with fairness. That is, to what extent is it possible to say that an allocation of requirements to stakeholders is fair when there are multiple stakeholders, each with their own idea of what the next set of requirements should be.

*This section is based on the work published by the author at RE 2008 [63] and RE Jounal [64].

Of course, as soon as one embarks upon a discussion of fairness, the issue of what one means by fairness must be addressed. What might appear to be manifestly fair to one person, may seem exceedingly unjust to another. In requirements engineering, there are several possible ways in which fairness can be defined. Rather than picking one of these in a somewhat arbitrary fashion, this study advocates a multi-objective search-based approach, in which a requirements engineer can incorporate all proposed models of fairness.

The approach proposes that each notion of fairness should form an objective in a multi-objective, Pareto optimal SBSE setting. Pareto optimality is well-suited to this scenario, because it makes no assumptions about which objective takes priority. As will be seen, the approach advocated in the study can be used to explore the extent to which a solution can be fair according to all definitions of fairness. The optimising approach can also be said to reveal the inherent tensions and trade-offs between the different notions of fairness.

In this way, SBSE is used not to provide an optimal solution to some particular instantiation of the choice of fairness. Rather, it is used to provide insights by exploring the space of possible solutions and the relationships between them. The aim of this work is thus not to replace a decision maker with an automated tool that allocates requirements, but to provide a new form of decision aid; one that searches for optimal balances, guiding the decision maker. The approach can be used to reveal the structure of the optimisation problems that characterise the difficult balancing act faced by the requirements engineer. Like the previous section, the study concerns the requirements analysis setting in which there are many stakeholders, each with competing (and possibly conflicting interests).

The techniques for fairness analysis proposed in this study have been applied to a real world set of requirements from Motorola and the results are reported as part

of the validation of this work. The study also uses a data set from the literature by Greer and Ruhe, together with synthetically created data that explore the behaviour of the optimisation algorithms used over a range of possible data configurations. The Motorola, Greer and Ruhe data sets are also used in Chapter 6 Tensioning Analysis.

To address the fairness analysis problem, the study adopts a search-based optimisation approach, to automate the exploration of the possible trade-offs and conflicts between various notions of fairness. The search explores the space of possible allocations of requirements for the next release of the system.

The problem of fairness in requirements allocation has two aspects:

1. What is a reasonable way to measure fairness?
2. To what extent can a solution be shown (to the stakeholders) to be a fair allocation of requirements

These two aspects are interrelated and complicated by the fact that there is no single accepted notion of fairness. For example, an allocation might be deemed to be fair were it to satisfy the same number of requirements for each stakeholder. However, this might be over-simplistic; perhaps the solution should give each stakeholder roughly equal value (as perceived by the stakeholder) or, alternatively, roughly equal cost should be spent in implementing each stakeholder's requirements.

This study addresses these issues. It is the first to introduce techniques for analysis of the trade-offs between different stakeholders' notions of fairness in requirements allocation, where there are multiple stakeholders with potentially conflicting requirement priorities and also possibly different views of what would constitute fair and equitable solution.

The study shows that using a multi-objective Pareto optimal search for optimal allocations of requirements, it is possible to treat each candidate notion of fairness as a separate optimisation objective in its own right. The study shows that, using this multi-objective approach, it is possible to explore the trade-offs between different notions of fairness and to attempt to locate solutions that balance these trade-offs.

The result is feedback to the decision maker that serves two purposes: it allows the decision maker to see where there are potential problems in balancing concepts of fairness among stakeholders and it allows the decision maker to demonstrate to the stakeholder that the solution adopted is fair according to multiple fairness criteria.

In this way, the ability to automatically search for optimal regions of the 'fairness space' has applications in negotiation, mediation and conflict resolution during the requirements analysis process. It provides an unbiased and thorough exploration of trade-offs and tensions within the multi-dimensional and complex space of stakeholders and their requirements.

6.3.2 Fitness Function

Fairness is a deceptively simple concept; its implementation is complicated because the definition of fairness may have several equally valid, but possibly conflicting formulations. In order to capture and optimise fairness, a new aspect is explored: *Fairness in Requirements Assignments*. The principal motivation of fairness analysis is try to balance the requirement fulfilments between the stakeholders. It could provide a convincing reference from the view of marketing and help the decision makers to maintain a record of fairness between the stakeholders. It may also play a role in mediation, negotiation and dispute resolution.

Three factors are considered in this study, namely, the number, the value and the

cost of the requirements fulfilled for each stakeholder. The aim is to calculate the absolute amount and the percentage of each factor that is present in a proposed MONRP solution. More formally, the two combinations studied in this work are:

1. Fairness on absolute *number* of fulfilled requirements:

$$\text{Maximise} \qquad \overline{NA}$$

$$\text{Minimise} \qquad \sigma(NA)$$

where $\overline{NA}$ is the mean value of the vector NA.

The vector $NA = \{NA_1, \cdots, NA_m\}$ represents the absolute number of fulfilled requirements for each stakeholder, where $NA_j = |R_j|$. Thus, the aim is to maximise the average absolute number of fulfilled requirements for all the stakeholders whilst minimising the standard deviation of the absolute number fulfilled requirements for each stakeholder.

2. Fairness on absolute *value* of fulfilled requirements:

$$\text{Maximise} \qquad \overline{VA}$$

$$\text{Minimise } \sigma(VA) \text{ where } VA_j = \sum_{i=1}^{n} value(r_i, c_j) \cdot x_i$$

The vector $VA = \{VA_1, \cdots, VA_m\}$ represents the fulfilled value for each stakeholder. In this vector, similarly, $VA_j (1 \leq j \leq m)$ is the j^{th} stakeholder's fulfilled value.

This objective function rewards solutions for which each stakeholder obtains the same value. It penalises solutions the more they depart from this equitable outcome.

6.3.3 Experimental Set Up

6.3.3.1 Data Sets

This section describes the test data sets used to fulfil the research tasks of fairness analysis in requirements assignments. There are three data sets used in our experiments.

The first data set is generated randomly with 30 requirements and 5 stakeholders according to the problem model. The values and costs are assigned as follows: random choices were made for value and cost; the range of costs were from 1 through to 9 inclusive (zero cost is not permitted). The range of values were from 0 to 5 inclusive (zero value is permitted, indicating that the stakeholder places no value on, i.e. does not want, this requirement).

The second and the third data sets are "Motorola data set" and "Greer and Ruhe data set" which have been introduced in Section 6.2.3.1.

6.3.3.2 Algorithmic Tuning

Each algorithm was run for a maximum of 50,000 function evaluations. The number of executions of each algorithm was 30 times for each data set. The parameter tuning options of the NSGA-II algorithm we used here are the same as those described as in Chapter 4 Section 4.2.2.3. In the Two-Archive algorithm, the total capacity of the archives was set to 500. An archive was chosen with a probability that was set to 0.6.

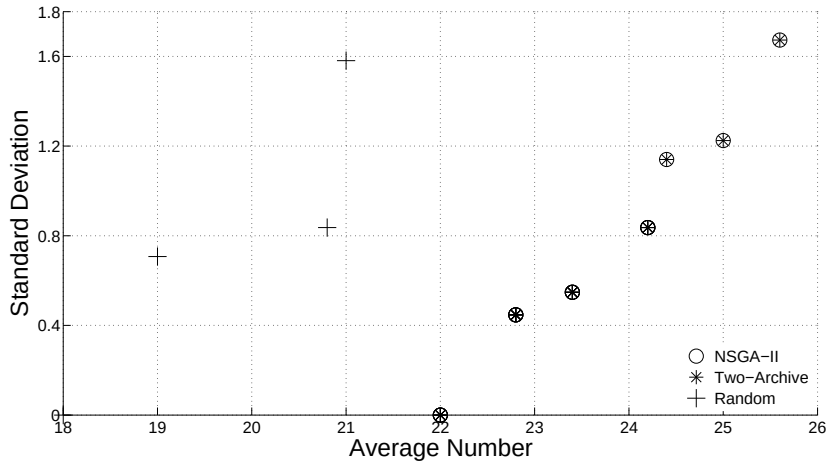
6.3.4 Results and Analysis

In this section, we present the results of applying the different optimisation algorithms to different problem instances. They are NSGA-II, Two-Archive and Random Search. In order to compare their performances, two experiments concerned with fairness on absolute *number* of fulfilled requirements and fairness on absolute *value* of fulfilled requirements were conducted. That is, the first and the second fitness functions defined.

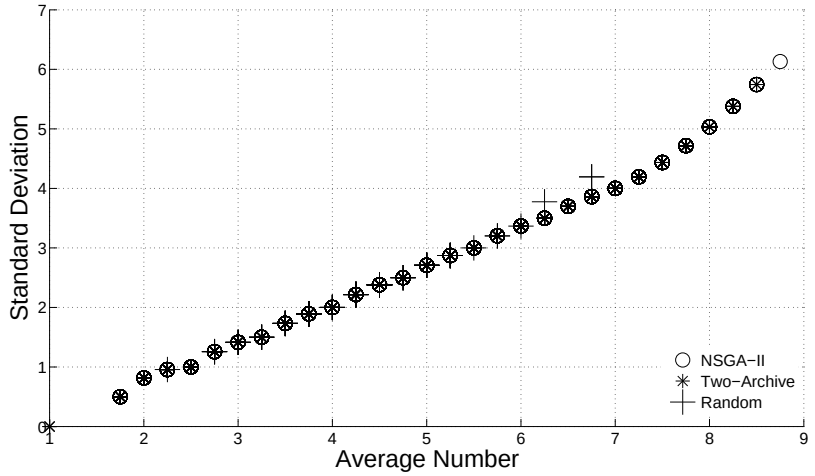
The results shown in Figures 6.4 and 6.5 respectively. The final non-dominated solutions for each technique are represented by the '◯', '$*$' and '$+$' symbols plotted in the figures. These solutions are the best ones over all runs. Each point represents an optimal subset of requirements solution for the next release.

The two objectives in the experiments are the same as described in Section 6.3.2: a) minimise the standard deviation of the absolute number or value of fulfilled requirements for each stakeholder and b) maximise the overall average number or value of fulfilled requirements for all stakeholders.

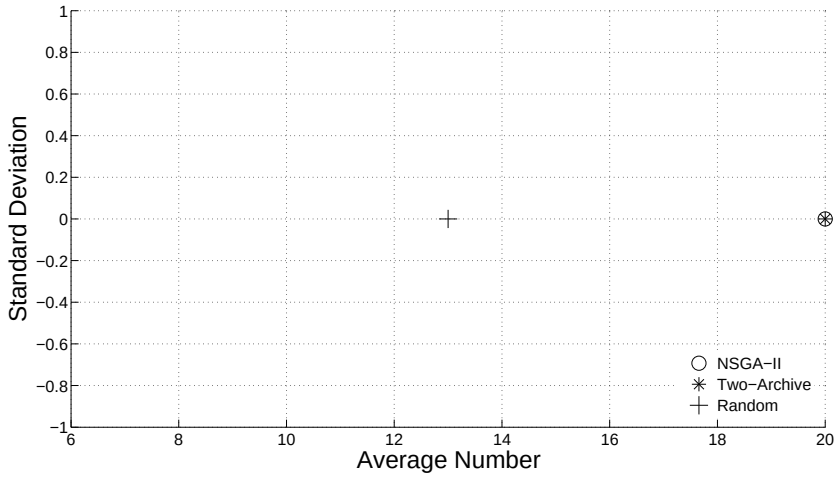
The results of the first experiment are shown in Figure 6.4. In Figure 6.4 (a), the symbol '◯' denotes the Pareto front produced by the NSGA-II algorithm, while the '$*$' symbol denotes the results generated by the Two-Archive algorithm. On these fronts, the standard deviation of fulfilled requirements increases with overall average number. This implies that the more the requirements are fulfilled overall, the less fairness is provided to the stakeholders in general. This is partly because the stakeholders in these data sets demand different numbers of requirements. As the number of the selected requirements increases, it becomes easier for the algorithm to adjust the allocations of fulfilled requirements to different stakeholders to obtain a lower standard deviation (more fairness).



(**a**) Result for Random Data Set

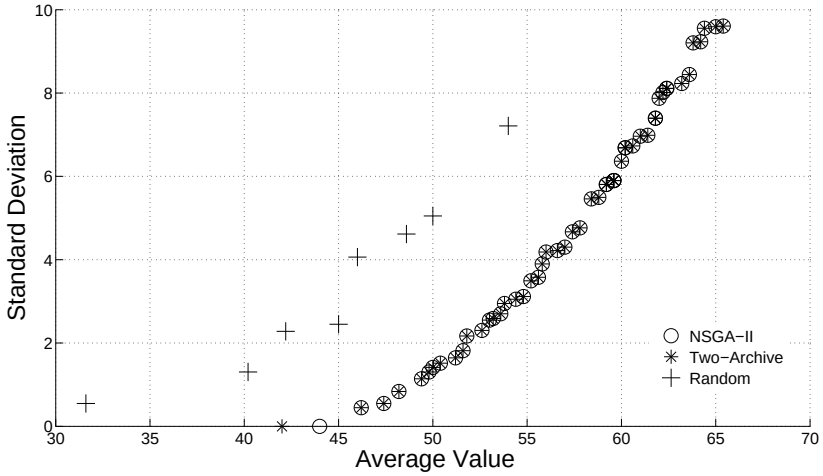


(**b**) Result for Motorola Data Set

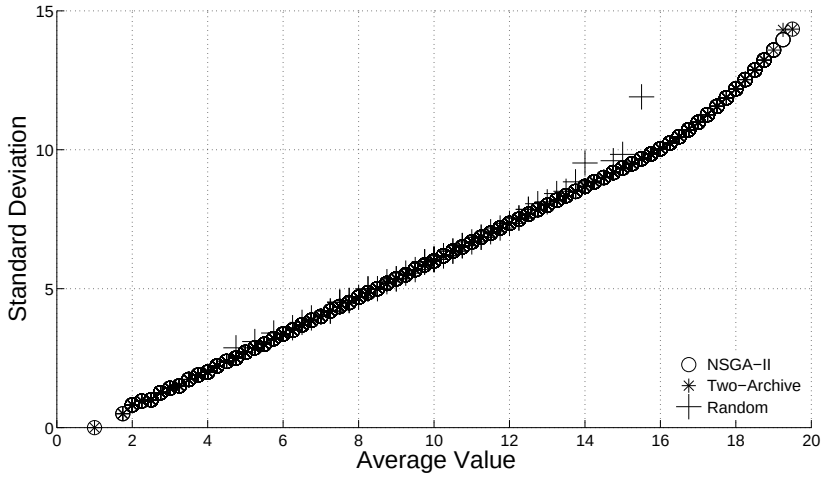


(**c**) Result for Greer and Ruhe Data Set

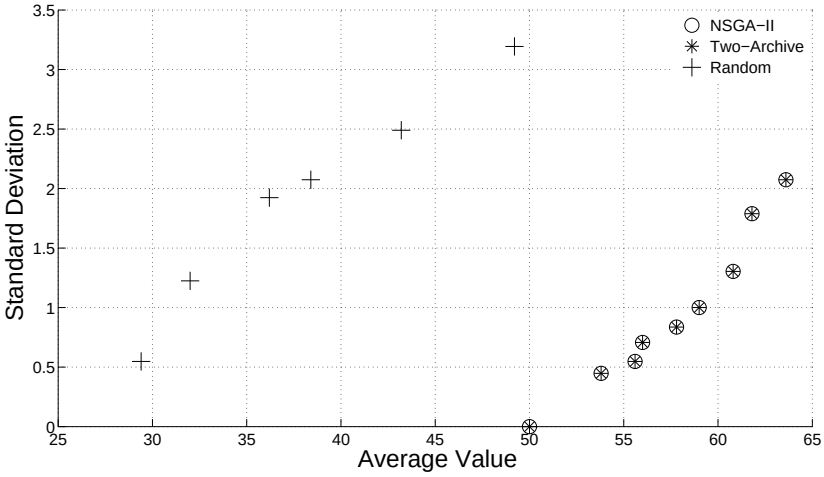
Figure 6.4: Comparative Results of Fairness on Absolute *Number* of Fulfilled Value and Cost



(**a**) Result for Random Data Set



(**b**) Result for Motorola Data Set



(**c**) Result for Greer and Ruhe Data Set

Figure 6.5: Comparative Results of Fairness on Absolute *Value* of Fulfilled Requirements

In Figure 6.4 (a), solutions along the X-axis with zero standard deviation show that the NSGA-II and two-archive algorithms are able to obtain subsets of requirements that fulfil each stakeholder with the same number of requirements.

In Figure 6.4 (b), we observe that this sort of "perfectly-fair" solution moves towards the left along the X-axis. This is because of the difference between the sparsity pattern of the Stakeholder-Requirement matrix of these two data sets. In the Motorola data set, every requirement is demanded by only one stakeholder exclusively, and the fourth stakeholder requests only a single requirement. This pattern dramatically increases the difficulty for search-based techniques to obtain the "perfectly-fair" solution that fulfils each stakeholder with only one requirement.

On the other hand, the result for the Greer and Ruhe data set in Figure 6.4 (c) shows the standard deviation remains at zero throughout the search. This is also because of the distribution of the data, which, in this case is perfectly uniform. That is, in the Greer and Ruhe data set, every stakeholder requests every requirement, so all stakeholders would have an equal number of fulfilled requirements, no matter which requirements are selected in the next release.

Figure 6.5 illustrates the results for the second experiment. On the Pareto fronts of these results, a similar trend is observed: the degree of fairness decreases as the overall coverage increases.

We also observe that there is no gap between the two sets of solutions. The Pareto fronts of each solution share many common points and almost overlap completely. Therefore, from the figure we can see the Pareto front consists of the markers '⊛'. This means that these two sets of solutions have a large intersection and the search techniques perform better than the Random Search. The results generated by the Random Search could not reach the Pareto front as can be seen by direct observation

of the figures.

In terms of performance of two search algorithms, it is difficult to distinguish between them visually, simply by looking at the figures displayed. Therefore, a Reference Pareto front was constructed and used in this study to compare the Pareto fronts produced by different algorithms. Consisting of the best solutions of each technique, the Reference Pareto front denotes the best available approximation to the *real* Pareto front. The Pareto fronts generated by the different search techniques may partly contribute to the Reference Pareto front. Therefore, one of the measurements to compare Pareto fronts is to count the number of solutions on the Reference Pareto front, namely the solutions that are not dominated by the Reference Pareto front. The results of this analysis are shown in Table 6.6 and Table 6.7.

In Tables 6.6 and 6.7, the results provide a more quantitative analysis than the figures. For the all data sets, NSGA-II performs best, taking the lion's share of the Reference Pareto front. In the Random data set and Greer and Ruhe data set, the NSGA-II and Two-Archive algorithms share almost the same amount of the Pareto front. Indeed, both succeed in covering almost the entire front, showing very similar performance.

However, the Motorola data set reveals a slightly different story. There is no algorithm that covers the entire Reference Pareto front. The solutions from NSGA-II provide the largest share of the Reference front. Moreover, the results from the Two-Archive algorithm also contribute many solutions to the Reference Pareto front. In addition, there is an interesting observation in the results from the application of the three algorithms to the Motorola data set: there are few duplicate solutions obtained by the three algorithms. In this case, there are almost no elements in common in the three sets of solutions and the methods preserve wide diversity of Pareto solutions. For this reason, in Table 6.6 and 6.7, adding up the numerical

Table 6.6: Percentage of solutions on the Reference Pareto front (Fairness on Absolute Number of Fulfilled Requirements)

	NSGA-II	Two-Archive	Random
Random Data Set	100%	100%	0%
Motorola Data Set	63.17%	32.69%	4.36%
Greer and Ruhe Data Set	100%	100%	0%

Table 6.7: Percentage of solutions on the Reference Pareto front (Fairness on Absolute Value of Fulfilled Requirements)

	NSGA-II	Two-Archive	Random
Random Data Set	100%	98.08%	0%
Motorola Data Set	55.92%	44.34%	0.40%
Greer and Ruhe Data Set	100%	100%	0%

values together in '*Motorola Data Set*' row is almost equal to 100%.

Another observation shows that the Random Search even contributes to the Reference Pareto front in some cases. The algorithms having the occasional good or bad performance may be due to the different characteristics and scale size of data sets. In terms of Random Search, this may occur when the size of the data set is comparatively small and therefore denotes a relatively easy optimisation problem.

The experiments show that as more requirements are fulfilled, less fairness is provided to the stakeholders. This is partly due to the high variation in the stakeholders' number of requirements in the examined data sets. Fortunately, as the number of the selected requirements increases, the algorithm has more scope in which to search for optimally fair solutions. It was also observed that the quality of the final solutions in terms of fairness is partly dependent upon the sparsity pattern of the Stakeholder-

Requirement matrices. This is also the case for the search algorithm, i.e. sparser Stakeholder-Requirement matrices tend to make the problem more difficult for the search algorithm.

6.4 Summary

This chapter introduces the concept of tensioning and fairness among multiple stakeholders in requirements analysis and optimisation for the first time.

Tensioning analysis formulates the multi-stakeholder requirements analysis problem as a MOOP, treating each stakeholder's set of requirements as a separate objective to calculate and investigate the extent of satisfaction for each stakeholder. This is the first time this single-objective-per-stakeholder formulation has been proposed.

The study caters for tensioning of the relationship between the stakeholders and resources as well as the natural internal tensioning among the stakeholders. These scenarios let the decision maker explore and optimise the trade-offs allowed by the instantiation of the problem with which they are faced.

The chapter presents an empirical study and statistical analysis in which meta-heuristic search techniques are applied to the new formulations of the problem. We compare the performance of the Two-Archive algorithm, the NSGA-II algorithm and a Random Search. These three algorithms were compared according to a set of convergence and diversity metrics on a set of scalable testing data sets including both those synthetically generated to provide limiting-case data and real world industrial data. The results reveal that the Two-Archive algorithm outperformed the others in convergence as the scale of the problems increased.

The study provides a simple visual method to show the tensioning of the relation-

ships between the stakeholders and resources as well as the internal tensioning among the stakeholders. We envisage that the animated Kiviat diagrams introduced in the chapter can be used by a decision maker to explore the effect of budgetary pressure on multiple stakeholder satisfaction. The decision maker might, for example, use these diagrams to help decide upon the position to adopt in external negotiations with stakeholders. The requirements engineer can also see where modest increases in budget can result in significant increases in satisfaction. This analysis may be useful in internal negotiations over budgetary allowance for the project.

Though the study has presented results for real world requirements data sets, more work is required to explore these potential applications of this work. For example, in the case where there are hundreds of thousands of users in an development project, single-objective-per-stakeholder formulation may not be used directly in this context. We can extend our work to deal with these stakeholders first. The stakeholders who share similar interests and goals can be clustered together and divided into several groups in order to reduce the number of objectives.

In the second part, we discussed the concept of fairness in requirements analysis and optimisation. Three fairness models were introduced to balance the requirement fulfilments between stakeholders. The work reported here is the first to address the issue of balancing fairness among different definitions. The formulations adopted cover simplified scenarios. However, even with the relatively simple formulations adopted in this study, it has been possible to use search-based optimisation techniques to reveal tensions between fairness definitions.

The experiments upon which this study reports demonstrate that search-based techniques can be applied to real world data sets and illustrate the way in which they reveal hidden tensions implicit in these data sets. We presented the results of a comparative study of Random Search and two more sophisticated, evolutionary multi-

objective search techniques. The results validate the overall approach, showing that the more sophisticated techniques comfortably outperform Random Search.

Chapter 7

Conclusions and Future Work

In this chapter, we first conclude the thesis by reviewing the outcomes and contributions of search-based requirements selection and optimisation approach introduced in this thesis. We then discuss some of the threats to validity of the findings and limitations of this thesis. Finally, we outline some potential avenues for future work.

7.1 Summary

The intention of the thesis was to widen and explore the scope of the release planning as a problem within Search-based Software Engineering. The contributions were to provide new approaches to support and improve requirements analysis and optimisation. The motivation was to formulate requirements selection and analysis problems as search-based optimisation problems and, thereby, to provide an automatic, flexible framework. This may be useful for decision making support in the RE process. Based on the search-based requirements selection and optimisation framework, the contributions of the research work presented in the thesis can be summarised as follows:

7.1.1 Value/Cost Trade-off in Requirements Selection

The goal of this study was to introduce a novel formulation for a search-based multi-objective requirements selection. Previous work considered only single objective formulations. In the multi-objective formulation, the problem is to select the optimal or near optimal subset of requirements and the two objectives are (1) to maximise overall *Value* for the stakeholders and (2) to minimise total *Cost* required for the satisfaction of stakeholders' requirements.

Search techniques were applied to find approximations of the Pareto-optimal set. This allows the decision maker to select the preferred solution from the Pareto front according to different contexts. Empirical studies investigated the relative performance of the algorithms adopted in three scales of synthetic data sets and explored the 'critical point' at which the metaheuristic search techniques can outperform a random search. That is, once the critical mass exceeds a predetermined critical point, it becomes worthwhile to apply search techniques to the problem.

Then, we extended the basic formulation by taking requirements change into consideration, introducing the Today/Future Importance Analysis (T/FIA) model. This study was based on the real world data set from Ericsson in a three-objective formulation in order to select optimal subset of requirements both for today and future scenarios. We found that the resulting Pareto front captured the trade-offs between the two competing objectives: *value* for today and *value* for the future. It also can provide valuable insights into the structure of the data and highlight relationships between objectives.

7.1.2 Requirements Interaction Management

This study presented requirements interaction management using search-based techniques for the first time. Two empirical studies (a dependency impact study and a scale study) were conducted to simulate the requirements selection process under five common types of requirements dependencies (*And*, *Or*, *Precedence*, *Value-related* and *Cost-related*). The objectives were to investigate the influence of requirements dependence on the automated requirements selection process and to validate the feasibility of the proposed framework.

In the empirical studies, the four data sets were designed in different scales and densities of the stakeholder-requirement matrix. The Dependency Impact Study evaluated the impacts of five different dependency types and the Scale Study reported results concerning the performance of the two algorithms (the NSGA-II algorithm and an archive based NSGA-II algorithm) as the data sets increase in size.

7.1.3 Multi-Stakeholder Tensioning Analysis

We also focused on stakeholder relationships. The problem was formulated as a Multi-Objective Optimisation Problem (MOOP), treating each stakeholder's set of requirements as a separate objective. The overall goal was to calculate and investigate the extent of satisfaction for each stakeholder. This is the first study to propose such a single-objective-per-stakeholder formulation.

The study catered for tensioning of the relationship between the stakeholders and the resources as well as the natural internal tensioning among the stakeholders. We carried out an empirical study and a statistical analysis in which metaheuristic search techniques were applied to the new formulations of the problem. The

results suggested that the Two-Archive algorithm outperforms the NSGA-II algorithm when the objectives increase in size. The study also provided Kiviat diagrams to visualise the discovered solutions in the multi-dimensional solution space. These scenarios let the decision maker explore and optimise the trade-offs allowed by the instantiation of the problem with which they are faced. We adapted the traditional static Kiviat diagram to create an animated Kiviat diagram that allows the decision maker to explore the dynamic changes in inter-stakeholder tension as budgetary pressure increases.

7.1.4 Multi-Stakeholder Fairness Analysis

The objective of this study was to support investigation of the trade-offs in various notions of fairness between multiple stakeholders by using multi-objective search techniques. We provided ways to define and measure fairness. Each notion of fairness formed a fitness function. Then, Pareto optimal multi-objective search approaches were adapted to answer the question "To what extent is it possible to say that a solution can be shown (to the stakeholders) to be a fair allocation of requirements?" The results of empirical studies illustrate the way in which they reveal hidden tensions implicit in the data sets. The results showed that as more requirements are fulfilled, less fairness is provided to the stakeholders. We also presented results of a comparative study to determine the most suitable algorithm for this problem and from that, the NSGA-II algorithm showed the best performance.

7.2 The Tools Used in This Thesis

The multi-objective search-based requirements selection and optimisation was supported by tools which was developed using the MATLAB system (*R2007a*). The main programming language is Matlab script and all the data sets were formatted into .mat or .dat files. The system was installed on an Intel Core 2 Duo processor 2.26 GHz with 4Gb RAM. The main source code for the tools is reported in the Appendix.

The structure of the tools can be divided into three major parts: (1). Data set generation and initialisation; (2). Selection process; (3). Result representation and visualisation, which are illustrated in Figures 7.1, 7.2 and 7.3 separately.

In the first part, there are three sources of raw data sets categorised by the tools. After being initialised and formatted, the requirement attributes are transferred to measurable criteria. These data sets can be used as the inputs to the selection process.

In the selection process, according to the fitness function, seven search-based techniques are applied to provide three main functionalities in the framework. In addition, changes to the system can also be involved in this phase and be evolved for the next iteration.

In the third part, the optimal or near-optimal solutions are generated and the implicit characteristics of the problem are exposed. The software engineer can investigate the insight achieved and the solutions provided based on the visualisations in order to help fully understand the problem, give feedback and finally arrive at the practical solutions. In addition, the best techniques can be selected based on performance metrics.

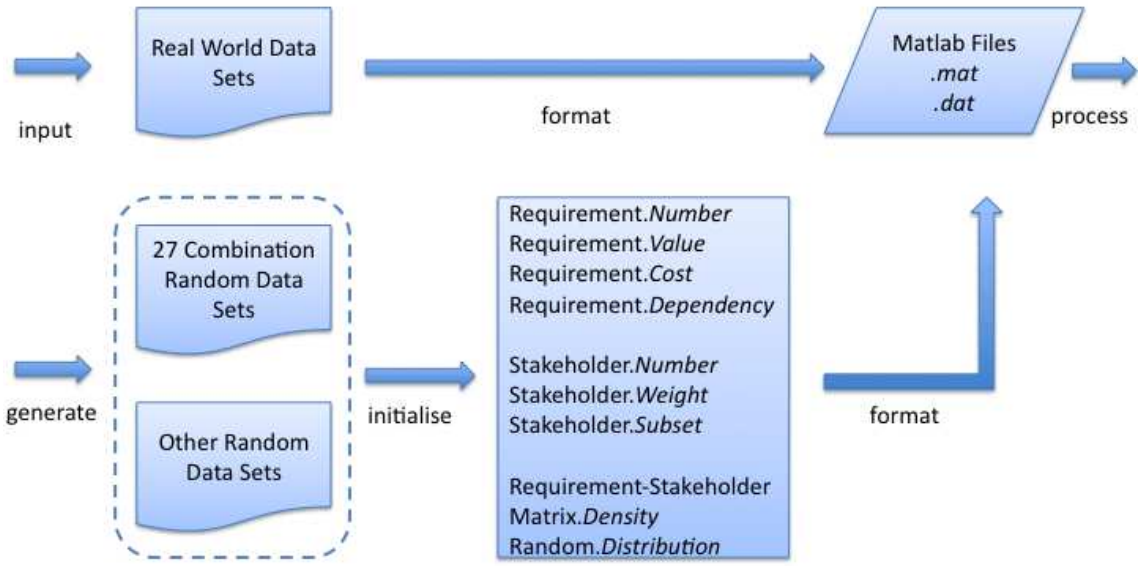


Figure 7.1: Data Sets Generation and Initialisation

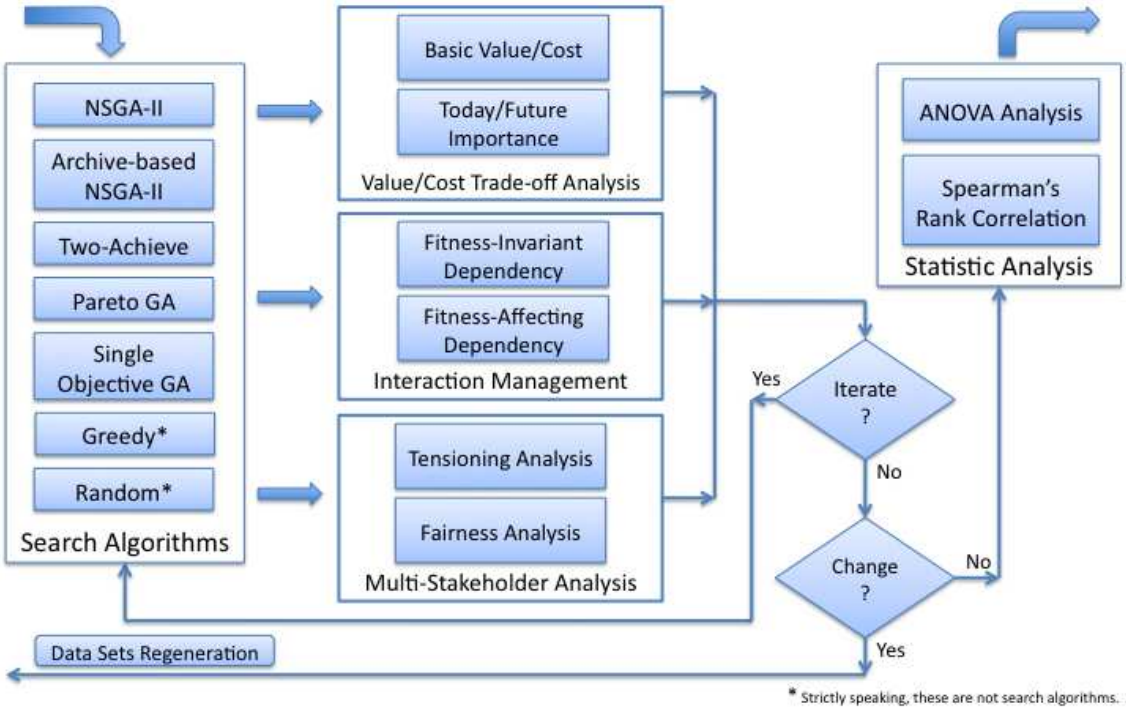


Figure 7.2: Selection Process

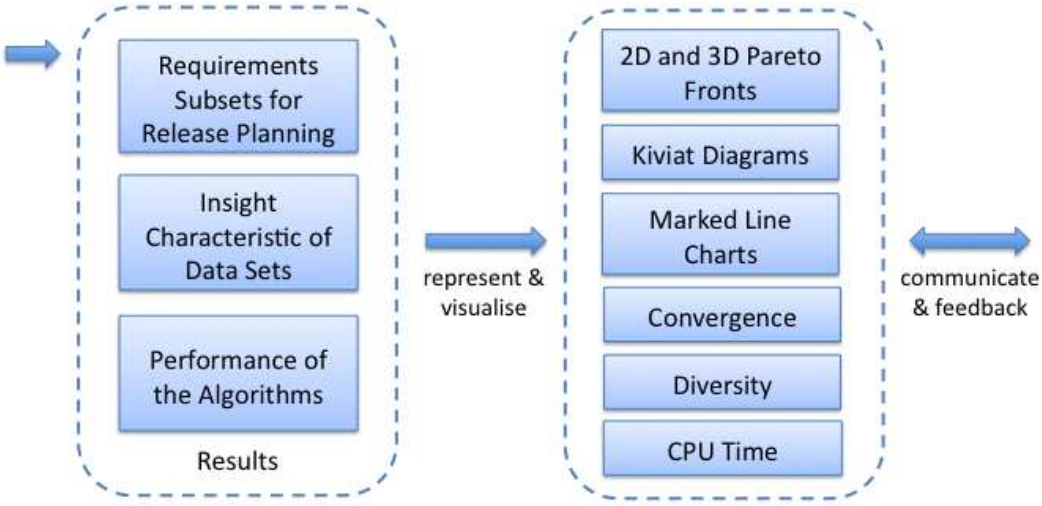


Figure 7.3: Results Representation and Visualisation

7.3 Threats to Validity and Limitations

The results of the current research are subject to limitations which are inherent in any empirical study. This section sets out these threats to validity and limitations, indicating how they may affect the degree to which it is possible to generalise the results. It is important to consider these threats to validity so that the work is contained within a complete and full understanding of its limitations. Four threats to potential validity of findings are considered and discussed in this section: construct validity, internal validity, external validity and conclusion validity.

Construct validity "concerns generalising the result of the experiment to the concept or theory behind the experiment." [195]. It reflects the degree to which the measures used in the study accurately measure the properties of the entity under investigation.

In this thesis, the objects studied are sets of requirements and their properties. Their properties of interest are the *value* and *cost* 'measurements' associated with each requirements. These measurements on which we base the optimisation come from estimates. It is common for software engineers to rely on estimates and it is helpful in the provision of decision support to use these estimates to explore trade-offs of requirements selection. Indeed, the results of the analysis rely on the quality of the estimates. However, this might lead to one possible construct validity issue when these estimates are unreliable. Another issue is that not all the projects have clearly visible stakeholders who evaluate existing requirements. Thus the estimated inputs to our Framework are unavailable.

External Validity is the degree to which the research findings can be generalised to outside world. "Threats to external validity are conditions that limit our ability to generalise the results of our experiment to industrial practice." [195].

In the thesis, two kinds of data sets were used in the empirical study. In order to explore the behaviour of the algorithms for the different sizes of problem, synthetic data sets were generated to compare performance. These data sets clearly cannot be said to test the performance of real world data sets. Some of them were only used to test the performance of the algorithms adopted for search-based optimisation. The "27 combination random data sets" were generated to provide a better coverage of potential instances that remained uncaptured by the real world case studies. However, as discussed in Chapter 5, these data sets still cannot be said to be representative. There is missing information in the "27 combination random data sets" model. Such as, only uniform distribution is adopted in the generation of random numbers. In the real world, the data structure might be followed by other probability distributions. In addition, the different types of requirement relationships may also have different dependency density levels. All of these factors are

likely to impact on the results obtained.

The real world data sets used in the thesis come from Motorola, Ericsson and from a previous paper by Greer and Ruhe [75]. These data sets from industry and the literature are relatively more realistic. However, they might not be regarded as a fair representative of all real cases. In follow up studies, more real world data sets will be required to better cover the wider context of requirements applications in general. However, it is known to be difficult to obtain real world data sets; they are typically considered confidential by the companies that own them.

In addition, our visualisation approach is also limited to multi-stakeholder requirements scenarios with relatively few stakeholders. Once the real world scenario has more than about 10 stakeholders, the visualisations we propose and the animations we introduce will become hard to read. Furthermore, the search-based optimisation techniques we use do not, presently, scale well beyond four or five objectives, that is, the performance deteriorates according to a set of convergence and diversity metrics. However, this remains a current topic of open research in the optimisation research community. Fortunately, even in the real world scenarios accounting for many stakeholders, it will still be possible to efficiently adopt the approach introduced in the thesis. The stakeholders who share similar interests and goals can be clustered together to reduce the number of objectives.

Internal validity is the degree to which the values of the dependent variable (the study outcome) are only the results of the manipulation of the independent variable. "Threats to internal validity are influences that can affect the independent variable with respect to causality, without the researcher's knowledge." [195].

In the case of this research topic, the primary comparison tests for statistically significant results concerned the relative performance of the algorithms. The NSGA-II

algorithm and its variation, the Two-Archive algorithm, Pareto GA, Single Objective GA, Greedy algorithm and Random Search were used in this thesis. The first two search algorithms (NSGA-II and the Two-Archive) denote one popular and one comparatively new multi-objective search algorithm. The dependent variables measured are convergence and diversity which are two widely used measures of the effectiveness of multi-objective optimisation algorithms.

The significance level chosen was the 95% level, denoting our balance of concerns with respect to type I and type II statistical errors. This is also the choice adopted in most other research on Empirical Software Engineering. It is a sensible balance between the two types of error for research such as this, where adoption of a technique can improve business performance but where safety critical issues are not direct consequences of any such adoption. That is, one does not want to abandon potentially valuable new techniques by setting the alpha level to a more stringent requirement. However, one does want to ensure that there is a relatively robust assessment that the evidence points to a causal effect of the dependent variable on the independent variable.

Evolutionary algorithms have been shown to be effective and efficient in a wide range of problems but, like any other metaheuristic algorithms, there is no guarantee that they are the best available solution for a given problem. As a result, there might be better performing algorithms for the applications.

Although identification of the best algorithms for the problem could be a separate subject, we feel the chosen algorithms are appropriate to address the main research questions in the studies. Our results have demonstrated that the evolutionary optimisation algorithms are superior to Random Search. While this is a relatively low threshold to aim for, it does provide a validity check on the approach. Also, in the absence of any alternative technique, Random Search is a natural choice for a base

line validity check.

A number of assumptions have been made regarding the operators as well as parameter values of the search algorithms which can influence the results. However, all assumptions are the same for all algorithms to minimise environmental factors during the experimentations. Performance of the algorithms could have been improved by individual fine tuning empirically or through systematic experimentation. This is beyond the scope of the current research but could be suggested as an avenue for future research.

Conclusion validity is the degree to which conclusions reached are reasonable for the given results. "Threats to the conclusion validity are concerned with issues that affect the ability to draw the right conclusions about relations between the treatment and the outcome of an experiment." [195].

The results provided in the thesis are based upon the study of real world and synthetic data sets to support the claim, but this is merely initial evidence. So it would be too strong to claim that the approach will save requirements engineers' time or that it will help to ensure that the results of requirements analysis please the stakeholders. However, we believe that the modest claim that the multi-objective search-based requirements selection and optimisation framework is feasible for the given evidence.

7.4 Future Work

This thesis presented initial work of Search-based Requirements Analysis and Optimisation for release planning problem. Hence, the results of the thesis provided are not "final and conclusive". This section describes several possibilities for future

work.

The study will focus on adapting the search-based formulations to cater for more 'messy' real world scenarios in which requirements are partially unclear and subject to change and for which domain specific parameters are both constrained and subject to estimation error. These augmented scenarios would pose a significant challenge to any approach. However, there are grounds for optimism.

Requirements prioritisation is another topic in which we are interested. Search techniques might be the best qualified candidates to overcome the scalability problem in requirements prioritisation.

Future work will also consider the ways in which the approach can be scaled up to handle scenarios with many more stakeholders than considered here. In this work, automated optimising clustering techniques will be considered as a way to identify groups of stakeholders with similar goals. Such an approach can be used in combination with the multi-stakeholder optimisation approach introduced in this thesis to explore the ways in which different groups of stakeholders might interact, compete and collaborate (facilitated by automated decision support) to agree upon the most mutually agreeable set of requirements.

We will verify these findings by applying other search techniques and also classical optimisation techniques (such as, integer programming, non-linear programming) to more real world problems. This will provide valuable feedback to researchers and practitioners in search techniques as well as to the RE community. This, in turn, may give rise to the need for the development of more efficient solution techniques.

Finally, we will provide a generic and multi-perspective framework of search-based requirements optimisation. The work will be concerned with finding ways to package and deliver the framework in a way that can be used by working requirements

engineers in the context of RE tool sets. It will need to be effective, interactive and non-technical. This will be a challenge, but the results and visualisations presented here give some cause to believe it is achievable.

Appendix A

Source Code

```
%% Cost/Value and Today/Future Important Analysis

%% record time
starttime = cputime;
%% set path
addpath ('/Users/Yuanyuan/Documents/MATLAB/Ericsson_3obj/NSGA2');
addpath('/Users/Yuanyuan/Documents/MATLAB/Ericsson_3obj/Dataset');
addpath('/Users/Yuanyuan/Documents/MATLAB/Ericsson_3obj/Experiment');
addpath('/Users/Yuanyuan/Documents/MATLAB/Ericsson_3obj/data');
addpath('/Users/Yuanyuan/Documents/MATLAB/Ericsson_3obj');
%% rand seed
rand('state',sum(100*clock)*rand(1));
%% global data
%  ds: dataset including customers and requirements.
%  nsga2: algorithm model.
global ds;
global nsga2;
```

```
ds = [];
nsga2 = [];
nsga2.chr_front = [];
%% init Dataset
Dataset();
index = 1;
for i = 1 : 30
    index = NSGA2(500,100,3,index);
end

NSGA2_find_pareto();
[~,variable] = size(nsga2.chr_front);

a = nsga2.chr_front(:,ds.req+3);
a = -a;
scatterbar3(nsga2.chr_front(:,ds.req+1),nsga2.chr_front(:,ds.req+2),a,20)
grid on

xlabel('Value for Today','FontSize',16);
ylabel('Value for the Future','FontSize',16);
zlabel('Cost','FontSize',16);
set(gca,'FontSize',16);

runningtime = cputime-starttime;
```

```
function [  ] = Dataset()

global ds;

```

```
ds.now = [];
ds.future = [];

ds.cus = 0;
ds.now = 0;

ds.now = load ('data/now.dat');
ds.future = load('data/future.dat');

[ds.cus,ds.req] = size(ds.now);

Dataset_init_cost();
```

```
function scatterbar3(X,Y,Z,width)
% plot 3D scatter bar graph.

[r,c] = size(Z);
for j = 1 : r,
    for k = 1 : c,
        if ~isnan(Z(j,k))
            drawbar(X(j,k),Y(j,k),Z(j,k),width/2)
        end
    end
end

zlim = [min(Z(:)) max(Z(:))];
if zlim(1) > 0
    zlim(1) = 0;
end
```

```
if zlim(2) < 0
    zlim(2) = 0;
end
axis([min(X(:))-width max(X(:))+width min(Y(:))...
    -width max(Y(:))+width zlim])
caxis([min(Z(:)) max(Z(:))])

function drawbar(x,y,z,width)

h(1) = patch([-width -width width width]+x,...
    [-width width width -width]+y,[0 0 0 0],'w');
h(2) = patch(width.*[-1 -1 1 1]+x,...
    width.*[-1 -1 -1 -1]+y,z.*[0 1 1 0],'w');
h(3) = patch(width.*[-1 -1 -1 -1]+x,...
    width.*[-1 -1 1 1]+y,z.*[0 1 1 0],'w');
h(4) = patch([-width -width width width]+x,...
    [-width width width -width]+y,[z z z z],'w');
h(5) = patch(width.*[-1 -1 1 1]+x,...
    width.*[1 1 1 1]+y,z.*[0 1 1 0],'w');
h(6) = patch(width.*[1 1 1 1]+x,...
    width.*[-1 -1 1 1]+y,z.*[0 1 1 0],'w');
set(h,'facecolor','flat','FaceVertexCData',z)
```

```
%% Requirements Interaction Management Experiment using NSGA II

%% set path
addpath ('/Users/Yuanyuan/Documents/MATLAB/Dependence/NSGA2');
addpath('/Users/Yuanyuan/Documents/MATLAB/Dependence/Dataset');
addpath('/Users/Yuanyuan/Documents/MATLAB/Dependence/Experiment');
```

```
addpath('/Users/Yuanyuan/Documents/MATLAB/Dependence/Draw');
%% rand seed
rand('state',sum(100*clock)*rand(1));
%% global data
%  ds: dataset including customers and requirements.
%  nsga2: algorithm model.
global ds;
global nsga2;
ds = [];
nsga2 = [];
%% init Dataset
REQUIRMENT_SIZE = 3;      %(1-3)
CUSTOMER_SIZE = 3;        %(1-3)
SELECTION_DENSITY = 3;    %(1-3)
DP_AND_DENSITY = 0.3;     %(0-1)
DP_OR_DENSITY = 0.3;      %(0-1)
DP_ORDER_DENSITY = 0.3;   %(0-1)
DP_VALUE_DENSITY = 0.3;   %(0-1)
DP_COST_DENSITY = 0.3;    %(0-1)

 Dataset(SELECTION_DENSITY,  ...
         CUSTOMER_SIZE,...
         REQUIRMENT_SIZE,...
         DP_AND_DENSITY,...
         DP_OR_DENSITY,...
         DP_ORDER_DENSITY,...
         DP_VALUE_DENSITY,...
         DP_COST_DENSITY);
```

```
35 %% load data
36 load 'nsga2_1.mat' ;
37 ds.req = 35;
38 ds.cus = 1;
39 ds.data = R(1,:);
40 ds.cost = R(2,:);
41 ds.weight = C;
42 %% Setup Experiment
43 POPULATION = 500;
44 GENERATION = 100;
45 OBJECTIVE = 2;
46 DEP_VALUE_COST = 0;
47 %% Run Experiment
48 exp = 1;
49 switch exp
50     case 1
51         exp1();
52 % Exp1: Experiment in normal mode, display the front of the last generation.
53     case 2
54         exp2();
55 % Exp2: Compare and, or, order dependence with normal front using last generati
56     case 3
57         exp3();
58 % Exp3: Compare and, or, order dependence with normal front using archive.
59     case 4
60         exp4();
61 % Exp4: Compare and, or, order dependence between archive and last generation.
62     case 5
```

```
        exp5();
% Exp5: Compare cost, value, dependence with normal with last generation.
end
```

```
function [  ] = Dataset(density,cus, req,dp_and_per, ...
                        dp_or_per,dp_order_per,...
                        dp_value_per,dp_cost_per)
% create dataset according to number of requirements, customer, density
% and dependence relationship

global ds;
ds.data = [];
ds.cost = [];
ds.weight = [];
ds.dp_and_per = dp_and_per;
ds.dp_or_per = dp_or_per;
ds.dp_order_per = dp_order_per;
ds.dp_value_per = dp_value_per;
ds.dp_cost_per = dp_cost_per;

switch density
    case 1
        id = 0.01; ad = 0.33;
    case 2
        id = 0.34; ad = 0.66;
    case 3
        id = 0.98; ad = 0.98;
end

```

```
26 switch req
27     case 1
28         iM = 1; aM = 100;
29     case 2
30         iM = 101; aM = 250;
31     case 3
32         iM = 251; aM = 600;
33 end
34
35 switch cus
36     case 1
37         im = 2; am = 5;
38     case 2
39         im = 6; am = 20;
40     case 3
41         im = 21; am = 50;
42 end
43
44 ds.density = id + (ad - id) * rand(1);
45 ds.req = round(iM + (aM - iM) * rand(1));
46 ds.cus = round(im + (am - im) * rand(1));
47
48 while ds.req <= ds.cus
49     ds.req=round(iM + (aM - iM) * rand(1));
50     ds.cus=round(im + (am - im) * rand(1));
51 end
52
53 % init dataset
```

```
54 Dataset_init_value();
55 Dataset_clean_useless_cus();
56 Dataset_init_cost();
57 Dataset_init_weight();
58
59 % init dep data
60 Dataset_init_dep_and();
61 Dataset_init_dep_or();
62 Dataset_init_dep_order();
63 Dataset_init_dep_value();
64 Dataset_init_dep_cost();
```

```
1  function [] = Dataset_init_weight()
2
3  global ds;
4  min_weight = 1;
5  max_weight = 100;
6
7  for j = 1 : ds.cus
8      weight(j)=round(min_weight + (max_weight - min_weight) * rand(1));
9  end
10 total = sum (weight);
11
12 for i = 1 : ds.cus
13     ds.weight(i) = weight(i) / total;
14 end
```

```
1  function [ ] = Dataset_init_value( )
2
```

```
global ds;

min_value = 1;
max_value = 5;

ds.data=sprand(ds.cus,ds.req,ds.density);

% set value
ra=find(ds.data);
for i=1: length(ra)
    ds.data(ra(i))=round(min_value + (max_value - min_value)*rand(1));
end

ds.data=full(ds.data);
```

```
function [ ] = Dataset_init_cost( )

global ds;
min_cost = 1;
max_cost = 9;

% set cost
for j = 1 : ds.req
    ds.cost(j) = round(min_cost + (max_cost - min_cost) * rand(1));
end
```

```
function [ ] = Dataset_clean_useless_cus( )
%delete customers of 0 value

```

```
global ds;
i=1;

while i <= ds.cus
    if sum(ds.data(i,:)) == 0
        ds.data(i,:) = [];
    else i = i + 1;
    end
    [ds.cus,tmp] = size(ds.data);
end
```

```
function [] = Dataset_init_dep_value( )

global ds;
ds.dp_value = [];

a = ds.req * ds.dp_value_per;
ra_value_density = a / (ds.req * ds.req);
ds.ra_dep_value_density = ra_value_density;

ds.dp_value = sprand(ds.req,ds.req,ds.ra_dep_value_density);

ds.dp_value(find(ds.dp_value~=0)) = 1;
ds.dp_value = full(ds.dp_value);

for i = 1 : ds.req
    for j = 1 : ds.req
        if(ds.dp_value(i,j) == 1)
            m = round(1 + (4 - 1) * rand(1));
```

```
            if(m == 1)
                ds.dp_value(i,j) = -0.2;
            elseif(m == 2)
                ds.dp_value(i,j) = -0.4;
            elseif(m == 3)
                ds.dp_value(i,j) = 0.2;
            elseif(m == 4)
                ds.dp_value(i,j) = 0.4;
            end
        end
    end
end
```

```
function [ output_args ] = Dataset_init_dep_order( input_args )

global ds;
ds.dp_order=[];

a = ds.req * ds.dp_order_per;
ra_order_density = a / (ds.req * ds.req);
ds.ra_dep_order_density = ra_order_density;

ds.dp_order = sprand(ds.req,ds.req,ds.ra_dep_order_density);

ds.dp_order(find(ds.dp_order~=0)) = 1;
ds.dp_order(find(ds.dp_and~=0)) = 0;
ds.dp_order(find(ds.dp_or~=0)) = 0;
ds.dp_order = full(ds.dp_order);

```

```
for i = 1 : ds.req
    for j = 1 : ds.req
        if(i == j)
            ds.dp_order(j,i) = 0;
        elseif (ds.dp_order(i,j) == 1 && i~=j)
            ds.dp_order(j,i) = 0;
        end
    end
end

for i = 1 : ds.req
    for j =1 : ds.req
        f = [];
        if (ds.dp_order(i,j) == 1)
            f(1) = i;
            if(check_deadlock(f,j) == 1)
                ds.dp_order(i,j) = 0;
            end
        end

    end
end

%check dependence deadlock
function r = check_deadlock(f,j)
global ds;
r = 0;
tmp_r = 0;
```

```
[~,b] = size(f);

for m = 1 : 5
    if(ds.dp_order(j,m) == 1)
        for n = 1 : b
            if(f(n) == m)
                ds.dp_order(j,m) = 0;
                tmp_r = 1 ;
            end
        end

        if(tmp_r ~= 1)
            f(b + 1) = j;
            r = check_deadlock(f,m);
            if(r == 1)
                break;
            end
        end

    end
end
```

```
function [ ] = Dataset_init_dep_or( )

global ds;
ds.dp_or = [];

a = ds.req * ds.dp_or_per;
ra_or_density = a / (ds.req * ds.req);
```

```
8
9  ds.ra_dep_or_density = ra_or_density;
10 ds.dp_or = sprandsym(ds.req,ra_or_density);
11 ds.dp_or(find(ds.dp_or~=0)) = 1;
12 ds.dp_or(find(ds.dp_and~=0)) = 0;
13 ds.dp_or = triu(ds.dp_or,1);
14 ds.dp_or = full(ds.dp_or);
```

```
1  function [ ] = Dataset_init_dep_cost( )
2
3  global ds;
4  ds.dp_cost = [];
5
6  a = ds.req * ds.dp_cost_per;
7  ra_cost_density = a / (ds.req * ds.req);
8  ds.ra_dep_cost_density = ra_cost_density;
9
10 ds.dp_cost = sprand(ds.req,ds.req,ds.ra_dep_cost_density);
11
12 ds.dp_cost(find(ds.dp_cost~=0)) = 1;
13 ds.dp_cost = full(ds.dp_cost);
14
15 for i = 1 : ds.req
16     for j = 1 : ds.req
17         if(ds.dp_cost(i,j) == 1)
18             m = round(1 + (4 - 1) * rand(1));
19             if(m == 1)
20                 ds.dp_cost(i,j) = -0.2;
21             elseif(m == 2)
```

```
                ds.dp_cost(i,j) = -0.4;
            elseif(m == 3)
                ds.dp_cost(i,j) = 0.2;
            elseif(m == 4)
                ds.dp_cost(i,j) = 0.4;
            end
        end
    end
end
```

```
function [ ] = Dataset_init_dep_and( )

global ds;
ds.dp_and = [];

a = ds.req * ds.dp_and_per;
ra_and_density = a / (ds.req * ds.req);

ds.ra_dep_and_density = ra_and_density;

ds.dp_and = sprandsym(ds.req,ra_and_density);
ds.dp_and(find(ds.dp_and~=0)) = 1;
ds.dp_and = triu(ds.dp_and,1);

ds.dp_and = full(ds.dp_and);
```

```
function [  ] = Dataset(density,cus, req,dp_and_per, ...
                        dp_or_per,dp_order_per,...
                        dp_value_per,dp_cost_per)
```

```
% create dataset according to number of requirements, customer, density
% and dependence relationship

global ds;
ds.data = [];
ds.cost = [];
ds.weight = [];
ds.dp_and_per = dp_and_per;
ds.dp_or_per = dp_or_per;
ds.dp_order_per = dp_order_per;
ds.dp_value_per = dp_value_per;
ds.dp_cost_per = dp_cost_per;

switch density
    case 1
        id = 0.01; ad = 0.33;
    case 2
        id = 0.34; ad = 0.66;
    case 3
        id = 0.98; ad = 0.98;
end

switch req
    case 1
        iM = 1; aM = 100;
    case 2
        iM = 101; aM = 250;
    case 3
```

```
32          iM = 251; aM = 600;
33  end
34
35  switch cus
36      case 1
37          im = 2; am = 5;
38      case 2
39          im = 6; am = 20;
40      case 3
41          im = 21; am = 50;
42  end
43
44  ds.density = id + (ad - id) * rand(1);
45  ds.req = round(iM + (aM - iM) * rand(1));
46  ds.cus = round(im + (am - im) * rand(1));
47
48  while ds.req <= ds.cus
49      ds.req=round(iM + (aM - iM) * rand(1));
50      ds.cus=round(im + (am - im) * rand(1));
51  end
52
53  % init dataset
54  Dataset_init_value();
55  Dataset_clean_useless_cus();
56  Dataset_init_cost();
57  Dataset_init_weight();
58
59  % init dep data
```

```
Dataset_init_dep_and();
Dataset_init_dep_or();
Dataset_init_dep_order();
Dataset_init_dep_value();
Dataset_init_dep_cost();
```

```
%% exp1
clean_exp();

NSGA2(POPULATION,GENERATION,OBJECTIVE,DEP_VALUE_COST);

name = ...
    sprintf('Population: %d, Generation: %d, Objectives %d, No Dependence',...
    POPULATION,GENERATION,OBJECTIVE);
NSGA2_display(nsga2.chr_front,ds,1,200,'+b', name);
```

```
%% exp2
clean_exp;

NSGA2(POPULATION,GENERATION,OBJECTIVE,DEP_VALUE_COST);

and_all = NSGA2_filter_dep_and(nsga2.chr_set);
or_all = NSGA2_filter_dep_or(nsga2.chr_set);
order_all = NSGA2_filter_dep_order(nsga2.chr_set);

and_or_all = NSGA2_filter_dep_or(and_all);
and_or_order_all = NSGA2_filter_dep_order(and_or_all);

and_front = NSGA2_filter_front_raw(and_all);
```

```
or_front = NSGA2_filter_front_raw(or_all);
order_front = NSGA2_filter_front_raw(order_all);
and_or_order_front = NSGA2_filter_front_raw(and_or_order_all);

c_and = Dataset_count_dep_and();
c_or = Dataset_count_dep_or();
c_order = Dataset_count_dep_order();
c_all = c_and + c_or + c_order;
```

```
%% exp3
clean_exp();

NSGA2(POPULATION,GENERATION,OBJECTIVE,DEP_VALUE_COST);

and_all = NSGA2_filter_dep_and(nsga2.archive);
or_all = NSGA2_filter_dep_or(nsga2.archive);
order_all = NSGA2_filter_dep_order(nsga2.archive);

and_or_all = NSGA2_filter_dep_or(and_all);
and_or_order_all = NSGA2_filter_dep_order(and_or_all);

and_front = NSGA2_filter_front_raw(and_all);
or_front = NSGA2_filter_front_raw(or_all);
order_front = NSGA2_filter_front_raw(order_all);
and_or_order_front = NSGA2_filter_front_raw(and_or_order_all);

c_and = Dataset_count_dep_and();
c_or = Dataset_count_dep_or();
c_order = Dataset_count_dep_order();
```

```
21 c_all = c_and + c_or + c_order;
22
23 name = sprintf('Archive, And Dependence: %d',c_and);
24 subplot(2,2,1);
25 NSGA2_display(nsga2.chr_front,ds,1,200,'+b', name);
26 NSGA2_display(and_front,ds,1,200,'or', name);
27
28 name = sprintf('Archive, Or Dependence: %d',c_or);
29 subplot(2,2,2);
30 NSGA2_display(nsga2.chr_front,ds,1,200,'+b',name);
31 NSGA2_display(or_front,ds,1,200,'or', name);
32
33 name = sprintf('Archive, Order Dependence: %d',c_order);
34 subplot(2,2,3);
35 NSGA2_display(nsga2.chr_front,ds,1,200,'+b', name);
36 NSGA2_display(order_front,ds,1,200,'or', name);
37
38 name = sprintf('Archive, And-Or-Order Dependence: %d', c_all);
39 subplot(2,2,4);
40 NSGA2_display(nsga2.chr_front,ds,1,200,'+b', name);
41 NSGA2_display(and_or_order_front,ds,1,200,'or', name);
```

```
1 %% exp4
2 clean_exp;
3
4 NSGA2(POPULATION,GENERATION,OBJECTIVE,DEP_VALUE_COST);
5
6 and_all = NSGA2_filter_dep_and(nsga2.chr_set);
7 or_all = NSGA2_filter_dep_or(nsga2.chr_set);
```

```
order_all = NSGA2_filter_dep_order(nsga2.chr_set);

and_or_all = NSGA2_filter_dep_or(and_all);
and_or_order_all = NSGA2_filter_dep_order(and_or_all);

and_front=NSGA2_filter_front_raw(and_all);
or_front=NSGA2_filter_front_raw(or_all);
order_front=NSGA2_filter_front_raw(order_all);
and_or_order_front=NSGA2_filter_front_raw(and_or_order_all);

and_all_archive = NSGA2_filter_dep_and(nsga2.archive);
or_all_archive = NSGA2_filter_dep_or(nsga2.archive);
order_all_archive = NSGA2_filter_dep_order(nsga2.archive);

and_or_all_archive = NSGA2_filter_dep_or(and_all_archive);
and_or_order_all_archive = NSGA2_filter_dep_order(and_or_all_archive);

and_front_archive = NSGA2_filter_front_raw(and_all_archive);
or_front_archive = NSGA2_filter_front_raw(or_all_archive);
order_front_archive = NSGA2_filter_front_raw(order_all_archive);
and_or_order_front_archive = NSGA2_filter_front_raw(and_or_order_all_archive);

c_and = Dataset_count_dep_and();
c_or = Dataset_count_dep_or();
c_order = Dataset_count_dep_order();
c_all = c_and + c_or + c_order;
```

```
%% exp5
clean_exp;
```

```
3
4  DEP_VALUE_COST = 0;
5  NSGA2(POPULATION,GENERATION,OBJECTIVE,DEP_VALUE_COST);
6
7  name = sprintf('No Dependence');
8  subplot(2,2,1);
9  NSGA2_display(nsga2.chr_front,ds,1,150,'+b', name);
10
11 DEP_VALUE_COST = 1;
12 NSGA2(POPULATION,GENERATION,OBJECTIVE,DEP_VALUE_COST);
13
14 c_value = Dataset_count_dep_value();
15 name = sprintf('Value Dependency');
16 subplot(2,2,2);
17 NSGA2_display(nsga2.chr_front,ds,1,150,'+b', name);
18
19 DEP_VALUE_COST = 2;
20 NSGA2(POPULATION,GENERATION,OBJECTIVE,DEP_VALUE_COST);
21 c_cost = Dataset_count_dep_cost();
22 name = sprintf('Cost Dependency');
23 subplot(2,2,3);
24 NSGA2_display(nsga2.chr_front,ds,1,150,'+b', name);
25
26 DEP_VALUE_COST = 3;
27 NSGA2(POPULATION,GENERATION,OBJECTIVE,DEP_VALUE_COST);
28 c_value = Dataset_count_dep_value();
29 c_cost = Dataset_count_dep_cost();
30 c_all = c_and + c_value + c_cost;
```

```
name = sprintf('Value-Cost Dependency');
subplot(2,2,4);
NSGA2_display(nsga2.chr_front,ds,1,150,'+b', name);
```

```
function [ result ] = NSGA2_filter_dep_and( in_chr_set )

global ds;
result = [];
id = 1;
[a,b] = size(in_chr_set);

for m = 1 : a
    chr_set = in_chr_set(m,:);
    pass = 1;
    for i = 1 : ds.req
        for j = 1 : ds.req

            if ds.dp_and(i,j) == 1
                if chr_set(i) ~= chr_set(j)
                    pass = 0;
                end
            end

        end
    end

    if(pass == 1)
        result(id,:) = chr_set;
        id = id+1;
```

```
    end
end

[a,b] = size(result);
if(a == 0)
    fprintf('Empty result !!! \n');
end
```

```
function [ result ] = NSGA2_filter_dep_or( in_chr_set )

global ds;
result = [];
id = 1;
[a,b] = size(in_chr_set);

for m = 1 : a
    chr_set = in_chr_set(m,:);
    pass = 1;
    for i = 1 : ds.req
        for j =1 : ds.req
            if (ds.dp_or(i,j) == 1)
                if(chr_set(i) == chr_set(j) && chr_set(i) ~= 0)
                    pass = 0;
                end
            end
        end
    end
    if(pass == 1)
      result(id,:) = chr_set;
```

```
      id = id+1;
    end
end

[a,b] = size(result);
if(a == 0)
    fprintf('Empty result !!! \n');
end
```

```
function [ result ] = NSGA2_filter_dep_order( in_chr_set )

global ds;
result = [];
id = 1;
[a,b] = size(in_chr_set);

for m = 1 : a
    chr_set = in_chr_set(m,:);
    pass = 1;

    for i = 1 : ds.req
        for j = 1 : ds.req

            if (ds.dp_and(i,j) == 1)
                if(chr_set(i) == 0 && chr_set(j) == 1)
                    pass = 0;
                end
            end
        end
```

```
    end

    if(pass == 1)
      result(id,:) = chr_set;
      id = id + 1;
    end
end

[a,b] = size(result);
if(a == 0)
    fprintf('Empty result !!! \n');
end
```

```
function [ hsca ] = NSGA2_display(chr_set,ds,i,si,mark,name)

yy = 1;
figure(i)
hold on
[a,b] = size(chr_set);

for i = 1 : a
    hsca = scatter(chr_set(i,ds.req + 1),chr_set(i,ds.req + 2),si,mark);
end

title(name, 'FontSize',20,'FontWeight','b');
xlabel('Value','FontSize',20);
ylabel('-Cost','FontSize',20);

set(gca,'FontSize',14);
```

```
%% Draw results by NSGA-II with AND dependence
clean_exp;

and_front = NSGA2_filter_dep_and(nsga2.chr_front);
name = sprintf('And Dependency');

hsca = NSGA2_display(nsga2.chr_front,ds,1,200,'+k', name);
hscb = NSGA2_display(and_front,ds,1,200,'ok', name);

hs(1) = hsca;
hs(2) = hscb;
legend(hs,'NSGA-II without dependency','NSGA-II with dependency');

grid on;
```

```
%% Draw results by Archived based NSGA-II with AND dependence
clean_exp;

and_all = NSGA2_filter_dep_and(nsga2.archive);
and_front_arc = NSGA2_filter_front_raw(and_all);
and_front = NSGA2_filter_dep_and(nsga2.chr_front);

name = sprintf('And Dependency');
hsca = NSGA2_display(and_front_arc,ds,1,100,'*k',name);
hscb = NSGA2_display(and_front,ds,1,200,'ok', name);

hs(1) = hsca;
hs(2) = hscb;
```

```
legend(hs,...
    'Archive based NSGA-II with dependency','NSGA-II with dependency');
grid on;
```

```
%% Draw results by NSGA-II with OR dependence
clean_exp;

or_front = NSGA2_filter_dep_or(nsga2.chr_front);
name = sprintf('Or Dependency');

hsca = NSGA2_display(nsga2.chr_front,ds,1,200,'+k', name);
hscb = NSGA2_display(or_front,ds,1,200,'ok', name);

hs(1) = hsca;
hs(2) = hscb;
legend(hs,'NSGA-II without dependency','NSGA-II with dependency');
grid on;
```

```
%% Draw results by Archived based NSGA-II with OR dependence
clean_exp;

or_all = NSGA2_filter_dep_or(nsga2.archive);
or_front_arc = NSGA2_filter_front_raw(or_all);
or_front = NSGA2_filter_dep_or(nsga2.chr_front);

name = sprintf('Or Dependency');
hsca = NSGA2_display(or_front_arc,ds,1,100,'*k', name);
hscb = NSGA2_display(or_front,ds,1,200,'ok', name);

```

```
hs(1) = hsca;
hs(2) = hscb;
legend(hs,...
    'Archive based NSGA-II with dependency','NSGA-II with dependency');
grid on;
```

```
%% Draw results by NSGA-II with Precedence dependence
clean_exp;

order_front = NSGA2_filter_dep_order(nsga2.chr_front);

name = sprintf('Precedence Dependency');
hsca = NSGA2_display(nsga2.chr_front,ds,1,200,'+k', name);
hscb = NSGA2_display(order_front,ds,1,200,'ok', name);

hs(1) = hsca;
hs(2) = hscb;
legend(hs,'NSGA-II without dependency','NSGA-II with dependency');
grid on;
```

```
%% Draw results by Archived based NSGA-II with Precedence dependence
clean_exp;

order_all = NSGA2_filter_dep_order(nsga2.archive);
order_front_arc = NSGA2_filter_front_raw(order_all);
order_front = NSGA2_filter_dep_order(nsga2.chr_front);

name = sprintf('Precedence Dependency');
hsca = NSGA2_display(order_front_arc,ds,1,100,'*k', name);
```

```
hscb = NSGA2_display(order_front,ds,1,200,'ok', name);

hs(1) = hsca;
hs(2) = hscb;
legend(hs,...
    'Archive based NSGA-II with dependency','NSGA-II with dependency');
grid on;
```

```
%% Draw results by NSGA-II with AND, OR & Precedence dependence
clean_exp;

and_front = NSGA2_filter_dep_and(nsga2.chr_front);
and_or_front = NSGA2_filter_dep_or(and_front);
and_or_order_front = NSGA2_filter_dep_order(and_or_front);

name = sprintf('And, Or and Precedence Dependencies');
hsca = NSGA2_display(nsga2.chr_front,ds,1,200,'+k', name);
hscb = NSGA2_display(and_or_order_front,ds,1,200,'ok', name);

hs(1) = hsca;
hs(2) = hscb;
legend(hs,...
    'NSGA-II without dependencies','NSGA-II with dependencies');
grid on;
```

```
%% Draw results by Archived based NSGA-II with AND, OR & Precedence dependence
clean_exp;

and_front = NSGA2_filter_dep_and(nsga2.chr_front);
```

```
and_or_front = NSGA2_filter_dep_or(and_front);
and_or_order_front = NSGA2_filter_dep_order(and_or_front);

and_all = NSGA2_filter_dep_and(nsga2.archive);
and_or_all = NSGA2_filter_dep_or(and_all);
and_or_order_all = NSGA2_filter_dep_or(and_or_all);
and_or_order_front_arc = NSGA2_filter_front_raw(and_or_order_all);

name = sprintf('And, Or and Precedence Dependencies');
hsca = NSGA2_display(and_or_order_front_arc,ds,1,100,'*k', name);
hscb = NSGA2_display(and_or_order_front,ds,1,200,'ok', name);

hs(1) = hsca;
hs(2) = hscb;
legend(hs,...
    'Archive based NSGA-II with dependencies','NSGA-II with dependencies');
grid on;
```

```
%% Draw results by NSGA-II and Archived based NSGA-II with all dependences
clean_exp;

and_front = NSGA2_filter_dep_and(nsga2.chr_front);
and_or_front = NSGA2_filter_dep_or(and_front);
and_or_order_front = NSGA2_filter_dep_order(and_or_front);

and_all = NSGA2_filter_dep_and(nsga2.archive);
and_or_all = NSGA2_filter_dep_or(and_all);
and_or_order_all = NSGA2_filter_dep_or(and_or_all);
and_or_order_front_arc = NSGA2_filter_front_raw(and_or_order_all);
```

```
12
13 name = sprintf('And, Or and Precedence Dependencies');
14 hsca = NSGA2_display(and_or_order_front_arc,ds,1,100,'*k', name);
15 hscb = NSGA2_display(and_or_order_front,ds,1,200,'ok', name);
16 hscc = NSGA2_display(nsga2.chr_front,ds,1,200,'+k', name);
17
18 hs(1) = hsca;
19 hs(2) = hscb;
20 hs(3) = hscc;
21 legend(hs,'Archive based NSGA-II with dependencies',...
22     'NSGA-II with dependencies','NSGA-II without dependencies');
23 grid on;
```

```
1 %%Multi-stakeholder Tensioning Analysis
2 % K Customer M requirements s dataset
3 load(s);
4 [K,M] = size(R);
5 K = K - 1;
6
7 ss = [];
8 budget = 0;
9 for i = 1 : M
10     if sum(R(1 : K,i)) > 0
11         ss(i) = 1;
12     else ss(i) = 0;
13     end
14 end
15
16 for i = 1 : M
```

```
17      budget = budget + R(K + 1,i) * ss(i);
18  end
19  budget = budget * 0.7;
20
21  path(path,'/Users/Yuanyuan/Documents/MATLAB/Tension/random');
22  path(path,'/Users/Yuanyuan/Documents/MATLAB/Tension/NSGA-II');
23  path(path,'/Users/Yuanyuan/Documents/MATLAB/Tension/two-archive');
24
25  front_randomm = randomm(K,M,budget,s);
26  front = front_randomm(:,1 : M + K);
27
28  [a,temp] = size(front);
29
30  front_two_archive = two_archive(K,M,budget,s);
31  [b,temp] = size(front_two_archive);
32  front(a+1 : a+b,:) = front_two_archive(:,1 : M+K);
33
34  front_nsga2 = nsga_2(K,M,budget,s);
35  [c,temp] = size(front_nsga2);
36  front(a+b+1:a+b+c,:) = front_nsga2(:,1:M+K);
37  front(1:a,M+K+2) = 2;
38  front(a+1:a+b,M+K+2) = 3;
39  front(a+b+1:a+b+c,M+K+2) = 4;
40
41
42  F = [];
43  [R,temp] = size(front);
44
```

```
individual = [];
for i = 1 : R
    front(i,M+K+1) = 1;
end

for i = 1 : R
    if front(i,M+K+1) == 1
        individual(i).n = 0;

        for j = 1 : R
            if front(j,M+K+1) == 1
                dom_less = 0;
                dom_equal = 0;
                dom_more = 0;

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

                if dom_less == 0 && dom_equal ~= K
                    front(j,M+K+1) = 0;

```

```
                elseif dom_more == 0 && dom_equal ~= K
                    individual(i).n = individual(i).n + 1;
                end

            end
            if individual(i).n == 0
                front(i,M + K + 1) = 1;
            else
                front(i,M + K + 1) = 0;
            end
        end
    end
end

yy = 1;
f_random = 0;
f_two_archive = 0;
f_nsga2 = 0;
for i = 1 : R
    if front(i,M+K+1) == 1
        F(yy,:) = front(i,:);
        if front(i,M+K+2) == 2
            f_random = f_random+1;
        end
        if front(i,M+K+2) == 3
            f_two_archive = f_two_archive+1;
        end
        if front(i,M+K+2) == 4
```

```
101             f_nsga2 = f_nsga2+1;
102         end
103         yy = yy + 1;
104     end
105 end
106
107 %calculate diversity of pareto front
108 div_randomm = dalta(front_randomm,K,M,F);
109 div_two_archive = dalta(front_two_archive,K,M,F);
110 div_nsga2 = dalta(front_nsga2,K,M,F);
111
112 save(s2,'a','b','c','f_random','f_two_archive','f_nsga2',...
113     'F','div_randomm','div_two_archive','div_nsga2','-ASCII');
114
115
116 %calculate convergence distance of pareto front
117 for i = 1 : K
118     range(i) = max(F(:,M+i)) - min(F(:,M+i));
119 end
120 [fs,variable] = size(F);
121 su = 0;
122 for i = 1 : a
123     sum_min=10000;
124
125     for j = 1 : fs
126         sum1 = 0;
127         for k = 1 : K
128             if range(k) > 0
```

```
129                 sum1 = sum1+((front_randomm(i,M+k)-F(j,M+k))/range(k))^2;
130             else sum1 = sum1 + 0;
131             end
132             sum2 = sqrt(sum1);
133             if sum2 < sum_min
134                 sum_min = sum2;
135             end
136         end
137         su = su + sum_min;
138     end
139 end
140 dist_randomm = su / a;
141
142 su = 0;
143 for i = 1 : b
144     sum_min = 10000;
145     for j = 1 : fs
146         sum1 = 0;
147         for k = 1 : K
148             sum1 = sum1+((front_two_archive(i,M+k)-F(j,M+k))/range(k))^2;
149         end
150         sum2 = sqrt(sum1);
151         if sum2 < sum_min
152             sum_min = sum2;
153         end
154     end
155     su = su + sum_min;
156 end
```

```
dist_two_archive = su / b;

su = 0;
for i = 1 : c
    sum_min = 10000;

    for j = 1 : fs
        sum1 = 0;
        for k = 1 : K
            sum1 = sum1 + ((front_nsga2(i,M+k)-F(j,M+k))/range(k))^2;
        end
        sum2 = sqrt(sum1);
        if sum2 < sum_min
            sum_min = sum2;
        end
    end
    su = su + sum_min;
end
dist_nsga2 = su / c;

save(s2,'a','b','c','f_random','f_two_archive','f_nsga2',...
    'std_div_randomm','std_div_two_archive','std_div_nsga2',...
    'dist_randomm','dist_two_archive','dist_nsga2','-ASCII');
```

```
function [ dalta ] = dalta(x,K,M,F)
%% calculate diversity of Pareto fronts
% K customer as well as objectives M requirements F reference front x front
[aa,temp] = size(x); % aa number of solutions

```

```
y = []; % original distance vector
res = [];
dis = [];

if aa == 1
    dis(1) = 0;
else
    y = pdist(x(:,M+1 : M+K));
    dis(1) = min(y(1 : aa-1));
    p = 1;
    p2 = 0;
    res(1,:) = y(1 : aa-1);
    for i = 2 : aa - 1
        res(i,1 : i-1) = res(1 : i-1,i-1);
        p = p + aa - i + 1;
        p2 = p + aa - i - 1;
        res(i,i : aa - 1) = y(1,p : p2);
        dis(i) = min(res(i,1 : aa - 1));
    end
end

avg_d = mean(dis);

res_w=[];
f_w=[];

for i = 1 : K % find the extreme solutions on F and x
    [~,b] = sort(x(:,M+i),'descend'); % aa solutions in total
```

```
    [~,d] = sort(F(:,M+i),'descend');
    res_w(i,1:K) = x(b(1),M+1:M+K);
    f_w(i,1:K) = F(d(1),M+1:M+K);
end

d_extr = [];
dis_v = [];
for i = 1 : K % compute the distance between extremes
    dis_v(1,:) = res_w(i,:);
    dis_v(2,:) = f_w(i,:);

    d_extr(i) = pdist(dis_v);
end

sum_d_extr = sum(d_extr);

if aa == 1
    st = 0;
else
    for i = 1 : aa-1
        st(i) = abs(avg_d-dis(i));
    end
end
dalta = (sum_d_extr + sum(st))/(sum_d_extr + (aa-1)*avg_d);
```

```
function [ index ] = NSGA2(pop,gen,obj,index)
% NSGA2 main function

global nsga2;
```

```
5  global ds;
6
7  nsga2.pop = pop;
8  nsga2.gen = gen;
9  nsga2.obj = obj;
10 nsga2.chr_set = [];
11
12 %Initialise the population
13 NSGA2_init_var(pop);
14
15 nsga2.chr_set = NSGA2_non_domination_sort(nsga2.chr_set,ds.req);
16 nsga2.archive = [];
17
18 nsga2.archive=vertcat(nsga2.archive,nsga2.chr_set);
19
20 for i = 1 : nsga2.gen
21     pool = round(nsga2.pop/2);
22     tour = 5;
23     parent_chromosome = NSGA2_tournament_selection(pool,tour);
24
25     offspring_chromosome = ...
26         NSGA2_genetic_operator(parent_chromosome,ds.req);
27
28     [main_pop,~] = size(nsga2.chr_set);
29     [offspring_pop,~] = size(offspring_chromosome);
30     intermediate_chromosome(1 : main_pop,:) = nsga2.chr_set;
31     intermediate_chromosome(main_pop + 1 : main_pop + offspring_pop,...
32         1 : nsga2.obj + ds.req) = offspring_chromosome;
```

```
33
34      intermediate_chromosome = ...
35          NSGA2_non_domination_sort(intermediate_chromosome,ds.req);
36
37      nsga2.chr_set = ...
38          NSGA2_replace_chromosome(intermediate_chromosome,ds.req,pop);
39      nsga2.archive = vertcat(nsga2.archive,nsga2.chr_set);
40  end
41
42  [a,b] = NSGA2_filter_front_raw(nsga2.chr_set);
43  nsga2.chr_front(index:index+b-2,:) = a;
44  index = index + b - 1;
```

```
1   function [ ] = NSGA2_init_var(pop)
2   % intialize population
3
4   global nsga2;
5   global ds;
6
7   obj_flag = ds. req + nsga2.obj;
8
9   for i = 1 : pop
10      % Initialize the decision variables
11      for j = 1 : ds.req
12          nsga2.chr_set(i,j) = round(rand(1));
13      end
14      % Evaluate the objective function
15      nsga2.chr_set(i,ds.req + 1: obj_flag) = ...
16          NSGA2_eval_obj(nsga2.chr_set(i,:));
```

```

end
```

```
function [ f ] = NSGA2_non_domination_sort(x,V)
% sort the popultion based on non-domination

[N,temp] = size(x);
M = 3;
front = 1;

F(front).f = [];
individual = [];
for i = 1 : N
    individual(i).n = 0;
    individual(i).p = [];
% find the first pareto front
    for j = 1 : N
        dom_less = 0;
        dom_equal = 0;
        dom_more = 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
```

```
27          end
28
29
30          if dom_less == 0 && dom_equal ~= M
31              individual(i).p = [individual(i).p j];
32          elseif dom_more == 0 && dom_equal ~= M
33              individual(i).n = individual(i).n + 1;
34          end
35      end
36      if individual(i).n == 0
37          x(i,M + V + 1) = 1;
38          F(front).f = [F(front).f i];
39      end
40  end
41  % find the subsequent fronts
42  while ~isempty(F(front).f)
43      Q = [];
44      for i = 1 : length(F(front).f)
45          if ~isempty(individual(F(front).f(i)).p)
46              for j = 1 : length(individual(F(front).f(i)).p)
47                  individual(individual(F(front).f(i)).p(j)).n = ...
48                      individual(individual(F(front).f(i)).p(j)).n - 1;
49                  if individual(individual(F(front).f(i)).p(j)).n == 0
50                      x(individual(F(front).f(i)).p(j),M + V + 1) = ...
51                          front + 1;
52                      Q = [Q individual(F(front).f(i)).p(j)];
53                  end
54              end
```

```
55          end
56      end
57      front =  front + 1;
58      F(front).f = Q;
59  end
60  [temp,index_of_fronts] = sort(x(:,M + V + 1));
61  for i = 1 : length(index_of_fronts)
62      sorted_based_on_front(i,:) = x(index_of_fronts(i),:);
63  end
64  current_index = 0;
65  % find the crowding distance for each individual in each front
66  for front = 1 : (length(F) - 1)
67      objective = [];
68      distance = 0;
69      y = [];
70      previous_index = current_index + 1;
71      for i = 1 : length(F(front).f)
72          y(i,:) = sorted_based_on_front(current_index + i,:);
73      end
74      current_index = current_index + i;
75  % sort each individual based on the objective
76      sorted_based_on_objective = [];
77
78      for i = 1 : M
79          [sorted_based_on_objective, index_of_objectives] = ...
80              sort(y(:,V + i));
81          sorted_based_on_objective = [];
82          for j = 1 : length(index_of_objectives)
```

```
83              sorted_based_on_objective(j,:) = y(index_of_objectives(j),:);
84          end
85          f_max = ...
86              sorted_based_on_objective(length(index_of_objectives), V + i)
87          f_min = sorted_based_on_objective(1, V + i);
88          y(index_of_objectives(length(index_of_objectives)),M + V + 1 + i)...
89              = Inf;
90          y(index_of_objectives(1),M + V + 1 + i) = Inf;
91          for j = 2 : length(index_of_objectives) - 1
92              next_obj    = sorted_based_on_objective(j + 1,V + i);
93              previous_obj    = sorted_based_on_objective(j - 1,V + i);
94              if (f_max - f_min == 0)
95                  y(index_of_objectives(j),M + V + 1 + i) = Inf;
96              else
97                  y(index_of_objectives(j),M + V + 1 + i) = ...
98                      (next_obj - previous_obj)/(f_max - f_min);
99              end
100         end
101     end
102     distance = [];
103     distance(:,1) = zeros(length(F(front).f),1);
104     for i = 1 : M
105         distance(:,1) = distance(:,1) + y(:,M + V + 1 + i);
106     end
107     y(:,M + V + 2) = distance;
108     y = y(:,1 : M + V + 2);
109     z(previous_index:current_index,:) = y;
110 end
```

```
f = z();
```

```
function [ f ] = NSGA2_genetic_operator(parent_chromosome,V)
% generate offsprings through crossover and mutation operators

[N,~] = size(parent_chromosome);

M = 3;
p = 1;

for i = 1 : N
    child_1 = [];
    child_2 = [];
% select parents
    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
```

```
28      parent_1 = parent_chromosome(parent_1,:);
29      parent_2 = parent_chromosome(parent_2,:);
30  % crossover
31      if rand(1) < 0.8
32          u=1 + round(rand(1)*(V-1));
33          for j = 1 : u
34              child_1(j) = parent_1(j);
35              child_2(j) = parent_2(j);
36          end
37          for j= u+1 : V
38              child_1(j) = parent_2(j);
39              child_2(j) = parent_1(j);
40          end
41      else
42          for j = 1 : V
43              child_1(j) = parent_1(j);
44              child_2(j) = parent_2(j);
45          end
46      end
47  % mutation
48      for k = 1 : V
49          if rand(1) < 1/V
50              if child_1(k) == 1
51                  child_1(k) = 0;
52              else
53                  child_1(k) = 1;
54              end
55              if child_2(k) == 1
```

```
                child_2(k) = 0;
            else
                child_2(k) = 1;
            end
        end
    end
    child_1(:,V + 1: M + V) = NSGA2_eval_obj(child_1);
    child_2(:,V + 1: M + V) = NSGA2_eval_obj(child_2);

    child(p,:) = child_1;
    child(p+1,:) = child_2;
    p = p + 2;
end
f = child;
```

```
function [ f ] = NSGA2_replace_chromosome(intermediate_chromosome,V,pop)
%replace the chromosome according to pareto front rank and crowding
%distance

[N,temp] = size(intermediate_chromosome);
M = 3;

[~,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));
```

```
15
16 previous_index = 0;
17 for i = 1 : max_rank
18     current_index = max(find(sorted_chromosome(:,M + V + 1) == i));
19     if current_index > pop
20         remaining = pop - previous_index;
21         temp_pop = ...
22             sorted_chromosome(previous_index + 1 : current_index, :);
23         [~,temp_sort_index] = ...
24             sort(temp_pop(:, M + V + 2),'descend');
25         for j = 1 : remaining
26             f(previous_index + j,:) = temp_pop(temp_sort_index(j),:);
27         end
28         return;
29     elseif current_index < pop
30         f(previous_index + 1 : current_index, :) = ...
31             sorted_chromosome(previous_index + 1 : current_index, :);
32     else
33         f(previous_index + 1 : current_index, :) = ...
34             sorted_chromosome(previous_index + 1 : current_index, :);
35         return;
36     end
37     previous_index = current_index;
38 end
```

```
1 function [ front,yy ] = NSGA2_filter_front_raw(chr_set)
2 % find the Pareto front from one run
3
4 global nsga2;
```

```
5  global ds;
6
7  yy = 1;
8
9  [~, b] = size(chr_set);
10
11 chr_set(:,b) = [];
12 chr_set(:,b-1) = [];
13
14 chr_set = NSGA2_non_domination_sort(chr_set,ds.req);
15 [a, b] = size(chr_set);
16 for i = 1 : a
17     if chr_set(i,ds.req + nsga2.obj + 1) == 1
18         flag = 0;
19
20         for j = 1 : yy-1
21             if (chr_set(i,ds.req+1) == front(j,ds.req+1)...
22                     && chr_set(i,ds.req+2) == front(j,ds.req+2)...
23                     && chr_set(i,ds.req+3) == front(j,ds.req+3))
24                 flag = 1;
25                 break;
26             end
27         end
28
29         if flag == 0
30             front(yy,:) = chr_set(i,:);
31             yy = yy + 1;
32
```

```
        end
    end
end
```

```
function [ fitness ] = NSGA2_eval_obj(solution)
% evalute each individual based on fitness function
global ds;

%% fitness function one
sum1 = 0;
for i = 1 : ds.req
    for j = 1 : ds.cus
        sum1 = sum1 + ds.now(j,i) * solution(i);
    end
end
fitness(1) = sum1;
%% fitness function two
sum2 = 0;
for i = 1 : ds.req
    for j = 1 : ds.cus
        sum2 = sum2 + ds.future(j,i) * solution(i);
    end
end
fitness(2) = sum2;
%% fitness function three
sum3 = 0;
for i = 1 : ds.req
    sum3 = sum3 + ds.cost(i) * solution(i);
end
```

```
fitness(3) = -sum3;
```

```
function [ ] = NSGA2_find_pareto()
% find the Pareto front from all the iterations
global nsga2;
global ds;

front = [];
[R, variable] = size(nsga2.chr_front);
individual = [];

for i = 1 : R
    if nsga2.chr_front(i,ds.req+4) == 1
        individual(i).n = 0;

        for j = 1 : R
            if nsga2.chr_front(j,ds.req+4) == 1
                dom_less = 0;
                dom_equal = 0;
                dom_more = 0;

                for k = 1 : 3
                    if nsga2.chr_front(i, ds.req+k)...
                            < nsga2.chr_front(j,ds.req+k)
                        dom_less = dom_less + 1;
                    elseif nsga2.chr_front(i, ds.req+k)...
                            == nsga2.chr_front(j, ds.req+k)
                        dom_equal = dom_equal + 1;
                    else
```

```
                        dom_more = dom_more + 1;
                    end
                end

                if dom_less == 0 && dom_equal ~= 3
                    nsga2.chr_front(j,ds.req+4) = 0;
                elseif dom_more == 0 && dom_equal ~= 3
                    individual(i).n = individual(i).n + 1;
                end
            end
        end
        if individual(i).n == 0
            nsga2.chr_front(i,ds.req+4) = 1;
        else
            nsga2.chr_front(i,ds.req+4) = 0;
        end
    end
end

yy=1;
for i = 1 : R
    if nsga2.chr_front(i,ds.req+4) == 1
        flag = 0;
        for j = 1 : yy-1
          if nsga2.chr_front(i,ds.req+1) == nsga2.chr_front(j,ds.req+1) &&...
              nsga2.chr_front(i,ds.req+2)==nsga2.chr_front(j,ds.req+2) &&...
              nsga2.chr_front(i,ds.req+3)==nsga2.chr_front(j,ds.req+3)
            flag = 1;
```

```
                break;
            end
        end
        if flag == 0
            front(yy,:) = nsga2.chr_front(i,:);
            yy = yy + 1;
        end
    end
end

nsga2.chr_front = front;
```

```
function f=two_archive(M,V,budget,s)
%% The two-archive algorithm
pop = 300;
gen = 100;

% Initialize the population
chromosome = initialize_variables(pop,M,V,budget,s);
chromosome = non_domination_solution(chromosome,M,V);

CA = [];
DA = [];

for i = 1 : gen

    CA_DA = collect_non_dominated_archives(CA,DA,chromosome,M,V);

    [a,b] = size(CA_DA);
```

```
18      c = CA_DA(a,1);
19      d = CA_DA(a,2);
20
21      [C,temp] = size(CA);
22      [D,temp] = size(DA);
23
24      CA = CA_DA(1 : c, :);
25      DA = CA_DA(c+1 : c+d, :);
26      CA_DA = [];
27      [C,temp] = size(CA);
28      [D,temp] = size(DA);
29
30      limit = 500;
31
32      if (C+D) > limit
33          for ii = 1 : D
34              shortest = 1000;
35              for jj = 1 : C
36                  if DA(ii,V + M + 6) > Dist(jj,ii,CA,DA,M,V)
37                      DA(ii,V + M + 6) = Dist(jj,ii,CA,DA,M,V);
38                  end
39              end
40          end
41          pp = 1;
42
43          while (C+D) > limit
44              [~,index_sort] = sort(DA(:,V + M + 6));
45
```

```
46              for p = 1 : length(index_sort)
47                  if index_sort(p) == pp
48                      DA(p,:) = [];
49                      pp = pp + 1;
50                      break;
51                  end
52              end
53              [D, variable] = size(DA);
54          end
55      end
56
57      offspring_chromosome = ...
58          genetic_operator(chromosome,CA,DA,M,V, budget, s);
59      chromosome = non_domination_solution(offspring_chromosome,M,V);
60
61      [C, temp] = size(CA);
62      [D, temp] = size(DA);
63  end
64
65  save ca_solution.txt CA -ASCII
66  save da_solution.txt DA -ASCII
67
68  [C, temp] = size(CA);
69  [D, temp] = size(DA);
70
71  j = 1;
72  for i = 1 : C
73      flag = 0;
```

```
74
75      for jj = 1 : j-1
76          flag = 0;
77          for k = 1 : M
78              if CA(i,V+k) == f(jj,V+k)
79                  flag = flag+1;
80              end
81          end
82          if flag == M
83              break;
84          end
85      end
86
87      if flag < M
88          f(j,:) = CA(i,:);
89          j = j + 1;
90      end
91
92  end
93  [j,temp] = size(f);
94
95  for i = 1 : D
96      flag = 0;
97
98      for jj = 1 : j-1
99          flag = 0;
100         for k = 1 : M
101             if DA(i,V+k) == f(jj,V+k)
```

```
                flag = flag + 1;
            end
        end
        if flag == M
            break;
        end
    end

    if flag < M
        f(j+1,:) = DA(i,:);
        j = j + 1;
    end
end
save ('two_archive results','CA','DA');
```

```
function [f] = Dist(j,i,CA,DA,M,V)

sum = 0;
for k = 1 : M
    sum = sum + (DA(i,V+k)-CA(j,V+k))^2;
end
f = sqrt(sum);
```

```
function [f] = evaluate_objective(x,budget,s)

load(s);
f = [];
[m,M] = size(R);
m = m - 1;
```

```
sum3 = 0;

for i = 1 : M
    if sum(R(1:m,i)) > 0
        s(i) = 1;
    else s(i) = 0;
    end
    sum3 = sum3 + R(m + 1,i) * x(i) * s(i);
end

if (sum3 <= budget)
    for j = 1 : m
        sum1 = 0; sum2 = 0;
        for i = 1 : M

            sum1 = sum1 + R(j,i);
            sum2 = sum2 + R(j,i) * x(i);
        end

        f(j) = sum2 / sum1;
    end
else
    for j = 1 : m
        f(j) = 0;
    end
end
```

```
%% Greedy for Value/Cost trade-off Analysis
starttime=cputime;
```

```
3
4  clear;
5
6  pro = 1;
7  obj = 1;
8  dig = 0;
9  global le_greedy_value;
10 global le_greedy_ratio;
11
12 switch pro
13     case 1
14         load ('15_40.mat');
15     case 2
16         load ('50_80.mat');
17     case 3
18         load ('100_140.mat');
19     case 4
20         load ('100_20.mat')
21     case 5
22         load ('100_25.mat');
23     case 6
24         load ('2_200.mat');
25     case 9
26         load ('100_15.mat');
27     case 8
28         load ('100_30.mat');
29     case 7
30         load ('4_35.mat');
```

```
end

[m,M]=size(R);
m=m-1; % M requirements; m customers
dot = [];

if pro == 7
    C = ones(1,4);
end
for i = 1 : M
    dig = i + 0;
    dot = select_greedy(obj,dig,m,M,R,C);
    if obj == 1
        le_greedy_value = plot(dot(1),dot(2),'kx');
    elseif obj == 3
        le_greedy_ratio = plot(dot(1),dot(2),'k+');
    end
    hold on
    dot = [];
end
runningtime=cputime-starttime
```

```
function f = select_greedy (obj,dig,m,M,R,C)

f = [];

b = [];  % newly sorted array
ip = []; % the index of sorted array
summ = [];
```

```
ratio = [];

for i = 1 : M
    sum1 = 0;
    for j = 1 : m
        sum1 = sum1 + R(j,i) * C(j);
    end
    summ(i) = sum1;
end

max_value = max(summ);
min_value = min(summ);

max_cost = max(R(m+1,:));
min_cost = min(R(m+1,:));

for i = 1 : M
    noma_value(i) = (summ(i) - min_value) / (max_value - min_value);
    noma_cost(i) = (R(m+1,i) - min_cost) / (max_cost - min_cost);
    ratio(i) = noma_value(i) / noma_cost(i);
end

if obj == 1 % highest value
    [b,ip]=sort(summ);
    f(1) = sum(b(M-dig+1:M));
    x = zeros(1,M);
    for aa = M:-1:M- dig +1
        %for aa = 1 : dig
```

```
36          x(ip(aa)) = 1;
37      end
38      sum2=0;
39      for i = 1 : M
40          sum2=sum2+R(m+1,i)*x(i);
41      end
42      f(2) = -sum2;
43
44  elseif obj == 2 % lowest cost
45      [b,ip]=sort(R(m+1,:));
46
47      sum_value = 0;
48      for aa = 1 : dig
49          sum_value = sum_value + summ(ip(aa));
50      end
51      f(1) = sum_value;
52
53      sum2=0;
54      sum2=sum(b(1:dig));
55      f(2)= -sum2;
56
57  elseif obj == 3   % value/cost ratio
58      sum_value = 0;
59      sum_cost = 0;
60      [b,ip]= sort(ratio,'descend');
61      for aa = 1 : dig
62          sum_value = sum_value + summ(ip(aa));
63          sum_cost = sum_cost + R(m+1,ip(aa));
```

```
64      end
65      f(1) = sum_value;
66      f(2) = -sum_cost;
67
68  end
```

Bibliography

[1] IBM Rational DOORS (Dynamic Object Oriented Requirements System), http://www.telelogic.com/Products/doors/.

[2] IBM Rational Focal Point, http://www.telelogic.com/products/focalpoint/.

[3] IBM Rational Tau, http://www.telelogic.com/products/tau/.

[4] Volere Requirements Specification Template, http://www.volere.co.uk/template.htm.

[5] Volere Stakeholder Analysis Template, http://www.volere.co.uk/.

[6] T. AlBourae, G. Ruhe, and M. Moussavi. Lightweight Replanning of Software Product Releases. In *Proceedings of the 1st International Workshop on Software Product Management (IWSPM '06)*, pages 27–34, Minneapolis, MN, USA, 12-12 September 2006. IEEE Computer Society.

[7] B. Alenljung. *Envisioning a Future Decision Support System for Requirements Engineering – A Holistic and Human-centred Perspective*. PhD thesis, Department of Computer and Information Science, Linköpings University, 2008.

[8] B. Alenljung and A. Persson. Portraying the Practice of Decision-Making in Requirements Engineering: A Case of Large Scale Bespoke Development. *Requirements Engineering*, 13(4):257–279, November 2008.

[9] I. Alexander and L. Beus-Dukic. *Discovering Requirements: How to Specify Products and Services*. John Wiley & Sons, February 2009.

[10] I. F. Alexander. A Taxonomy of Stakeholders: Human Roles in System Development. *International Journal of Technology and Human Interaction*, 1(1):23–59, 2005.

[11] C. Alves and J. Castro. CRE: A Systematic Method for COTS Components Selection. In *Proceedings of the XV Brazilian Symposium on Software Engineering (SBES '01)*, pages 193–207, Rio de Janeiro, Brazil, 3-5 October 2001. ACM.

[12] Amandeep, G. Ruhe, and M. Stanford. Intelligent Support for Software Release Planning. In *Proceedings of the 5th International Conference on Product Focused Software Process Improvement (PROFES '04)*, volume 3009 of *LNCS*, pages 248–262, Kansai Science City, Japan, 5-8 April 2004. Springer.

[13] R. J. Anderson, P. Beame, S. Burns, W. Chan, F. Modugno, D. Notkin, and J. D. Reese. Model Checking Large Software Specifications. *IEEE Transactions on Software Engineering*, 24(7):498–520, July 1998.

[14] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The Next Release Problem. *Information and Software Technology*, 43(14):883–890, December 2001.

[15] P. Baker, M. Harman, K. Steinhöfel, and A. Skaliotis. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM '06)*, pages 176–185, Philadelphia, Pennsylvania, 24-27 September 2006. IEEE Computer Society.

[16] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison Wesley, October 1999.

[17] P. Berander and P. Jnsson. Hierarchical Cumulative Voting (HCV) - Prioritization of Requirements in Hierarchies. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE) - Special Issue on Requirements Engineering Decision Support*, 16(6):819–849, 2006.

[18] D. Beuche, A. Birk, H. Dreier, A. Fleischmann, H. Galle, G. Heller, D. Janzen, I. John, R. T. Kolagari, T. von der Maßen, and A. Wolfram. Using Requirements Management Tools in Software Product Line Engineering: The State of the Practice. In *Proceedings of the 11th International Software Product Line Conference (SPLC '07)*, pages 84–93, Kyoto, Japan, 10-14 September 2007. IEEE Computer Society Press.

[19] H. Beyer and K. Holtzblatt. *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann Publishers Inc., 1998.

[20] P. Beynon-Davies, C. Carne, H. Mackay, and D. Tudhope. Rapid Application Development (RAD): An Empirical Review. *European Journal of Information Systems*, 8(3):211–223, September 1999.

[21] B. Boehm, P. Bose, E. Horowitz, and M. J. Lee. Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach. In *Proceedings of the 17th International Conference on Software Engineering (ICSE '95)*, pages 243–253, Seattle, Washington, US, 24-28 April 1995. ACM.

[22] I. K. Bray. *Introduction to Requirements Engineering*. Addison Wesley, 2002.

[23] P. Carlshamre. Release Planning in Market-Driven Software Product Development: Provoking an Understanding. *Requirements Engineering*, 7(3):139–151, 2002.

[24] P. Carlshamre and B. Regnell. Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes. In *Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA '00)*, pages 961–965, London, UK, 4-8 September 2000. IEEE Computer Society.

[25] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. Natt och Dag. An Industrial Survey of Requirements Interdependencies in Software Product Release Planning. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE '01)*, 2001.

[26] P. Checkland. Soft Systems Methodology: A Thirty Year Retrospective. *Systems Research and Behavioral Science*, 17(1):11–58, 2000.

[27] P. Checkland and J. Scholes. *Soft Systems Methodology in Action*. John Wiley & Sons, 1990.

[28] P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[29] B. H. C. Cheng and J. M. Atlee. Research Directions in Requirements Engineering. In *Proceedings of the 29th International Conference on Software Engineering and Future of Software Engineering (ICSE/FOSE '07)*, pages 285–303, Minneapolis, MN, 19-27 May 2007.

[30] L. Chung and K. Cooper. COTS-Aware Requirements Engineering and Software Architecting. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP '04)*, pages 57–63, Las Vegas, Nevada, USA, 21-24 June 2004. CSREA Press.

[31] L. Chung and K. Cooper. Defining Goals in a COTS-aware Requirements Engineering Approach: Regular Paper. *Systems Engineering*, 7(1):61–83, March 2004.

[32] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*, volume 5 of *International Series in Software Engineering*. Springer, 1999.

[33] P. Coad and E. Yourdon. *Object Oriented Analysis*. Yourdon Press, 1991.

[34] E. F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, 4(4):394–434, December 1979.

[35] C. A. Coello Coello. Evolutionary Multiobjective Optimization: A Historical View of the Field. *IEEE Computational Intelligence Magazine*, 1(1):28–36, Febuary 2006.

[36] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, May 2002.

[37] Y. Collette and P. Siarry. *Multiobjective Optimization: Principles and Case Studies*. Decision Engineering. Springer, 2004.

[38] C. Corbridge, G. Rugg, N. P. Major, N. R. Shadbolt, and A. M. Burton. Laddering: Technique and Tool use in Knowledge Acquisition. *Knowledge Acquisition*, 6(3):315–341, September 1994.

[39] V. Cortellessa, I. Crnkovic, F. Marinelli, and P. Potena. Experimenting the Automated Selection of COTS Components Based on Cost and System Requirements. *Journal of Universal Computer Science*, 14(8):1228–1255, 2008.

[40] V. Cortellessa, F. Marinelli, and P. Potena. Automated Selection of Software Components Based on Cost/Reliability Tradeoff. In *Proceedings of the 3rd European Workshop on Software Architecture (EWSA '06)*, volume 4344 of *LNCS*, pages 66–81, Nantes, France, 4-5 September 2006. Springer.

[41] V. Cortellessa, F. Marinelli, and P. Potena. An Optimization Framework for "Build-or-Buy" Decisions in Software Architecture. *Computers & Operations Research*, 35(10):3090–3106, October 2008.

[42] J. L. Cybulski and K. Reed. Requirements Classification and Reuse: Crossing Domain Boundaries. In *Proceedings of the 6th International Conference on Software Reuse: Advances in Software Reusability (ICSR '00)*, volume 1844 of *LNCS*, pages 1–8, Vienna, Austria, 27-29 June 2000. Springer.

[43] Å. G. Dahlstedt and A. Persson. Requirements Interdependencies - Moulding the State of Research into A Research Agenda. In *Proceedings of the 9th International Workshop on Requirements Engineering: Foundation for Software Quality (RefsQ '03)*, Klagenfurt/Velden, Austria, 16-17 June 2003.

[44] Å. G. Dahlstedt and A. Persson. *Engineering and Managing Software Requirements*, chapter 5 Requirements Interdependencies: State of the Art and Future Challenges, pages 95–116. Springer Berlin Heidelberg, 2005.

[45] A. Dardenne, S. Fickas, and A. van Lamsweerde. Goal-Directed Concept Acquisition in Requirements Elicitation. In *Proceedings of the 6th International Workshop on Software Specification and Design*, pages 14–21, Como, Italy, 25-26 October 1991. IEEE Computer Society Press.

[46] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.

[47] C. Darwin. *On the Origin of Species*. John Murray, London, 1859.

[48] A. M. Davis. Operational Prototyping: A New Development Approach. *IEEE Software*, 9(5):70–78, September 1992.

[49] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.

[50] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[51] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[52] S. Easterbrook. *What is Requirements Engineering?*, chapter 1, pages 2–18. Draft book chapter, 2004.

[53] S. Easterbrook and M. Chechik. A Framework for Multi-Valued Reasoning Over Inconsistent Viewpoints. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 411–420, Toronto, Ontario, Canada, 12-19 May 2001. IEEE Computer Society Press.

[54] S. Easterbrook, A. Finkelstein, J. Kramer, and B. Nuseibeh. Co-Ordinating Distributed Viewpoints: The Anatomy of a Consistency Check. *Concurrent Engineering*, 2(3):209–222, 1994.

[55] D. J. Farmer, N. H. Packard, and A. S. Perelson. The Immune System, Adaptation, and Machine Learning. *Physica D*, 22:187–204, 1986.

[56] M. S. Feather, S. L. Cornford, J. D. Kiper, and T. Menzies. Experiences using Visualization Techniques to Present Requirements, Risks to Them, and

Options for Risk Mitigation. In *Proceedings of the International Workshop on Requirements Engineering Visualization (REV '06)*, pages 10–10, Minnesota, USA, 11 September 2006. IEEE.

[57] M. S. Feather, J. D. Kiper, and S. Kalafat. Combining Heuristic Search, Visualization and Data Mining for Exploration of System Design Space. In *Proceedings of the 14th Annual International Symposium on Systems Engineering (INCOSE '04)*, Toulouse, France, 20-24 June 2004.

[58] M. S. Feather and T. Menzies. Converging on the Optimal Attainment of Requirements. In *Proceedings of the 10th IEEE International Conference on Requirements Engineering (RE '02)*, pages 263–270, Essen, Germany, 9-13 September 2002. IEEE.

[59] A. Finkelstein. Requirements Engineering: A Review and Research Agenda. In *Proceedings of the first Asia Pacific Conference on Software Engineering*, pages 10–19, Tokyo, Japan, 7-9 December 1994. IEEE CS Press.

[60] A. Finkelstein. Advanced Software Engineering Course, 2002.

[61] A. Finkelstein and W. Emmerich. The Future of Requirements Management Tools. *Information Systems in Public Administration and Law*, 138, 2000.

[62] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency Handling in Multiperspective Specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, August 1994.

[63] A. Finkelstein, M. Harman, S. A. Mansouri, J. Ren, and Y. Zhang. "Fairness Analysis" in Requirements Assignments. In *Proceedings of the 16th IEEE International Requirements Engineering Conference (RE '08)*, pages 115–124, Barcelona, Catalunya, Spain, 8-12 September 2008. IEEE Computer Society.

[64] A. Finkelstein, M. Harman, S. A. Mansouri, J. Ren, and Y. Zhang. A Search based Approach to Fairness Analysis in Requirement Assignments to Aid Negotiation, Mediation and Decision Making. *Requirements Engineering Journal (RE '08 Special Issue)*, 2009.

[65] A. Finkelstein, M. Ryan, and G. Spanoudakis. *Software Package Requirements and Procurement.* IEEE Computer Society Press, March 1996.

[66] R. W. Floyd. Assigning meanings to Programs. *American Mathematical Society*, pages 19–32, 1967.

[67] M. Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional, November 2002.

[68] X. Franch and N. A. Maiden. Modelling Component Dependencies to Inform Their Selection. In *Proceedings of the 2nd International Conference on COTS-Based Software Systems (ICCBSS '03)*, volume 2580 of *LNCS*, pages 81–91, Ottawa, Canada, 10-12 February 2003. Springer.

[69] M.-C. Gaudel. Structuring and Modularizing Algebraic Specifications: the PLUSS Specification Language, Evolutions and Perspectives. In A. Finkel and M. Jantzen, editors, *Proceedings of the 9th Annual Syposium on Theoretical Aspects of Computer Science (STACS '92)*, volume 577 of *LNCS*, pages 13–18, Cachan, France, 13-15 February 1992. Springer.

[70] T. Gilb. *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering using Planguage.* Butterworth-Heinemann, 2005.

[71] T. Gilb. What's Wrong With Agile Methods Some Principles and Values to Encourage Quantification. Methods & Tools, 2007.

[72] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 5:533–549, 1986.

[73] J. A. Goguen and C. Linde. Techniques for Requirements Elicitation. In *Proceedings of IEEE International Symposium on Requirements Engineering (RE '93)*, pages 152–164, San Diego, CA, USA, 4-6 January 1993. IEEE.

[74] O. C. Z. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proceedings of the 1st International Conference on Requirements Engineering (RE '94)*, pages 94–101, Colorado Springs, Colorado, USA, 18-21 April 1994. IEEE Computer Society.

[75] D. Greer and G. Ruhe. Software Release Planning: An Evolutionary and Iterative Approach. *Information & Software Technology*, 46(4):243–253, March 2004.

[76] M. Harman. The Current State and Future of Search Based Software Engineering. In L. Briand and A. Wolf, editors, *Proceedings of International Conference on Software Engineering / Future of Software Engineering 2007 (ICSE/FOSE '07)*, pages 342–357, Minneapolis, Minnesota, USA, 20-26 May 2007. IEEE Computer Society.

[77] M. Harman and J. A. Clark. Metrics Are Fitness Functions Too. In *Proceedings of the 10th IEEE International Symposium on Software Metrics (METRICS '04)*, pages 58–69, Chicago, USA, 11-17 September 2004. IEEE Computer Society.

[78] M. Harman and B. F. Jones. Search-based Software Engineering. *Information & Software Technology*, 43(14):833–839, December 2001.

[79] M. Harman, J. Krinke, J. Ren, and S. Yoo. Search Based Data Sensitivity Analysis Applied to Requirement Engineering. In *Proceedings of the 11th*

Annual Conference on Genetic and Evolutionary Computation (GECCO '09), pages 1681–1688, Montreal, Canada, 8-12 July 2009. ACM.

[80] M. Harman, A. Skaliotis, and K. Steinhöfel. Search-based Approaches to the Component Selection and Prioritization Problem. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1951–1952, Seattle, Washington, USA, 8-12 July 2006. ACM.

[81] M. Harman, S. Swift, and K. Mahdavi. An Empirical Study of the Robustness of Two Module Clustering Fitness Functions. In H.-G. Beyer and U.-M. O'Reilly, editors, *Proceedings of the International Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1029–1036, Washington DC, USA, 25-29 June 2005. ACM.

[82] R. L. Haupt and S. E. Haupt. *Practical Genetic Algorithms*. WileyBlackwell (2nd Edition), 2004.

[83] C. Heitmeyer, R. Jeffords, and B. Labaw. Automated Consistency Checking of Requirements Specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(3):231–261, July 1996.

[84] A. M. Hickey and A. M. Davis. Requirements Elicitation and Elicitation Technique Selection: A Model for Two Knowledge-Intensive Software Development Processes. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS '03)*, pages 96–106, Island of Hawaii, 6-9 January 2003.

[85] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM (CACM)*, 12(10):576–580, October 1969.

[86] J. M. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[87] J. Horn and N. Nafpliotis. Multiobjective Optimization using the Niched Pareto Genetic Algorithm. Technical Report IllIGAL 93005, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, IL, 1993.

[88] P. Hsia, D. Kung, and C. Sell. Software requirements and acceptance testing. *Annals of Software Engineering*, 3(1):291–317, January 1997.

[89] E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. Springer, 2nd edition, 2005.

[90] A. Hunter and B. Nuseibeh. Analyzing Inconsistent Specifications. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE '97)*, pages 78–86, Annapolis, USA, 6-10 January 1997. IEEE Computer Society.

[91] IEEE. *IEEE Guide for Software Requirements Specifications*. Institute of Electrical and Electronics Engineers, 1998.

[92] INCOSE. What is Systems Engineering?

[93] S. Jacobs. Introducing Measurable Quality Requirements: A Case Study. In *Proceedings of the 4th IEEE International Symposium on Requirements Engineering (RE '99)*, pages 172–179, Limerick, Ireland, 7-11 June 1999. IEEE Computer Society.

[94] N. A. M. James Lockerbie. Extending i* to Fit with the Requirements World. In *Proceedings of the 3rd International i* Workshop*, pages 67–70, Recife, Brazil, 11-12 February 2008. CEUR-WS.org.

[95] A. W. Johnson and S. H. Jacobson. A Class of Convergent Generalized Hill Climbing Algorithms. *Applied Mathematics and Computation*, 125(2-3):359–373, January 2002.

[96] H.-W. Jung. Optimizing Value and Cost in Requirements Analysis. *IEEE Software*, 15(4):74–78, July/August 1998.

[97] J. Karlsson. Software Requirements Prioritizing. In *Proceedings of the Second International Conference on Requirements Engineering (RE '96)*, pages 110–116, Colorado Springs, CO, USA, 15-18 April 1996. IEEE Computer Society.

[98] J. Karlsson. *A Systematic Approach for Prioritizing Software Requirements*. PhD thesis, Linköping University, 1998.

[99] J. Karlsson, S. Olsson, and K. Ryan. Improved Practical Support for Large-scale Requirements Prioritizing. *Requirements Engineering Journal*, 2(1):51–60, 1997.

[100] J. Karlsson and K. Ryan. A Cost-Value Approach for Prioritizing Requirements. *IEEE Software*, 14(5):67–74, 1997.

[101] J. Karlsson, C. Wohlin, and B. Regnell. An Evaluation of Methods for Prioritizing Software Requirements. *Information & Software Technology*, 39(14-15):939–947, 1998.

[102] L. Karlsson, Å. G. Dahlstedt, B. Regnell, J. N. och Dag, and A. Persson. Requirements Engineering Challenges in Market-Driven Software Development - An Interview Study with Practitioners. *Information & Software Technology*, 49(6):588–604, 2007.

[103] M. G. Kendall and J. D. Gibbons. *Rank Correlation Methods*. Charles Griffin, 5th edition, 1990.

[104] R. Kennedy and J. Eberhart. A New Optimizer using Particle Swarm Theory. In *Proceedings of the 6th International Symposium on Micro Machine and Human Science (MHS '95)*, pages 39–43, Nagoya, Japan, 1995.

[105] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.

[106] D. Knjazew. *OmeGA: A Competent Genetic Algorithm for Solving Permutation and Scheduling Problems*, volume 6 of *Genetic Algorithms and Evolutionary Computation*. Springer, 2002.

[107] J. D. Knowles, D. W. Corne, and M. J. Oates. The Pareto-Envelope based Selection Algorithm for Multiobjective Optimization. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature (PPSN '00)*, volume 1917 of *LNCS*, pages 839–848, Paris, France, September 2000. Springer.

[108] K. W. Kolence. The Software Empiricist. *ACM SIGMETRICS Performance Evaluation Review*, 2(2):31–36, June 1973.

[109] K. W. Kolence and P. J. Kiviat. Software Unit Profiles & Kiviat Figures. *ACM SIGMETRICS Performance Evaluation Review*, 2(3):2–12, September 1973.

[110] J. Kontio. OTSO: A Systematic Process for Reusable Software Component Selection. Technical Report CS-TR-3478, University of Maryland, June 1995.

[111] J. Kontio. A Case Study in Applying a Systematic Method for COTS Selection. In *Proceedings of the 18th International Conference on Software Engineering (ICSE '96)*, pages 201–209, Berlin, Germany, 25-30 March 1996. IEEE Computer Society.

[112] G. Kotonya and I. Sommerville. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, 1998.

[113] L. Lamport. A Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, May 1994.

[114] R. Land, L. Blankers, M. Chaudron, and I. Crnković. COTS Selection Best Practices in Literature and in Industry. In *Proceedings of the 10th International Conference on Software Reuse: High Confidence Software Reuse in Large Systems*, volume 5030 of *LNCS*, pages 100–111, Beijing, China, 2008. Springer.

[115] S. Lauesen. *Software Requirements: Styles & Techniques*. Addison-Wesley Professional, January 2002.

[116] E. Letier and A. van Lamsweerde. Deriving Operational Software Specifications from System Goals. *ACM SIGSOFT Software Engineering Notes*, 27(6):119–128, November 2002.

[117] C. Li, M. van den Akker, S. Brinkkemper, and G. Diepen. Integrated Requirement Selection and Scheduling for the Release Planning of a Software Product. In *Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ '07)*, volume 4542 of *LNCS*, pages 93–108, Trondheim, Norway, 11-12 June 2007. Springer.

[118] R. W. Lichota, R. L. Vesprini, and B. Swanson. PRISM Product Examination Process for Component Based Development. In *Proceedings of the 5th International Symposium on Assessment of Software Tools (SAST '97)*, pages 61–69, Pittsburgh, PA, USA, 3-5 June 1997. IEEE.

[119] H. Lin, A. Lai, R. Ullrich, M. Kuca, J. Shaffer-Gant, S. Pacheco, K. Dalton, K. McClelland, W. Watkins, and S. Khajenoori. COTS Software Selection Process. Sandia Report SAND 2006-0478, Sandia National Laboratories, May 2006.

[120] J. Lockerbie and N. Maiden. REDEPEND: Extending i* Modelling into Requirements Processes. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE '06)*, pages 354–355, Minneapolis/St. Paul, Minnesota, USA, 11-15 September 2006. IEEE Computer Society.

[121] N. A. Maiden, V. Croce, H. Kim, G. Sajeva, and S. Topuzidou. SCARLET: Integrated Process and Tool Support for Selecting Software Components. *Component-Based Software Quality*, 2693:85–98, 2003.

[122] N. A. Maiden, S. V. Jones, S. Manning, J. Greenwood, and L. Renou. Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study. In *Proceedings of the 16th International Conference on Advanced Information Systems Engineering*, volume 3084 of *LNCS*, pages 368–383, Riga, Latvia, 7-11 June 2004. Springer.

[123] N. A. Maiden, H. Kim, and C. Ncube. Rethinking Process Guidance for Selecting Software Components. In *Proceedings of the 1st International Conference on COTS-Based Software Systems (ICCBSS '02)*, volume 2255 of *LNCS*, pages 151–164, Orlando, FL, USA, 4-6 February 2002. Springer.

[124] N. A. Maiden and G. Rugg. ACRE: Selecting Methods for Requirements Acquisition. *Software Engineering Journal*, 11(3):183–192, May 1996.

[125] N. A. Maiden, N. Seyff, P. Grunbacher, O. Otojare, and K. Mitteregger. Making Mobile Requirements Engineering Tools Usable and Useful. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE '06)*, pages 26–35, Minneapolis/St. Paul, Minnesota, USA, 11-15 September 2006. IEEE Computer Society.

[126] N. A. M. Maiden. CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements. *Automated Software Engineering*, 5(4):419–446, 1998.

[127] N. A. M. Maiden and S. Robertson. Integrating Creativity into Requirements Processes: Experiences with an Air Traffic Management System. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE '05)*, pages 105–114, Minneapolis/St. Paul, Minnesota, USA, 11-15 September 2005. IEEE Computer Society.

[128] A. Mohamed, G. Ruhe, and A. Eberlein. COTS Selection: Past, Present, and Future. In *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 103–114, 26-29 March 2007.

[129] H. Muehlenbein and G. Paaß. From Recombination of Genes to the Estimation of Distributions I, Binary Parameters. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN '96)*, volume 1141 of *LNCS*, pages 178–188, Berlin, Germany, 22-26 September 1996. Springer.

[130] P. Naur. Proofs of Algorithms by General Snapshots. *BIT Numerical Mathematics*, 6(4):310–316, 1966.

[131] C. Ncube, J. Lockerbie, and N. Maiden. Automatically Generating Requirements from i* Models: Experiences with a Complex Airport Operations System. In *Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ '07)*, volume 4542 of *LNCS*, pages 33–47, Trondheim, Norway, 11-12 June 2007. Springer.

[132] C. Ncube and N. A. Maiden. Guidance for Parallel Requirements Acquisition and COTS Software Selection. In *Proceedings of the 4th IEEE International Symposium on Requirements Engineering (RE '99)*, page 133, Limerick, Ireland, 7-11 June 1999. IEEE Computer Society.

[133] C. Ncube and N. A. Maiden. PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm. In *Proceedings of the 2nd International Workshop on Component-Based Software Engineering*, pages 1–12, Los Angeles, CA, USA, 17-18 May 1999. IEEE.

[134] A. Ngo-The and G. Ruhe. A Systematic Approach for Solving the Wicked Problem of Software Release Planning. *Soft Computing*, 12:95–108, 2008.

[135] B. Nuseibeh and S. M. Easterbrook. Requirements Engineering: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering (ICSE '00) - Future of SE Track*, pages 35–46, Limerick, Ireland, 2000. ACM.

[136] B. Nuseibeh, J. Kramer, and A. Finkelstein. A Framework for Expressing the Relationships between Multiple Views in Requirements Specifications. *IEEE Transactions on Software Engineering*, 20(10):760–773, October 1994.

[137] P. A. Oberndorf, L. Brownsword, E. Morris, and C. S. (Editors). Workshop on COTS-Based Systems. Special Report CMU/SEI-97-SR-019, Software Engineering Institute, Carnegie Mellon University, November 1997.

[138] J. N. och Dag. *Elicitation and Management of User Requirements in Market-Driven Software Development*. PhD thesis, Lund University, January 2002.

[139] H. Ohiwa, K. Kawai, A. Shiomi, and N. Takeda. KJ-Editor: A Collaboration Environment for Brain Storming and Consensus Forming. In *Proceedings of the Fifth International Conference on Human-Computer Interaction*, volume 2 of *IV. Help and Learning*, pages 939–942, Orlando, Florida, USA, 8-13 August 1993.

[140] A. Osyczka. Multicriteria Optimization for Engineering Design. In *Design Optimization*, pages 193–227, Orlando, Florida, USA, 1985. Academic Press.

[141] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, 1998.

[142] B. C. Phillips and S. M. Polen. Add Decision Analysis to Your COTS Selection Process. *The Journal of Defense Software Enginering*, April 2002.

[143] K. Pohl. *Process-Centered Requirements Engineering*. Research Studies Press, 1996.

[144] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*. Prentice-Hall, 1991.

[145] K. Praditwong and X. Yao. A New Multi-Objective Evolutionary Optimisation Algorithm: The Two-Archive Algorithm. In *Proceedings of he 2006 International Conference on Computational Intelligence and Security (CIS '06)*, volume 1, pages 286–291, Guangzhou, China, 3-6 November 2006. IEEE Press.

[146] S. K. Probert. Requirements Engineering, Soft Systems Methodology and Workforce Empowerment. *Requirements Engineering*, 4(2):85–91, July 1999.

[147] B. Ramesh, T. Powers, C. Stubbs, and M. Edwards. Implementing Requirements Traceability: A Case Study. In *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering (RE '95)*, pages 89–95, York, UK, 27-29 March 1995. IEEE Computer Society.

[148] B. Regnell and O. Eklundh. A Market-driven Requirements Engineering Process: Results from an Industrial Process Improvement Programme. *Requirements Engineering*, 3:121–129, December 1998.

[149] H. W. J. Rittel and M. M. Webber. Dilemmas in a General Theory of Planning. *Policy Sciences*, 4:155–169, 1973.

[150] S. Robertson and J. Robertson. Requirements Fundamentals: the Basis for Effective Testing, 2000.

[151] S. Robertson and J. C. Robertson. *Mastering the Requirements Process*. Addison Wesley, 2nd edition, 2006.

[152] W. Robinson and S. Fickas. Automated Support for Requirements Negotiation, March 1994.

[153] W. N. Robinson, S. D. Pawlowski, and V. Volkov. Requirements Interaction Management. Technical Report 99-7, Georgia State University, August 1999.

[154] W. N. Robinson, S. D. Pawlowski, and V. Volkov. Requirements Interaction Management. *ACM Computing Surveys (CSUR)*, 35(2):132–190, June 2003.

[155] W. N. Robinson and V. Volkov. Requirement Conflict Restructuring, August 1999.

[156] B. Roy. *Multicriteria Methodology for Decision Aiding*, volume Nonconvex Optimization and Its Applications. Springer, August 1996.

[157] G. Rugg, C. Corbridge, N. P. Major, A. M. Burton, and N. R. Shadbolt. A Comparison of Sorting Techniques in Knowledge Acquisition. *Knowledge Acquisition*, 4(3):279–291, September 1992.

[158] G. Ruhe. Intelligent Support for Selection of COTS Products. In *Proceedings of the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, volume 2593 of *LNCS*, pages 34–45, Erfurt, Germany, 7-10 October 2002. Springer.

[159] G. Ruhe. Software Release Planning. *Handbook Software Engineering and Knowledge Engineering*, 3:1–21, 2003.

[160] G. Ruhe, A. Eberlein, and D. Pfahl. Quantitative WinWin: a New method for Decision Support in Requirements Negotiation. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 159–166, Ischia, Italy, 15-19 July 2002. ACM.

[161] G. Ruhe and D. Greer. Quantitative Studies in Software Release Planning under Risk and Resource Constraints. In *Proceedings of the International Symposium on Empirical Software Engineering (ISESE '03)*, pages 262–270, Rome, Italy, 29 September-4 October 2003. IEEE.

[162] G. Ruhe and A. Ngo-The. Hybrid Intelligence in Software Release Planning. *International Journal of Hybrid Intelligent Systems*, 1(1-2):99–110, April 2004.

[163] G. Ruhe and M. O. Saliu. The Art and Science of Software Release Planning. *IEEE Software*, 22(6):47–53, November 2005.

[164] G. Ruhe and M. O. Saliu. The Science and Practice of Software Release Planning. *IEEE Software*, 2005.

[165] T. L. Saaty. *The Analytic Hierarchy Process, Planning, Priority Setting, Resource Allocation*. McGraw-Hill, 1980.

[166] D. Sahni. A Controlled Experiment on Analytical Hierarchy Process and Cumulative Voting. Master's thesis, School of Engineering Blekinge Institute of Technology, 2007.

[167] C. Salinesi and E. Kornyshova. Choosing a Prioritization Method – Case of IS Security Improvement. In *Forum Proceedings of the 18th Conference on Advanced Information Systems Engineering (CAiSE Forum '06) Theme: Trusted Information Systems*, pages 51–55, Luxembourg, 5-9 June 2006. CEUR-WS.org.

[168] M. O. Saliu and G. Ruhe. Bi-Objective Release Planning for Evolving Software Systems. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 105–114, Dubrovnik, Croatia, 3-7 September 2007. ACM.

[169] J. M. Schraagen, S. F. Chipman, and V. L. Shalin, editors. *Cognitive Task Analysis*. CRC Press, 2000.

[170] H.-P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, 1981.

[171] J. Siddiqi and M. C. Shekaran. Requirements Engineering: The Emerging Wisdom. *IEEE Software*, 13(2):15–19, March 1996.

[172] S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, 1980.

[173] I. Sommerville, T. Rodden, P. Sawyer, R. Bentley, and M. Twidale. Integrating Ethnography into the Requirements Engineering Process. In *Proceedings of the IEEE International Symposium on Requirements Engineering (RE '03)*, pages 165–173, San Diego, CA, USA, 4-6 January 1993. IEEE.

[174] N. Srinivas and K. Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[175] L. Sullivan. Quality Function Deployment. *Quality Progress*, June issue:39–50, 1986.

[176] A. Sutcliff and A. Gregoriades. Validating Functional System Requirements with Scenarios. In *Proceedings of the 10th Anniversary IEEE Joint Inter-*

national Conference on Requirements Engineering (RE '02), pages 181–190, Essen, Germany, 9-13 September 2002. IEEE Computer Society.

[177] R. B. Svensson, T. Gorschek, and B. Regnell. Quality Requirements in Practice: An Interview Study in Requirements Engineering for Embedded Systems. In *Proceedings of the 15th International Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ '09)*, volume 5512 of *LNCS*, pages 218–232, Amsterdam, Netherlands, 8-9 June 2009. Springer.

[178] F. Szidarovsky, M. E. Gershon, and L. Dukstein. *Techniques for Multiobjective Decision Making in Systems Management*. Elsevier, New York, 1986.

[179] F. B. Tan and M. G. Hunter. The Repertory Grid Technique: A Method for the Study of Cognition in Information Systems. *Management Information Systems Quarterly*, 26(1):39–57, January 2002.

[180] J. M. Thompson, M. P. E. Heimdahl, and S. P. Miller. Specification-Based Prototyping for Embedded Systems. In O. Nierstrasz and M. Lemoine, editors, *Proceedings of the 7th European Software Engineering Conference held jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering ESEC/FSE*, volume 24 of *LNCS*, pages 163–179, Toulouse, France, November 1999. Springer-Verlag / ACM Press.

[181] T. Tourwé, W. Codenie, N. Boucart, and V. Blagojević. Demystifying Release Definition: From Requirements Prioritization to Collaborative Value Quantification. In *Proceedings of 15th Internatinal Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ '09)*, volume 5512 of *LNCS*, pages 37–44, Amsterdam, Netherlands, 8-9 June 2009. Springer.

[182] A. M. Turing. Checking a Large Routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69, Cambridge, UK, June 1949.

[183] M. van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal. Flexible Release Composition using Integer Linear Programming. Technical Report UU-CS-2004-063, Institute of Information and Computing Sciences, Utrecht University, December 2004.

[184] M. van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal. Determination of the Next Release of a Software Product: an Approach using Integer Linear Programming. In O. Belo, J. Eder, J. F. e Cunha, and O. Pastor, editors, *Proceedings of the 17th conference on Advanced Information Systems Engineering (CAiSE '05)*, CEUR Workshop Proceedings, pages 119–124, Porto, Portugal, 13-17, June 2005. CEUR-WS.org.

[185] M. van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal. Flexible Release Planning using Integer Linear Programming. In *Proceedings of the 11th International Workshop on Requirements Engineering for Software Quality (RefsQ '05)*, pages 247–262, Porto, Portugal, 13-14 June 2005. Essener Informatik Beitrage.

[186] M. van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal. Software Product Release Planning through Optimization and What-If Analysis. *Information and Software Technology*, 50(1-2):101–111, January 2008.

[187] A. van Lamsweerde. Formal Specification: A Roadmap. In *Proceedings of the International Conference on Software Engineering (ICSE '00) - Future of SE Track*, pages 147–159, Limerick, Ireland, 4-11 June 2000. ACM.

[188] A. van Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. In *Proceedings of the 22nd International Conference on Software engineering (ICSE '00)*, pages 5–19, Limerick, Ireland, 4-11 June 2000.

[189] A. van Lamsweerde, R. Darimont, and E. Letier. Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, November 1998.

[190] S. Viller and I. Sommerville. Social Analysis in the Requirements Engineering Process: From Ethnography to Method. In *Proceedings of the 4th IEEE International Symposium on Requirements Engineering (RE '99)*, pages 6–13, Limerick, Ireland, 7-11 June 1999. IEEE Computer Society.

[191] K. E. Wiegers. Automating Requirements Management. *Software Development*, 7, July 1999.

[192] K. E. Wiegers. First Things First: Prioritising Requirements. *Software Development*, 7(9):48–53, 1999.

[193] K. E. Wiegers. *Software Requirements*. Microsoft Press, Redmond, WA, USA, 2003.

[194] R. J. Wirfs-Brock and R. E. Johnson. Surveying Current Research in Object-Oriented Design. *Communications of the ACM (CACM)*, 33(9):104–124, September 1990.

[195] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*, volume 6 of *International Series in Software Engineering*. Springer, 2000.

[196] R. Yeh and P. Ng. Software Requirements – A Management Perspective. *System and Software Requirements Engineering*, pages 450–461, 1990.

[197] E. S. K. Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE '97)*, pages 226–235, Annapolis, MD, USA, 6-10, January 1997. IEEE Computer Society Press.

[198] K. Zachos and N. A. Maiden. ART-SCENE: Enhancing Scenario Walkthroughs With Multi-Media Scenarios. In *Proceedings of the 12th IEEE International Conference on Requirements Engineering (RE '04)*, pages 360–361. IEEE Computer Society, 6-10 September 2004.

[199] P. Zave. An Operational Approach to Requirements Specification for Embedded Systems. *IEEE Transactions on Software Engineering*, 8(3):250–269, May 1982.

[200] P. Zave. Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys (C SUR)*, 29(4):315–321, December 1997.

[201] Y. Zhang, A. Finkelstein, and M. Harman. Search Based Requirements Optimisation: Existing Work & Challenges. In *Proceedings of the 14th International Working Conference, Requirements Engineering: Foundation for Software Quality (RefsQ '08)*, volume 5025 of *LNCS*, pages 88–94, Montpellier, France, 16-17 June 2008. Springer.

[202] Y. Zhang, M. Harman, and S. A. Mansouri. The Multi-Objective Next Release Problem. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1129–1137, London, UK, 7-11 July 2007. ACM.

[203] Z. Zhang. Effective Requirements Development - A Comparison of Requirements Elicitation Techniques. In *Proceedings of the 15th Software Quality*

Management Conference: (SQM '07), pages 225–240, Tampere, Finland, 31 July - 3 August 2007. British Computer Society.

[204] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001.