UCL DEPARTMENT OF
COMPUTER SCIENCE

Research Note
RN/11/12

Search Based Optimization of Requirements Interaction Management

29 April 2011

Yuanyuan Zhang

Mark Harman

Soo Ling Lim

Abstract

Requirements optimization has been widely studied in the SBSE literature. However, previous approaches have not handled requirements interactions, such as the dependencies that may exist between requirements, *and*, *or*, *precedence*, *cost*– and *value*–based constraints. To introduce and evaluate a Multi-Objective Search Based Requirements Selection technique, using chromosome repair and to evaluate it on both synthetic and real world data sets, in order to assess its effectiveness and scalability. The paper extends and improves upon our previous conference paper on Requirements Interaction Management. The popular Multi-Objective Evolutionary Algorithm NSGA-II was used to produce baseline data for each data set in order to determine how many solutions on the Pareto front fail to meet five different requirement interaction constraints. The results for this baseline data are compared to those obtained using the Archive–based approach previously studied and the repair–based approach introduced in this paper. The repair–based approach was found to produce more points on the Pareto front and a better spread of results than the previously studied Archive-based approach. The repair–based approach was also found to scale almost as well as the previous approach. There is evidence to indicate that the repair–based algorithm introduced in this paper is a suitable technique for extending previous work on Requirements Optimization to handle the requirement interaction constraints inherent in requirement interactions arising from dependencies, *and*, *or*, *precedence*, *cost*– and *value*–based constraints.

RN/11/12
29 April 2011

Research
Note

Search Based Optimization of Requirements Interaction Management

Yuanyuan Zhang, Mark Harman & Soo Ling Lim

Telephone: +44 (0) 20 7679 1056
Fax: +44 (0)171 387 1397
Electronic Mail: {yuanyuan.zhang, m.harman, sooling.lim}@cs.ucl.ac.uk
URL: http://www.cs.ucl.ac.uk/people/Yuanyuan.Zhang.html

Abstract

Requirements optimization has been widely studied in the SBSE literature. However, previous approaches have not handled requirements interactions, such as the dependencies that may exist between requirements, *and*, *or*, *precedence*, *cost*– and *value*–based constraints. To introduce and evaluate a Multi-Objective Search Based Requirements Selection technique, using chromosome repair and to evaluate it on both synthetic and real world data sets, in order to assess its effectiveness and scalability. The paper extends and improves upon our previous conference paper on Requirements Interaction Management. The popular Multi-Objective Evolutionary Algorithm NSGA-II was used to produce baseline data for each data set in order to determine how many solutions on the Pareto front fail to meet five different requirement interaction constraints. The results for this baseline data are compared to those obtained using the Archive–based approach previously studied and the repair–based approach introduced in this paper. The repair–based approach was found to produce more points on the Pareto front and a better spread of results than the previously studied Archive-based approach. The repair–based approach was also found to scale almost as well as the previous approach. There is evidence to indicate that the repair–based algorithm introduced in this paper is a suitable technique for extending previous work on Requirements Optimization to handle the requirement interaction constraints inherent in requirement interactions arising from dependencies, *and*, *or*, *precedence*, *cost*– and *value*–based constraints.

Keywords

Requirements, RIM, NSGA-II, repair method, dependency, Search-based Software Engineering

Department of Computer Science
University College London
Gower Street
London WC1E 6BT, UK

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 the evolution process.

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" [5].

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. 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" [35].

Robinson et al. [35] defined requirements interaction as:

> "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."

RIM consists of a series of activities related to requirement dependencies which are complex and challenging tasks.

Few previous authors [3, 12, 15] focus on the role of requirements dependencies in the solution space. However, dependencies can have a very strong impact on the development process in a typical real world project. Bagnall et al. [3] 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 [15] extended the work by adding the *And* dependency type together with *Precedence* as the constraints in their EVOLVE model. Franch and Maiden [12] applied the *i** approach to model dependencies for COTS component selection.

Requirements dependency management can be considered as a constraint satisfaction problem from a new perspective. In this paper, the Search-based approaches allow for the most common types of requirement dependencies. The objective is to investigate the influences of requirement dependencies on the automated requirements selection process for release planning.

Although search based techniques can find good solutions for unconstrained or simple constrained optimization problems, they might encounter difficulties while solving highly constrained problems. In terms of the RIM, the strength of constraints depends on the number and complexity of interactions between the requirements. The tighter the constraints are, the more difficult the problem is to solve. In order to meet the challenge and to generate feasible optimal solutions, two improved techniques are used: one is an archive based version of NSGA-II; the other is a standard evolutionary algorithm with constraint handling technique – the 'repair' method. A real world large scale data set RALIC and the synthetic data sets previously studied [41] are adopted in this paper to evaluate the approach.

The study is based on the assumption that the dependence identification activity has been completed. Here we present the most common interaction types found in the requirements literature. These will be studied in this paper. The paper will show how multi-objective SBSE can be adapted to take account of RIM.

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.

The rest of the paper is organized as follows. In Section 2 the problem is formalized as an SBSE problem, while Section 3 describes the data sets and algorithms used. Section 4 presents the results for dependence aware requirements optimization and discusses the findings. Section 5 describes the context of related work in which the current paper is located. Section 6 concludes the paper.

2 Fitness Function

In the context of Value/Cost-based requirements assignments analysis, the dependencies among requirements need to be accounted for within the fitness function. This section describes our fitness computation and how we incorporate RIM into this fitness.

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

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

The requirements array R is defined by:

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

First, we formalize the RIM constraints that were listed informally in the introduction to this paper.

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

Or Define an equivalence relation φ on the requirements array R such that $r(i,j) \in \varphi$ (equivalently $r(j,i) \in \varphi$) means that at most 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

$$\xi \bigcap \varphi = \emptyset \ \wedge \ \xi \bigcap \chi = \emptyset$$

in order to guarantee consistency in the requirements dependency relationship.

A set of stakeholders for a software system or service denoted by $C = \{c_1, \ldots, c_m\}$. 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: $Weight = \{w_1, \ldots, w_m\}$ where $w_j \in [0,1]$ and $\sum_{j=1}^{m} w_j = 1$.

The resources needed to implement a particular requirement can be transformed into cost terms. The resultant cost vector for the set of requirements r_i $(1 \leq i \leq n)$ is denoted by: $Cost = \{cost_1, \ldots, cost_n\}$

In the real world, different stakeholders have different needs and perspectives. That is, not all requirements are equally important for a given stakeholder. 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.

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

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

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

The fitness function with dependency constraints is defined as follows:

$$\text{Maximize } f_1(\overrightarrow{x}) = \sum_{i=1}^{n} score_i \cdot x_i \tag{2}$$

$$\text{Maximize } f_2(\overrightarrow{x}) = -\sum_{i=1}^{n} cost_i \cdot x_i \tag{3}$$

subject to

$$x_i = x_j \text{ for all pairs } r(i,j) \in \xi \text{ (And constraints)}$$
$$x_i \neq x_j \ \vee \ x_i = x_j = 0 \text{ for all pairs } r(i,j) \in \varphi \text{ (Or constraints)}$$
$$x_i = 1 \wedge x_j = 1 \ \vee \ x_i = 1 \wedge x_j = 0 \ \vee \ x_i = x_j = 0$$
$$\text{for all pairs } r(i,j) \in \chi \text{ (Precedence constraints)}$$

In terms of Value-related and Cost-related requirements dependencies, they cannot be transformed from dependencies into constraints. So the fitness values of a solution are changed directly when there exists a Value-related or Cost-related dependency, as follows:

$$\text{If } x_i = x_j = 1 \text{ for all pairs } r(i,j) \in \psi \Rightarrow$$
$$\text{Update Fitness Value of } f_1(\overrightarrow{x})$$
$$\text{If } x_i = x_j = 1 \text{ for all pairs } r(i,j) \in \omega \Rightarrow$$
$$\text{Update Fitness Value of } f_2(\overrightarrow{x})$$

3 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 requirements interaction management.

3.1 Data Sets

3.1.1 RALIC data sets

The RALIC project was a software project in University College London (UCL), initiated to replace the existing access control systems at UCL and consolidate the new system with library access and borrowing [24]. RALIC stands for Replacement Access, Library and ID Card. It was a combination of development and customization of an off-the-shelf system. The project duration was 2.5 years and the system has been in deployment for over two years.

The stakeholder priority data for RALIC was collected in previous work using the StakeNet stakeholder analysis method [25] as follows. The stakeholders were asked to recommend people whom they think are stakeholders in the project. Their recommendations were then used to build a social network, where the stakeholders were nodes and

their recommendations were links. Finally, social network measures such as betweenness centrality, degree centrality, and PageRank were used to rank the stakeholders. The output was a prioritized list of stakeholders and their roles. In this paper, the output produced by PageRank was used to weight the stakeholders, as previous study found it to be effective [25]. The PageRank weight of the stakeholder j is the w_j in equation 1.

PageRank is the algorithm used by Google to rank documents [30]. Using PageRank, stakeholders who were strongly recommended by many important stakeholders were important, and the recommendations of a highly important stakeholder were given more weight, which, in turn, made their recommended stakeholders more important.

The stakeholder requirements data sets were collected in previous work using the StakeRare requirements elicitation method [24] described as follows. Two data sets were used in this work: PointP and RankP.

- For the PointP dataset, there are 143 requirements and 77 stakeholders. Each stakeholder identified by StakeNet was asked to distribute 100 points among the requirements they want [24]. The stakeholders were asked to allocate more points to the requirements that were more important to them. Some stakeholders made arithmetic errors while allocating points such that their total points were greater or less than 100. As such, the ratings were normalized such that each stakeholder's allocated points added up to 100. The PointP value for each stakeholder used in this paper is divided by 100 so that all the values are within the same range of 0 to 1.
- For the RankP dataset, there are 143 requirements and 79 stakeholders. Each stakeholder identified by StakeNet were asked to provide a list of requirements, and rank the requirements with numeric priorities (1 for the most important requirement) [24]. Stakeholders also marked "X" for requirements they actively do not want. The rankings were normalized such that the sum of all the ranks from each stakeholder adds up to 1. The rating "X" (from stakeholders actively not wanting a requirement) was converted to 0.

The RALIC data sets are publicly available at

http://www.cs.ucl.ac.uk/staff/S.Lim/phd/dataset.html. Detail descriptions about the data sets can be found in [23].

In the previous work, the requirements in RankP and PointP are organized into a hierarchy of three levels: project objective, requirement, and specific requirement [24]. Achieving all the leaf requirements implies that the parent requirement is achieved. As such, the leaf requirements with the same parent requirement have *And* relationships among each other. Only the leaf requirements are considered in this work, as the higher level requirements are achieved when the leaf requirements are achieved. Some requirements are in conflict, i.e., stakeholders pointed out that achieving the requirement means that another requirement cannot be achieved. Conflicting requirements have *Or* relationships between one another.

The cost data was derived from the RALIC post implementation report. The cost of each requirement was calculated as the time spent by the project team on the requirement during the project, in terms of person hours. For requirements that were not implemented, the cost was estimated by finding the cost of implemented requirements with equivalent effort, and validating the result with the project team.

3.1.2 27 combination random data sets

The "27 combination random data sets" used in previous studies [40] were adopted in this studies. 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. These 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, 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 1 lists the combination of all cases schematically. As can be seen in Table 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 behavior 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

Table 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 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

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 behavior 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.

To explore this, in the empirical studies, we generated four data sets exhibiting different scales and densities using the approach to data set generation depicted in Table 1 and Table 2. We named the sets A, B, C and D. In the A data set, the parameter choices were chosen to be "medium" and the density of the stakeholder-requirement matrix was also chosen to be "medium". The parameters of the data set were randomly generated within the given scale intervals. More concretely, 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 specific scales chosen are listed in Table 3.

In the four data sets that were generated, all the requirements were initially created to be independent. To introduce the dependency, relationships among requirements are added randomly but with respect to constraints. 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 represent the five requirements dependency types.

Table 3: Scale of A, B, C and D data sets: exploration of the configuration space for RIM

	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:** $C_l\ R_l\ D_h$

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

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

The above three dependency arrays are bit vectors which compactly store individual boolean values, as flags to indicate the relationship between 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.

Table 4: synthetic and real world data sets: choosing a variety of RIM distributions

	Name of Data Set	No. of Stakeholders	No. of Requirements	Density of Matrix
Synthetic	A	11	230	0.53
	B	34	50	0.39
	C	4	258	0.51
	D	21	412	0.98
Real World	PointP	77	143	0.14
	RankP	79	143	0.10

The number of requirements, stakeholders and the densities of requirement-stakeholder matrix for all the six (real world and synthetic) data sets are listed in Table 4.

3.2 Algorithms

The search algorithms used in this work were NSGA-II [11], Archive based NSGA-II and a repair method for constraint handling.

The modification of Archive based NSGA-II is inspired by Praditwong and Yao's Two-Archive multi-objective evolutionary optimization algorithm [33]. Keeping the final set of non-dominated solutions is good enough for general multi-objective optimization work. However, when we take account for requirements dependencies, the selected optimal non-dominated solutions might not respect the dependency constraints. As a result some solutions might have to be eliminated as the infeasible solutions. This may mean rejecting otherwise 'optimal' solutions in favor of previously considered and otherwise less optimal solutions.

In order to preserve these potential candidate solutions, this paper introduces an archive-based variation of the NSGA-II algorithm to retain near optimal solutions (maintaining diversity and quantity of the solutions) based on weak constraints.

When solving highly constrained problems, we discovered that the archive based NSGA-II previously used [41] may fail to maintain diversity and convergence of the Pareto front. Therefore, we introduce a repair method to directly convert the infeasible solutions to the feasible solutions.

Repair methods is one type of the constraint handling techniques. It is computational effective and performs well in many combinatorial optimization problems (like the knapsack problem). Because these problems' constraints and decision variables are usually easily characterized and an infeasible solution is relatively easy to repair.

RIM based requirements selection and optimization is a multi-objective knapsack problem, for which previous work [13] has indicated that a repair-based approach is likely to be effective. Hence our repair method is specifically constructed to produce a good Pareto front for RIM-constrained problem. According to the generic framework for constrained optimization problem proposed by Venkatraman and Yen [39], there are two phases for solving constrained optimization problem. Phase one is constraint satisfaction. For any solution, in each generation, it first

checks the constraint violation. If the solution is infeasible, the repair method is used to repair the chromosome. The infeasible solution is replaced by the repaired solution in the population. At this stage, the fitness functions are not considered and all the solutions' fitness values are not evaluated. Phase two is constrained optimization. Using fitness functions and search techniques (NSGA-II in this paper) we seek to find good feasible solutions in one generation. After iterating the process, the optimal solutions in the last generation will be collected to compare the performance of the Archive based NSGA-II algorithm.

The repair method works as follows:

$\exists\, x_i, x_j$ for pair $r(i,j) \in \xi$ (And constraints). If decision vectors x_i and x_j violate And constraint, then make the value of x_j equal the value of x_i;

$\exists\, x_i, x_j$ for pair $r(i,j) \in \varphi$ (Or constraints). If decision vectors x_i and x_j violate Or constraint, then make the value of $x_j = 0$;

$\exists\, x_i, x_j$ for pair $r(i,j) \in \chi$ (Precedence constraints). If decision vectors x_i and x_j violate Precedence constraint, then make the value of $x_j = 0$.

All search-based approaches were run for a maximum of 50,000 fitness function evaluations. The 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). 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). In the archive based NSGA-II algorithm, the total capacity of the archives was set to 500.

4 Empirical Studies and Results

This section presents the experiments carried out to investigate the results in the presence of requirement dependencies and to compare the performance of three methods.

There are two types of empirical study: a Dependency Impact Study (DIS) and a Scale Study (SS). Real world data sets – RALIC project are used for DIS. Synthetic data sets are applied for both DIS and SS: Data set A is used throughout the DIS experiments in order to set up a uniform baseline for comparison; 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. Because there are only two kinds (*And* and *Or*) of dependencies in the RALIC data sets, we will discuss synthetic data set A and real world data sets RALIC separately.

Three experiments were conducted for data set A in 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 relationships as a whole in order to seek to investigate the difference among the solutions generated by the two algorithms.

The NSGA-II algorithm, the archive based NSGA-II algorithm and repair method will be applied to the RALIC data sets to inspect the performance of the approaches and to find the feasible solutions.

In SS, we report results concerning the performance of the three 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.

There are a number of factors in the empirical studies including data sets, techniques and requirement dependencies. The detailed information contained in each factor is listed as follow:

- 6 data sets

  Synthetic: A, B, C and D

  Real world: RALIC PointP and RankP

- 4 techniques

  NSGA-II (without considering dependencies – baseline)

  NSGA-II (non satisfying solutions deleted)

  Archive based NSGA-II

  Repair method

- 5 dependencies

  And, *Or*, *Precedence*, *Value-related* and *Cost-related*

This means that there are 120 possible graphs of results. However not all dependencies are relevant for all data sets. The real world data set, RALIC, only contain *And* and *Or* dependencies. So there are 96 sets of results from our experiments. It is impossible to present all of them in the paper, therefore, DIS and SS are designed to illustrate the research results.

4.1 Dependency Impact Study

4.1.1 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.

4.1.2 And, Or and Precedence for Data Set A

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 for data set A. The results generated by the standard NSGA-II algorithm are shown in Figures 1, 2 and 3; and the results from the archive based NSGA-II algorithm are shown in Figures 4, 5 and 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. They are also marked in grey to distinguish them from the others (which do take account of dependencies).

In Figures 1, 2 and 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 1 when the *And* dependency relationships exist among the requirements. '◯' symbols indicate solutions which respect the *And* dependence. As can be seen from the graph, all the '◯' solutions still fall on the Pareto front composed of grey '+' 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

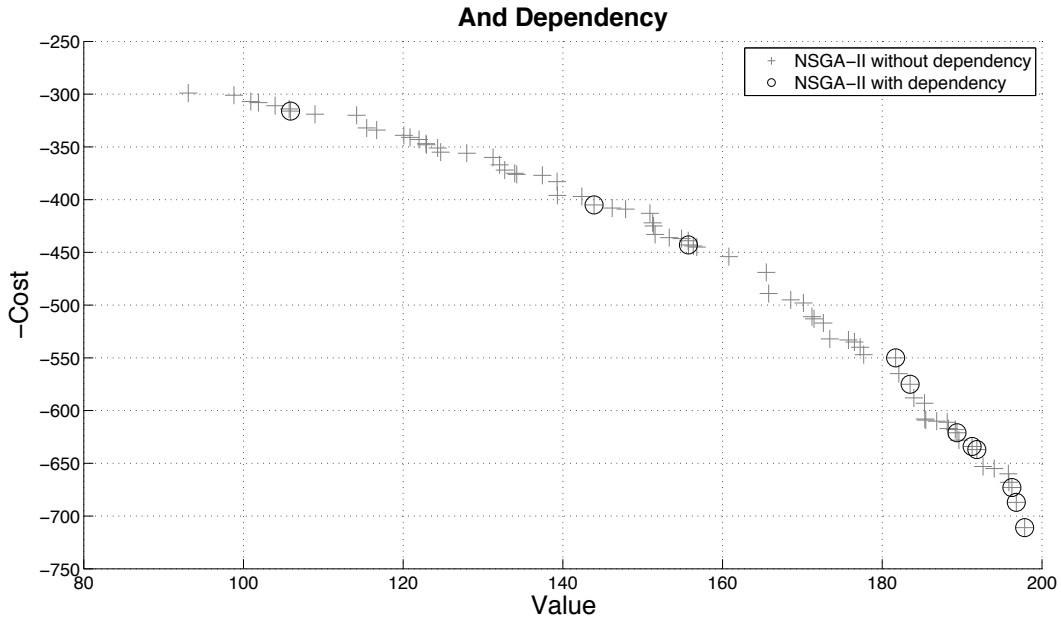

Figure 1: *And* dependency, data set A, NSGA-II

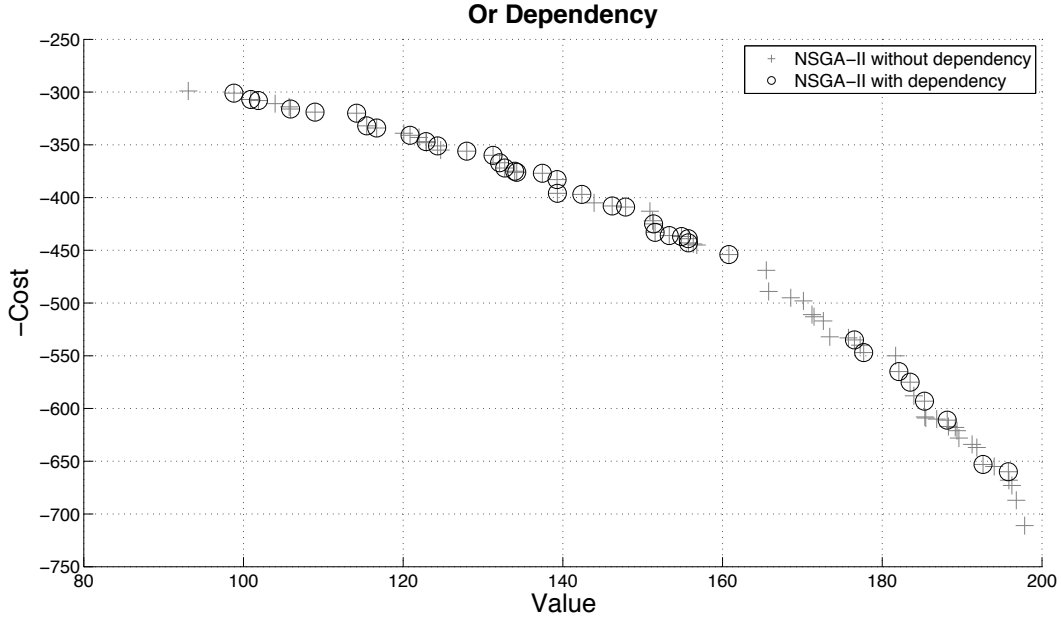

Figure 2: *Or* dependency, data set A, NSGA-II

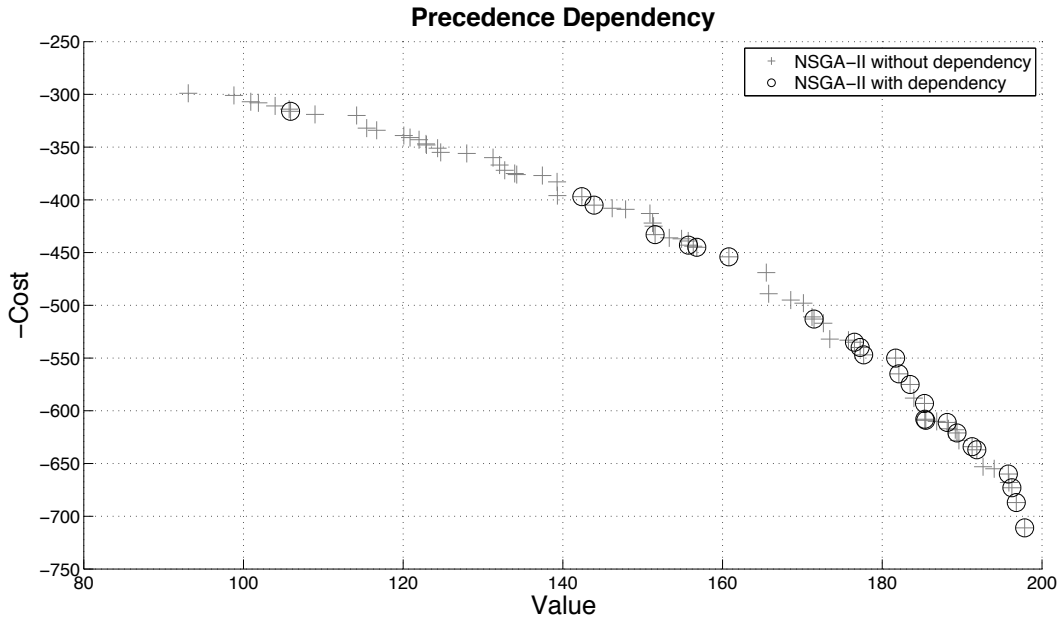

Figure 3: *Precedence* dependency, data set A, NSGA-II

drawn from this graph is that the distribution of '◯' solutions is neither as smooth nor as uniform as the '+' 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) on the Pareto front under *And* dependency constraints.

Compared to the results of *Or* and *Precedence* dependencies in Figures 2 and 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: 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 optimization. The latter two denote problems for which it is relatively easy to find the solutions that satisfy the constraints.

To explore these findings in more detail, we designed a more robust adaptive algorithm for both tight and loose constraints; the archive-based NSGA-II algorithm. The results for three types of constraints are shown in Figures 4, 5 and 6 respectively. The '∗' denotes the solution generated by the archive-based version of the NSGA-II algorithm and the '◯' by NSGA-II.

From Figure 4, we can see the archive based '∗' solutions actually reach all the points on the previous '◯' 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 and 6.

Finally, all three dependencies were taken into consideration to access their overall combined impact. In the Figure 7, the '◯' solutions denote the final results that satisfy all the dependencies constraints. Combining *And*, *Or* and *Precedence* dependencies together, the constraints become much tighter. It is easy to see that very few '◯' solutions remain on the Pareto front based on the NSGA-II algorithm. By contrast, Figure 8 shows a smooth, relatively non-interrupted Pareto front, consisting of '∗' solutions, generated by archive based NSGA-II.

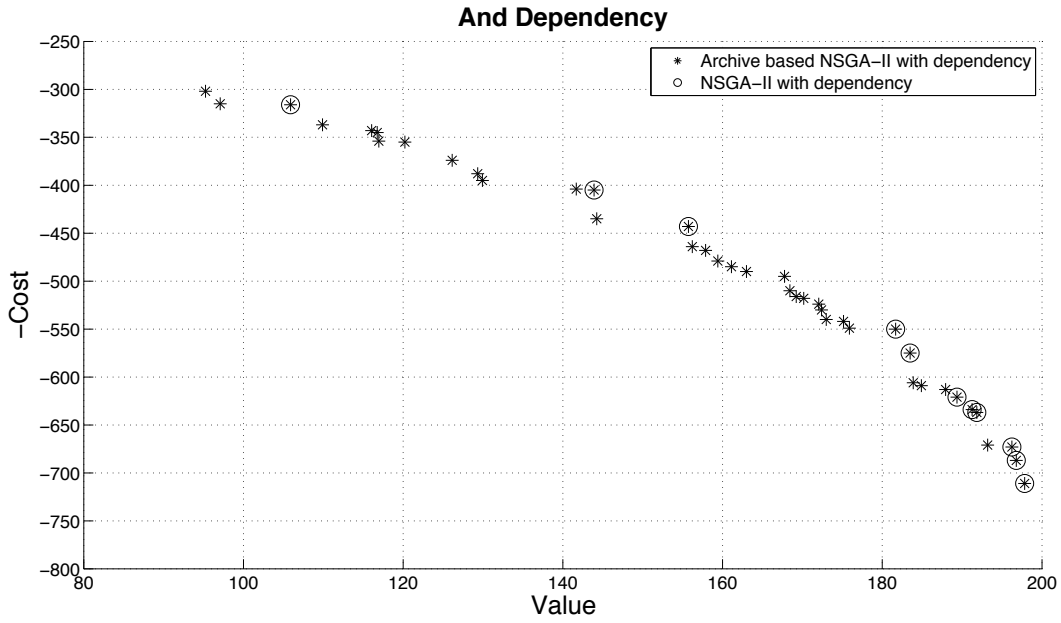

Figure 4: *And* dependency, data set A, NSGA-II and Archive based NSGA-II

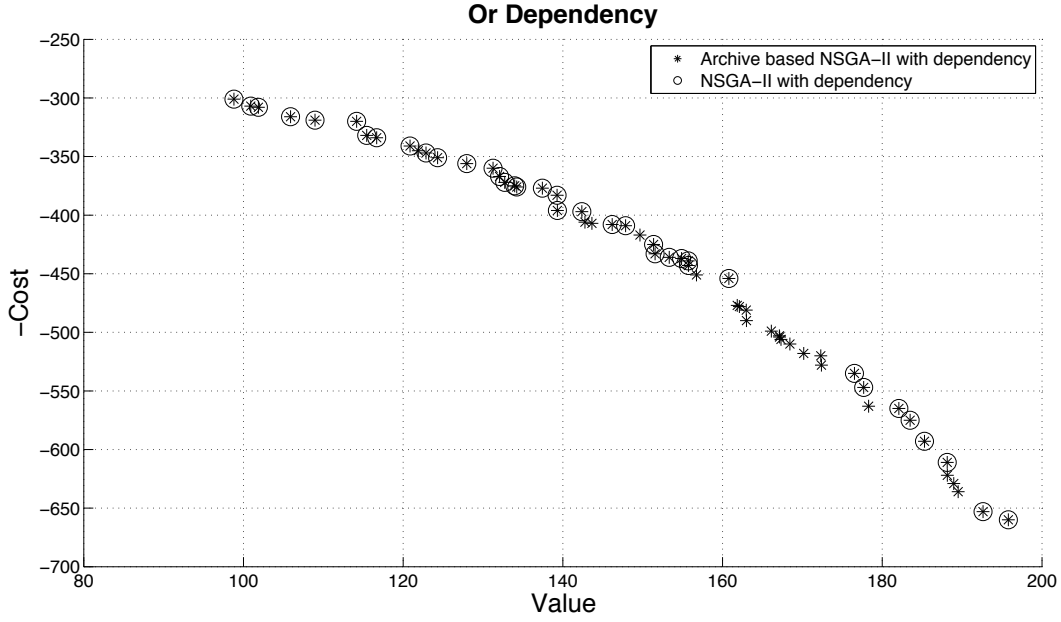

Figure 5: *Or* dependency, data set A, NSGA-II and Archive based NSGA-II

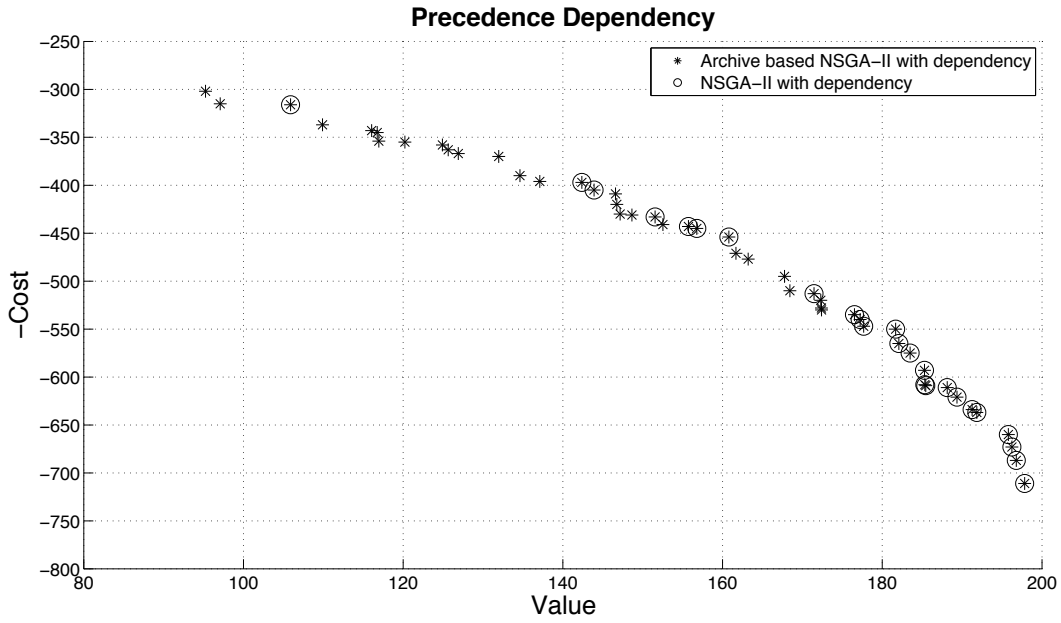

Figure 6: *Precedence* dependency, data set A, NSGA-II and Archive based NSGA-II

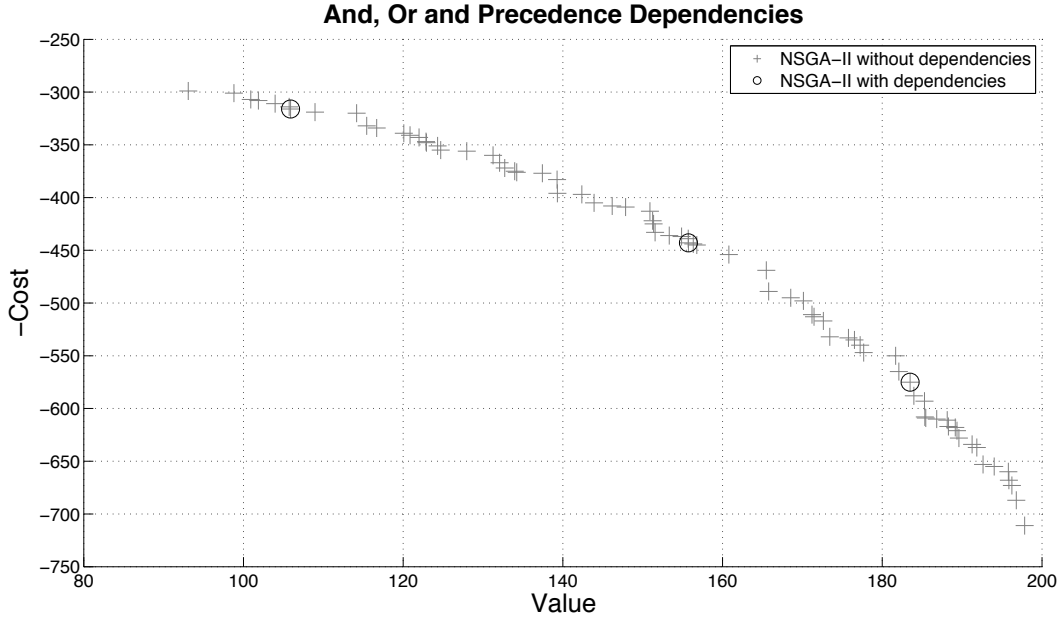

Figure 7: *And*, *Or* and *Precedence* dependencies, data set A, NSGA-II

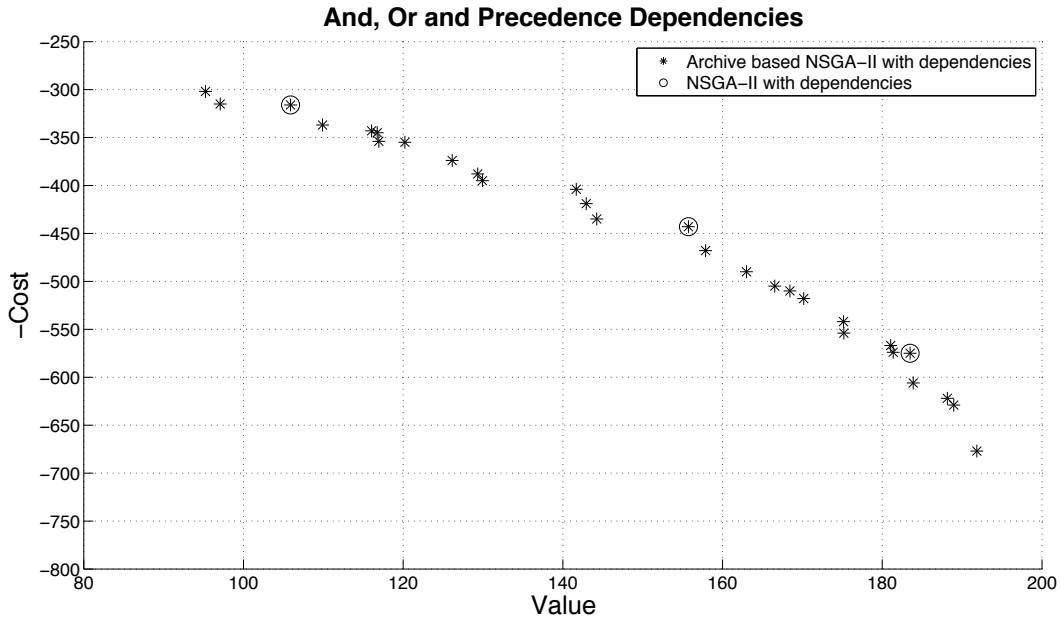

Figure 8: *And*, *Or* and *Precedence* dependencies, data set A, NSGA-II and Archive based NSGA-II

4.1.3 And and Or for Data Set RALIC

In this section we present the results applied to data set RALIC to handle *And* and *Or* dependencies. Because of the highly tight constraints, NSGA-II and archive based NSGA-II could not find a single feasible solution in the experiments. Feasible solutions that satisfy *And* and *Or* constraints only generated successfully by the repair method. The results are shown in Figure 9 and 10.

The grey '+' symbol represents the solutions found by the NSGA-II algorithm without regard to requirement dependencies. The '•' symbol represents the feasible solutions generated by the repair method. As explained in Section 3.1, the stakeholders in the data set RALIC assigned two kinds of requirements ratings for each requirement, namely, PointP and RankP. Figure 9 is the PointP data set while Figure 10 is the RankP data set.

As can be seen from Figure 9, unlike the results produced by the archive based NSGA-II algorithm, all the '•′ solutions do not completely fall on the Pareto front composed of grey '+' symbols. The repair method explored the solution space and successfully found solutions in the feasible region. Moreover, there is a substantial increase in the number of '•′ solutions found compared to the number of '+' solutions. Another observation from this graph is that the diversity of solutions is significantly improved.

Figure 10 presents the results for the RankP data set, also follow a similar pattern and for which we made the same observations. The Pareto front comprised of '•′ solutions is smooth and relatively non-interrupted. The repair method also helped to find the two extreme points of the Pareto front. An important observation is that the '•′ solutions on the graph dominate the great majority of the '+' solutions (without dependencies), which demonstrates that the repair method not only can generated the robust solutions for RIM but also can be used to enhance the (convergency) quality of solutions in the problems without constraints.

In terms of computational effectiveness, we measured the execution time of the NSGA-II algorithm without dependencies and the repair method dealing with *And* and *Or* constraints. The results listed in Table 5 are the average value of 20 executions. The unit of time measured is CPU time.

Data Set	NSGA-II without dependencies	Repair method with dependencies
PointP	350.05	359.59
RankP	348.06	362.67

Table 5: Average CPU Time of Data Sets

From the table we can see, the amount of computational time for finding infeasible solutions and repairing them

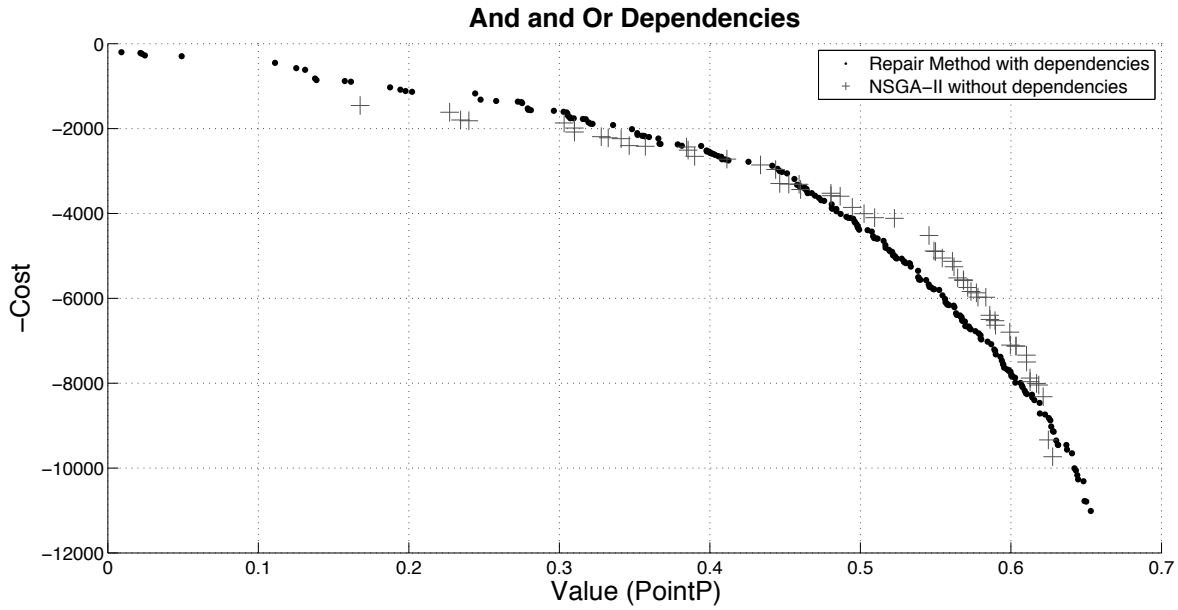

Figure 9: *And* and *Or* dependencies, RALIC Data Set – PointP, Repair Method

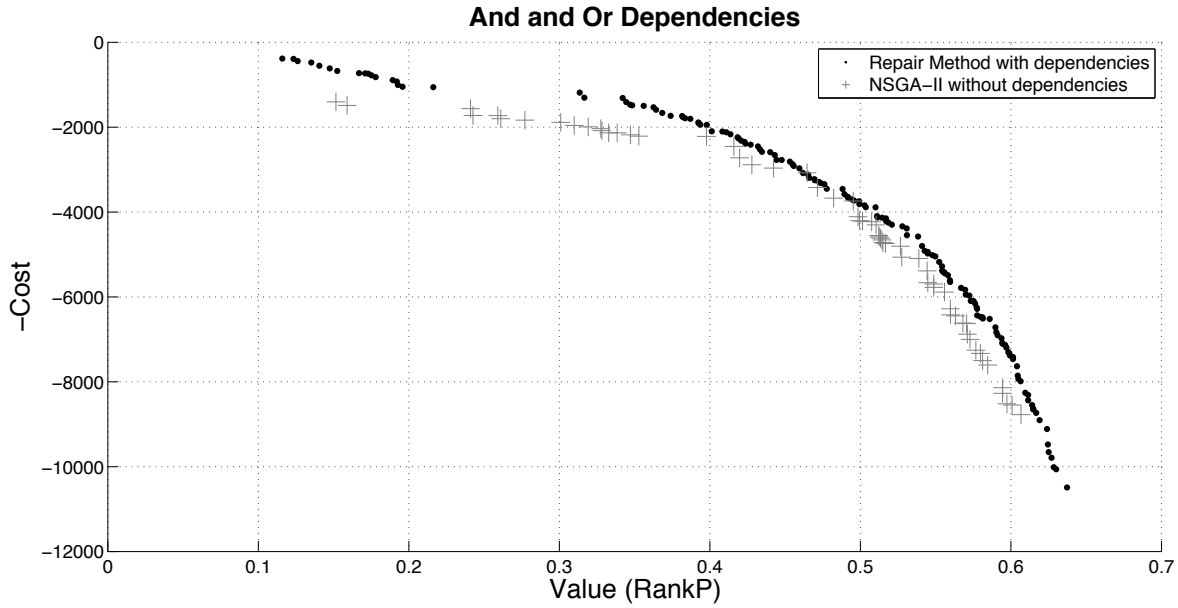

Figure 10: *And* and *Or* dependencies, RALIC Data Set – RankP, Repair Method

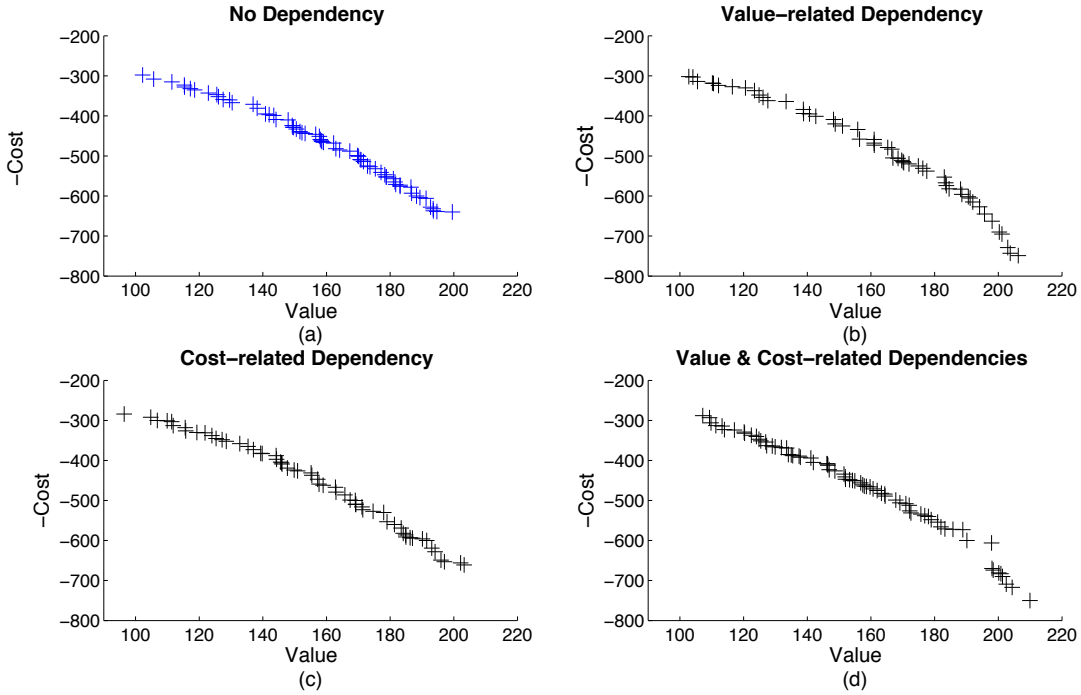

Figure 11: *Value-related* and *Cost-related* dependencies, data set A, NSGA-II

to feasible ones is minor compared to the total execution time. So the repair method is a computationally effective algorithm for solving this RIM-based highly constrained optimization problem.

In this way, the repair method can provide stable, fruitful and computationally effective solutions, which are not merely 'good enough' but also 'robust enough' under the strict constraints that characterize the problem.

4.1.4 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 11. 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.

We observe that the shapes of the four Pareto fronts produced are different. They are not like the previous results of the first three dependencies: eliminating solutions on the unconstrained Pareto fronts or using a 'degenerate' front are not viable for *Value/Cost-related* constraints. *Value/Cost-related* relationships 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.

4.2 Scale Study

In this section, we report on the second empirical study – the Scale Study. The results are presented in the Figures 12, 13 and 14. As described at the beginning of Section 4, 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 3 and Table 4.

In this study, all three dependency constraints are considered together. The results 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; '•' solutions are generated by the repair method.

When the problem is gradually scaled up, from the graphs we can see that the number of the '◯' solutions consistently

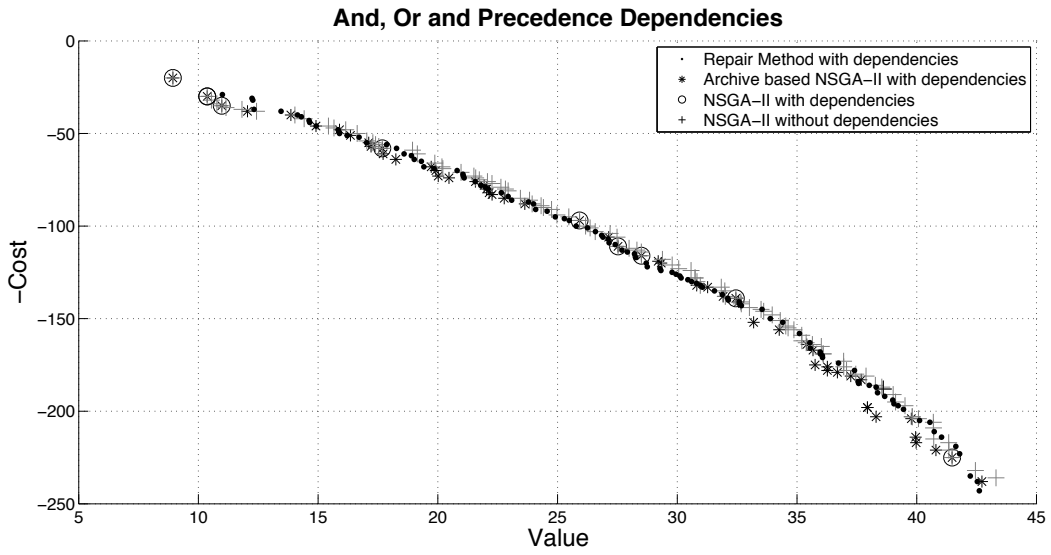

Figure 12: *And*, *Or* and *Precedence* dependencies, data set B, NSGA-II, Archive based NSGA-II and Repair Method

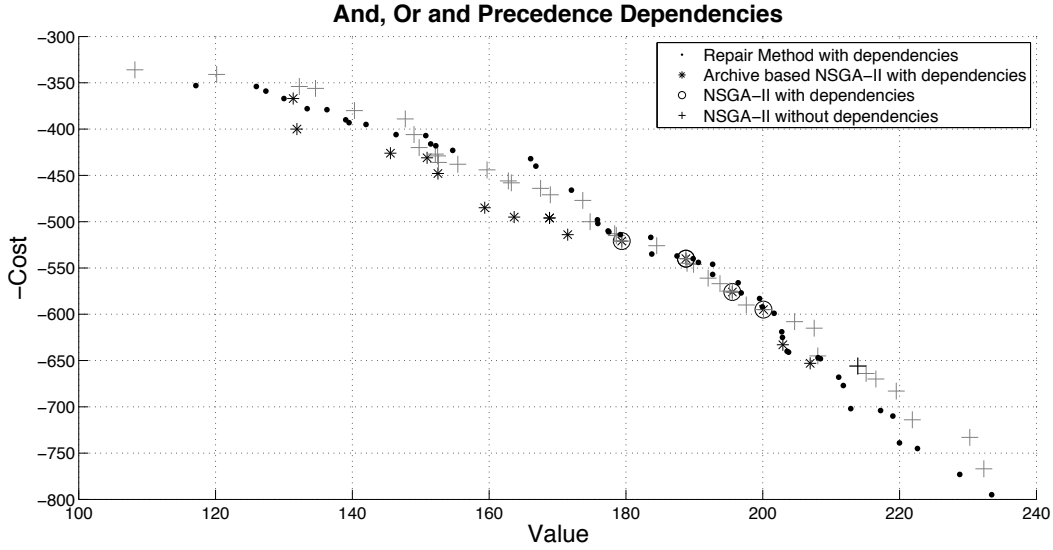

Figure 13: *And*, *Or* and *Precedence* dependencies, data set C, NSGA-II, Archive based NSGA-II and Repair Method

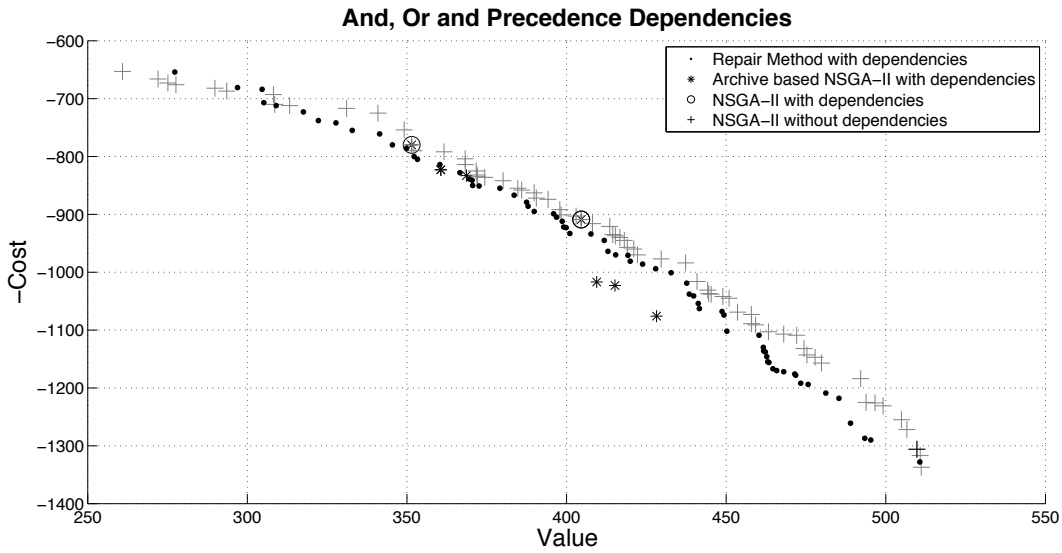

Figure 14: *And*, *Or* and *Precedence* dependencies, data set D, NSGA-II, Archive based NSGA-II and Repair Method

and rapidly decreases. As illustrated in Figure 14, the '◯' solutions have a poor spread over the Pareto front. The '∗' solutions fill some of the gaps among the '◯' solutions and produce a relatively smooth Pareto front. The '•' solutions fill all the gaps among the '◯' solutions and constitute the highly 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 14 than in Figure 12. The 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.

In contrast, the '•' solutions are capable of moving towards the upper right part of the solution space. The repair method even found a number of solutions that dominate the grey '+' solutions. These results indicate that the repair method is able to 'repair' the gaps which open up in the Pareto front and provide good solutions both in quantity and in quality (convergency and diversity) when RIM constraints are imposed.

5 Related Work

Dependence analysis is a part of the overall traceability problem for requirements engineering. The task of requirement traceability is to identify and document traceability links among requirements and between requirements and following SE activities in both a forwards and backwards direction [14]. Requirements traceability is crucial for the success of the system. It enables detection of the conflicting requirements and reduction of missing requirements. Furthermore, it can track the progress of a project, assess the impact of various changes and provide complete information in the SE lifecycle.

There are many ways to represent traceability links. The traceability matrix [34] and cross references [14] are both regarded as good practice which have been widely used in industry. In addition, a large number of requirements tools support traceability management [14]. One of the most well known tools is DOORS (Dynamic Object Oriented Requirements System) [1]. Pohl [31] 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. [20] opened up the discussion on supporting requirements dependencies in the requirements selection process. Robinson et al. [35] provided the basic concepts and scope of Requirements Interaction Management (RIM). They introduced the RIM process in general and a historical perspective of RIM. Carlshamre and Regnell [4] described a two-dimensional (*scope* and *explicitness*) representation to investigate different types of dependencies. Subsequently, Carlshamre et al. [5] 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 [9, 10] provided an overview of research work concerning comparing and validating the different requirements dependencies classification frameworks. There was some work that suggested

that proper treatment of RIM should take account of different types of requirement interactions [5, 15, 20, 9, 36].

In terms of the relevant methods proposed for constraint handling using optimization techniques, there are generally five types of approaches [7]:

1. penalty functions [37];
2. preserving the feasibility of solutions [21];
3. repair methods [13];
4. separation of constraints and objectives [32];
5. hybrid methods [2].

The most commonly used of these are penalty functions and repair methods [6]. Penalty functions were first introduced by Courant [8], who quantified the extent of constraint violation in an infeasible solution and assign a certain amount of penalty to the objective functions. Penalty functions can deal with both equality and inequality constraints. The simplest way to handle infeasible solutions is to discard them for the next iteration. The method is called "death penalty" and is computationally efficient because no further calculation needed. However, the drawback is no information can be extracted from the infeasible solutions to guide the search towards the feasible region.

Homaifar et al. [16] then proposed static penalty methods which refine the degree of constraint violation into different levels. A weight factor is assigned to each constraint, which is the penalty coefficient defined by the user. The performance of the method relies heavily on the proper selection of penalty coefficient parameter. In order to reduce the burden of choosing parameters, several authors have proposed dynamic and adaptive penalty methods. For example, Michalewicz and Attia [27] and Carlson Skalak et al. [38] presented the methods inspired by the cooling analog employed by simulated annealing [17]. The penalty is initially small and its impact on the objective functions is consequently minor. As it subsequently increases over time, infeasible solutions are also gradually penalized more severely. This mimics the cooling process, in which early stages of the search are less constrained while subsequent phases become progressively more constraints.

Adaptive penalty methods take a alternative route to alleviate the difficulty of determining a suitable penalty factor for infeasible solutions. They continue adjusting the penalty factor according to the feedback information from the search iteration. Stochastic Ranking was introduced by Runarsson and Yao [37]. In this approach, the penalty factor is not needed. It is substituted by a probability factor P_f that finds a balance between objective functions and the constraint violations. However, most of these penalty functions have the disadvantage that they may never generate feasible solutions if the problem is highly constrained.

By contrast, repair methods [13, 29] attempt to directly fix infeasible solutions using heuristics to guide the repair process. In other words, a feasible solution can be generated from an infeasible one. Compared to penalty functions, this repair–based approach introduces few additional parameters and can usually return feasible solutions. Combinatorial optimization problems such as graph coloring problem [19], knapsack problems [26] and traveling salesman problems [22] tend to be good candidates for the application of repair methods, because these problems' constraints and decision variables are usually easily characterized.

There are several repair heuristic schemes. Liepins et al. [13] first proposed a greedy repair technique for a number of constrained optimization problems and demonstrated that the computation time for repairing infeasible solutions is minor compared with overall search process time. Orvosh and Davis [29] presented a probability based greedy repair method to replace the original infeasible solutions with their corresponding repaired ones in the population. The method presented by Liepins et al. [13] did not replace the infeasible chromosome in the population so repair is only used to evaluate the fitness values. This type of method is named Baldwinian repair [18], in contracts to Lamarckian repair [18], which replaces the infeasible solutions by repaired ones.

Ishibuchi et al. [18] compared these two types of repair (Lamarckian and Baldwinian) on multi-objective 0/1 knapsack problems and found that the Baldwinian repair methods outperform the Lamarckian on the knapsack problems studied.

Michalewicz and Nazhiyath [28] presented a co-evolution and repair system called Genocop III. The idea is to develop two types of populations. One population maintains feasible solutions (to linear constraints), while the other maintains a feasible solutions that satisfy all the (linear and nonlinear) constraints. The repair algorithm was used to convert those infeasible solutions in the first population into feasible ones in the second. Chootinan and Chen [6] used gradient information extracted from the constraints to repair the infeasible solutions.

The overall conclusion of the literature on the use of repair and approaches to handle infeasible solutions indicated that these constraint–handling issues are inherently problem–specific [7] and that experimentation is required in order to determine the most promising way to handle constraints in an effective and efficient manner. It is in providing this experimentation for constraints involved in Requirement Interaction Management that the present paper seeks to make a contribution.

6 Summary

This paper 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 real world RALIC project and a "27 combination random data sets" model were adopted to develop a procedure in order to better understand real world situations. In the synthetic data set, 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; 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 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 optimization. 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 smaller scale to a relatively larger one were applied. The results showed that repair method could produce a smooth, relatively non-interrupted Pareto front compared to NSGA-II and archive based NSGA-II. When the data set was gradually scaled up, the number of solutions generated by the latter consistently and rapidly decreases. Instead, repair method could still find better feasible (even dominating) solutions both in quality (diversity and convergence) 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 into 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.

References

[1] IBM Rational DOORS (Dynamic Object Oriented Requirements System), http://www.telelogic.com/Products/doors/.

[2] Hojjat Adeli and Hojjat Adeli. Augmented Lagrangian Genetic Algorithm for Structural Optimization. *Journal of Aerospace Engineering*, 7(1):104–118, 1994.

[3] 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.

[4] Pär Carlshamre and Björn 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.

[5] Pär Carlshamre, Kristian Sandahl, Mikael Lindvall, Björn Regnell, and Johan 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.

[6] Piya Chootinan and Anthony Chen. Constraint Handling in Genetic Algorithms using A Gradient-based Repair Method. *Computers and Operations Research*, 33(8):2263–2281, August 2006.

[7] Carlos A. Coello Coello. Theoretical and Numerical Constraint-handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, January 2002.

[8] Richard Courant. Variational Methods for the Solution of Problems of Equilibrium and Vibrations. *Bulletin of the American Mathematical Society*, 49(1):1–23, 1943.

[9] Åsa G. Dahlstedt and Anne 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.

[10] Åsa G. Dahlstedt and Anne Persson. *Engineering and Managing Software Requirements*, chapter 5 Requirements Interdependencies: State of the Art and Future Challenges, pages 95–116. Springer Berlin Heidelberg, 2005.

[11] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[12] Xavier Franch and Neil 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.

[13] W.D. Potter G.E. Liepins. *A Genetic Algorithm Approach to Multiple-fault Diagnosis*, chapter 17, pages 237–250. Van Nostrand Reinhold, New York, 1991.

[14] Orlena C. Z. Gotel and Anthony 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.

[15] Des Greer and Günther Ruhe. Software Release Planning: An Evolutionary and Iterative Approach. *Information & Software Technology*, 46(4):243–253, March 2004.

[16] Abdollah Homaifar, Charlene X. Qi, and Steven H. Lai. Constrained Optimization Via Genetic Algorithms. *International Transactions of the Society for Modeling and Simulation*, 62(4):242–253, April 1994.

[17] Chii-Ruey Hwang. Simulated annealing: Theory and applications. *Acta Applicandae Mathematicae*, 12(1):108–111, 1988.

[18] Hisao Ishibuchi, Shiori Kaige, and Kaname Narukawa. Comparison between Lamarckian and Baldwinian Repair on Multiobjective 0/1 Knapsack Problems. In *Proceedings of the 3rd International Conference on Evolutionary Multi-Criterion Optimization*, LNCS 3410, pages 370–385, Guanajuato, Mexico, 9-11 March 2005. Springer.

[19] Tommy R. Jensen and Bjarne Toft. *Graph Coloring Problems*. Wiley-Interscience, 1994.

[20] Joachim Karlsson, Stefan Olsson, and Kevin Ryan. Improved Practical Support for Large-scale Requirements Prioritizing. *Requirements Engineering Journal*, 2(1):51–60, 1997.

[21] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 7(1):19–44, 1999.

[22] E. L. Lawler, Jan Karel Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, 1985.

[23] S.L. Lim. Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation. *PhD thesis*, 2010.

[24] S.L. Lim and A. Finkelstein. StakeRare: Using Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation. *IEEE Transactions on Software Engineering*, in press.

[25] S.L. Lim, D. Quercia, and A. Finkelstein. StakeNet: using social networks to analyse the stakeholders of large-scale software projects. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 295–304. ACM, 2010.

[26] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[27] Zbigniew Michalewicz and Naguib F. Attia. Evolutionary Optimization of Constrained Problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 98–108. World Scientific, 1994.

[28] Zbigniew Michalewicz and Girish Nazhiyath. Genocop III: A Co-evolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints. In *Proceedings of IEEE International Conference on Evolutionary Computaion (CEC '95)*, pages 647–651, Perth, Australia, 29 November - 1 December 1995. IEEE.

[29] David Orvosh and Lawrence Davis. Using a Genetic Algorithm to Optimize Problems with Feasibility Constraints. In *Proceedings of the 1st IEEE World Congress on Computational Intelligence*, pages 548–553, Orlando, USA, 27-29 June 1994.

[30] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. 1999.

[31] Klaus Pohl. *Process-Centered Requirements Engineering*. Research Studies Press, 1996.

[32] David Powell and Michael M. Skolnick. Using Genetic Algorithms in Engineering Design Optimization with Non-Linear Constraints. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 424–431, Urbana, USA, 17-21 July 1993. Morgan Kaufmann Publishers Inc.

[33] K. Praditwong and Xin 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.

[34] 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.

[35] William N. Robinson, Suzanne D. Pawlowski, and Vecheslav Volkov. Requirements Interaction Management. Technical Report 99-7, Georgia State University, August 1999.

[36] William N. Robinson, Suzanne D. Pawlowski, and Vecheslav Volkov. Requirements Interaction Management. *ACM Computing Surveys (CSUR)*, 35(2):132–190, June 2003.

[37] Thomas P. Runarsson and Xin Yao. Stochastic Ranking for Constrained Evolutionary Optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.

[38] S. Carlson Skalak, R. Shonkwiler, S. Babar, and M. Aral. Annealing A Genetic Algorithm Over Constraints. In *Proceeding of IEEE International Conference on Systems, Man, and Cybernetics*, pages 3931 – 3936, San Diego, CA , USA, 11-14 Oct 1998. IEEE.

[39] Sangameswar Venkatraman and Gary G. Yen. A Generic Framework for Constrained Optimization using Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 9(4):424–435, August 2005.

[40] Yuanyuan Zhang, Enrique Alba, Juan J. Durillo, Sigrid Eldh, and Mark Harman. Today/Future Importance Analysis. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO '10)*, pages 1357–1364, Portland, USA, 7-11 July 2010. ACM. To appear.

[41] Yuanyuan Zhang and Mark Harman. Search Based Optimization of Requirements Interaction Management. In *Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE '10)*, pages 47–56, Benevento, Italy, 7-9 September 2010. IEEE.