Information and Software Technology 55 (2013) 126–152

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Empirical evaluation of search based requirements interaction management

Yuanyuan Zhang *, Mark Harman, Soo Ling Lim

University College London, Malet Place, London WC1E 6BT, UK

ARTICLE INFO

Article history:
Available online 21 April 2012

Keywords:
Requirements
RIM
NSGA-II
Repair method
Dependency
Search-Based Software Engineering

ABSTRACT

Context: Requirements optimization has been widely studied in the Search Based Software Engineering (SBSE) literature. However, previous approaches have not handled requirement interactions, such as the dependencies that may exist between requirements, *and*, *or*, *precedence*, *cost*- and *value*-based constraints.
Objective: 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.[1]
Method: 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.
Results: The repair based approach was found to produce more solutions on the Pareto front and better convergence and diversity of results than the previously studied NSGA-II and archive-based NSGA-II approaches based on Kruskal–Wallis test in most cases. The repair based approach was also found to scale almost as well as the previous approach.
Conclusion: 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.

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. This relationship can directly affect 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" [2].

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" [3].

Robinson et al. [3] 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. 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.

* Corresponding author. Tel.: +44 (0) 20 7679 1056; fax: +44 (0) 20 3108 5040.
E-mail addresses: yuanyuan.zhang@cs.ucl.ac.uk (Y. Zhang), m.harman@cs.ucl.ac.uk (M. Harman), sooling.lim@cs.ucl.ac.uk (S.L. Lim).

[1] This paper is an invited extension of the previous conference paper that appeared at SSBSE 2010 [1]. The primary novelty of this paper over its predecessor is the introduction of the repair based approach and the extension of the evaluation to include the real world data set (RALIC) in addition to the synthetic data used in the conference version.

0950-5849/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2012.03.007

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	Requirement R_1 is selected before selecting 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

Requirements dependency can have a very strong impact on the development process. For example, Bagnall et al. [4] considered the *Precedence* dependency, 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. The authors showed that stakeholder subset selection process could be affected by *Precedence* dependency. Greer and Ruhe [5] considered *And* and *Precedence* dependencies. The proposed system provided candidate subsets of requirements for the next release problem. As with previous work, the authors use a single objective formulation, taking two type dependencies and the resource budget as the constraints. More recently, Tonella et al. [6] considered *Precedence* dependency in the requirements prioritization process. Similar to Bagnall et al. [4], *Precedence* relations were represented in a directly acyclic graph and treated as constraints for the ordering of the requirements.

And and *Precedence* dependencies were considered in the literature. However, little work has been studied on all five dependencies together. Proper treatment of RIM should take account of all the different types of requirement interactions. In this paper, we treat RIM problem as a constraint satisfaction problem and propose three search-based algorithms to handle all of the five common types of requirement dependencies for the first time. 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.

The paper will show how multi-objective SBSE can be adapted to take account of RIM. 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 [1] are adopted in this paper to evaluate the approach.

The rest of the paper is organized as follows. In Section 2 the problem is formalized as an SBSE problem. Section 3 describes the data sets, algorithms used and the performance metrics. 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. Problem formulation

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:

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

A set of stakeholders for a software system or service is 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 $(1 \leqslant j \leqslant m)$ 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 $r_i (1 \leqslant i \leqslant n)$ 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 $(1 \leqslant j \leqslant m)$ assigns a *value* to requirement r_i $(1 \leqslant i \leqslant n)$ 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 $(1 \leqslant i \leqslant n)$ 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.

Next, we define the RIM constraints that were listed informally in the introduction to this paper.

And	Define a pair of requirements (i, j) and a set ξ such that $(i, j) \in \xi$ means that r_i is selected if and only if requirement r_j has to be chosen
Or	Define a pair of requirements (i, j) and a set φ such that $(i, j) \in \varphi$ (equivalently $(j, i) \in \varphi$) means that at most one of r_i, r_j can be selected
Precedence	Define a pair of requirements (i, j) and a set χ such that $(i, j) \in \chi$ means that requirement r_i has to be implemented before requirement r_j
Value-related	Define a pair of requirements (i, j) and a set ψ such that $(i, j) \in \psi$ 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 pair of requirements (i, j) and a set ω on the requirements array R such that $(i, j) \in \omega$ means that if the requirement r_i is selected, then its inclusion affects the cost of implementing requirement r_j

In addition, the sets 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.

The fitness function with dependency constraints is defined as follows:

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

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

where decision vector $x = \{x_1, \ldots, x_i, \ldots, x_n\} \in \{0, 1\}$ determines the requirements that are to be be satisfied. In the vector, x_i is 1 if requirement r_i is selected and 0 otherwise.

subject to

$x_i = x_j(i,j) \in \xi$ (And constraints)

$x_i \neq x_j \vee x_i = x_j = 0(i,j) \in \varphi$ (Or constraints)

$x_i >= x_j(i,j) \in \chi$ (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:

$$\text{If } x_i = x_j = 1 \; r(i,j) \in \psi \Rightarrow$$

$$f_1(\vec{x}) = \sum_{k=1}^{n} score_k \cdot x_k + (score_i + score_j) \cdot IF$$

$$\text{If} x_i = x_j = 1 \; r(i,j) \in \omega \Rightarrow$$

$$f_2(\vec{x}) = -\sum_{k=1}^{n} cost_k \cdot x_k + (cost_i + cost_j) \cdot IF$$

where *IF* is a set {−0.4, −0.2, 0.2, 0.4}, which stands for Influence Factor. When calculating the requirements' inclusion influence, a value is randomly selected from *IF* set.

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 [7]. 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 was deployed in 2009.

The stakeholder priority data for RALIC was collected in previous work using the StakeNet stakeholder analysis method [8] 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 [8]. The PageRank weight of the stakeholder j is the w_j in Eq. (1).

PageRank is the algorithm used by Google to rank documents [9]. 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 [7] described as follows. Two data sets were used in this work: PointP and RankP.

- For the PointP data set, there are 143 requirements and 77 stakeholders. Each stakeholder identified by StakeNet was asked to distribute 100 points among the requirements they want [7]. 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 0–1.
- For the RankP data set, 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) [7]. Stakeholders also marked "X" for requirements they actively do not want. The marking "X" (from stakeholders actively not wanting a requirement) was converted to 0. Fractional ranking was used such that if a tie in ranks occurs, the mean of the ranks involved was assigned to each of the tied items. For example, if a stakeholder ranks r_1 as 1, r_2 as 1, and r_3 as 2, the ranks of r_1 and r_2 become 1.5 and the rank of r_3 becomes 3. As lower rank values correspond to higher importance, the rank values were inverted by subtracting them from the maximum rank +1. Following the previous example, the inverted ranks for r_1 and r_2 are $(3 + 1) - 1.5 = 2.5$ and the inverted rank for r_3 is $(3 + 1) - 3 = 1$. The rankings were normalized such that the sum of all the ranks from each stakeholder adds up to 1. This was done by dividing the inverted rank with the sum of all the inverted ranks. Following the previous example, the inverted and normalized ranks for r_1 and r_2 are $\frac{2.5}{2.5+2.5+1} = 0.42$, and the inverted and normalized rank for r_3 is $\frac{1}{2.5+2.5+1} = 0.17$.

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 [10].

In the previous work, the requirements in RankP and PointP are organized into a hierarchy of three levels: project objective, requirement, and specific requirement [7]. A (parent) requirement is satisfied once all its (child) specific requirements are satisfied. As such, child requirements with the same parent requirement have *And* relationships among each other. Only the child requirements are considered in this work, as the higher level requirements are achieved when the child 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 [11] were adopted in this study. There are 4 of the 27 data sets used in the empirical studies. They 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).

The number of stakeholders (S) and the number of requirements (R) are divided into three categories, namely, small scale, medium scale and large scale; the Density of the stakeholder-requirement matrix (D) is defined as low, medium and high level. It is calculated as follow:

$$D = \left(\sum_{i=1}^{S} \text{Number of Requirements selected by Stakeholder } i\right) \Big/ (S \cdot R)$$

As can be seen in Table 1, the data set divides the range of a variable into a finite number of non-overlapping intervals of unequal width. With three variables (S, R, D) each taking three levels, there are 27 combination possibilities in total. Table 2 lists the combination of all cases schematically.

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 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.

Four data sets were generated and used in the empirical studies. They exhibit different scales and densities, according to the approach to data set generation depicted in Tables 1 and 2. We named the sets A, B, C and D. In the data set A, 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 data sets B, C and D were generated.

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 $Cos(i, j)$ ($1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant n$) are defined to represent the five requirements dependency types.

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

$And(i, j) = 1 \wedge And(j, i) = 1$ 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.

The specific scales chosen for data sets A, B, C and D are listed in Table 3. According to the scale range in Table 2, real world data sets PointP and RankP are constrained to a "large" scale in the number of stakeholders, a "medium" scale in the number of requirements and with a "low" density of stakeholder-requirement matrix. Therefore, data sets PointP and RankP are labeled as $S_l R_m D_{low}$ in Table 3. 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 Non-dominated Sorting Genetic Algorithm-II (NSGA-II), archive based NSGA-II and a repair method based NSGA-II for constraints handling.

Table 1
Scale range of "27 combination random data sets".

	Small	Medium	Large
No. of stakeholders (S)	2–5	6–20	21–50
No. of requirements (R)	1–100	101–250	251–600
	Low	Medium	High
Density of matrix (D)	0.01–0.33	0.34–0.66	0.67–1.00

Table 2
27 Combination random data sets.

	R_{small}			R_{medium}			R_{large}		
S_{small}	S_s	R_s	D_{low}	S_s	R_m	D_{low}	S_s	R_l	D_{low}
	S_s	R_s	D_m	S_s	R_m	D_m	S_s	R_l	D_m
	S_s	R_s	D_h	S_s	R_m	D_h	S_s	R_l	D_h
S_{medium}	S_m	R_s	D_{low}	S_m	R_m	D_{low}	S_m	R_l	D_{low}
	S_m	R_s	D_m	S_m	R_m	D_m	S_m	R_l	D_m
	S_m	R_s	D_h	S_m	R_m	D_h	S_m	R_l	D_h
S_{large}	S_l	R_s	D_{low}	S_l	R_m	D_{low}	S_l	R_l	D_{low}
	S_l	R_s	D_m	S_l	R_m	D_m	S_l	R_l	D_m
	S_l	R_s	D_h	S_l	R_m	D_h	S_l	R_l	D_h

Table 3
Scale of synthetic and real world data sets: exploration of the configuration space for RIM.

	R_{small}	R_{medium}	R_{large}
S_{small}			**C**:$S_s\ R_l\ D_m$
S_{mediun}		**A**:$S_m\ R_m\ D_m$ **PointP**, **RankP**: $S_l\ R_m\ D_{low}$	
S_{large}	**B**: $S_l\ R_s\ D_m$		**D**: $S_l\ R_l\ D_h$

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

3.2.1. NSGA-II

NSGA-II was introduced by Deb et al. [12]. It incorporates elitism to maintain the solutions of the Pareto front found. 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 [12]. The process is described in Algorithm 1.

Algorithm 1. NSGA-II (the main loop) [12] Deb (2001)

Input: t, P_0, Q_0 and N
Output: S
1 Let $t = 0$;
2 **While** t <= MAX_GENERATION **do**
3 Let $R_t = P_t \cup Q_t$;
4 Let F = fast-non-dominated-sort (R_t);
//$F = (F_1, F_2, \ldots, F_i, \ldots)$
5 Let $P_{t+1} = \emptyset$ and $i = 1$;
6 **While** $|P_{t+1}| + |F_i| \leqslant N$ **do**
7 Apply crowding-distance-assignment (F_i);
8 Let $P_{t+1} = P_{t+1} \cup F_i$;
9 Let $i = i + 1$;
10 **end**
11 $Sort(F_i, \prec_n)$;
12 Let $P_{t+1} = P_{t+1} \cup F_i[1: (N - |P_{t+1}|)]$;
13 Let Q_{t+1}= make-new-pop(P_{t+1});
14 Let $t = t + 1$;
15 **end**
16 Let S = constraint-violation-checking (P_t);

Each solution is assigned a fitness value based on the level of non-domination. Between two solutions with differing non-domination levels, the one with the lower (better) rank is preferred. In this way, the solutions in better rank are given higher fitness values.

A measure of crowding distance is used by NSGA-II to provide an estimation of the density of solutions belonging to the same rank. Solutions with higher crowding distance are assigned a higher fitness compared to those with lower crowding distance, aiming to promote diversity within the population.

In terms of constraint handling, in this paper, the NSGA-II algorithm does not consider and handle the dependency constraints during the evolutionary process. Rather, on completion of the algorithm, a separate post-processing phase checks for constraint violations. The solutions that respect the constraints are extracted from the last generation of the algorithm.

3.2.2. Archive based NSGA-II

The modification of archive based NSGA-II is inspired by Praditwong and Yao's two-archive multi-objective evolutionary optimization algorithm [13]. Keeping the last generation of non-dominated solutions is good enough for general multi-objective optimization work. However, when we take into account 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 infeasible solutions. This may mean rejecting otherwise 'optimal' solutions in favor of previously considered and otherwise less optimal solutions.

To preserve these potential candidate solutions, this paper introduces an archive-based variation of the NSGA-II algorithm to retain near optimal solutions. The factor that distinguishes it from NSGA-II is that it archives the best solutions from all the generations during the evolutionary process. Thus there are greater number of candidate solutions for the constraint violation checking than those in the NSGA-II algorithm. The archive process are represented in Algorithm 2.

Algorithm 2. Archive based NSGA-II (the main loop)

Input: t, P_0, Q_0, N and *Archive*
Output: S
1 Let $t = 0$;
2 Initialize *Archive* to the empty set;
3 **while** t <= MAX_GENERATION **do**
4 Let $R_t = P_t \cup Q_t$;
5 Let F = fast-non-dominated-sort (R_t);
//$F = (F_1, F_2, \ldots, F_i, \ldots)$
6 Let $P_{t+1} = \emptyset$ and $i = 1$;
7 **while** $|P_{t+1}| + |F_i| \leqslant N$ **do**
8 Apply crowding-distance-assignment (F_i);
9 Let $P_{t+1} = P_{t+1} \cup F_i$;
10 Let $i = i + 1$;
11 **end**
12 $Sort(F_i, \prec_n)$;
13 Let = $P_{t+1} \cup F_i[1: (N - |P_{t+1}|)]$;
14 Add P_{t+1} to *Archive*;
15 Let Q_{t+1}= make-new-pop (P_{t+1});
16 Let $t = t + 1$;
17 **end**
18 Let S = constraint-violation-checking (*Archive*);

3.2.3. Repair method based NSGA-II

When solving highly constrained problems, we discovered that the archive based NSGA-II previously used [1] 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.

RIM based requirements selection and optimization is a multi-objective knapsack problem, for which previous work [14] 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 [15], there are two phases for solving constrained optimization problems. Phase one is constraint satisfaction. For every solution generated, the algorithm first checks for constraint violations. If the solution is infeasible, the repair method is used to repair the chromosome. Any 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 evolutionary optimization process. Using fitness functions to guide a search for optimal or near optimal solutions. The main process is presented in Algorithm 3.

The repair method works as follows:

$\exists\ x_i, x_j$ for pair $(i, j) \in \xi$ (*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 $x_j = 0$.

Algorithm 3. Repair method based NSGA-II (the main loop)

```
Input: t, P0, Q0 and N
Output: Pt
Let t = 0;
while t <= MAX_GENERATION do
  Apply constraint-violation-checking (Pt, Qt);
  Apply repair method (Pt, Qt);
  Let Rt = Pt ∪ Qt;
  Let F = fast-non-dominated-sort (Rt);
    //F = (F1, F2, ..., Fi, ...)
  Let Pt+1 = ∅ and i = 1;
  while |Pt+1| + |Fi| ⩽ N do
    Apply crowding-distance-assignment (Fi);
    Let Pt+1 = Pt+1 ∪ Fi;
    Let i = i + 1;
  end
  Sort(Fi, ≺n);
  Let = Pt+1 ∪ Fi[1: (N − |Pt+1|)];
  Add Pt+1 to Archive;
  Let Qt+1 = make-new-pop(Pt+1);
Let t = t + 1;
end
```

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 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).

3.3. Performance metrics

In this section, three performance metrics are introduced to measure the quality of the Pareto fronts produced by the search-based algorithms in the paper.

3.3.1. Convergence

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.

In the case of requirements selection and optimization 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. The reference pareto front is the best pareto front obtainable using all of the algorithms in concert. It is formed from the union of the pareto fronts of each algorithm. From this union, any dominated solutions are removed. The front so-formed is a 'reference' in the sense that it contains the best results obtainable by any and all of the algorithms. As such, it is the best approximation available from the experiments against which to compare each algorithm.

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

3.3.2. Diversity

The metric Δ [16] measures the extent of distribution in the obtained solutions and spread achieved between approximated solutions and the reference Pareto front.

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

where $k (1 \leqslant k \leqslant M)$ is the number of objectives for a multi-objective algorithm. There are two objectives in the paper, so $M = 2$. 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. $d_j (1 \leqslant j \leqslant N-1)$ is the Euclidean distance between consecutive solutions, $\bar{d}$ 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 $(d_j = \bar{d})$ and also include all the extreme solutions on the reference front $(d_k = 0)$.

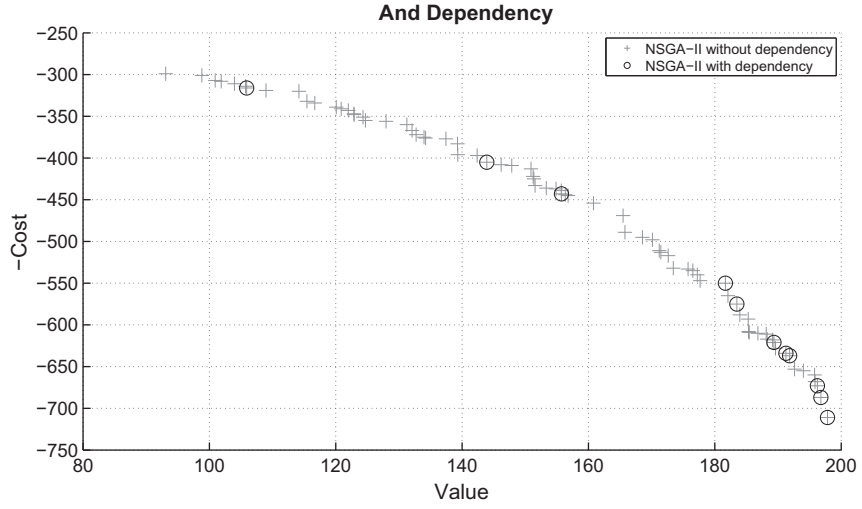

Fig. 1. *And* dependency, data set A, NSGA-II, 1-run solutions.

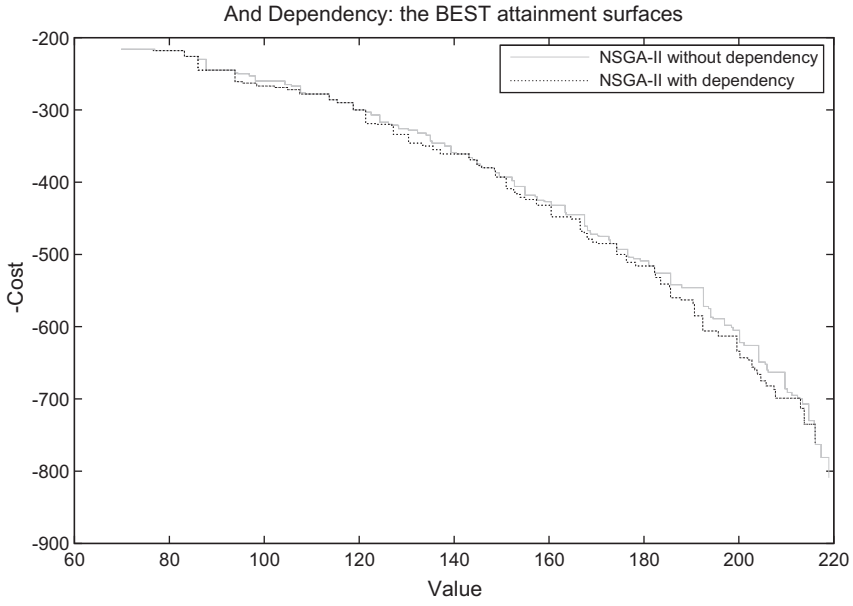

Fig. 2. *And* dependency, data set A, NSGA-II, 20-run best attainment surface.

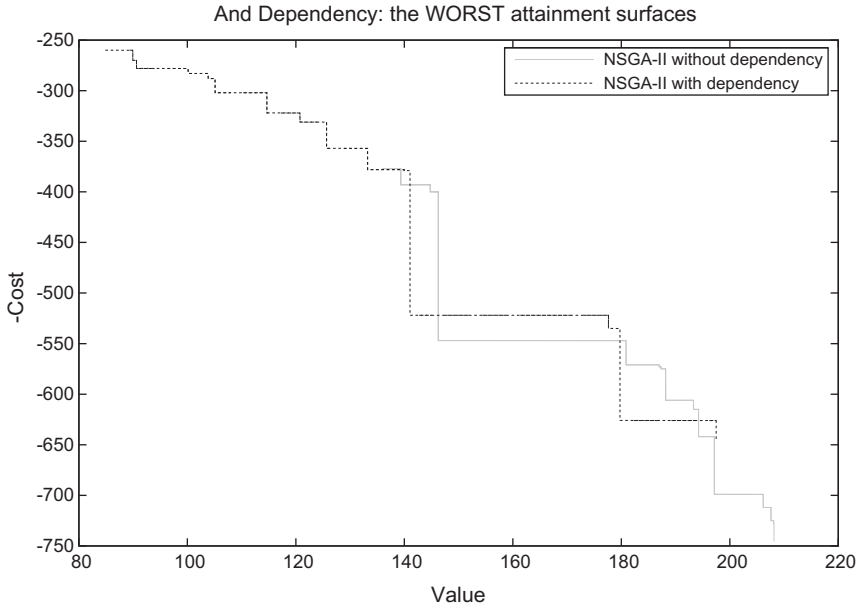

Fig. 3. *And* dependency, data set A, NSGA-II, 20-run worst attainment surface.

3.3.3. Number of solutions

The metric *Num* is the number of solutions on the Pareto front that are obtained by an algorithm. It reports the results in a quantitative manner. The greater the value of *Num*, the better the algorithm's results.

4. Empirical studies and results

4.1. Research questions

Our studies ask two research questions, which we define here, describing how our experiments are designed to address them.

RQ1: How do five dependency types impact on the requirements selection process?

To answer RQ1, a Dependency Impact Study (DIS) was carried out. The RALIC project data sets are used for DIS. Synthetic data set A is also used throughout DIS in order to establish a uniform baseline for the comparison.

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*). In data set A, we will discuss the two scenarios separately. In the RALIC data sets, there are two types (*And* and *Or*) dependencies, so we will only discuss fitness-invariant dependency.

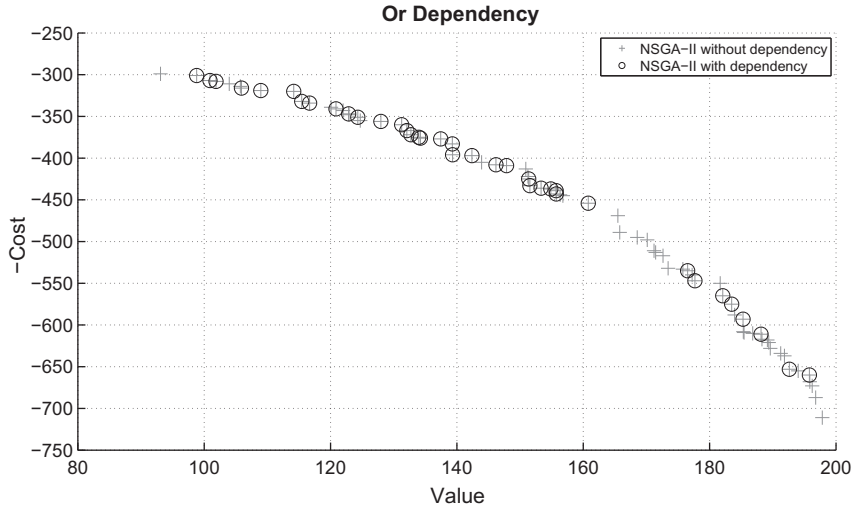

Fig. 4. *Or* dependency, data set A, NSGA-II, 1-run solutions.

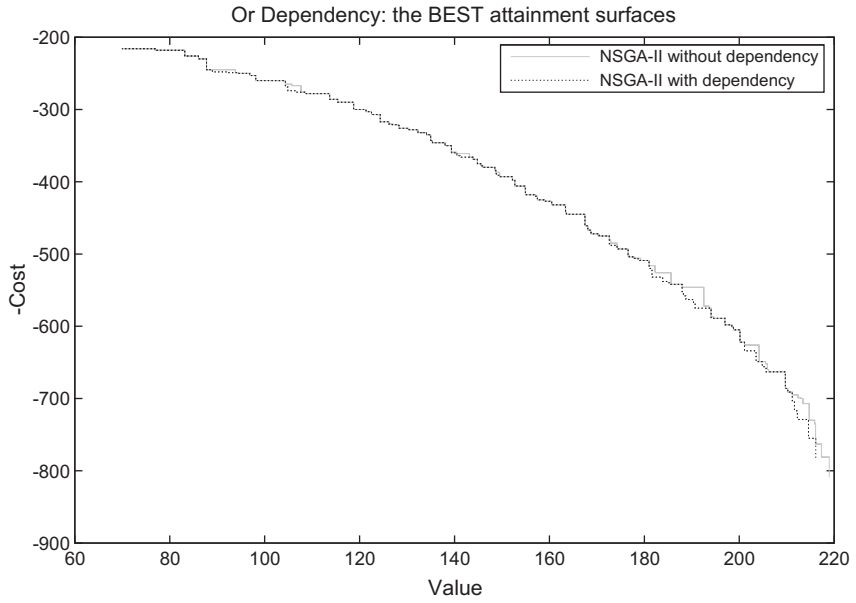

Fig. 5. *Or* dependency, data set A, NSGA-II, 20-run best attainment surface.

Four 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 three fitness-invariant dependencies together in order to seek to investigate the difference among the solutions generated by the two algorithms.
4. Considering two fitness-affecting dependencies together to investigate the difference among the solutions.

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

RQ2: How effective are the algorithms as the data sets increase in size?

The effectiveness is measured using the convergence of solutions, the diversity of solutions and the number of solutions on the Pareto front.

To investigate the performance of three algorithms, a Scale Study (SS) is conducted. There are three data sets B, C and D used for SS with the number of stakeholders ranging from 4 to 34 and the number of requirements ranging from 50 to 412.

To cope with the stochastic nature of the evolutionary process, in both studies, each experiment was repeated 20 times. The same dependency density levels were used for all five dependencies.

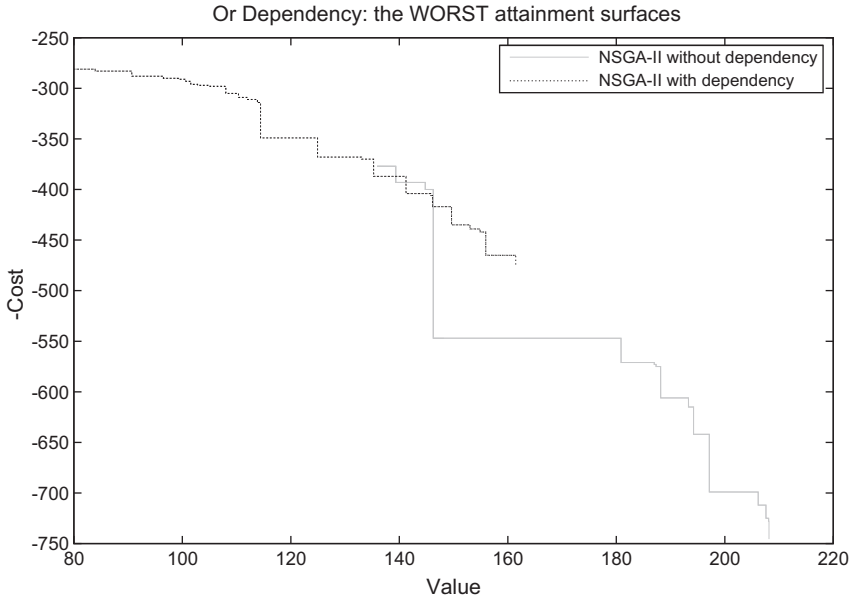

Fig. 6. *Or* dependency, data set A, NSGA-II, 20-run worst attainment surface.

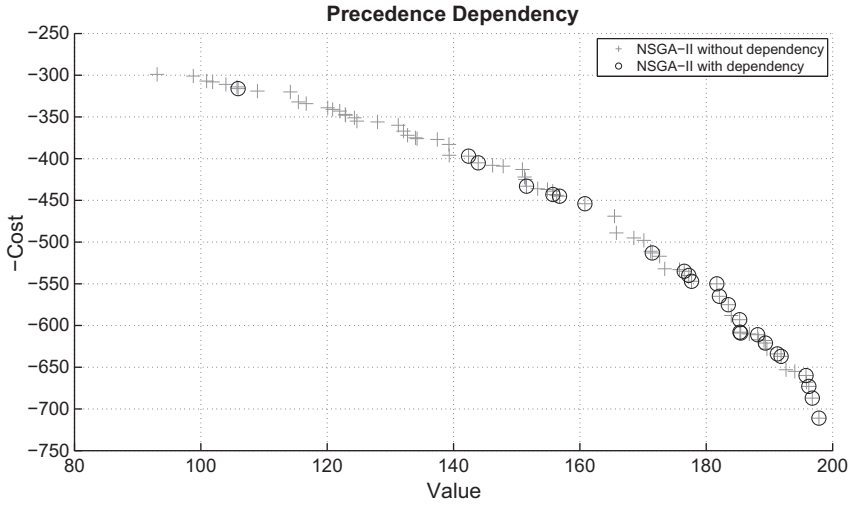

Fig. 7. *Precedence* dependency, data set A, NSGA-II.

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 contains *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.

4.2. Dependency impact study

4.2.1. 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. To deal with the stochastic nature of search techniques, the results of both the single execution of the algorithms and the attainment surfaces of 20 executions are presented in Figs. 1–24. The results of the single execution generated by the standard NSGA-II algorithm are shown in Figs. 1, 4 and 7, and the results from the archive based NSGA-II algorithm are shown in Figs. 10, 13 and 16 separately. The

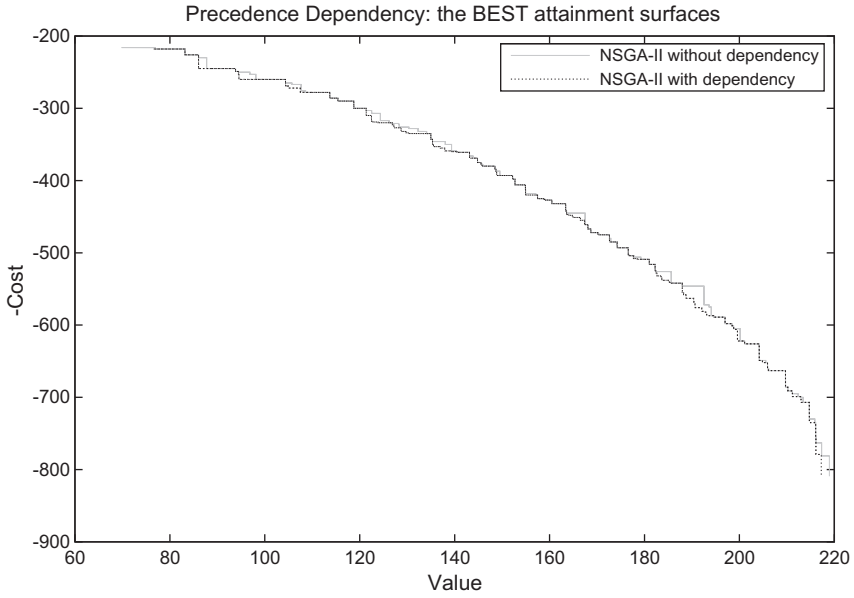

Fig. 8. *Precedence* dependency, data set A, NSGA-II, 20-run best attainment surface.

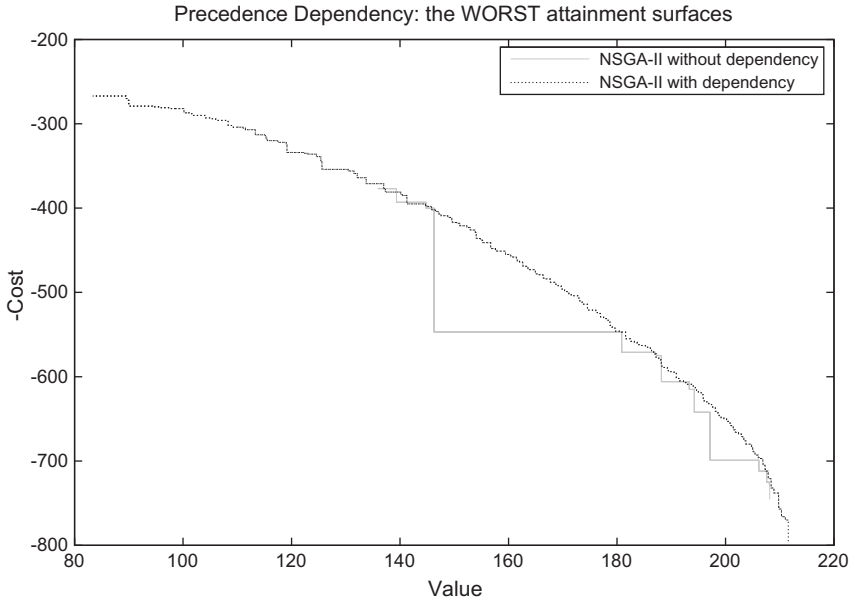

Fig. 9. *Precedence* dependency, data set A, NSGA-II, 20-run worst attainment surface.

results of the attainment surfaces are presented in the rest of the figures.

For the single execution results, 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 gray to distinguish them from the others (which do take account of dependencies).

In Figs. 1, 4 and 7, we observe that the shapes of Pareto fronts, consisting of a number of gray '+' symbols, are the same. These are solutions generated by the NSGA-II algorithm without consideration of requirements dependency relationships 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 Fig. 1 for which *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 gray '+' 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 their failure to meet dependency constraints. Another obvious observation 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 on the Pareto front under *And* dependency constraints.

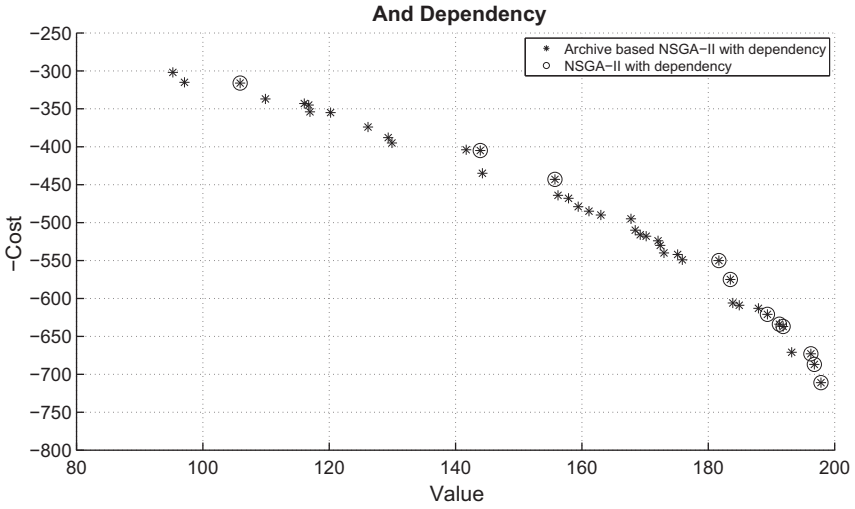

Fig. 10. *And* dependency, data set A, NSGA-II and archive based NSGA-II, 1-run solutions.

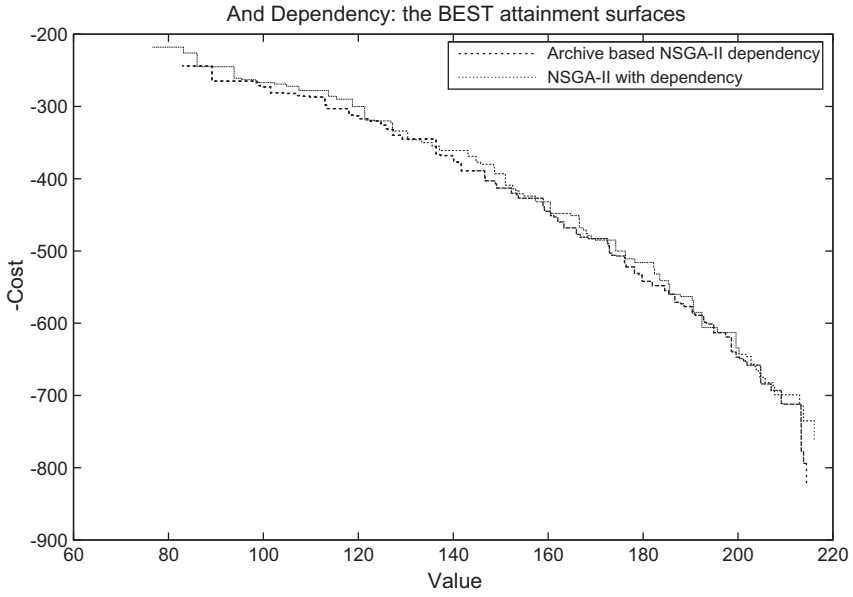

Fig. 11. *And* dependency, data set A, NSGA-II and archive based NSGA-II, 20-run best attainment surface.

Comparing to the results of *Or* and *Precedence* dependencies in Figs. 4 and 7, 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 solutions. Moreover, the distribution is more continuous, exhibiting a few small gaps among the solutions.

The results of the single execution discussed above do give an idea of how good is the individual solutions found, but information on how they tend to be distributed is largely lost. In order to gain some insight into how a search algorithm typically performs, the attainment surfaces are also depicted in the studies. An attainment surface is "the family of tightest goal vectors [17]" that likely to be attained by the non-dominated solution set. In fact, it is a boundary in the search space separating the non-dominated solutions from those that are dominated from the multiple executions.

The most important boundaries – the best and the worst attainment surfaces of NSGA-II for *And*, *Or* and *Precedence* dependencies are illustrated in Figs. 2 and 3, 5 and 6, 8 and 9 respectively.

In the figures, the solid gray lines represent the attainment surfaces of NSGA-II without dependency and the dotted gray lines are the ones of NSGA-II with dependency. The best attainment surfaces are plotted in Figs. 2, 5 and 8. As can be seen, the solutions were never attained in the area located up and to the right of the best attainment surfaces in any of the executions. We observe that the best attainment surfaces of NSGA-II with dependency are located exactly on or to the lower left of the those of NSGA-II without dependency. This is because NSGA-II with dependency only applied constraint violation checking mechanism to the solutions in the last generation of the algorithm. That is, all the feasible solutions under the constraints were extracted from the best attainment surfaces of NSGA-II without dependency (the solid gray lines).

From Figs. 3, 6 and 9, we can see that the solutions were also never attained in the area located down and to the left of the worst attainment surfaces, which might give us some ideas of worst case performance of the algorithms. Thus, in between these two bound-

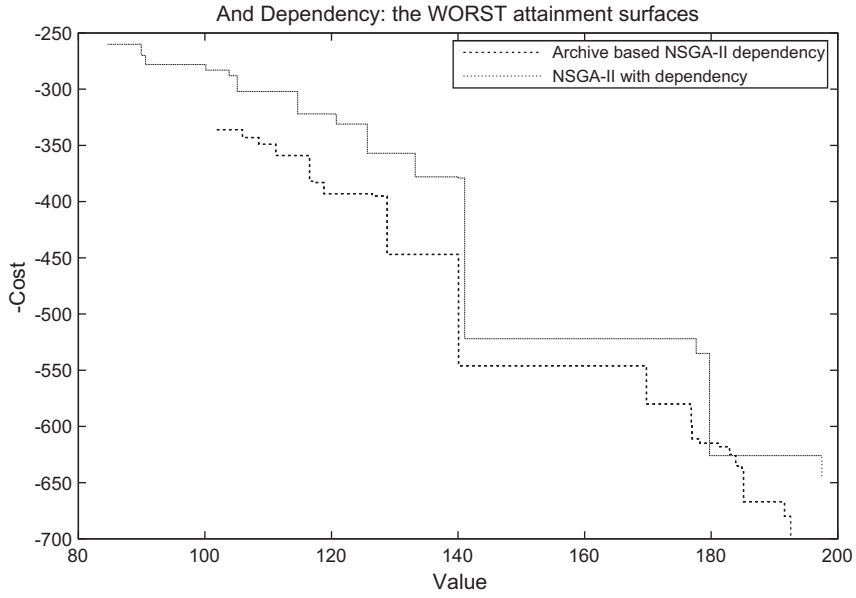

Fig. 12. *And* dependency, data set A, NSGA-II and archive based NSGA-II, 20-run worst attainment surface.

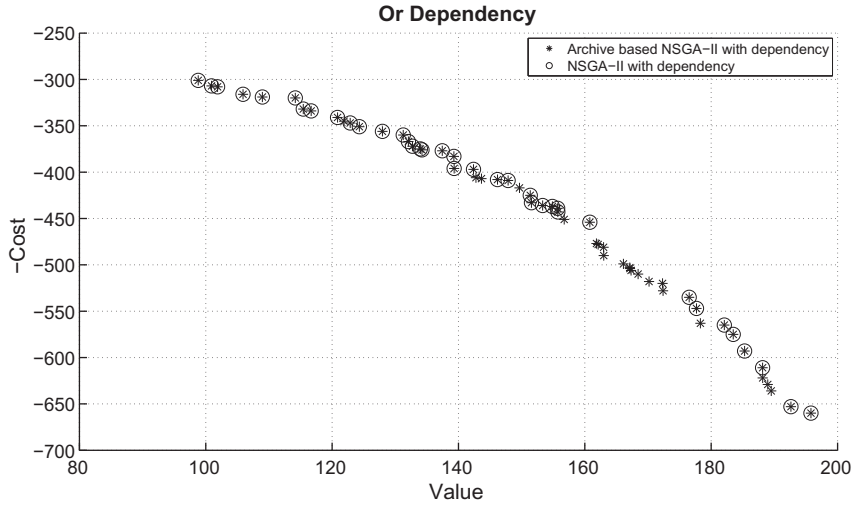

Fig. 13. *Or* dependency, data set A, NSGA-II and archive based NSGA-II, 1-run solutions.

aries (the best and the worst attainment surfaces) represent what was attained by all the executions. Similar to the best attainment surfaces, we also observe that the worst attainment surfaces of NSGA-II with dependency are located exactly on or to the upper right of the those of NSGA-II without dependency.

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 Figs. 10, 13 and 16. The '$*$' denotes the solution generated by the archive-based version of the NSGA-II algorithm and the '○' by NSGA-II.

From Fig. 10, 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 '$*$' Pareto front generated may appear to become less optimal when compared to the gray '+' front. However it discovers a larger number of good solutions to fill the gaps while also 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 Figs. 13 and 16.

The 20-execution attainment surfaces of archive based NSGA-II are shown in Figs. 11 and 12, 14 and 15, 17 and 18. The attainment surfaces of archive based NSGA-II are presented using dashed black lines.

The figures show that the best attainment surfaces of archive based NSGA-II mainly locate to the lower left part of the those of NSGA-II. We also observe that there are still a small number of solutions attained by archive based NSGA-II that dominate the solutions by NSGA-II. However, notice that, the worst attainment surface of archive based NSGA-II in Fig. 18 has very small coverage. Because there are only a few solutions on the worst surface.

Finally, all three dependencies were taken into consideration to access their overall combined impact. In Fig. 19, the '○' solutions

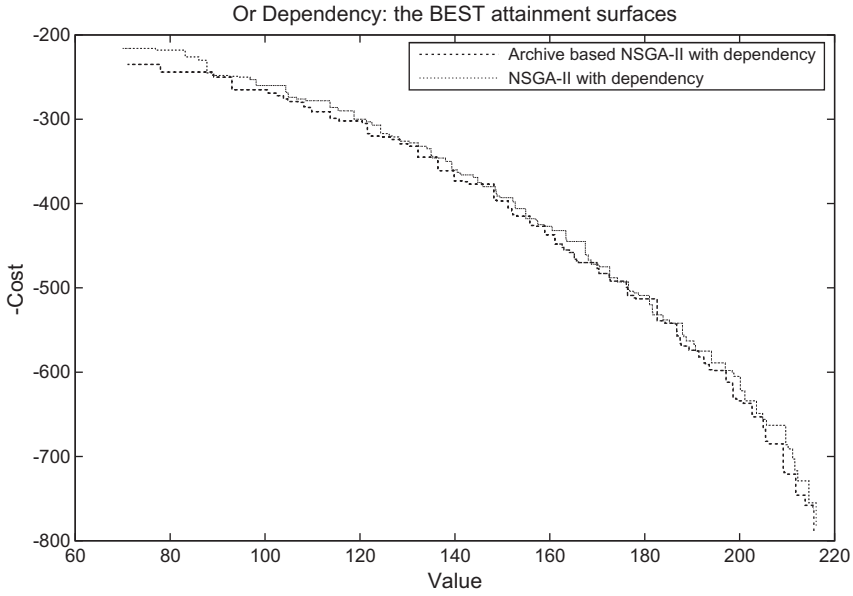

Fig. 14. *Or* dependency, data set A, NSGA-II and archive based NSGA-II, 20-run best attainment surface.

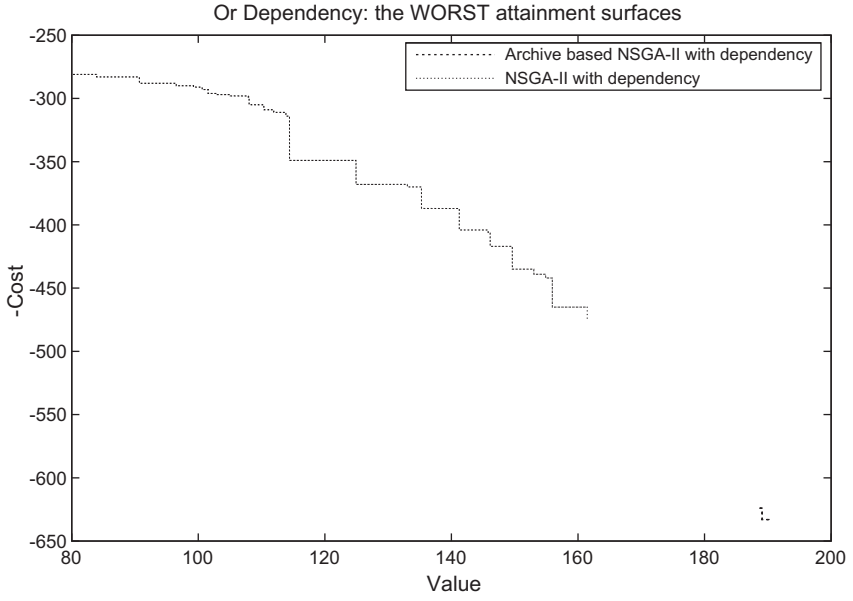

Fig. 15. *Or* dependency, data set A, NSGA-II and archive based NSGA-II, 20-run worst attainment surface.

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, Fig. 22 shows a smooth, relatively noninterrupted Pareto front, consisting of '∗' solutions, generated by archive based NSGA-II.

The results of the attainment surfaces are presented in Figs. 20, 21, 23 and 24. Similar to the attainment surfaces of individual dependence, in Fig. 20, all the attainable solutions generated by NSGA-II with dependencies are located to the lower left of the surface of NSGA-II without dependencies in the solution space. In Fig. 23, the best attainment surfaces of NSGA-II and archive based NSGA-II cross each other. We can see that two algorithms have equal performance in finding the extreme regions of the surface. In terms of the worst attainment surfaces shown in Figs. 21 and 24, it is much easier to identify big 'gaps' in the distribution of solutions.

4.2.2. 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 in this real world data set, NSGA-II and archive based NSGA-II could not find a single feasible solution in these experiments. All the feasible solutions that satisfy *And* and *Or* constraints were generated by the repair method. The results of a single execution are depicted in Figs. 25 and 28 and the attainment surfaces are illustrated in Figs. 26, 27, 29 and 30.

The gray '+' 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

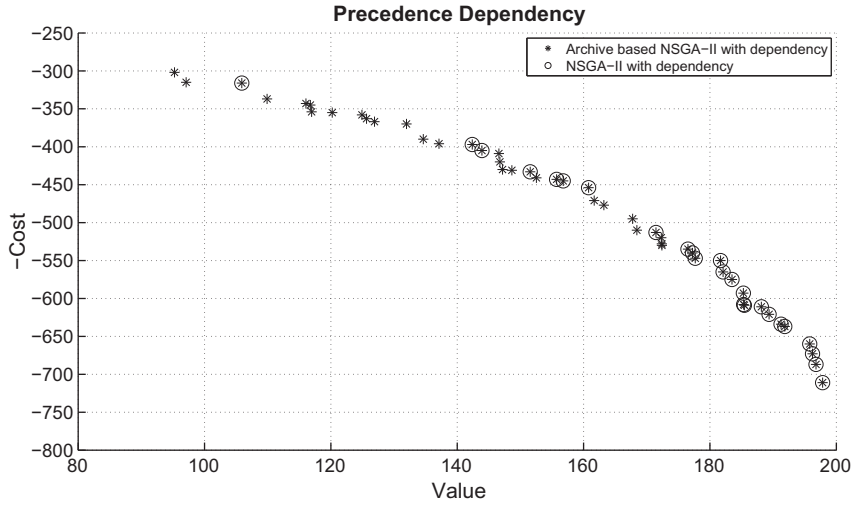

Fig. 16. *Precedence* dependency, data set A, NSGA-II and archive based NSGA-II, 1-run solutions.

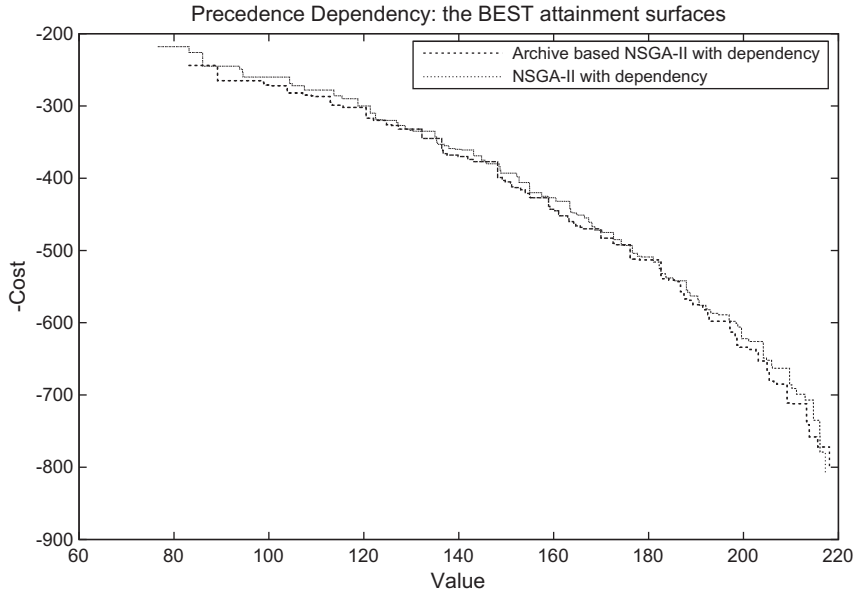

Fig. 17. *Precedence* dependency, data set A, NSGA-II and archive based NSGA-II, 20-run best attainment surface.

set RALIC assigned two kinds of requirements ratings for each requirement, namely, PointP and RankP. Fig. 25 is the PointP data set while Fig. 28 is the RankP data set.

As can be seen from Fig. 25, unlike the results produced by the archive based NSGA-II algorithm, all the '•' solutions do not completely fall on the Pareto front composed of gray '+' 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.

Fig. 28 presents the results for the RankP data set. These also follow a similar pattern and for which we made the same observations. The repair method 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 generate robust solutions for RIM but also can be used to enhance the quality of solutions for these problems that are free from constraints.

From the graphs of attainment surfaces, we can see more clearly that the attainment surfaces emphasize the distribution of the solutions achieved, and also indicate the quality of the individual points. For instance, in Fig. 29, two horizontal lines appear in the left and middle upper part of the best attainment surface, which represent the interruptions occurring. These interruptions can also be observed from the single execution results in Fig. 28.

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 second.

From the table we can see that the amount of computational time for finding infeasible solutions and repairing them is small compared to the total execution time. So the repair method is a

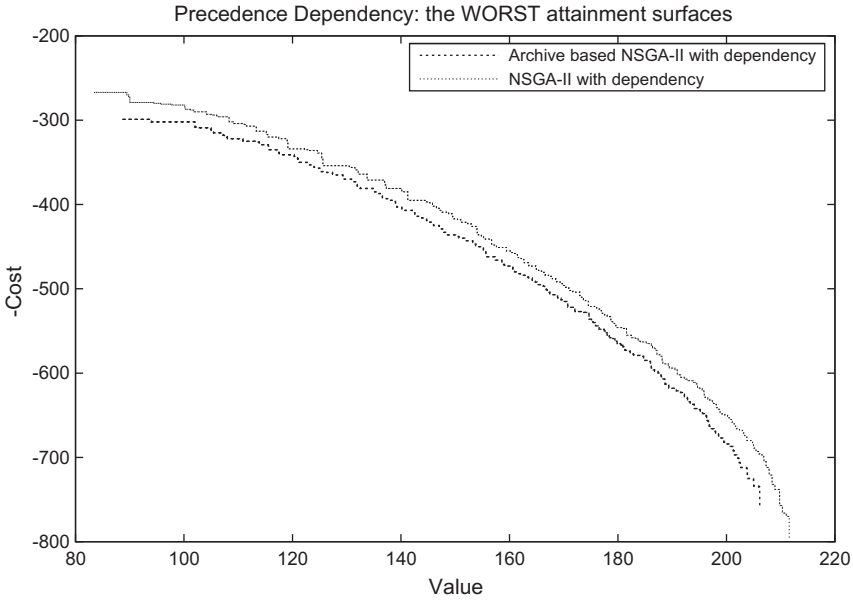

Fig. 18. *Precedence* dependency, data set A, NSGA-II and archive based NSGA-II, 20-run worst attainment surface.

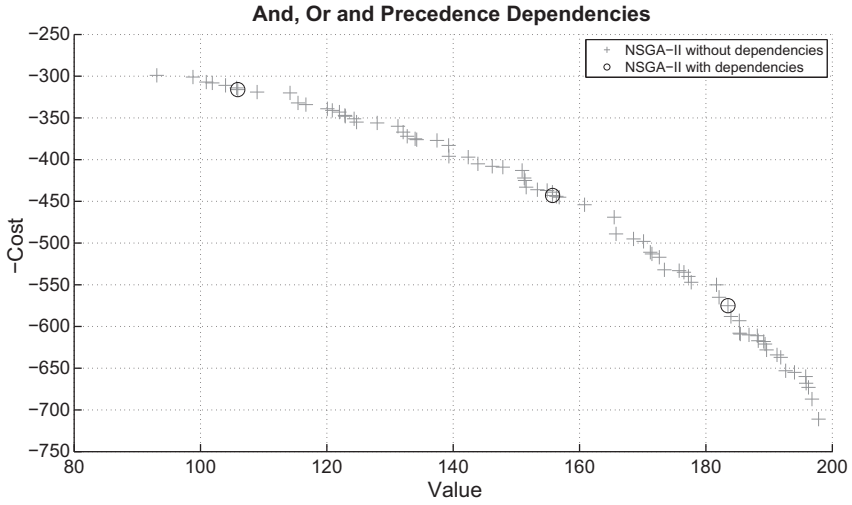

Fig. 19. *And*, *Or* and *Precedence* dependencies, data set A, NSGA-II, 1-run solutions.

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.

4.2.3. 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 Fig. 31. 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 is 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. Decision makers may have preferences towards certain regions or shapes of Pareto fronts. These different shapes can be used and compared in the sensitivity analysis based on the changes of *Value* or *Cost* for the decision support.

4.2.4. Answer to RQ1

The shapes of Pareto fronts (the results of the single execution) and the attainment surfaces (the results of 20 executions) are affected by the different dependency constraints to a different extent. The *And* dependency 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.

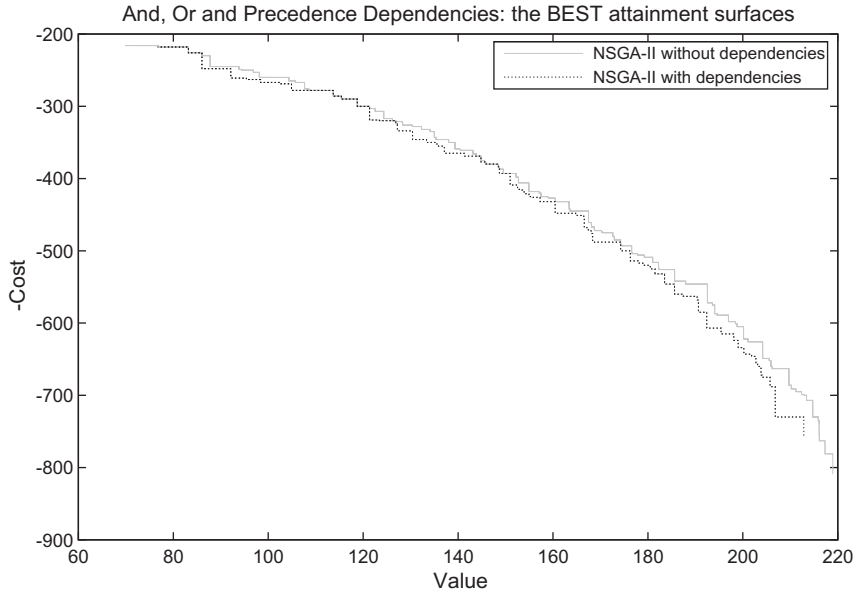

Fig. 20. *And*, *Or* and *Precedence* dependencies, data set A, NSGA-II, 20-run best attainment surface.

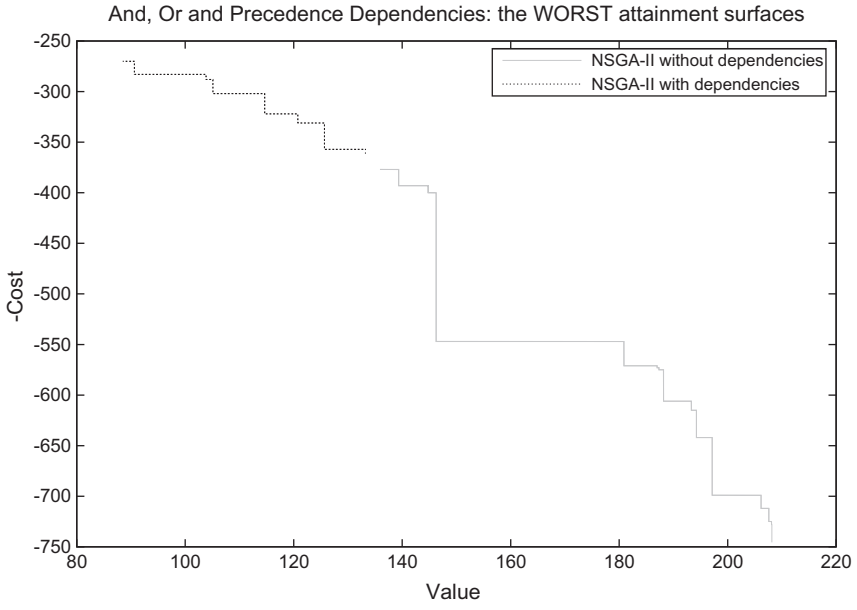

Fig. 21. *And*, *Or* and *Precedence* dependencies, data set A, NSGA-II, 20-run worst attainment surface.

When considering three fitness-invariant dependencies together, the fewer solutions could be found than those of the individual dependency.

4.3. Scale Study

In this section, we report on the second empirical study – the Scale Study. As described at the beginning of Section 4, each algorithm was 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 Tables 3 and 4.

4.3.1. Three fitness-invariant dependencies for data set B, C and D

The results are plotted in three graphs for each data set: the single execution, the best attainment surface and the worst attainment surface. The results of the single execution are presented in the Figs. 32, 35 and 38. The results of the best and the worst attainment surfaces for the three data sets are shown in Figs. 33 and 34, 36 and 37, 39 and 40 respectively.

In this study, all three dependency constraints are considered together. In the single execution figures, the gray 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 are produced by the archive based NSGA-II algorithm; '•' 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 and rapidly decreases. As illustrated in Fig. 38, the '○' solutions have a poor spread over the Pareto front. The '∗' solutions fill some of the gaps left by the '○' solutions and produce a relatively smooth

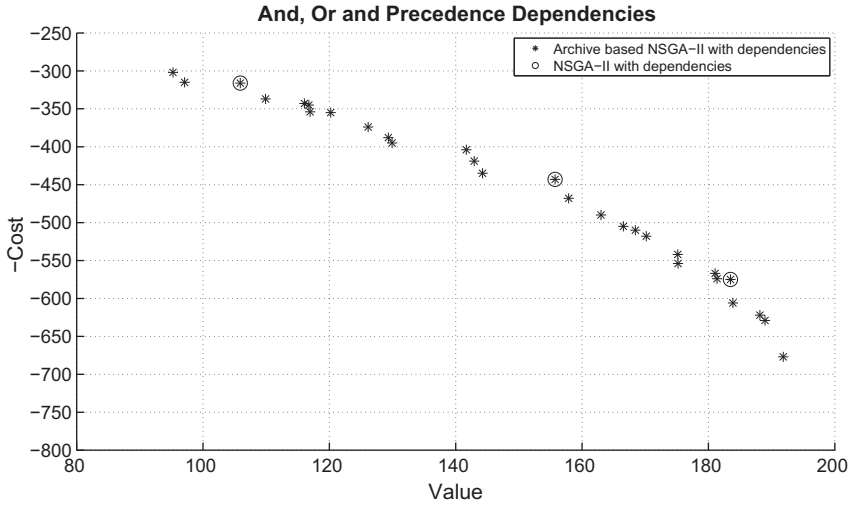

Fig. 22. *And*, *Or* and *Precedence* dependencies, data set A, NSGA-II and archive based NSGA-II, 1-run solutions.

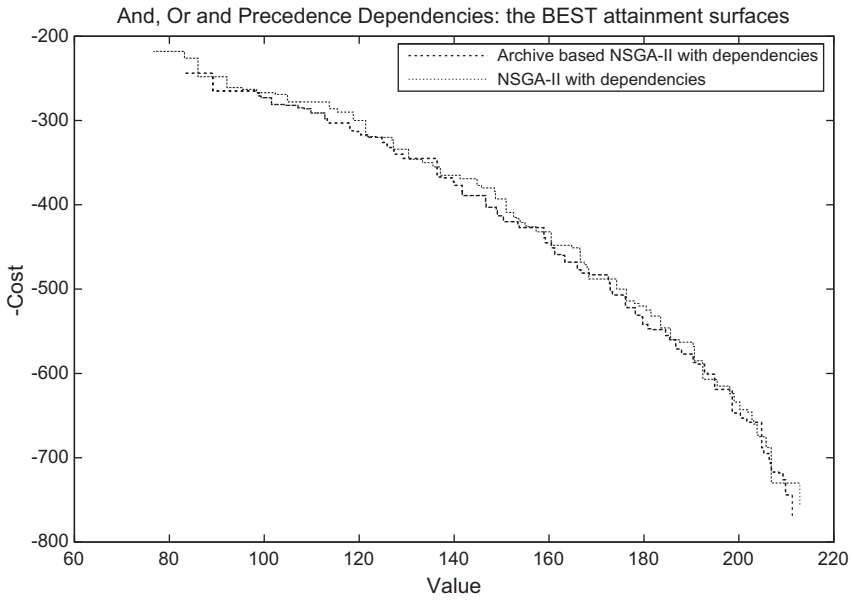

Fig. 23. *And*, *Or* and *Precedence* dependencies, data set A, NSGA-II and archive based NSGA-II, 20-run best attainment surface.

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 '∗' Pareto front is wider in Fig. 38 than in Fig. 32. 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 gray '+' 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 when RIM constraints are imposed.

In terms of the best attainment surfaces, we observe that the surfaces of three algorithms are very close to each other in Fig. 33. When increasing the scale of the data set, from Fig. 36 we can see, the three surfaces are easily distinguishable by the performance of the algorithms. In Fig. 39, the surface generated by the repair method has much wider ranges of *X* and *Y* axes than others. That is, a greater number of solutions can be delivered and the repair method has improved performance in the extreme regions of the surface. The worst attainment surfaces shown in Figs. 34, 37 and 40 visualize the worst-case results achieved by the algorithms.

4.3.2. Answer to RQ2

Fig. 41 shows boxplots of the performance metrics of the algorithms for data sets B, C and D in 20 executions. Each row of subfigures indicates the results for one data set. Each column presents the results for one metric. The NSGA-II with dependencies algorithm is referred as '*NSGA-II*' in the subfigures. The archive based NSGA-II with dependencies algorithm is referred as '*Archive*'. Similar, the repair method with dependencies algorithm is shown as '*Repair*'.

Considering the differences in the results for the algorithms' convergence in Fig. 41 (the first column), we observe that

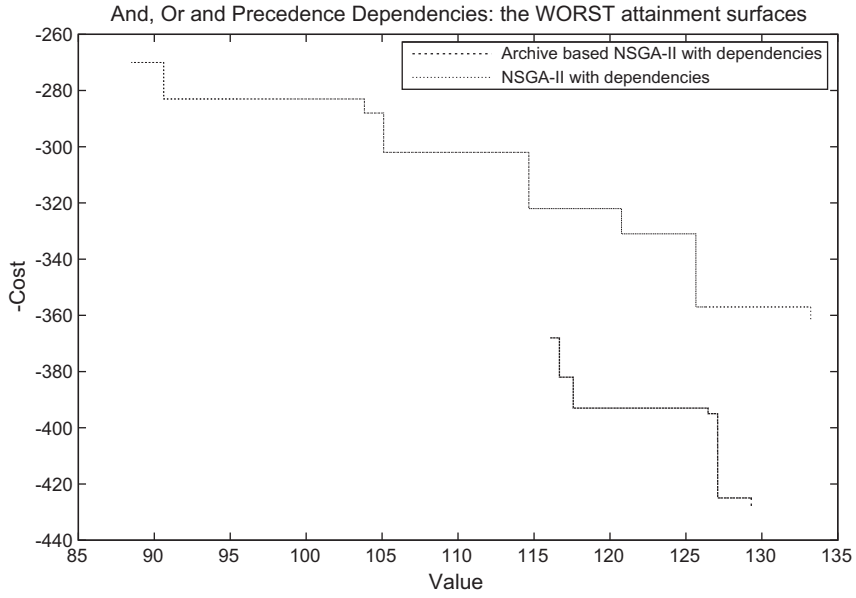

Fig. 24. *And*, *Or* and *Precedence* dependencies, data set A, NSGA-II and archive based NSGA-II, 20-run worst attainment surface.

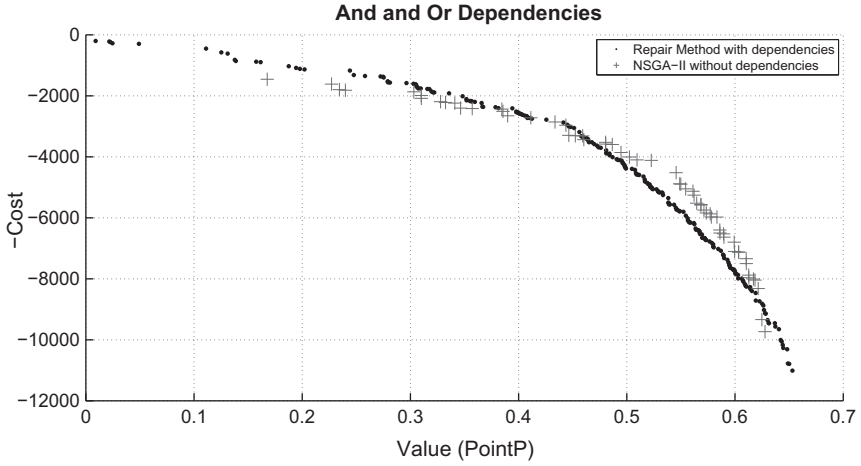

Fig. 25. *And* and *Or* dependencies, RALIC data set – PointP, repair method, 1-run solutions.

NSGA-II and the repair method have relatively better convergence performance than archive based NSGA-II. In addition, the boxplot is used as an indicator of spread. The length of the box for the repair method is shorter than others, which indicates that standard deviation of convergence obtained from 20-run repair method is lower than those of other algorithms.

In terms of the diversity metric (the second column in Fig. 41), NSGA-II has the worst performance and the repair method outperforms in data sets B and D. Similarly, the repair method also has better performance in terms of the number of solutions in each execution. The NSGA-II algorithm only generates a limited number of solutions compared to others.

To explore further to see if the three algorithms' performance is significantly different, a statistical analysis was performed. Since the results do not belong to any particular distribution, the non-parametric Kruskal–Wallis test was carried out to compare the performance of the three algorithms based on three metrics described in Section 3.3. The null hypothesis is that all three algorithms have the same performance. Rejection of the null hypothesis by Kruskal–Wallis test can tell us whether the three algorithms' performance was significantly different to one another.

The statistical analysis results for three data sets are presented in Table 6. A larger *Chi-Square* indicates a greater confidence in rejecting the null hypothesis. *df* is the degrees of freedom. There are three algorithms compared for each metric, so the value of *df* is 2. *Prob* > *Chi-Square* is the significance level of *Chi-Square*. The smaller the *Prob* > *Chi-Square* (<0.05), the stronger the evidence to reject the null hypothesis. The results show that all the *Prob* > *Chi-Square* of the metrics are much less than 0.05. That is, three algorithms do have significantly different performance in data set B, C and D.

5. Related work

Dependency 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 [18].

On the one hand, the identified traceability links among requirements can be easily used to extract requirements dependencies.

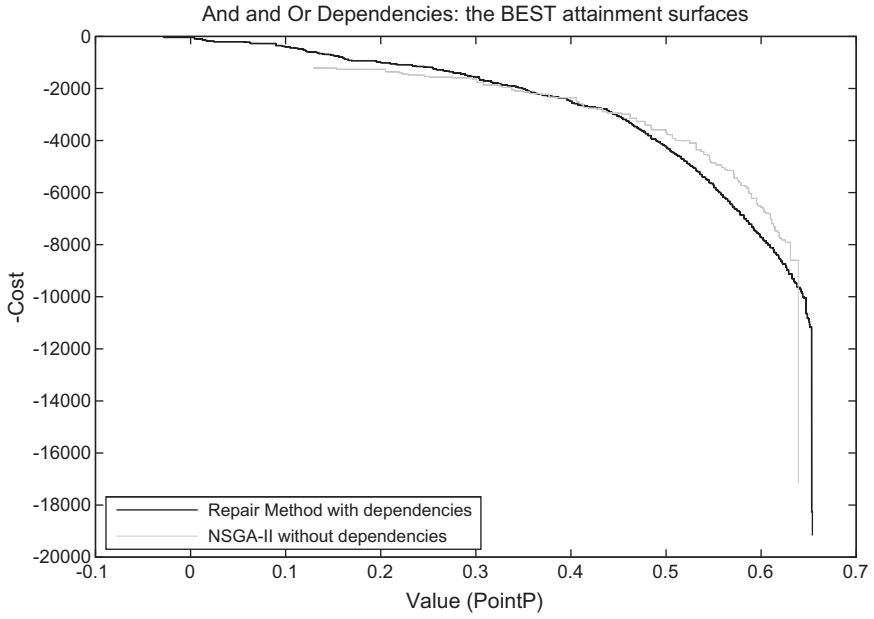

Fig. 26. *And* and *Or* dependencies, RALIC data set – PointP, repair method, 20-run best attainment surface.

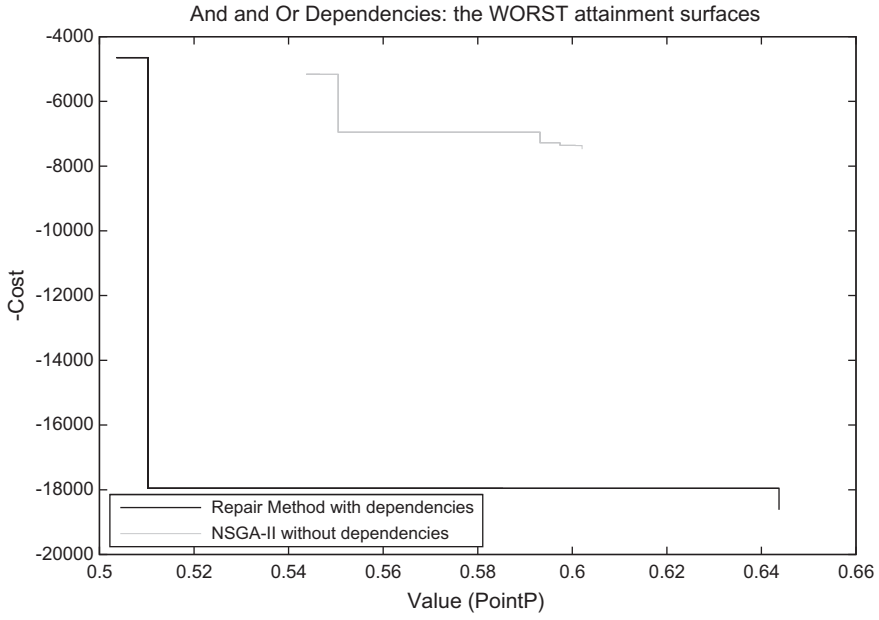

Fig. 27. *And* and *Or* dependencies, RALIC data set – PointP, repair method, 20-run worst attainment surface.

Thus, these traceability links may help and support the dependence identification activity. On the other hand, dependency relationship also directly affects the requirements traceability management.

There are many ways to represent traceability links. The traceability matrix [19] and cross references [18] are both regarded as good practice which have been widely used in industry. In addition, a large number of requirements tools support traceability management [18]. One of the most well known tools is DOORS (Dynamic Object Oriented Requirements System) [20]. Pohl [21] 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. [22] opened up the discussion on supporting requirements dependencies in the requirements selection process. Robinson et al. [3] 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 [23] described a two-dimensional (*scope* and *explicitness*) representation to investigate different types of dependencies. Subsequently, Carlshamre et al. [2] 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 [24,25] provided an overview of research work concerning comparing and validating the different requirements dependencies classification frameworks. There was some work

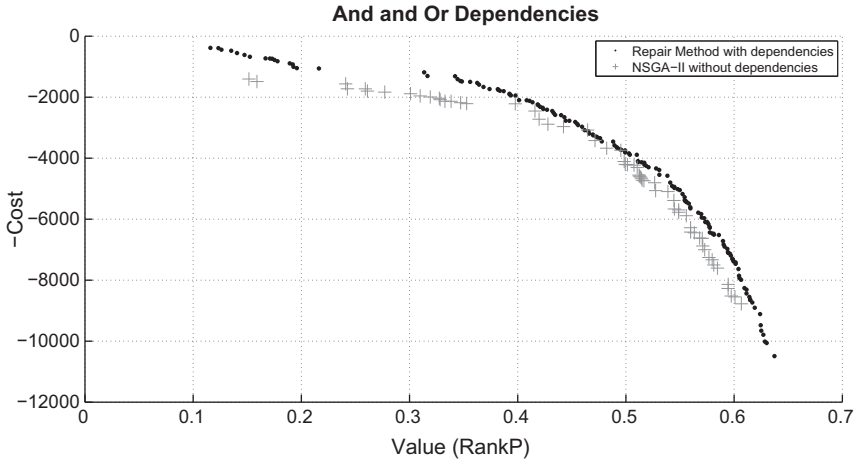

Fig. 28. *And* and *Or* dependencies, RALIC data set – RankP, repair method, 1-run solutions.

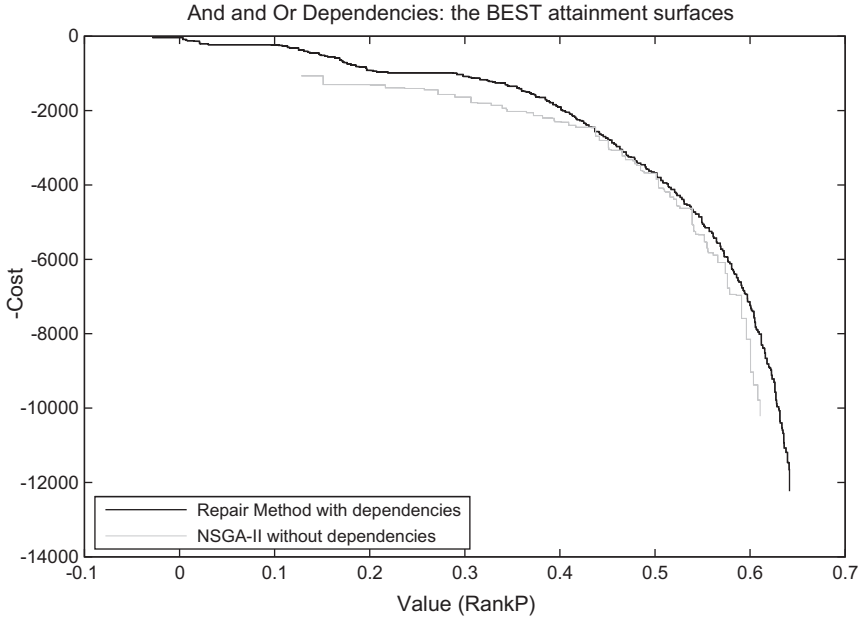

Fig. 29. *And* and *Or* dependencies, RALIC data set – RankP, Repair Method, 20-run best attainment surface.

that suggested that proper treatment of RIM should take account of different types of requirement interactions [2,5,22,24,26].

In the work on search based requirements optimization, Bagnall et al. [4] suggested the term Next Release Problem (NRP) for requirements release planning. The task was to find a subset of stakeholders whose requirements are to be satisfied by various metaheuristic optimization algorithms. Harman et al. [27] considered requirements problems as a feature subset selection problems, presenting results on a single objective formulation for a real world data set from Motorola. Zhang et al. [28] provided a multi-objective formulation of the NRP to optimize value and cost. The authors presented the results of an empirical study into the suitability of GA based multi-objective search techniques. More recently, del Sagrado et al. [29] extended the work by applying Ant Colony Optimization in NRP without dependency. Durillo et al. [30] also studied NRP with the intention of comparing the performance of three different multi-objective algorithms. In addition, search based requirements optimization includes not only NRP requirements selection, but also requirements prioritization. For example, Tonella et al. [6] proposed an interactive GA based method to prioritize the requirements for a software system.

In terms of the relevant methods proposed for constraint handling using optimization techniques, there are generally five types of approaches [31]:

1. penalty functions [32];
2. preserving the feasibility of solutions [33];
3. repair methods [14];
4. separation of constraints and objectives [34];
5. hybrid methods [35].

The most commonly used of these are penalty functions and repair methods [36]. Penalty functions were first introduced by Courant [37], 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

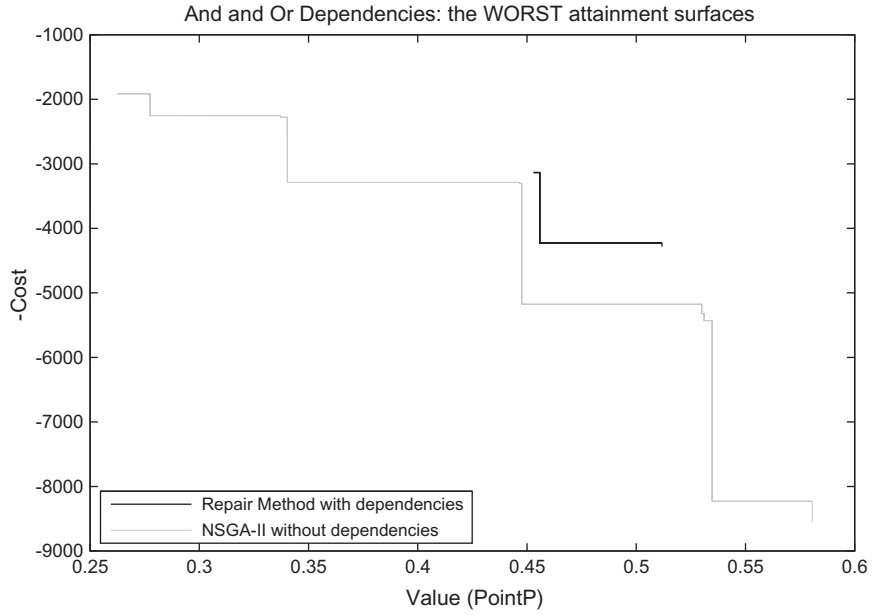

Fig. 30. *And* and *Or* dependencies, RALIC data set – RankP, repair method, 20-run worst attainment surface.

Table 5
Average CPU time of data sets.

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

called "death penalty" and is computationally efficient because no further calculation is 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. [38] 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 [39] and Carlson Skalak et al. [40] presented the methods inspired by the cooling analog employed by simulated annealing [41]. 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

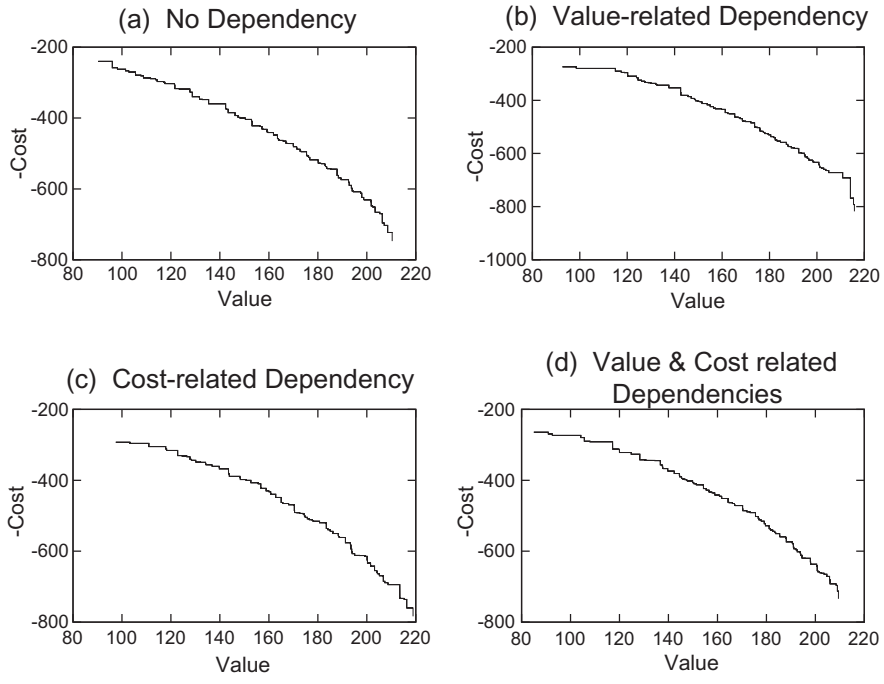

Fig. 31. *Value-related* and *Cost-related* dependencies, data set A, NSGA-II.

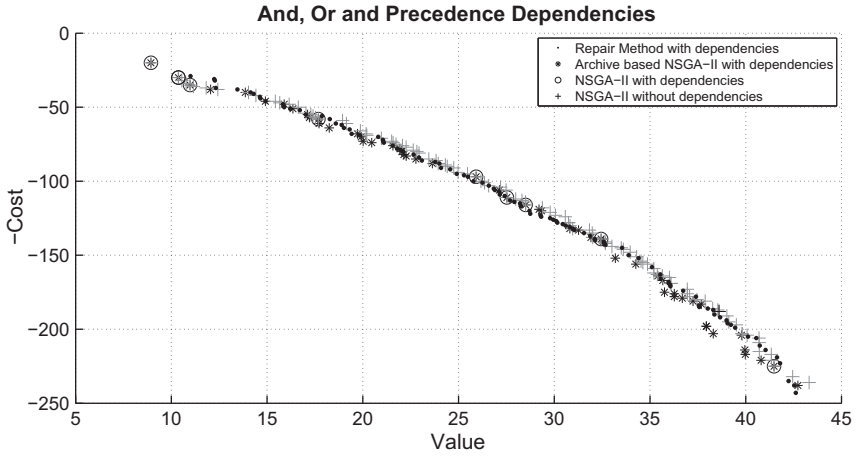

Fig. 32. *And*, *Or* and *Precedence* dependencies, data set B, NSGA-II, archive based NSGA-II and repair method, 1-run solutions.

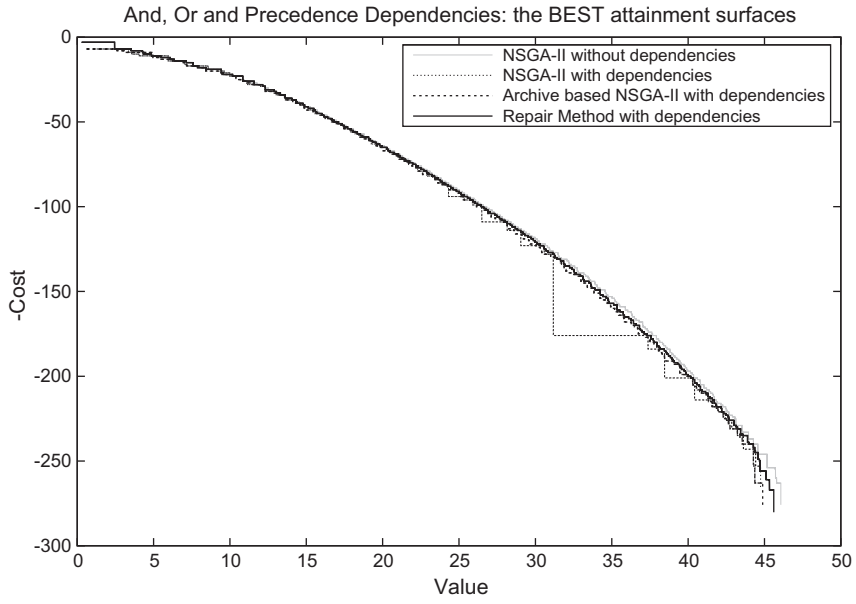

Fig. 33. *And*, *Or* and *Precedence* dependencies, data set B, NSGA-II, archive based NSGA-II and repair method, 20-run best attainment surface.

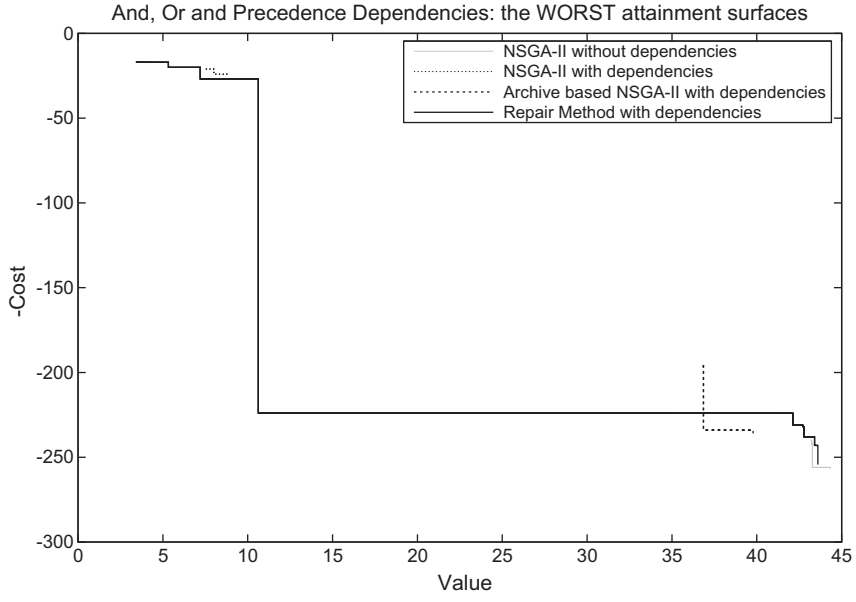

Fig. 34. *And*, *Or* and *Precedence* dependencies, data set B, NSGA-II, archive based NSGA-II and repair method, 20-run worst attainment surface.

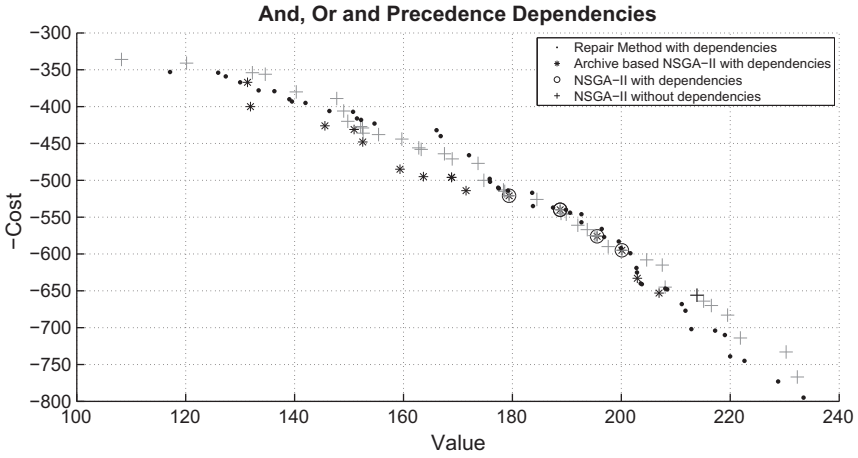

Fig. 35. *And*, *Or* and *Precedence* dependencies, data set C, NSGA-II, archive based NSGA-II and repair method, 1-run solutions.

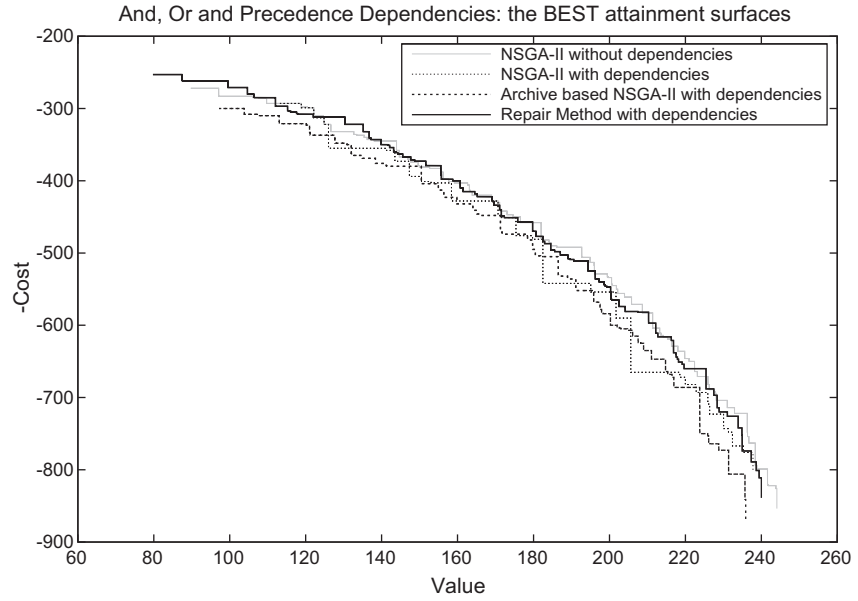

Fig. 36. *And*, *Or* and *Precedence* dependencies, data set C, NSGA-II, archive based NSGA-II and repair method, 20-run best attainment surface.

penalized more severely. This mimics the cooling process, in which early stages of the search are less constrained while subsequent phases become progressively more constrained.

Adaptive penalty methods take an 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 [32]. 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 [14,42] 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 [43], knapsack problems [44] and traveling salesman problems [45] tend to be good candidates for the application of repair methods, because these problems' constraints and decision variables are usually easily characterized and an infeasible solution is relatively easy to repair.

There are several repair heuristic schemes. Liepins et al. [14] 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 [42] 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. [14] 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 [46], in contracts to Lamarckian repair [46], which replaces the infeasible solutions by repaired ones.

Ishibuchi et al. [46] 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.

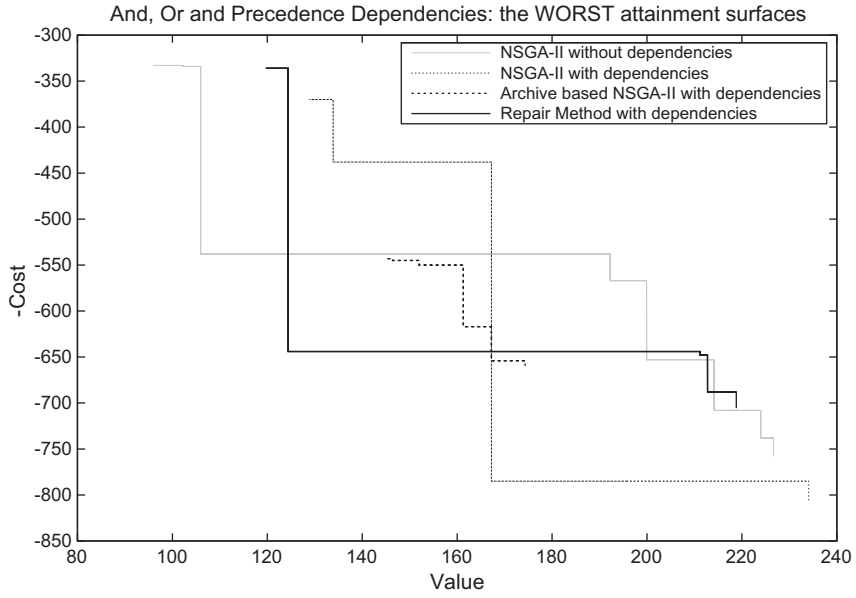

Fig. 37. *And*, *Or* and *Precedence* dependencies, data set C, NSGA-II, archive based NSGA-II and repair method, 20-run worst attainment surface.

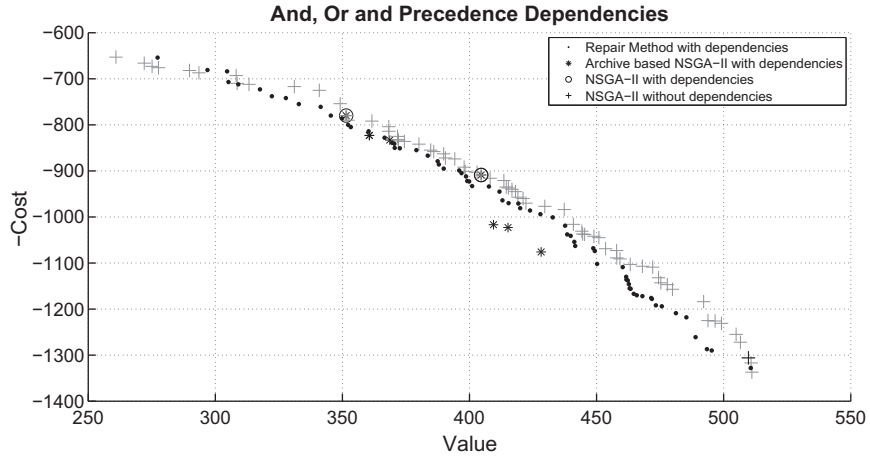

Fig. 38. *And*, *Or* and *Precedence* dependencies, data set D, NSGA-II, archive based NSGA-II and repair method,1-run solutions.

Michalewicz and Nazhiyath [47] 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 [36] 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 [31] 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 three search techniques when the problem becomes larger and more challenging.

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

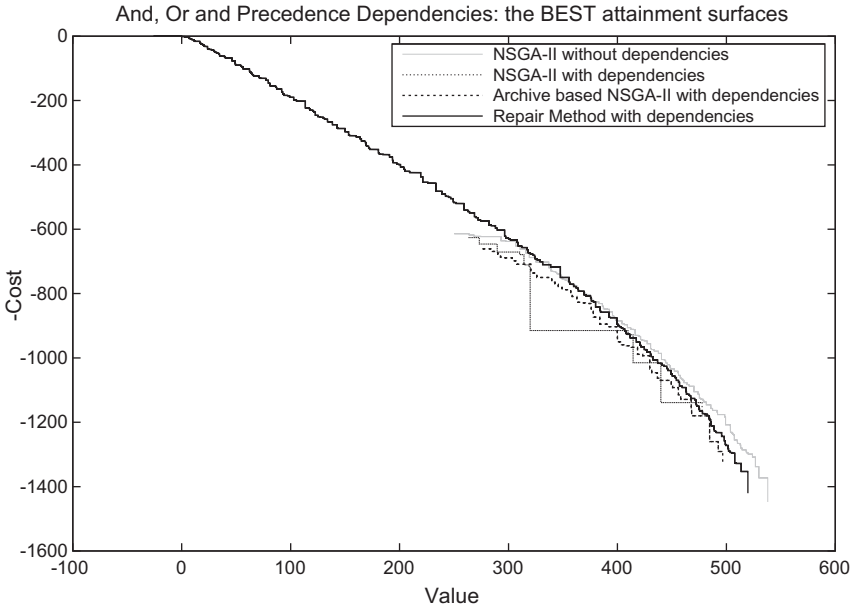

Fig. 39. *And*, *Or* and *Precedence* dependencies, data set D, NSGA-II, archive based NSGA-II and repair method, 20-run best attainment surface.

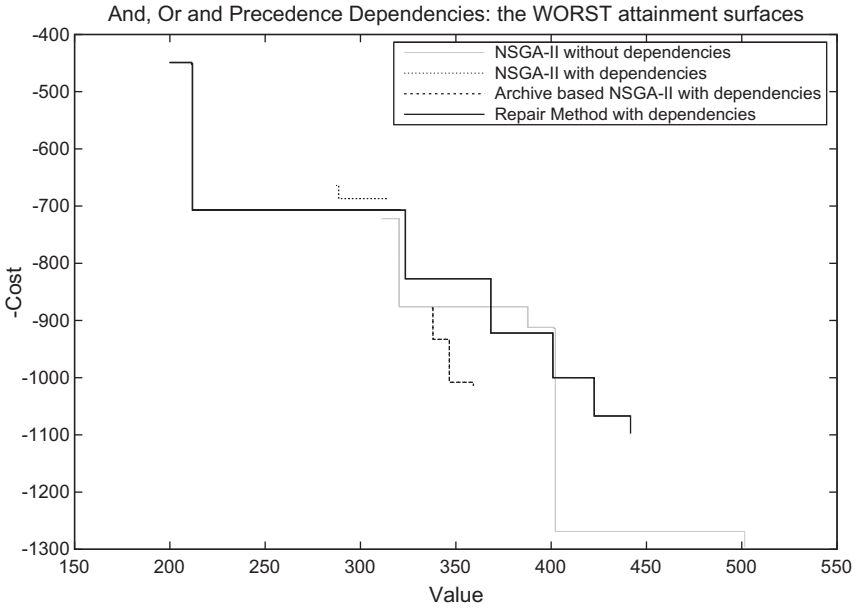

Fig. 40. *And*, *Or* and *Precedence* dependencies, data set D, NSGA-II, archive based NSGA-II and repair method, 20-run worst attainment surface.

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 statistical analysis showed that repair method could produce better convergence and diversity of results 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 a number of better feasible (even dominating) solutions.

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.

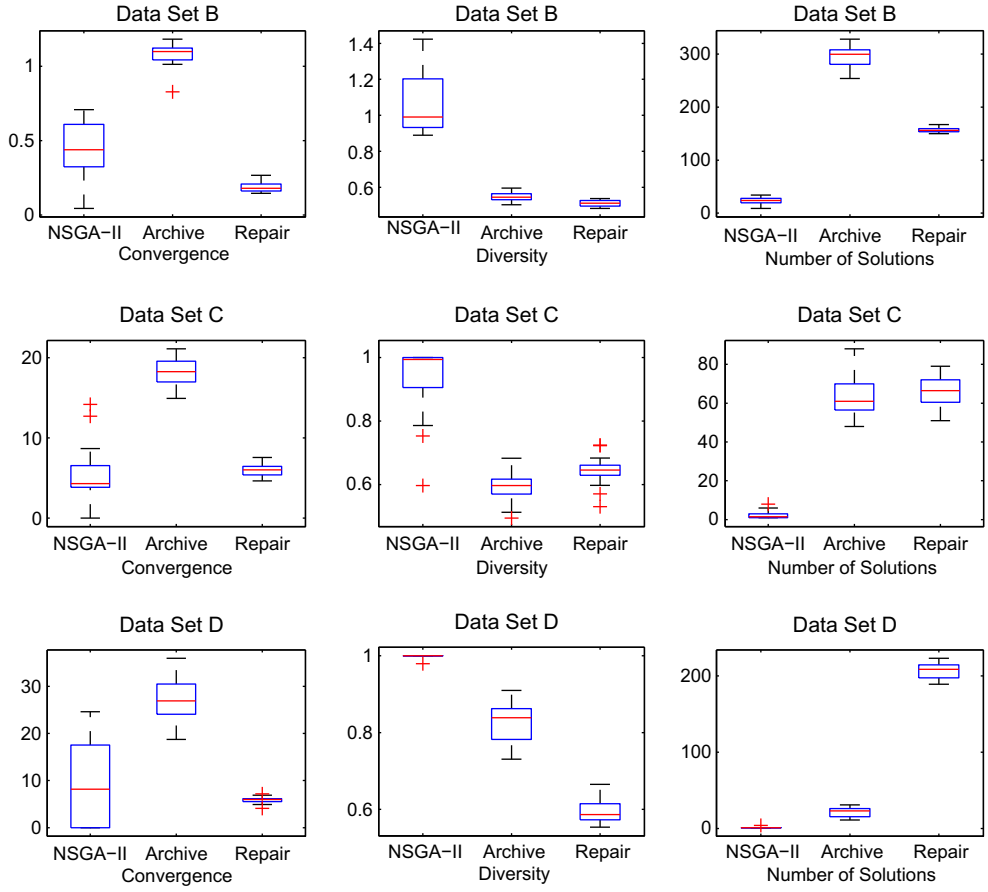

Fig. 41. Boxplots of convergence, diversity and number of solutions, *And*, *Or* and *Precedence* dependencies, data set B, C and D, NSGA-II, archive based NSGA-II and repair method, 20 runs.

Table 6
Kruskal–Wallis test of three algorithms' performance for data set B, C and D.

Data set	Metric	Kruskal–Wallis test		
		Chi-Square	df	Prob > Chi-Square
B	Convergence	46.53	2	7.8891e−011
	Diversity	46.62	2	7.5142e−011
	No. of solutions	52.48	2	4.0113e−012
C	Convergence	41.6	2	9.2477e−010
	Diversity	38.69	2	3.9592e−009
	No. of solutions	40.5	2	1.607e−009
D	Convergence	37.27	2	8.064e−009
	Diversity	53.91	2	1.964e−012
	No. of solutions	53.95	2	1.9266e−012

References

[1] Y. Zhang, M. Harman, Search based optimization of requirements interaction management, in: Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE '10), IEEE, Benevento, Italy, 2010, pp. 47–56.

[2] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, J. Nattoch 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).

[3] W.N. Robinson, S.D. Pawlowski, V. Volkov, Requirements Interaction Management, Technical Report 99-7, Georgia State University, 1999.

[4] A.J. Bagnall, V.J. Rayward-Smith, I.M. Whittley, The next release problem, Information and Software Technology 43 (2001) 883–890.

[5] D. Greer, G. Ruhe, Software release planning: an evolutionary and iterative approach, Information & Software Technology 46 (2004) 243–253.

[6] P. Tonella, A. Susi, F. Palma, Using interactive GA for requirements prioritization, in: Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE '10), IEEE, Benevento, Italy, 2010, pp. 57–66.

[7] S. Lim, A. Finkelstein, StakeRare: using social networks and collaborative filtering for large-scale requirements elicitation, IEEE Transactions on Software Engineering, in press.

[8] S. Lim, D. Quercia, 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, vol. 1, ACM, pp. 295–304.

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

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

[11] Y. Zhang, E. Alba, J.J. Durillo, S. Eldh, M. Harman, Today/future importance analysis, in: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO '10), ACM, Portland, USA, 2010, pp. 1357–1364.

[12] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2002) 182–197.

[13] K. Praditwong, X. Yao, A new multi-objective evolutionary optimisation algorithm: the two-archive algorithm, Proceedings of he 2006 International Conference on Computational Intelligence and Security (CIS '06), vol. 1, IEEE Press, Guangzhou, China, 2006, pp. 286–291.

[14] W.P.G.E. Liepins, A genetic algorithm approach to multiple-fault diagnosis, Van Nostrand Reinhold, New York, pp. 237–250.

[15] S. Venkatraman, G.G. Yen, A generic framework for constrained optimization using genetic algorithms, IEEE Transactions on Evolutionary Computation 9 (2005) 424–435.

[16] K. Deb, Multi-objective optimization using evolutionary algorithms, John Wiley & Sons, 2001.

[17] C.M. Fonseca, P.J. Fleming, On the performance assessment and comparison of stochastic multiobjective optimizers, in: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN IV), Springer, 1996, pp. 584–593.

[18] O.C.Z. Gotel, A. Finkelstein, An analysis of the requirements traceability problem, in: Proceedings of the 1st International Conference on Requirements Engineering (RE '94), IEEE Computer Society, Colorado Springs, Colorado, USA, 1994, pp. 94–101.

[19] B. Ramesh, T. Powers, C. Stubbs, M. Edwards, Implementing requirements traceability: a case study, Proceedings of the 2nd IEEE International Symposium on Requirements Engineering (RE '95), IEEE Computer Society, York, UK, 1995, pp. 89–95.
[20] IBM Rational DOORS (Dynamic Object Oriented Requirements System), <http://www.telelogic.com/Products/doors/>.
[21] K. Pohl, Process-Centered Requirements Engineering, Research Studies Press, 1996.
[22] J. Karlsson, S. Olsson, K. Ryan, Improved practical support for large-scale requirements prioritizing, Requirements Engineering Journal 2 (1997) 51–60.
[23] P. Carlshamre, 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), IEEE Computer Society, London, UK, 2000, pp. 961–965.
[24] Å.G. Dahlstedt, 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.
[25] Å.G. Dahlstedt, A. Persson, Engineering and Managing Software Requirements, Springer, Berlin, Heidelberg, 2005, pp. 95–116.
[26] W.N. Robinson, S.D. Pawlowski, V. Volkov, Requirements interaction management, ACM Computing Surveys (CSUR) 35 (2003) 132–190.
[27] M. Harman, A. Skaliotis, 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), ACM, Seattle, Washington, USA, 2006, pp. 1951–1952.
[28] Y. Zhang, M. Harman, S.A. Mansouri, The multi-objective next release problem, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO '07), ACM, London, UK, 2007, pp. 1129–1137.
[29] J. del Sagrado, I.M. del Águila, F.J. Orellana, Ant colony optimization for the next release problem – a comparative study, in: Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE '10), IEEE, Benevento, Italy, 2010, pp. 67–76.
[30] J.J. Durillo, Y. Zhang, E. Alba, M. Harman, A.J. Nebro, A study of the bi-objective next release problem, Empirical Software Engineering 16 (2011) 29–60.
[31] C.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 (2002) 1245–1287.
[32] T.P. Runarsson, X. Yao, Stochastic ranking for constrained evolutionary optimization, IEEE Transactions on Evolutionary Computation 4 (2000) 284–294.
[33] S. Koziel, Z. Michalewicz, Evolutionary algorithms for constrained parameter optimization problems, Evolutionary Computation 7 (1999) 19–44.
[34] D. Powell, M.M. Skolnick, Using genetic algorithms in engineering design optimization with non-linear constraints, in: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufman Publishers Inc., Urbana, USA, 1993, pp. 424–431.
[35] H. Adeli, H. Adeli, Augmented lagrangian genetic algorithm for structural optimization, Journal of Aerospace Engineering 7 (1994) 104–118.
[36] P. Chootinan, A. Chen, Constraint handling in genetic algorithms using a gradient-based repair method, Computers and Operations Research 33 (2006) 2263–2281.
[37] R. Courant, Variational methods for the solution of problems of equilibrium and vibrations, Bulletin of the American Mathematical Society 49 (1943) 1–23.
[38] A. Homaifar, C.X. Qi, S.H. Lai, Constrained optimization via genetic algorithms, International Transactions of the Society for Modeling and Simulation 62 (1994) 242–253.
[39] Z. Michalewicz, N.F. Attia, Evolutionary optimization of constrained problems, in: Proceedings of the 3rd Annual Conference on Evolutionary Programming, World Scientific, 1994, pp. 98–108.
[40] S.C. Skalak, R. Shonkwiler, S. Babar, M. Aral, Annealing a genetic algorithm over constraints, Proceeding of IEEE International Conference on Systems, Man, and Cybernetics, IEEE, San Diego, CA, USA, 1998, pp. 3931–3936.
[41] C.-R. Hwang, Simulated annealing: theory and applications, Acta Applicandae Mathematicae 12 (1988) 108–111.
[42] D. Orvosh, L. Davis, Using a genetic algorithm to optimize problems with feasibility constraints, in: Proceedings of the 1st IEEE World Congress on Computational Intelligence, Orlando, USA, 1994, pp. 548–553.
[43] T.R. Jensen, B. Toft, Graph Coloring Problems, Wiley-Interscience, 1994.
[44] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, 1990.
[45] E.L. Lawler, J.K. Lenstra, A.H.G.R. Kan, D.B. Shmoys, The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, Wiley, 1985.
[46] H. Ishibuchi, S. Kaige, K. 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, vol. 3410, Springer, Guanajuato, Mexico, 2005, pp. 370–385.
[47] Z. Michalewicz, G. Nazhiyath, Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints, in: Proceedings of IEEE International Conference on Evolutionary Computation (CEC '95), IEE, Perth, Australia, 1995, pp. 647–651.