FLINDERS UNIVERSITY

College of Science and Engineering

DOCTORAL THESIS

A Mathematical Programming Approach to Considering Value Dependencies in Software Requirement Selection

---

Davoud MOUGOUEI

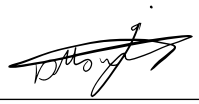
B.Eng. (Computer Engineering.), M.Sc. (Computer Science)

Thesis Submitted to Flinders University
for the degree of Doctor of Philosophy (Software Engineering)
College of Science and Engineering

June 6, 2018

Declaration of Authorship

I certify that without acknowledgment this thesis does not incorporate any material previously submitted for a degree or diploma in any university, and that to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where due reference is made in the text.

Signed: 

Date: June 6, 2018

"The only simple truth is that there is nothing simple in this complex universe. Everything relates. Everything connects."

Johnny Rich, The Human Script

Abstract

A Mathematical Programming Approach to Considering Value Dependencies in Software Requirement Selection

Software requirement selection aims to find an optimal subset of requirements with the highest value while respecting the project constraints. The value of a requirement however may depend, positively or negatively, on the presence or absence of other requirements in the optimal subset. Such *Value Dependencies* need to be considered in software requirement selection. However, the existing requirement selection works have mainly ignored value dependencies. This thesis presents a mathematical programming approach, referred to as *Dependency-Aware Requirement Selection* (DARS), for considering value dependencies in software requirement selection. The proposed approach includes four different selection methods: (i) an *Integer Programming* (IP) method, which takes into account the strengths of value dependencies; (ii) an Integer Linear Programming (ILP) method, which accounts for the strengths and qualities of value dependencies; (iii) a *Mixed Integer Programming* (MIP) method, that allows for partial selection of requirements; and (iv) a *Society-Oriented* method, which takes into account the social values of the requirements. Each method is comprised of three major components: (i) identification of value dependencies; (ii) modeling value dependencies; and (iii) integrating value dependencies into requirement selection. The main contributions of this thesis are validated by studying real-world software projects and extensive simulations with results highly suggestive of good prospects for application of DARS in industrial contexts as the approach reduces the risk of value loss posed by ignoring value dependencies among software requirements.

Acknowledgements

I would like to express my sincere gratitude to Professor David Powers for being a great support and a fantastic supervisor. Working with him was a very joyful and a great honor for me. His knowledge, patience, and trust gave me the opportunity to explore and examine new ideas to finally find my path toward completion of my thesis. My gratitude for his guidance goes beyond thanks. I would also like to thank Dr. Trent Lewis for his support and helpful reviews.

I also wish to thank Professor Jon Whittle, Professor Jerzy Filar, Professor Mark Wallace, and Dr. Asghar Moeini who kindly provided their comments and views on my work during my PhD candidature. I am thankful for their valuable insights which helped open up new horizons for my research.

I am especially thankful to my friends for providing me with their valuable feedbacks. They always encouraged me in difficult times and shared their stories with me. To name a few: Jenny, Mohammad, Amir Mahdi, Somaiyeh, Pouya, Mehdi, Nasim, Jameel, and Saeed.

I am thankful to the Australian government and Flinders University for generously supporting this thesis by an *Australian Government Research Training Program Scholarship* and *Elaine Martin Grant* as well as travel grants which helped me present my research at reputed venues including the 31^{st} *IEEE/ACM International Conference on Automated Software Engineering* (ASE 2016), the 39^{th} *International Conference on Software Engineering* (ICSE 2017), and the 30^{th} *Australian joint Conference on Artificial Intelligence* (AI 2017).

Last but not least, I am grateful to my dearest parents and lovely twin sisters for their unconditional love and support throughout all stages of my studies.

To my dearest parents and lovely twin sisters for unconditionally providing their love and encouragement.

Publications during the PhD Candidature

This thesis is the outcome of research carried out during the PhD candidature of the author at Flinders University. During the period, the main results of the thesis have been published in high quality peer reviewed journals and conferences. Moreover, some of our recent findings are under review for journal/conference publication. The details of the publications produced in relation to the thesis and their relevance to different chapters/sections of the thesis are discussed in detail in Section 1.3. We have listed these publications as follows.

(**P1**) D. Mougouei and D. M. W. Powers. Modeling and selection of interdependent software requirements using fuzzy graphs. *International Journal of Fuzzy Systems*, 19(6):1812–1828, Dec 2017

(**P2**) D. Mougouei. Factoring requirement dependencies in software requirement selection using graphs and integer programming. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 884–887. ACM, 2016

(**P3**) D. Mougouei, D. M. W. Powers, and A. Moeini. An integer linear programming model for binary knapsack problem with dependent item values. In W. Peng, D. Alahakoon, and X. Li, editors, *AI 2017: Advances in Artificial Intelligence: 30th Australasian Joint Conference, Melbourne, VIC, Australia, August 19–20, 2017, Proceedings*, pages 144–154. Springer International Publishing, Cham, 2017

(**P4**) D. Mougouei and D. M. W. Powers. The synergistic knapsack problem. *Fuzzy Optimization and Decision Making*, Under Review

(**P5**) D. Mougouei, D. M. Powers, and A. Moeini. Dependency-aware software release planning. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 198–200. ACM, 2017

(**P6**) D. Mougouei and D. M. W. Powers. An integer programming method for considering value-related dependencies in software requirement selection. *Information and Software Technology*, Under Review

(**P7**) D. Mougouei and D. M. W. Powers. Dependency-aware software release planning using fuzzy graphs and integer programming. *Engineering Applications of Artificial Intelligence*, Under Review

(**P8**) D. Mougouei and D. M. W. Powers. Dependency-aware software release planning through mining user preferences. *Expert Systems with Applications*, Under Review

(**P9**) D. Mougouei, H. Shen, and A. Babar. Partial selection of agile software requirements. *International Journal of Software Engineering & Its Applications*, 9(1):113–126, 2015

(**P10**) D. Mougouei and D. M. W. Powers. Paps: A scalable framework for prioritization and partial selection of security requirements. *International Journal of Approximate Reasoning*, Under Review

(**P11**) D. Mougouei and M. K. Yeung. Visibility requirements engineering for commercial websites. *International Journal of Software Engineering & Its Applications*, 8(8):11–18, 2014

(**P12**) D. Mougouei and D. M. W. Powers. Partial selection of software requirements. *International Conference on Computer Science, Engineering and Applications*, Accepted

(**P13**) D. Mougouei and D. M. W. Powers. An integer programming model for embedding social values into software requirement selection. *International Conference on Computer Science, Engineering and Applications*, Accepted

(**P14**) D. Mougouei and D. M. W. Powers. Gotm: a goal-oriented framework for capturing uncertainty of medical treatments. *Intelligent Systems Conference (IntelliSys) 2018*, Accepted

Invited Talk Related to the Thesis

- D. Mougouei. Considering value-related dependencies among requirements in software release planning: An integer programming approach. Faculty of Information Technology, Monash University, July 2017

Academic Community Involvement

- Reviewer: International Journal of Fuzzy Systems
- Program Committee Member: 24th Australian Software Engineering Conference
- Program Committee Member: 7th International Conference on Software Engineering and Applications (SEA-2018)
- Editor: Information and Computer Security
- Editor: Journal of Autonomous Intelligence
- Editor: Smart Construction Research
- Academic Member: Athens Institute for Education and Research
- Academic Member: Birouni Center for Education, Research, and Technology

Contents

[1] The author of the thesis has recently joined the Society-Oriented Software Design project at the Faculty of Information Technology, Monash University.

[2] Review of the existing requirement selection works, with regard to considering value dependencies, is presented in publications **(P1)**-**(P8)**.

[3]The contents of this chapter are presented in publications (**P1**) and (**P2**).
[4]The main results of this chapter are presented in publications (**P1**)-(**P8**).

[5] The results of this chapter are presented in publications (P3), (P4), and (P6)-(P12).
[6] The contents of this chapter are presented in publications (P3), (P4), (P6), (P8), (P10), (P12), and (P13).

[7]The author of the thesis has recently joined the Society-Oriented Software Design project at the Faculty of Information Technology, Monash University.

List of Figures

List of Tables

List of Abbreviations

The acronyms below are listed based on the order of first appearance in the thesis.

AV Accumulated Value.

EV Expected Value.

DARS Dependency-Aware Requirement Selection.

DARS-IP The Integer Programming method of DARS.

DARS-ILP The Integer Linear Programming Method of DARS.

DARS-MIP The mixed integer programming method of DARS.

DARS-SOC The society-oritented method of DARS.

OV Overall Value.

BKP-DIV The binary knapsack problem with dependent item values.

SKP Synergistic knapsack problem.

BKP Binary knapsack problem.

GRASP Greedy Randomized Adaptive Search Procedures.

LP Linear Programming.

BK Binary knapsack.

PCBK Precedence-Constrained Binary Knapsack.

SBK Stochastic Binary Knapsack.

OTKP Oregon trail knapsack problem.

SDP Selection deficiency problem.

NRP Next Release Problem.

FRIG Fuzzy Requirement Interdependency Graph.

LOI Level of Interdependency.

RAN Radio Access Network.

PMR Performance Management Recording.

PMS Precious Messaging System.

PDG Precedence Dependency Graph.

PDL Precedence Dependency Level.

NPDL Negative Precedence Dependency Level.

VDG Value Dependency Graph.

VDL Value Dependency Level.

NVDL Negative Value Dependency Level.

PAPS Pioritization and Partial Selection.

SRM Software Requirement Model.

SRL Software Requirement List.

FIS Fuzzy Inference System.

GFG Goal-based Fuzzy Grammar.

OBS Online Banking System.

RDS Required Degree of Satisfaction.

FCL Fuzzy Control Language.

GQM Goal Question Metric.

SVM Social Value Model.

List of Symbols

A glossary of the frequently used symbols in this thesis is given below.

R Set of requirements.

v_i Estimated value of a requirement r_i.

c_i Estimated cost of a requirement r_i.

r_i Requirement $r_i \in R$.

$p(r_i)$ The probability that users select or use requirement r_i.

$E(v_i)$ Expected value of a requirement r_i.

O Optimal subset.

$\tilde{O}$ Subset of excluded requirements.

G The Set of Software Goals.

μ Fuzzy membership function of requirements.

ρ Fuzzy membership function of value dependencies.

$\wedge$ Fuzzy AND operator.

I_i Impact of the requirements on the value of r_i.

$\vee$ Fuzzy OR operator.

v_i' Overall value of a requirement r_i.

η Measure of causal strength.

$M_{n\times k}$ Matrix of preferences for n requirements and k users.

ω_{-} The lower-bound of the confidence interval for Odds ratio.

ω_+ The upper-bound of the confidence interval for Odds ratio.

$\sigma(r_i,r_j)$ Specifies quality of a value dependency from r_i to r_j.

$\rho^{+\infty}(r_i,r_j)$ Strength of all positive value dependencies from r_i to r_j.

$\rho^{-\infty}(r_i,r_j)$ Strength of all negative value dependencies from r_i to r_j.

θ_i The penalty for a requirement r_i.

γ Price limit.

g_j Goal or subgoal $g_j \in G$.

$DC_g(x)$ the Impact of the Requirement/Goal x on Satisfaction of the Goal g.

$\oplus$ Fuzzy OR operator (taking maximimum).

$\otimes$ Fuzzy AND operator (taking minimum).

ζ_i The estimated effort for complete satisfaction of r_i.

ζ_i' The RELAX-ed effort for partial satisfaction of r_i.

$\sigma_t(r_i,r_j)$ Specifies quality of a value dependency of type t from r_i to r_j.

ρ_t Fuzzy membership function of value dependencies of type t.

$\rho_t^{+\infty}(r_i,r_j)$ Strength of all positive value dependencies of type t from r_i to r_j.

$\rho_t^{-\infty}(r_i,r_j)$ Strength of all negative value dependencies of type t from r_i to r_j.

$E(v_{i,t})$ The expected type t value of a requirement r_i.

β_t The minimum amount (lower-bound) required for the expected type t value of the selected requiremenets.

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Software requirement selection, also known as *Software Release Planning* [16, 17], aims to find an optimal subset of the requirements of a software project with the highest value while respecting the project constraints [18]. The values of the requirements however, may positively or negatively depend on the presence or absence of the other requirements [19, 20] in the optimal subset. Hence, it is important to consider *Value Dependencies* among requirements in software requirement selection [21, 22, 23, 24].

Moreover, as observed by Carlshamre *et al.* [21], requirement dependencies in general and value dependencies in particular are *fuzzy* [21] in the sense that the strengths of the requirement dependencies, including value dependencies, are imprecise and vary [18, 25, 26, 21] from large to insignificant [27] in real-world projects. Hence, it is important to consider not only the existence but the strengths of value dependencies [18, 21] and the imprecision associated with those dependencies in software requirement selection.

Although the need for considering value dependencies was observed as early as in 2001 [21], the existing requirement selection works have mainly ignored value dependencies by employing either the *Accumulated Value* (AV) [28, 22, 29, 30, 31] or the *Expected Value* (EV) as the optimality criterion for software requirement selection in software projects [32, 33, 34]. The latter take into account the uncertainty of the values through considering user preferences, yet they ignore value dependencies among requirements. To further clarify this, consider the following scenario.

let $R = \{r_1, r_2, r_3\}$ be a requirement set with the estimated values $v_1 = 10, v_2 = 20, v_3 = 30$, costs $c_1 = c_2 = c_3$, and probabilities of being selected by the users $p(r_1) = 0.9, p(r_2) = 0.9, p(r_3) = 0.4$. We assume that the values of r_2 and r_3 highly depend on the presence of r_1 thus ignoring r_1 would significantly impact the values of r_2 and r_3 in a negative way. The following examples show the disadvantages of using AV (Example 1.1) or EV (Example 1.2) as the optimality criterion in requirement selection. v_i and c_i in these examples denote the estimated value and cost of a requirement r_i respectively while $p(r_i)$ specifies the probability that r_i is selected by the users.

Example 1.1. Consider requirement selection for R when budget is available only for one of the requirements r_1, r_2, or r_3. When accumulated value (AV) is used as the optimality criterion, only r_3 with $v_3 = 30$ will be selected. As such, the value of r_3 may be negatively impacted, to a significant extent, by ignoring r_1 and this may result in value loss. Moreover, the accumulated value (AV) does not account for user preferences ignoring the fact that r_3 is less likely to be selected by the users compared to r_2 ($p(r_3) < p(r_2)$). Ignoring user preferences may also result in value loss.

Example 1.2. Consider using expected value (EV) as the optimality criterion in Example 1.1, where we have $E(v_1) = 0.9 \times 10 = 9$, $E(v_2) = 0.9 \times 20 = 18$, and $E(v_3) = 0.4 \times 30 = 12$. $E(v_i)$ denotes the expected value of a requirement r_i. It is clear that based on EV, requirement r_2 with the highest expected value will be selected. r_1 however will be ignored despite its significant impact on the value of r_2. Value loss hence may occur again as a result of ignoring r_1. Intuitively, users that would have used or purchased r_2 in the presence of r_1, now may change their minds in the absence of r_1. This cannot be captured by the selection methods that use EV as the measure of optimality.

Example 1.1 and Example 1.2 show that the selection methods that use AV/EV as the optimality criterion may ignore highly influential requirements if they are of smaller estimated/expected values compared to other requirements. Overall value on th other hand, as presented in this thesis, captures value dependencies among requirements.

Hence, the penalties of ignoring (selecting) requirements with positive (negative) impacts on the values of the selected requirements will be taken into account when making the decisions about selecting or ignoring the requirements.

Some of the extant requirement selection works have attempted to consider value dependencies by manually estimating the values of the requirement subsets. For n requirements thus $O(2^n)$ estimations might be needed in worst case [35] and $O(n^2)$ estimations may be needed when the estimations are limited to the pairs of requirements [22, 36, 19]. Such complexity further limits the practicality of these works, not to mention the issues around the accuracy of the manual estimations.

Moreover, these works do not specify how to estimate the amount of the increased or decreased values. Finally, the requirement selection works based on manual estimations of the values of the requirement subsets do not capture the directions of the influences. In other words, these methods do not distinguish among (a) requirement r_i influences the value of the requirement r_j and not the other way round, (b) r_j influences the value of r_i and not the other way round, and (c) both r_i and r_j influence the value of each other but to different extents.

1.2 Thesis Focus and Key Contributions

To effectively integrate value dependencies into software requirement selection, this thesis presents a mathematical programming approach, referred to as *Dependency-Aware Requirement Selection* (DARS). As outlined in Figure 1.1, the proposed approach includes four different selection methods: (i) an *Integer Programming* (IP) method, i.e. DARS-IP, which takes into account the strengths of value dependencies; (ii) an Integer Linear Programming (ILP) method, i.e. DARS-ILP, which extends the IP method mainly by accounting for the qualities of value dependencies; (iii) a *Mixed Integer Programming* (MIP) method, i.e. DARS-MIP, that allows for partial selection of requirements; and (iv) a *Society-Oriented* method, i.e. DARS-SOC, which takes into account the social values [37, 38] of the requirements.

As depicted in Figure 1.1, each method is comprised of three major components: (i) identification of value dependencies; (ii) modeling value dependencies; and (iii) integrating value dependencies into requirement selection. The identification and modeling components of DARS-MIP, however use the elements of the identification and modeling components of the DARS-ILP method respectively. Moreover, the identification component of DARS-SOC uses the elements of the identification component of the DARS-ILP for the identification of economic value dependencies.

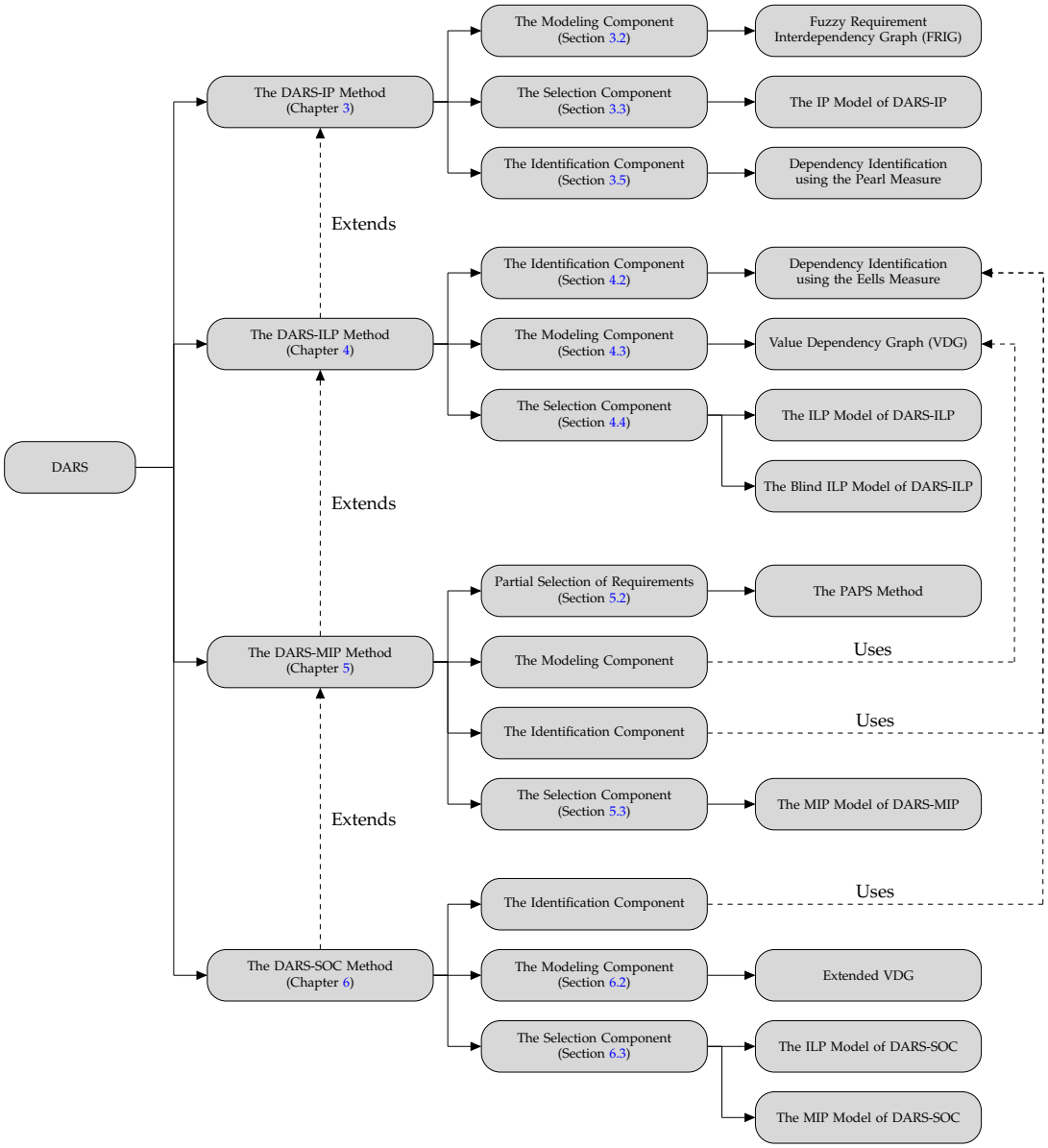


FIGURE 1.1: An overview of the main contributions of the thesis

The optimization models of the requirement selection methods presented in this thesis are convex/linear [39] and can be efficiently solved by the existing commercial solvers such as the *IBM CPLEX* [40]. We have implemented, solved, and tested all these optimization models using the *Concert Technology* and the *JAVA API of IBM CPLEX* [40]. The executable code for these models is available in *JAVA* and *OPL* languages and can be obtained from the website of DARS[1].

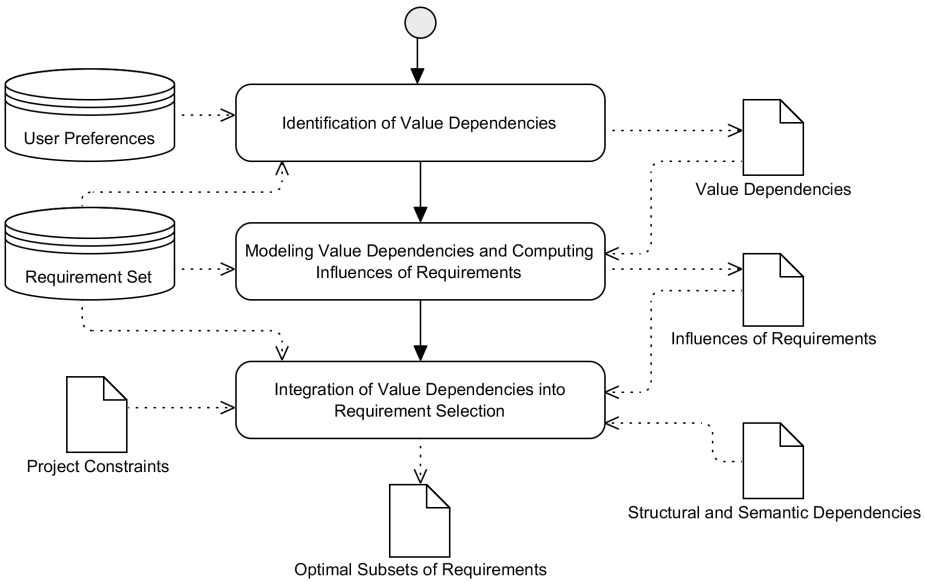


FIGURE 1.2: The main components of different methods of DARS.

Moreover, Figure 2.1 demonstrates the main components of the DARS-IP, DARS-ILP, DARS-MIP, and DARS-SOC methods and the relations among those components at the highest level of abstraction. As demonstrated, the proposed selection methods rely on the identification of value dependencies from the user preferences. The identified value dependencies then will be modeled for reasoning about the implicit value dependencies and computing the positive and negative influences of the requirements on the values of each other.

Eventually requirement selection will be performed subject to the project constraints and precedence dependencies, in order to find optimal or pareto optimal subsets of requirements by considering the impacts of selecting or ignoring requirements on the values of the requirements.

[1] http://bcert.org/projects/dars

1.2.1 The Integer Programming Method (DARS-IP)

The three main components of the proposed integer programming (IP) method of DARS (DARS-IP) are as follows:

(i) *Identification of value dependencies*. One of the most commonly adopted measures of causal strength referred to as *Pearl* measure [41] is used to specify the strengths of causal relations among user preferences for software requirements. Fuzzy membership functions are then used to estimate the strengths of the value dependencies using the identified causal relations;

(ii) *Modeling value dependencies*. We have demonstrated the use of fuzzy graphs [42] and their algebraic structure [43] for modeling the strengths of value dependencies and capturing the imprecision associated with those dependencies;

(iii) *Integrating value dependencies into requirement selection*. We have proposed an IP model which maximizes the overall value (OV) of a selected subset of requirements, where the strengths of the value dependencies are taken into account.

We show the practicality and validity of the IP method of DARS (DARS-IP) by studying a real-world software project and carrying out simulations. We also demonstrate why software vendors should take care with value dependencies among requirements, and how to employ DARS-IP to assist decision makers to comprehend the results, thus raising the following research questions.

(**RQ1**) What is the impact of using DARS-IP on the overall value of software products?

(**RQ2**) What is the relationship between maximizing the accumulated value and overall value of software products?

(**RQ3**) How effective is DARS-IP in mitigating the selection deficiency problem?

(**RQ4**) What is the impact of value dependencies on the performance of DARS-IP?

(**RQ5**) How practical is DARS-IP for software projects?

1.2.2 The Integer Linear Programming Method (DARS-ILP)

The ILP method of DARS (DARS-ILP) extends the DARS-IP method mainly by taking into account the qualities of value dependencies. In this regard, the dependency identification technique in DARS-IP is enhanced by (a) considering both the strengths and qualities of value dependencies and (b) using a formal significance test to understand the accuracy of the value dependencies.

Moreover, the modeling technique proposed in DARS-ILP is extended to capture not only the strengths but also the qualities of value dependencies, thus allowing for reasoning about simultaneous positive and negative impacts of the explicit and implicit value dependencies among the requirements. We have further presented a modified version of the Floyd-Warshall algorithm [44] capable of efficiently computing the positive and negative influences of the requirements on the values of each other based on the algebraic structure of fuzzy graphs.

Last but not least, the ILP model of the DARS-ILP method integrates both positive and negative value dependencies into software requirement selection by taking into account the qualities of value dependencies. The optimization model of the DARS-ILP method is linear and scalable to software projects with large number of requirements. The main components of the proposed DARS-ILP method are as follows:

(i) *Identification of value dependencies*. We have contributed a dependency identification technique that uses the Eells measure of causal strength [45] to estimate the strengths of value dependencies from causal relations among user preferences. The accuracy of such relations is determined by a formal significance test. Identified dependencies will then be used to infer implicit dependencies among requirements. We have further demonstrated using a Latent Multivariate Gaussian model [46] to generate samples of user preferences when collecting sufficient data on user preferences is not practical [46];

(ii) *Modeling value dependencies*. We have demonstrated the use of fuzzy graphs [42] and their algebraic structure [43] for modeling the strengths and qualities of value dependencies and capturing the imprecision associated with those dependencies. On this basis, value dependencies are modeled by fuzzy relations [21,

47, 26, 48], where strengths of those dependencies are captured by their corresponding fuzzy membership functions;

(iii) *Integrating value dependencies into requirement selection*. At the heart of DARS-ILP is an integer linear programming model, which maximizes the *Overall Value* (OV) of a selected subset of requirements, where user preferences and value dependencies identified from those preferences are taken into account. We have further contributed a *Blind* ILP model for DARS-ILP, which aims to mitigate the risk of value loss posed by ignoring the positive influences of the requirement on the values of each other. The Blind model does not require any information about value dependencies, thus it is, specially, useful for the projects in which the identification of the value dependencies is not practical.

We show the practicality and validity of the ILP method of DARS (DARS-ILP) by studying a real-world software project. We also demonstrate why software vendors should take care with value dependencies among requirements, and how to employ DARS-ILP to assist decision makers to comprehend the results, thus raising the following research questions about the ILP method of DARS.

(**RQ6**) How effective is DARS-ILP with respect to considering value dependencies?

(**RQ6.1**) How similar are solutions found by DARS-ILP to those found by the existing requirement selection methods?

(**RQ6.2**) What is the impact of using DARS-ILP on the overall value of software products?

(**RQ6.3**) What is the relationship between maximizing the accumulated value, expected value, and overall value of software products?

(**RQ6.4**) How effective is DARS-ILP in mitigating the value loss caused by ignoring (selecting) requirements with positive (negative) influence on the values of selected requirements?

We moreover, carry out extensive simulations to evaluate the performance of the DARS-ILP method in providing higher overall value and mitigating the value loss by

carrying out simulations for different levels of value dependencies, negative value dependencies, precedence dependencies, negative precedence dependencies, and budget. The following research questions thus will be raised.

(**RQ7**) How is the performance of DARS-ILP affected by changing value dependencies, precedence dependencies and project constraints?

(**RQ7.1**) What is the impact of the value dependencies on the performance of DARS-ILP in the presence of various budget constraints?

(**RQ7.2**) What is the impact of the negative value dependencies on the performance of DARS-ILP in the presence of various budget constraints?

(**RQ7.3**) What is the impact of the precedence dependencies on the performance of DARS-ILP in the presence of various budget constraints?

(**RQ7.4**) What is the impact of the negative precedence dependencies on the performance of DARS-ILP in the presence of various budget constraints?

(**RQ7.5**) What is the impact of the negative value dependencies in highly, moderately, or loosely interdependent value dependency graphs?

(**RQ7.6**) What is the impact of the negative precedence dependencies in highly or loosely interdependent precedence dependency graphs?

Finally, the following research questions pertaining to the scalability of the DARS-ILP are answered through applying DARS-ILP to a real-world software project and carrying out simulations.

(**RQ8**) What is the overhead of identification and modeling of value dependencies in DARS-ILP?

(**RQ9**) How scalable is the ILP model of DARS-ILP?

(**RQ9.1**) Is the ILP model scalable to large scale requirement sets?

(**RQ9.2**) What is the impact of budget on runtime?

(**RQ9.3**) What is the impact of precedence dependencies on runtime?

(**RQ9.4**) What is the impact of value dependencies on runtime?

1.2.3 The Mixed Integer Programming Method (DARS-MIP)

The DARS-ILP method, which is an enhanced version of the DARS-IP method, mitigates the value loss by taking into account the influences of the requirements on the values of each other. But the effectiveness of the DARS-ILP method in mitigating the value loss diminishes when the budget is tight or there exist several precedence relations among the requirements.

The reasons is that requirements with significant positive influences on the values of the selected requirements may have to be ignored due to their conflicts with other requirements or the lack of sufficient budget. Analogously, requirements with negative influences on the values of the requirements may need to be selected when they are required by other selected requirements. Hence, a value loss caused by ignoring (selecting) requirements with positive (negative) influences on the values of the selected requirements is foreseeable in DARS-ILP.

To mitigate this, we have proposed allowing for partial selection (satisfaction) of the requirements when that can be tolerated in software projects. When partial selection is integrated into DARS, requirements with positive (negative) influences on the values of the selected requirements can be partially selected rather than being fully ignored (selected). This can mitigate the value loss caused by ignoring (selecting) the requirements with positive (negative) influences. In doing so, we have presented a mixed integer programming (MIP) method referred to as the MIP method of DARS, i.e. DARS-MIP. Partial selection of requirements may or may not be tolerated and DARS-MIP handles both scenarios.

The optimization model of the DARS-MIP method finds an optimal investment policy that mitigates the value loss by allowing for increasing (decreasing) the investment in the requirements with significant positive (negative) influences on the values of the partially/fully selected requirements. The investment in each requirement r_i is bounded by the lower-bound cost and the upper-bound cost of r_i. The upper-bound cost of r_i is estimated by the stakeholders and then RELAX-ed, using a RELAX-ation technique proposed as part of a fuzzy method referred to as *Prioritization and Partial selection* (PAPS), to determine the lower bound cost of r_i.

The optimization model of the DARS-MIP method is linear and, therefore, scalable to software projects with large number of requirement. Application of the DARS-MIP method to real-world software projects is now under way as part of our ongoing research to further investigate the effectiveness of the method in mitigating the value loss in real-world settings.

1.2.4 The Society-Oriented DARS Method (DARS-SOC)[2]

The DARS-IP, DARS-ILP, and DARS-MIP methods focus on the economic values of software requirements and the dependencies among those values. However, there are several types of human values i.e. *Social Values*, as discussed in [38], with long term impacts on the society that are also important and need to be considered in software engineering activities including the requirement selection [37].

To address this, we have presented a society-oriented method for DARS, i.e. DARS-SOC, that accounts for the social values in dependency-aware software requirement selection. The proposed DARS-SOC method comprises two main optimization models, with different characteristics, that allow for embedding the social values and the dependencies among those values into software requirement selection. We have further demonstrated the use of fuzzy graphs and their algebraic structure for capturing different types of social values in modeling value dependencies.

Our proposed DARS-SOC method relies on the dependency identification component of DARS-ILP for identification of the economic value dependencies. But, to the best of our knowledge, there are not any techniques in the present literature for identification of social value dependencies. These dependencies may be identified manually for small requirement sets. But development of more sophisticated techniques is needed for automated identification of social value dependencies in medium to large scale requirement sets. This is, however, beyond the scope of this thesis.

[2]The author of the thesis has recently joined the Society-Oriented Software Design project at the Faculty of Information Technology, Monash University.

1.3 Publications and Thesis Outline

This thesis is the outcome of research carried out during the PhD candidature at Flinders university. During the period, the main results of the thesis have been published (or submitted for publication) in high quality peer reviewed journals and conferences. These publications are as listed below.

(**P1**) D. Mougouei and D. M. W. Powers. Modeling and selection of interdependent software requirements using fuzzy graphs. *International Journal of Fuzzy Systems*, 19(6):1812–1828, Dec 2017

(**P2**) D. Mougouei. Factoring requirement dependencies in software requirement selection using graphs and integer programming. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 884–887. ACM, 2016

(**P3**) D. Mougouei, D. M. W. Powers, and A. Moeini. An integer linear programming model for binary knapsack problem with dependent item values. In W. Peng, D. Alahakoon, and X. Li, editors, *AI 2017: Advances in Artificial Intelligence: 30th Australasian Joint Conference, Melbourne, VIC, Australia, August 19–20, 2017, Proceedings*, pages 144–154. Springer International Publishing, Cham, 2017

(**P4**) D. Mougouei and D. M. W. Powers. The synergistic knapsack problem. *Fuzzy Optimization and Decision Making*, Under Review

(**P5**) D. Mougouei, D. M. Powers, and A. Moeini. Dependency-aware software release planning. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 198–200. ACM, 2017

(**P6**) D. Mougouei and D. M. W. Powers. An integer programming method for considering value-related dependencies in software requirement selection. *Information and Software Technology*, Under Review

(**P7**) D. Mougouei and D. M. W. Powers. Dependency-aware software release planning using fuzzy graphs and integer programming. *Engineering Applications of Artificial Intelligence*, Under Review

(**P8**) D. Mougouei and D. M. W. Powers. Dependency-aware software release planning through mining user preferences. *Expert Systems with Applications*, Under Review

(**P9**) D. Mougouei, H. Shen, and A. Babar. Partial selection of agile software requirements. *International Journal of Software Engineering & Its Applications*, 9(1):113–126, 2015

(**P10**) D. Mougouei and D. M. W. Powers. Paps: A scalable framework for prioritization and partial selection of security requirements. *International Journal of Approximate Reasoning*, Under Review

(**P11**) D. Mougouei and M. K. Yeung. Visibility requirements engineering for commercial websites. *International Journal of Software Engineering & Its Applications*, 8(8):11–18, 2014

(**P12**) D. Mougouei and D. M. W. Powers. Partial selection of software requirements. *International Conference on Computer Science, Engineering and Applications*, Accepted

(**P13**) D. Mougouei and D. M. W. Powers. An integer programming model for embedding social values into software requirement selection. *International Conference on Computer Science, Engineering and Applications*, Accepted

(**P14**) D. Mougouei and D. M. W. Powers. Gotm: a goal-oriented framework for capturing uncertainty of medical treatments. *Intelligent Systems Conference (IntelliSys) 2018*, Accepted

Table 1.1 shows the relations between the publications during the PhD candidature ((**P1**)-(**P14**)) and different sections of the thesis.

The remainder of this thesis is organized as follows. Chapter 2 gives background information on the concepts discussed in the thesis. In this regard, the most important approaches to optimization and mathematical programming are briefly discussed. We further highlight the importance of considering value dependencies in the existing literature.

TABLE 1.1: Sections of the thesis and their corresponding publications (P1)-(P14).

Section	(P1)	(P2)	(P3)	(P4)	(P5)	(P6)	(P7)	(P8)	(P9)	(P10)	(P11)	(P12)	(P13)	(P14)
Chapter 1: Introduction														
Section 1.2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Chapter 2: Background and Related Work														
Section 2.2	✓	✓	✓	✓	✓	✓	✓	✓						
Chapter 3: The IP Method of DARS														
Section 3.2	✓	✓												
Section 3.3	✓													
Section 3.4	✓	✓												
Section 3.5	✓													
Chapter 4: The ILP Method of DARS														
Section 4.2	✓				✓	✓	✓	✓						
Section 4.3	✓	✓			✓	✓	✓	✓						
Section 4.4	✓		✓	✓	✓	✓	✓	✓						
Section 4.5						✓								
Section 4.6		✓					✓							
Section 4.7			✓	✓										
Chapter 5: The MIP Method of DARS														
Section 5.2									✓	✓	✓	✓		
Section 5.3			✓	✓		✓	✓	✓				✓		
Chapter 6: The Society-Oriented Method of DARS														
Section 6.2						✓		✓					✓	
Section 6.3			✓	✓		✓		✓		✓		✓	✓	
Chapter 7: Conclusions														
Section 7.3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

The chapter continues with a critical review of the main existing software requirement selection works with respect to considering aspects of value dependencies. The detailed review of these works is presented in publications (P1)-(P8). We have further characterized the existing requirement selection works by their corresponding selection methods: (i) *Binary Knapsack* (BK); (ii) *Precedence-Constrained Binary Knapsack* (PCBK); (iii) *Stochastic Binary Knapsack* (SBK); or (iv) *Increase-Decrease*.

Chapter 3 presents the IP method of DARS (DARS-IP). The DARS-IP method includes three main components: (i) identification of value dependencies; (ii) modeling value dependencies; and (iii) integrating value dependencies into requirement selection. The chapter focuses on considering the impacts of value dependencies on the value of an optimal subset of the requirements during a selection process. This is achieved by considering both the existence and the strengths of value dependencies in requirement selection. The main results of this chapter are presented in publications (P1) and (P2). Moreover, the chapter answers (RQ1)-(RQ5).

Chapter 4 presents the ILP method of DARS (DARS-ILP), which improves the DARS-IP method by integrating the qualities of value dependencies into the main components of DARS-IP. The optimization models proposed in DARS-ILP, are linear and scalable to software projects with large number of requirements. The main results of this chapter, as presented in publications **(P1)**-**(P8)**, answer the research questions **(RQ6)**-**(RQ9)**. The generalized form of the optimization models presented in DARS-ILP are further used in publications **(P3)** and **(P4)** to address the *Binary Knapsack Problem with Dependent Item Values* (BKP-DIV) [3], and the *Synergistic Knapsack Problem* (SKP) [4].

Chapter 5 presents the MIP method of DARS, which allows for partial selection (satisfaction) of requirements to further reduce the risk of value loss in software projects. The DARS-MIP method pursues the policy of partially selecting requirements rather than ignoring them or postponing them to the future. Moreover, a fuzzy method is presented to assist partial selection of the requirements. The main contributions of this chapter are presented in publications **(P3)**, **(P4)**, and **(P6)**-**(P12)**.

Chapter 6 presents the Society-Oriented method of DARS (DARS-SOC), which not only takes into account the economic values of the requirements but also accounts for the social values of those requirements. The proposed method embeds social values into software requirement selection using two different ILP models. The chapter further demonstrates the use of fuzzy graphs for modeling the dependencies among the social values of the requirements. The contents of this chapter are mainly presented in publications **(P3)**, **(P4)**, **(P6)**, **(P8)**, **(P10)**, **(P12)**, and **(P13)**.

Chapter 7 summarizes the main contributions of the thesis and concludes the results while highlighting the assumptions and the limitations of using DARS for software requirement selection. We have further discussed a selection of ongoing and future research in this chapter.

Chapter 2

Background and Related Work [1]

2.1 Background

There are different concepts used in this thesis in the area of combinatorial optimization [49], mathematical programming [50], and software requirement selection [32]. Hence, we have briefly introduced some of these concepts in this section. A summary of the key concepts/terms used in the thesis such as the *Integer Programming* (IP), *Integer Linear Programming* (ILP), *Mixed Integer Programming* (MIP), and requirement dependency can be seen in the *Word-Cloud* [51] of Figure 2.1.

FIGURE 2.1: The word-cloud of the thesis.

[1] Review of the existing requirement selection works, with regard to considering value dependencies, is presented in publications **(P1)**-**(P8)**.

2.1.1 Combinatorial Optimization

Combinatorial optimization problems involve a finite number of alternatives: given a ground set $E = \{e_1, ..., e_n\}$ and an objective function $f : 2^E \rightarrow \mathbb{R}$, the set of feasible solutions $S \subset 2^E$ is finite. In a maximization problem the optimization method searches for an optimal solution $s^* \in S$ such that $\forall s \in S$, $f(s^*) \geq f(s)$. Analogously, for a minimization problem the optimization method aims to find an optimal solution $s^* \in S$ such that $\forall s \in S$, $f(s^*) \leq f(s)$. To illustrate one amongst the most famous examples of combinatorial optimization problems, let us consider the *Binary Knapsack Problem* (BKP) [52]. The classical BKP[2] is concerned with finding an optimal subset of items with the highest value while respecting the capacity of the knapsack. In this case the ground set is the set E of items in the knapsack while S, is formed by all subsets of E whose sizes do not exceed the capacity of the knapsack, i.e. feasible subsets. Hence, an optimal solution for the classical binary knapsack problem is a feasible subset of E whose accumulated value is the highest among all other feasible subsets in S: $\forall s \in S f(s^*) \geq f(s)$. $f(s)$ gives the accumulated value of each subset.

Exact Optimization Methods

Exact optimization methods provide the optimum for any instance of the problem. Classical methods for exactly solving a combinatorial optimization problem are mainly *Branch-And-Bound* (appeared in the literature as early as in 1966 [53]) and *Dynamic Programming* (appeared in the literature as early as in 1952 [54]). These methods are based on the well-known concept of *Divide-And-Conquer* and thus are categorized as divide-and-conquer methods. Hence, both branch-and-bound and dynamic programming methods solve a problem by combining the solutions to its subproblems.

The main difference between these methods however resides in the way they partition a problem into subproblems. For a given optimization problem a branch-and-bound algorithm finds an optimal solution to the problem by dividing the problem into independent subproblems, solving the subproblems, and outputting as the optimal solution the best feasible solution found during the search.

[2] http://www.mathcs.emory.edu/ cheung/Courses/323/Syllabus/DynProg/knapsack1.html

On the contrary, the dynamic programming method can be used with a smaller set of optimization problems and more specifically those problems that can be divided into subproblems that are not independent [55]. In this context, a dynamic programming method avoids repeatedly solving common subproblems. That is, each subproblem is solved only once and then its optimal solution is saved in a table.

Approximation Methods

Approximation methods provide solutions of a certain optimality, for any instance of the problem, which means that the distance from the exact optimal solution is known. In other words, approximation methods provide a suboptimal solution with an approximation-guarantee on the optimality of the solution in a sense providing the best that can be done to give a guarantee of optimality for the solutions. Hence, approximation methods are specifically used to solve intractable combinatorial optimization problems for which finding the exact optimal solution is nearly infeasible.

The techniques used for finding approximate optimal solutions for computationally intractable problems are often the ones used for finding exact optimal solutions for tractable problems in a polynomial time. These techniques include:

- local search;
- greedy algorithms;
- sequential algorithms;
- linear programming and relaxation-based algorithms;
- dynamic programming algorithms;
- random algorithms.

More information on designing approximation algorithms can be found in [56].

Heuristics and Metaheuristics

Heuristic algorithms are developed to deal with hard combinatorial optimization problems of large scale for which finding the exact optimal solution is not computationally

tractable nor can an approximation with a reasonable guaranteed distance from the optimal be designed. Heuristics are employed in such circumstances to provide a solution that is "good enough". The heuristic methods are used to overcome the speed and resource limitations of exact and approximation methods. They however, sacrifice the quality of the solutions in favor of the speed and effective resource usage. In some cases in fact the heuristics may completely fail to provide a reasonable solution while in several cases they may find solutions near optimal.

Hence, heuristics do not guarantee optimality nor do they provide any guaranteed distance from the real optimal solution. In other words, there is no general framework behind the design of heuristics that is able to find good quality solutions for all problems. The effectiveness of a heuristic method relies on its ability to adapt to a particular realization, avoid entrapment at local optima, and exploit the basic structure of the problem [55].

Metaheuristics, however, are more generic variations of heuristics. The most frequently used metaheuristics include, but are not limited to, simulated annealing [57], evolutionary techniques such as genetic algorithms [58], ant colony optimization [59], scatter search and path-relinking [60], iterated local search [61], variable neighborhood search [62], and GRASP (Greedy Randomized Adaptive Search Procedures) [63].

One of the interesting definitions for metaheuristics is given in [64]: "A metaheuristic is an iterative master process that guides the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method."

Moreover, in a metaheuristics bibliography by Osman *et al.* [65] they define metaheuristics as: "metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space learning strategies are used to structure information in order to find efficiently near-optimal solutions."

Meta heuristics, which are general purpose heuristics designed for solving familiarities of optimization problems have long been used in many different fields of engineering. In software engineering however, using meta heuristic algorithms is mainly known as *Search-Based Software Engineering* [66].

2.1.2 Mathematical Programming

The term “programming" in the context of mathematical programming means planning activities that consume resources and/or meet requirements. Mathematical programming is mostly known as a technique used by decision makers to mathematically formulate optimization problems and develop optimal values of the decision variables. However, the application of mathematical programming goes beyond this. Three of the common sets of usages of mathematical programming are highlighted in [67]:

(i) Building problem insight: Mathematical programming helps thoroughly understand the problem and state a problem carefully as the model builder is required to specify decision variables, constraints, the objective function, data constraints and constraints that capture relationships among variables.

(ii) Numerical Mathematical Programming: Numerical usages of mathematical programming include prescription of solutions, prediction of consequences, demonstration of sensitivity, and solution of systems of equations. Although the most commonly thought of application of mathematical programming is to find the optimal decisions, there are many other applications. For instance, there are mathematical programming models, which have been never used to find the optimal decision, rather these models are employed to demonstrate what happens if certain factors are changed [67].

(iii) Solution Algorithm Development: Even though this is not a usage, mathematical programming models are often used by researchers as a framework for developing solution algorithms for solving problems. Research is also done on new formulation techniques and their ability to appropriately capture real-world problems.

Linear Programming

Linear programming, as it is known today, emerged out of the empirical programming needs of the Air Force during World War II. The general linear programming problem was devised by George B. Dantzig who also developed the *Simplex* method [68] of solving linear programming problems around 1947 [69].The method is still the most general and powerful-enough way of solving a large class of real-world problems that can be formulated as linear programming problems. The simplex method yields an exact optimal solution of the problem in a finite number of iterative steps. Soon after World War II, the great potentialities of linear programming were realized and there followed throughout the business, engineering, and scientific world, a rapidly expanding interest in the areas and methods of optimization [69].

A *Linear Programming* (LP) problem is a mathematical problem, where decision variables are selected so that a linear function of the decision variables is optimized while satisfying a simultaneous set of linear constraints. If the objective function or any of the constraints are not linear the problem is not characterized as LP anymore. Nonlinear optimization problems are generally more difficult to solve as they are of higher computational complexity.

Each LP problem comprises several essential components. First, there are n different unknowns $E = \{e_1, ..., e_n\}$ for which a decision variable x_i denotes the amount undertaken of its respective e_i. Second is a linear objective function which gives the total objective value $\sum_{i=1}^{n} x_i v_i$ for a feasible solution. To be feasible however a solution must simultaneously satisfy all the constraints that the problem of concern is subject to. All constraints of a linear programming problem must also be linear.

To demonstrate this consider one of the most common optimization problems referred to as the binary knapsack problem (BKP) [52]. The classical BKP is concerned with finding an optimal subset of items with the highest value ($\sum_{i=1}^{n} x_i v_i$), which is a linear objective, while respecting the capacity of the knapsack (C). In this case the ground set is the set E of items in the knapsack while S, is formed by all subsets of E whose sizes do not exceed the capacity of the knapsack ($\sum_{i=1}^{n} x_i s_i$), i.e. feasible subsets. This is specified by (2.2), which is also a linear constraint.

An optimal solution for the classical binary knapsack problem hence is a feasible subset of E whose accumulated value is the highest among all other feasible subsets in S: $\forall s \in S f(s^*) \geq f(s)$. $f(s)$ gives the accumulated value of each subset as given by (2.1)-(2.3). In these equations s_i and v_i denote size and value of the item e_i.

$$\text{Maximize} \sum_{i=1}^{n} x_i v_i \tag{2.1}$$

$$\text{Subject to} \sum_{i=1}^{n} x_i s_i \leq C \tag{2.2}$$

$$x_i \in \{0, 1\}, \qquad i = 1, ..., n \tag{2.3}$$

The binary knapsack problem hence is a linear programming (LP) problem since the objective function (2.1) as well as constraints (2.2) and (2.3) are all linear.

Integer Programming and Mixed Integer Programming

An integer program is a mathematical model in which all decision variables are integers. When at least one of the variables is relaxed to be a real number the model is referred to as a mixed integer programming model. It is worth mentioning that in some texts [70] the term integer programming comprises both of the aforementioned models while the term *Pure Integer Programming* is used to specify integer programming models with all integer variables.

It is clear that integer programming models and mixed integer programming models can be linear or nonlinear depending on the formulation of the objective function and constraints of the optimization models. For instance the classical formulation of the binary knapsack problem is a linear optimization problem with integer, in this case 0 and 1, variables x_i. Hence, the problem is an integer linear programming problem.

Convex Optimization

There are several real-world problems which cannot be formulated as a linear programming (LP) problem. However, they can still be solved efficiently if they can be formulated as a convex optimization problem [39] of form (2.4)-(2.5).

$$\text{Minimize } f_0(x) \tag{2.4}$$

$$\text{Subject to } f_0(x) \leq b_i, \quad i = 1, ..., n \tag{2.5}$$

The objective functions $f_0, ..., f_m : \mathbb{R}^n \rightarrow \mathbb{R}$ in (2.4) are all convex, meaning that they satisfy $f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y)$ for all $x,\ y \in \mathbb{R}^n$ and all $\alpha,\ \beta \in \mathbb{R}$ with $\alpha + \beta = 1$, $\alpha \geq 0, \beta \geq 0$.

For an optimization problem to be convex, the objective function must be convex, all inequality constraints, as in (2.5), must also be convex, and all equality constraints (if any) must be linear. It is clear that the linear programming problem is a special case of a convex problem where the objective function is linear.

Unfortunately, there is no general formula for solving convex optimization problems. But, as with linear programming problems, there are very effective methods of solving them. Interior-point methods work very well in practice, and in some cases can be proved to solve the problem to a specified accuracy with a number of operations bounded with polynomial of the problem dimensions [39]. There are techniques and tools such as IBM CPLEX that effectively solve different classes of convex optimization problems [39].

2.1.3 Value Dependencies among Software Requirements

It is widely recognized that the requirements of a software project influence the values of each other [19, 71, 20, 18]. Such influences are described in the literature as *value* dependencies [21, 22], *CVALUE* dependencies [21], *Increases/Decreases_value_of* dependencies [18, 23], and *Positive/Negative value* dependencies [24].

We use the term *value dependencies* consistently throughout this thesis. Identified explicitly or implicitly among software requirements, value dependencies are known to be of the most common types of requirement dependencies [21, 72].

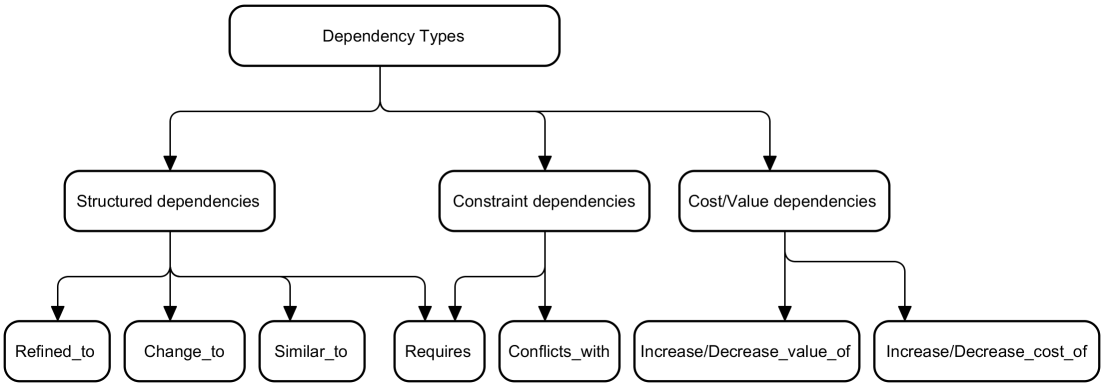


FIGURE 2.2: Dahlstedt's requirement dependency model.

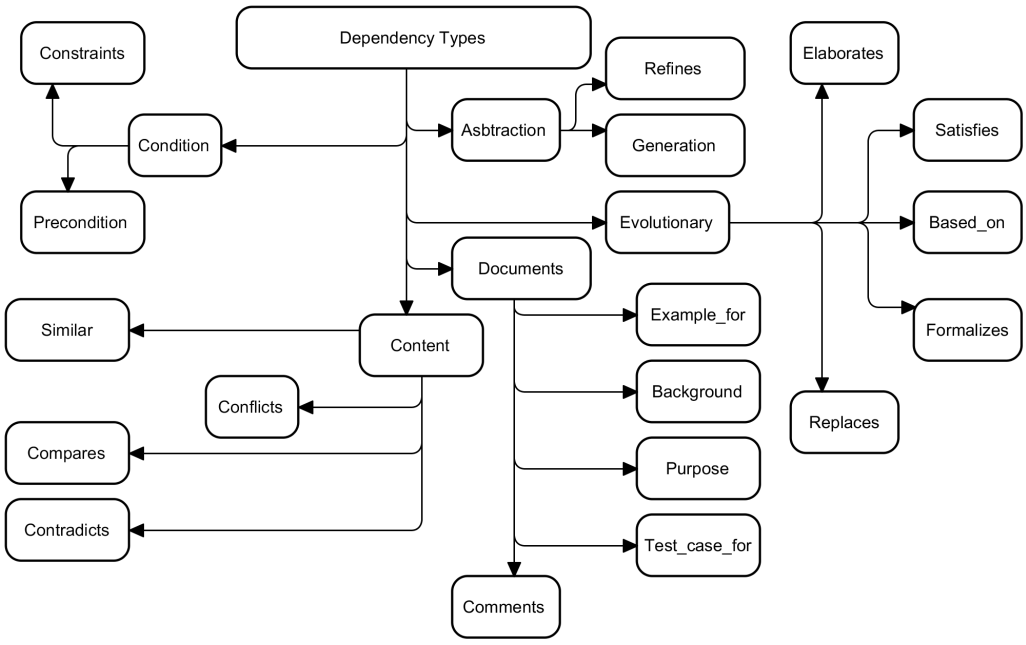


FIGURE 2.3: Pohl's requirement dependency model.

TABLE 2.1: Types of requirement dependencies.

Type	Description	Example
Intrinsic Dependencies		
Constrain [23, 73]	One requirement is a constraint of another requirement. This kind of dependency can represent crosscutting relationships among requirements.	• r_1 : System should respond in 3 seconds. • r_2 : Users search books. • r_1 constrains r_2.
Precede [23], Precondition [73], Requires [18]	If A precedes B, A is a precondition of B.	• r_1 : System authenticates users. • r_2 : Users search books. • r_1 precedes r_2.
Be_similar_to [23], Similar [73], Similar_to [18]	If two requirements share similar data information or complement each other	• r_1: Adding a new book record. • r_2: Modifying an old book record. • r_1 is similar to r_2.
Conflicts [23, 73], Conflicts_with [18]	One requirement negatively impacts another requirement.	• r_1: Increasing security. • r_2: Enhancing performance. • r_1 conflicts with r_2.
Be_exception_of [23]	One requirement describes an exceptional event of another requirement.	• r_1 : A user inputs a null user name. • r_2 : System authenticates user. • r_1 is an exception of r_2.
Evolve_into [23], Replaces [73], Based_on [73]	A requirement evolves into a new version This is used to trace and compare different versions of requirement documents.	• r_1 : Uploading reports using a desktop. • r_2 : Uploading reports using PDA. • r_1 evolves into r_2.
Refines [23, 73], Refines_to [18]	One requirement is refined to more specific requirements.	• r_1 : a valuer submits a valuation report. • r_2 : a valuer submits a report through website. • r_1 is refined as r_2.
Additional Cost/Value Dependencies		
Increase/Decrease_cost_of [23, 18]	The implementation of one requirement increases/decreases the implementation cost of another requirement.	• r_1: No response-time should exceed 5 seconds. • r_2: Search for books. • r_1 increases the cost of r_2.
Increase/Decrease_value_of [23, 18]	The implementation of one requirement increase/decrease the value of another requirement.	• r_1 : Users listen to music on a mobile phone. • r_2 : Users browse photos on a mobile phone. • r_1 Increases the value of r_2.

Based on the well-known dependency models from Dahlstedt [18] (Figure 2.2) and Pohl [73] (Figure 2.3), Zhang *et al.* have classified requirement dependency types into two categories [23]. The first category, referred to as *intrinsic* dependencies, reflect structural/semantic information of requirements. In a practical context, intrinsic dependencies may be used to discover some of the cost-related or value dependencies among the requirements of software projects [23]. In other words, cost-related and value dependencies may coexist with *intrinsic* dependencies. The second category of dependencies, referred to as *additional cost or value* dependencies, is introduced to capture cost-related or value dependencies in the absence of intrinsic dependencies. Table 2.1 lists requirement dependencies and their corresponding examples as given in Zhang's categorization [23].

Explicit value dependencies can be identified through pairwise comparisons among software requirements, which is a complex process [21]. However, several techniques have been devised [21, 74, 75] to reduce this complexity with varying degrees of efficiency. Based on a study [21] by Carlshamre *et al.*, identification of independent i.e. singular requirements, scanning for similar requirements, and identification of highly dependent requirements substantially reduce the effort required for identification of requirement dependencies [21]. Nonetheless, manual identification of value dependencies still remains difficult in practice and prone to human error.

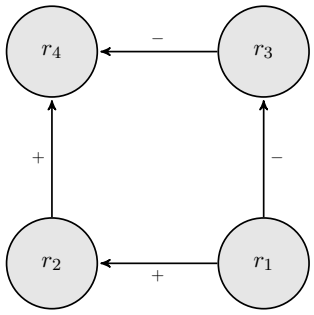


FIGURE 2.4: Value dependencies in Example 2.1

These can be mitigated by automating the dependency identification through exploiting the historical data about a software product as will be discussed in Chapter 3 and Chapter 4. Implicit value dependencies may also exist among requirements. Identifying these dependencies may be even more difficult than finding explicit value dependencies as such implicit or indirect dependencies are more difficult to trace among requirements by the experts. This further underpins the importance of using automated techniques and proper modeling to reason about implicit value dependencies.

We exploit the algebraic structure of fuzzy graphs for modeling value dependencies and reasoning about implicit value dependencies as will be discussed in Chapter 3 and Chapter 4. Implicit value dependencies hence can be inferred from properly modeled explicit value dependencies. Example 2.1 discusses explicit and implicit value dependencies in a typical money transfer system [76] characterized by a signed directed graph of Figure 2.4 [77].

Example 2.1. Consider a money transfer system with a set of identified requirements $R = \{r_1 : \textit{transfer money}, r_2 : \textit{enhance confidentiality}, r_3 : \textit{enhance performance}, r_4 : \textit{encrypt data}\}$. As given by Figure (2.4), requirements r_2 : *enhance confidentiality* and r_3 : *enhance performance* positively influence the value of requirement r_1 : *transfer money*. Hence, r_2 and r_3 are said to be value dependencies of r_1. In other words, the value of r_1 depends on the presence of r_2 and r_3 in the money transfer system. Requirement r_4 : *encrypt data* positively influences the value of r_2 : *enhance confidentiality* while it negatively influences the value of r_3 : *enhance performance* as encryption and performance are conflicting objectives in software products [78]. Thus, it is clear that r_4 has an implicit positive influence on r_1 in one way (r_1, r_2, r_4) while it negatively influences

the value of r_1 in another way (r_1, r_3, r_4) [79]. Thus, both implicit value dependencies (r_1, r_2, r_4) and (r_1, r_3, r_4) can simultaneously [79] exist. This example shows that: (a) a software requirement can implicitly influence the value of another requirement and (b) a requirement can influence the value of another requirement both positively and negatively.

Moreover, as observed by Carlshamre *et al.* [21] requirement dependencies in general and value dependencies in particular are fuzzy [21] in the sense that the strength of requirement dependencies are imprecise and vary from large to insignificant in the context of real-world projects [80, 20, 26, 26, 81].

2.2 Related Work

It is widely recognized that the requirements of a software projects influence the values of each other [19, 71, 20, 18]. Such influences are described in the literature as value dependencies [21, 22, 23, 24]. Value dependencies are fuzzy relations [21] with varying strengths (e.g. weak, moderate, strong) and qualities (positive or negative) which are imprecise and hard to specify [21, 2] in real-world software projects. Hence, software requirement selection methods should consider the qualities and strengths of explicit and implicit value dependencies while taking into account the imprecision of those dependencies.

Moreover, *Precedence Dependencies* such as *Requires* [18] and *Conflicts-With* [73], also have value implications. For instance, a requirement r_i requires (conflicts-with) r_j means that r_i cannot give any value if r_j is ignored (selected). Hence, it is also important to consider the value implications of precedence dependencies in software requirement selection. On this basis, we have characterized the following criteria for evaluating the existing software requirement selection works based on how they consider different aspects of value dependencies.

(**C1**) Taking into account explicit value dependencies.

(**C2**) Taking into account implicit value dependencies.

(**C3**) Taking into account qualities (positive or negative) of value dependencies.

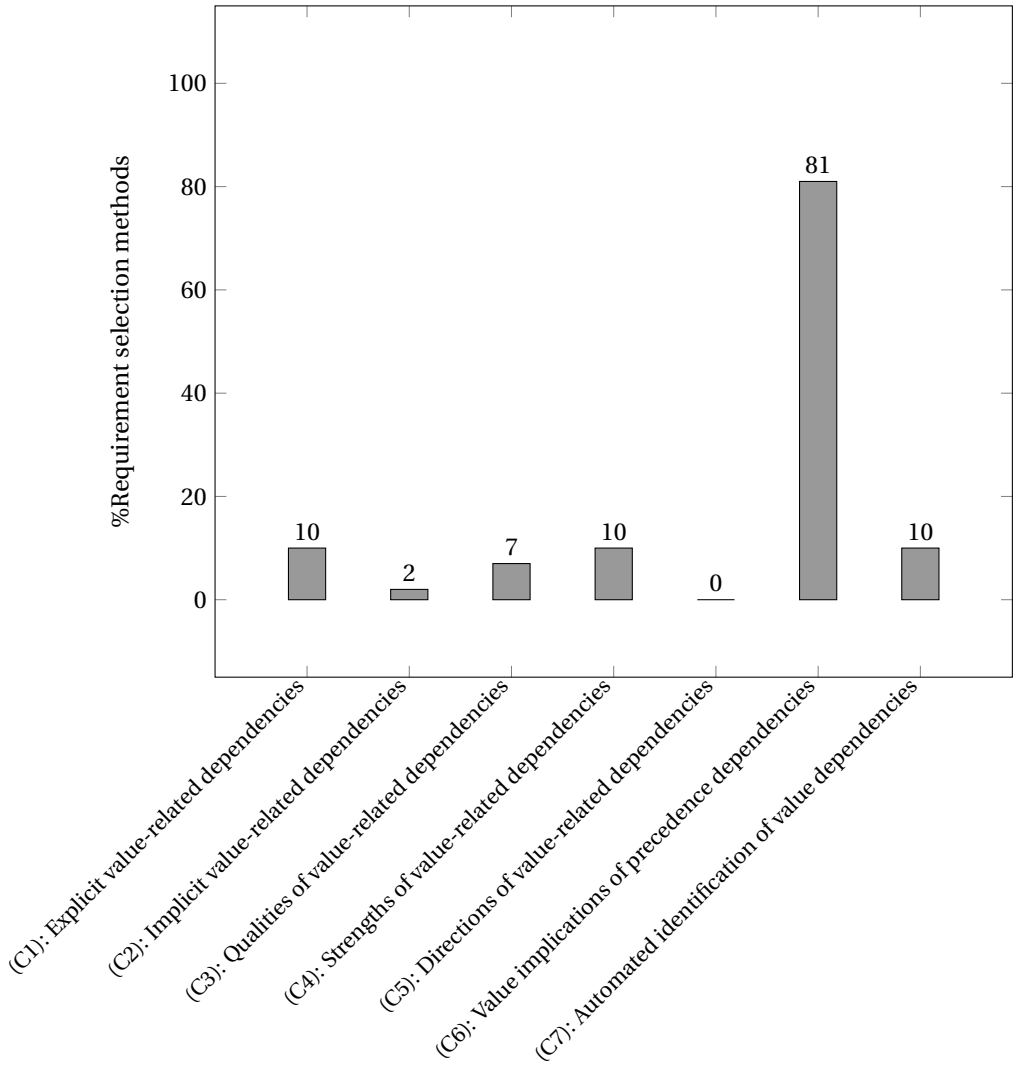


FIGURE 2.5: Percentages of the existing requirement selection methods that address the criteria **(C1)**-**(C7)**.

(**C4**) Taking into account strengths of value dependencies.

(**C5**) Taking into account directions of value dependencies.

(**C6**) Taking into account value implications of precedence dependencies.

(**C7**) Automated identification of value dependencies.

It can be seen in Figure 2.5 that the majority of the existing requirement selection works (around 81%), reviewed in this thesis, address the criterion **(C6)** by taking into account value implications of precedence dependencies while only around 10% of the existing

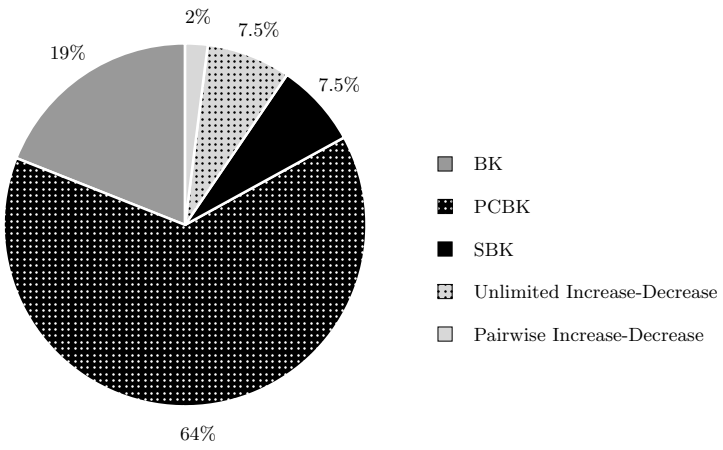


FIGURE 2.6: Percentages of the existing requirement selection works that are based on the BK, PCBK, SBK, or Increase-Decrease methods.

works account for explicit value dependencies (address criterion **(C1)**). This figure reduces even further to around 2% for works that consider implicit value dependencies (address criterion **(C2)**).

Another important criterion for evaluating the requirement selection works, with regard to considering value dependencies and their aspects, is **(C3)**: taking into account the quality of value dependencies. That is to consider if requirements positively or negatively influence the values of each other. This criterion is addressed by around 7% of the main requirement selection works.

Although around 10% of the main requirement selection works account for strengths of value dependencies and therefore address **(C4)**, none of them captures the directions of those dependencies (not addressing **(C5)**). Also, requirement selection works that consider the strengths of value dependencies undermine the criterion **(C7)** by relying on manually estimating the values of subsets of the requirements as demonstrated in Figure 2.5. This, however, limits the practicality of such methods as will be discussed in detail in Section 2.2.3.

Depending on their mathematical formulation the existing software requirement selection works are based on one of the four main categories of the *Binary Knapsack* (BK), *Precedence-Constrained Binary Knapsack* (PCBK), *Stochastic Binary Knapsack* (SBK), and *Increase-Decrease* methods as will be explained in the following subsections.

TABLE 2.2: Addressing the criteria (C1)-(C7) regarding different aspects of value dependencies by different selection methods and their corresponding works from the literature.

Selection method	Works from the literature	(C1)	(C2)	(C3)	(C4)	(C5)	(C6)	(C7)
BK	[82, 83, 84, 28, 85, 86, 87, 88]	NO	NO	NO	NO	NO	NO	NO
PCBK method	[89, 71, 36, 16, 90, 29, 91, 92, 93, 94, 95] [96, 97, 98, 99, 100, 101, 102, 103, 30, 104, 105] [106, 25, 107, 108, 109]	NO	NO	NO	NO	NO	YES	NO
SBK	[32, 33, 34]	NO	NO	NO	NO	NO	YES	NO
Increase-Decrease	[22, 36, 19] (pairwise)	YES	NO	YES	YES	NO	YES	YES
	[35] (unlimited)	YES	YES	NO	YES	NO	YES	YES

We further review the *Oregon Trail Knapsack Problem* (OTKP) [110], which is an attempt to consider value dependencies in the general problem of binary knapsack with dependent item values [3]. This method has not been used by any of the existing requirement selection works. But, it is worth discussing as it shows the importance of the problem of concern and the complexities of solving that problem.

Figure 2.6 shows percentages of the existing requirement selection works that are based on the BK, PCBK, SBK, or Increase-Decrease methods. Also, Table 2.2 lists these methods and their corresponding works from the literature. The selection methods and their corresponding works from the literature are compared in Table 2.2 based on the criteria (C1)-(C7).

2.2.1 The Binary Knapsack Method

The binary knapsack (BK) method is solely based on the classical formulation of the binary knapsack problem [111, 21] as given by (2.6)-(2.8). Let $R = \{r_1, ..., r_n\}$ be a set of identified requirements, where $\forall r_i \in R$ $(1 \leq i \leq n)$, v_i and c_i in (2.6)-(2.8) denote the value and the cost of r_i respectively. Also, b in (2.7) denotes the available budget.

$$\text{Maximize} \sum_{i=1}^{n} v_i x_i \tag{2.6}$$

$$\text{Subject to} \sum_{i=1}^{n} c_i x_i \leq b \tag{2.7}$$

$$x_i \in \{0, 1\}, \quad i = 1, ..., n \tag{2.8}$$

A decision variable x_i specifies whether requirement r_i is selected ($x_i = 1$) or not ($x_i = 0$). The objective of BK method as given by (2.6) is to find a subset of R that maximizes the accumulated value of a selected requirements ($\sum_{i=1}^{n} v_i x_i$) while entirely ignoring the value dependencies as well as the precedence dependencies among the requirements [82, 83, 84].

As demonstrated in figure 2.5, the binary knapsack (BK) method does not satisfy any of the criteria of considering value dependencies as it entirely ignores those dependencies. As shown in Figure 2.6, 19% of the main requirement selection works in the existing literature are based on the BK method.

2.2.2 The Precedence-Constrained Binary Knapsack Method

The precedence-constrained binary knapsack (PCBK) method, enhances the BK method by adding (2.12) to the optimization model of the BK method to account for precedence dependencies. A positive (negative) dependency from a requirement r_j to r_k is denoted by $x_j \leq x_k$ ($x_j \leq 1 - x_k$) in (2.12). Also, the decision variable x_i denotes whether a requirement r_i is selected ($x_i = 1$) or not.

$$\text{Maximize} \sum_{i=1}^{n} v_i x_i \tag{2.9}$$

$$\text{Subject to} \sum_{i=1}^{n} c_i x_i \leq b \tag{2.10}$$

$$x_i \in \{0, 1\}, \quad i = 1, ..., n \tag{2.11}$$

$$\begin{cases} x_j \leq x_k & \text{if } r_j \text{ positively depends on } r_k \\ x_j \leq 1 - x_k & \text{if } r_j \text{ negatively depends on } r_k, \quad j \neq k = 1, ..., n \end{cases} \tag{2.12}$$

However, when the PCBK method is used for requirement selection, value dependencies either have to be formulated as precedence relations (as in the PCBK method used in Chapter 3), or be ignored (as in the PCBK method used in Chapter 4).

When value dependencies are formulated as precedence constraints, however, all dependencies are treated as binary (0/1) relations and consequently a requirement cannot be selected even in the presence of a sufficient budget unless all of its dependent requirements are selected. This makes the PCBK method prone to the selection deficiency problem (SDP) [2] as explained earlier. As a result of the SDP, any increase in the number of dependencies would dramatically depreciate the accumulated value of selected requirements [22]. Therefore the SDP can severely impact the effectiveness of the PCBK method. In one study, Chen et al. [22] demonstrated that a 2% increase in the number of precedence dependencies would lead to almost a 10% decrease in the accumulated value of the optimal subset.

The SDP occurs if the condition of (2.13) holds. The dependency set D in (2.13) specifies the explicit dependencies among a set of requirements $R = \{r_1, ..., r_n\}$, where R is partitioned into two distinct subsets: (i) an optimal subset $O \subseteq R$ (selected requirements); and (ii) an excluded set $\tilde{O} \subseteq R$ (ignored requirements) such that $O \cap \tilde{O} = \emptyset$.

$$\exists\, r_i, r_j \in \tilde{O} : (r_i, r_j) \in D, (\sum_{r_k \in O} c_k) + c_i \leq b \qquad (2.13)$$
$$(\sum_{r_k \in O} c_k) + c_i + c_j > b$$

It is clear that when value dependencies are ignored, only value implications of the precedence dependencies (**C6**) such as *requires* [18] and *conflicts-with* [73] can be captured by the PCBK method. The majority (around 64%) of the main requirement selection works are based on this variation of the PCBK method as shown in Figure 2.6. Hence, the term "PCBK method" in this thesis, except in Chapter 3, refers to the variation of the PCBK method that ignores value dependencies. To demonstrate the impact of SDP on requirement selection, however, we use the other variation of the PCBK method, which models value dependencies as precedence relations in Chapter 3.

One may suggest considering some of the stronger value dependencies as precedence constraints. The effectiveness of such method is arguable as it may still be prone to SDP depending on the ratio of the strong value dependencies and also the definitions

of the strong and weak dependencies. Moreover, such analysis requires identification of value dependencies and their strengths, that has not been addressed by any of the existing works based on the PCBK method as shown in Table 2.2.

2.2.3 The Increase-Decrease Method

The Increase-Decrease selection method considers value dependencies among requirements through estimating the amount of the increased (decreased) values, which result from selecting different subsets of requirements. There are two different variations of the Increase-Decrease method in the existing literature.

The first variation of the Increase-Decrease method, referred to as the *Unlimited Increase-Decrease* method, as presented by Akker *et al.* [35] and given in (2.14)-(2.17) relies on estimating the values of the requirement subsets of unlimited sizes ($\{2, ..., n\}$ for n requirements).

In (2.14)-(2.17), for each subset $s_j \in S : \{s_1, ..., s_m\}, m \leq 2^n$, with n_j requirements, the difference between the estimated value of s_j (w_j) and the accumulated value of the requirements in s_j ($\sum_{r_k \in s_j} v_k$) is considered when computing the value of the selected requirements. y_j in (2.14) specifies whether a subset s_j is realized ($y_j = 1$) or not ($y_j = 0$). Also, constraint (2.16) ensures that $y_j = 1$ only if $\forall r_k \in s_j, x_k = 1$.

$$\text{Maximize} \sum_{i=1}^{n} v_i x_i + \sum_{j=1}^{m} (w_j - \sum_{r_k \in s_j} v_k)\, y_j \tag{2.14}$$

$$\text{Subject to } n_j y_j \leq \sum_{r_k \in s_j} x_k \tag{2.15}$$

$$\sum_{i=1}^{n} c_i x_i \leq b \tag{2.16}$$

$$x_i, y_j \in \{0, 1\}, \quad i = 1, ..., n, \quad j = 1, ..., m \tag{2.17}$$

The Unlimited Increase-Decrease method is complex and prone to human error as it relies on manual estimations for requirement subsets [2]. These estimations may get as complex as $O(2^n)$ for n requirements.

The second variation of the Increase-Decrease method [22, 36, 19] referred to as the *Pairwise Increase-Decrease* method, enhances the Unlimited Increase-Decrease method by limiting the estimations to pairs of requirements, as given in (2.18)-(2.23), thus reducing the complexity of the estimations to $O(n^2)$.

$$\text{Maximize } \sum_{i=1}^{n} v_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j w_{i,j} \tag{2.18}$$

$$\text{Subject to } \sum_{i=1}^{n} c_i x_i \leq b \tag{2.19}$$

$$y_{ij} \leq x_i \tag{2.20}$$

$$y_{ij} \leq x_j \tag{2.21}$$

$$y_{i,j} \geq x_i + x_j - 1 \tag{2.22}$$

$$x_i, y_{i,j} \in \{0,1\}, \quad i,j = 1,...,n \tag{2.23}$$

This complexity however cannot be tolerated for software projects with medium to large number of requirements. Moreover, relying on pairwise estimations results in ignoring implicit value dependencies as the directions of dependencies are not specified. For instance, consider requirements $R : \{r_1, r_2, r_3\}$ with the positive value dependencies from r_1 to r_2 and from r_2 to r_3.

An implicit positive value dependency from r_1 to r_3 can be inferred. An Increase-Decrease model, however, fails to capture this even if pairwise estimations identify that the value of r_1 AND r_2 (r_2 AND r_3) as a pair is higher than the accumulated value of r_1 and r_2 (r_2 and r_3). Hence, if no explicit value dependency is found between r_1 and r_3 the influence of r_3 on the value of r_1 will be ignored.

Finally, both the Unlimited and Pairwise Increase-Decrease methods rely on manual estimations for requirement subsets [2] and therefore they are prone to human error. This can be mitigated to some extent by repeating the estimations by different experts. But on the other hand, repeating the estimations introduces more complexity to the estimation process.

2.2.4 The Stochastic Binary Knapsack Method

The stochastic binary knapsack (SBK) requirement selection method maximizes the expected value of a requirement subset based on the formulation of the stochastic knapsack problem [112] as given by (2.24). In this equation, $E(v_i)$ denotes the expected value of a requirement r_i.

The work [32] for instance optimizes the expected value of a software product at different risk levels, where risk is defined as the summation of the covariances of the values of the requirements as given by (2.25). In this constraint, $cov(v_i, v_j)$ specifies covariance of the v_i and v_j and l denotes the tolerable risk level.

Risk is used in the SBK method to promote more diversified solutions (requirement subsets) through minimizing or limiting the variance of the value of the selected subset of requirements. This is achieved by preferring requirement subsets with negligible or negative linear correlations among requirements over requirement subsets with positive correlations among requirements. The reason is that positively correlated requirements increase the value of $\sum_{j=1}^{n} x_i x_j cov(v_i, v_j)$ while negatively correlated requirements decrease the value of $\sum_{j=1}^{n} x_i x_j cov(v_i, v_j)$ which respects (2.25). Minimizing or limiting the variance, therefore, results in choosing more diversified subsets of requirements.

Such diversification results in selecting variety of requirements that satisfy certain classes of users while may not be suitable to other users. This may help reduce the risk of value loss through diversification as most users will like some features of the software. But it may also result in choosing (ignoring) requirements that negatively (positively) influence the values of each other, thus reducing user satisfaction of requirements and resulting in value loss in another way.

Hence, requirement selection works such as [32, 33, 34], which are based on minimizing the variance of the value of the selected requirements, may result in value loss through selecting requirements that negatively influence the values of each other or ignoring requirements which positively influence the values of other requirements. In other words, enhancing diversification may conflict with reducing the value loss caused by ignoring (selecting) positive (negative) value dependencies of requirements.

$$\text{Maximize} \sum_{i=1}^{n} x_i E(v_i) \tag{2.24}$$

$$\text{Subject to} \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j cov(v_i, v_j) \leq l \tag{2.25}$$

$$\sum_{i=1}^{n} x_i c_i \leq b \tag{2.26}$$

$$x_i \in \{0, 1\}, \qquad i = 1, ..., n \tag{2.27}$$

Moreover, there are even problems with using (2.25) for the purpose of diversification (reducing the risk of value loss) in [32, 33, 34], which have been extensively discussed in the portfolio optimization literature [113, 114]. To mention a few, optimization methods based on covariance can only capture linear correlations ignoring non-linear correlations even if they are significant.

Also, by using (2.25) one is assuming the values of the requirements are jointly normally distributed. This assumption has been repeatedly violated in real-world investment problems such as in the area of portfolio optimization [115, 113, 114]. Furthermore, there is not any research in (to the best of our knowledge) to have suggested that values of software requirements are jointly normally distributed.

To summarize, as given by Table 2.2, software requirement selection works based on the SBK method, which constitute around 7.5% of the main requirement selection works, are the same as the works based on the PCBK method as far as considering aspects of value dependencies is concerned. That is, those requirement selection works only satisfy criterion (**C6**) by considering the value implications of precedence dependencies.

$$\text{Maximize} \sum_{i=1}^{n} x_i E(v_i) \tag{2.28}$$

$$\sum_{i=1}^{n} x_i c_i \leq b \tag{2.29}$$

$$x_i \in \{0, 1\}, \qquad i = 1, ..., n \tag{2.30}$$

In this thesis (2.28)-(2.30) specify the optimization model of the SBK method. In other words, we do not consider the constraint (2.25) as discussing diversification is beyond the scope of this thesis.

2.2.5 The Oregon Trail Knapsack Problem

Burg *et al.* [110] presented a variation of the knapsack problem with dependent item values [3] referred to as the Oregon Trail Knapsack Problem (OTKP), which was inspired from the Oregon Trail computer game, where players are asked to imagine preparing for a trek across the Oregon Trail. In order to make it across country, the travelers need to get good value for the supplies they purchase. They have a given amount of money to spend, and the weight of their supplies is bounded by the capacity of their wagon.

On this basis the formulation of the OTKP offered by Burg *et al.* [110] imposes cost and weight limits, defines the value of each item by a value function that allows for value of an item type to depend on the presence or absence of another item type in the knapsack. Hence value of an item type may be constant, or it may decrease for instance as more than one item of that type is taken. The rational behind this is that: one does not need an infinite number of items of a certain type, and thus they diminish in value as you take more of them.

More formally, the OTKP is formulated as given by (2.31)-(2.34), where x_j is an integer bounded by k_j while w and c denote the weight and cost limits respectively. Also, $f_j(v_j, x_j, x_{d_j})$ is the value function of type j, in which v_j is a constant value of the item j without considering the dependencies among item types and d_j gives the index of the type upon which the value of x_j depends.

An array of constant indices $d_1, ..., d_n$ thus, captures the dependencies among the item types. Also, x_{d_j} is a boolean variable with $x_{d_j} = 1$ when at least one item of type d_j is selected and we have $x_{d_j} = 0$ when no item of type d_j is in the knapsack. u_j gives the upper-bound for the number of items of type j.

$$\text{Maximize} \sum_{i=1}^{n} f_j(v_j, x_j, x_{d_j}) \tag{2.31}$$

$$\sum_{i=1}^{n} x_i w_i \leq w \tag{2.32}$$

$$\sum_{i=1}^{n} x_i c_i \leq c \tag{2.33}$$

$$x_j \in \{0, ..., u_j\} \qquad j = 1, ..., n \tag{2.34}$$

Burg *et al.* [110] combined constraint propagation techniques and domain pruning with classic branch and bound approaches to solve the optimization model of (2.31)-(2.34) with the value functions in (2.35)-(2.37), (2.35)-(2.37), and (2.41)-(2.43).

The value function (2.35)-(2.37) ensures that items of type j have value only if (a) at least one item of type d_j is present in the knapsack ($x_{d_j} > 0 \rightarrow k_{d_j} = 1$). k_{d_j} specifies if condition (a) is satisfied or not. The problem with this value function is that it does not consider strengths of value dependencies by treating all dependencies as binary relations. Moreover, (2.35)-(2.37) only capture the influences of items types on the values of items ignoring the fact that the values of items may also be impacted by the items of the same type to different extents.

$$f_j(v_j, x_j, x_{d_j}) = x_j v_j k_{d_j} \tag{2.35}$$

$$\begin{cases} k_{d_j} = 1 & \text{if } x_{d_j} > 0 \\ k_{d_j} = 0 & \text{if } x_{d_j} \leq 0 \end{cases} \tag{2.36}$$

$$x_j \in \{0, 1\}, \quad j = 1, ..., n \tag{2.37}$$

Equations (2.35)-(2.37) give the second value function proposed by Burg *et al.* [110], in which the value of an item j diminishes at the rate of α by selecting multiple items of its type. This value function enhances (2.35)-(2.37) by considering the strengths of value dependencies. But, it assumes that choosing multiple items of a certain type equally impact the value of an item of the same type. Such impact nevertheless, can be either positive or negative. But the function only considers negative impacts.

The function further ignores that choosing items of other types may also impact the values of the items of a certain type. Again, (2.38)-(2.40) only capture the influences of items types on the values of items ignoring that the individual items also impact the values of each other to various degrees.

$$f_j(v_j, x_j, x_{d_j}) = k_{d_j} \sum_{i=0}^{x_j - 1} \alpha^i v_j \tag{2.38}$$

$$\begin{cases} k_{d_j} = 1 & \text{if } x_{d_j} > 0 \\ k_{d_j} = 0 & \text{if } x_{d_j} \leq 0 \end{cases} \tag{2.39}$$

$$x_j \in \{0, 1\}, \quad j = 1, ..., n \tag{2.40}$$

The third value function proposed by Burg *et al.* [110] is given in (2.41)-(2.43), where the value associated with x_j type j items is diminished by a factor of β when type d_j items are present in the solution. In particular, the value of type j items can be reduced to zero when $\beta = 1$. This value function enhances (2.38)-(2.40) by considering the impact of selecting items of a different type on the value of an item of a specific type. But, this value function does not consider positive impacts of item types on the values of items. Moreover, the function in its present from does not account for the impact of multiple item types on the value of an item.

$$f_j(v_j, x_j, x_{d_j}) = x_j v_j - k_{d_j} \beta x_j v_j \tag{2.41}$$

$$\begin{cases} k_{d_j} = 1 & \text{if } x_{d_j} > 0 \\ k_{d_j} = 0 & \text{if } x_{d_j} \leq 0 \end{cases} \tag{2.42}$$

$$x_j \in \{0, 1\}, \quad j = 1, ..., n \tag{2.43}$$

In all the three value functions the problem of not considering value dependencies among individual requirements can be solved by considering each requirement as a different type. But, that will exponentially increase the complexity of any optimization model that adopts such value functions. Due to all the problems discussed above, the optimization model of the OTKP is not suitable to requirement selection problem.

Chapter 3

The Integer Programming Method (DARS-IP)[1]

3.1 Introduction

Owing to budget constraints, it is hardly if ever feasible to satisfy the entire set of the requirements of a software project [16]. Therefore, requirement selection is inevitable to find an optimal subset of the requirements with the highest value while respecting the project constraints [80, 116, 117, 118, 119]. This problem, also known as the *Next Release Problem* (NRP) [16], is mathematically formulated by the *Binary Knapsack Problem* (BKP) [3, 82, 83].

Based on the BKP formula, the existing requirement selection works aim to maximize the *Accumulated Value* (AV) of an optimal subset of the requirements on the assumption that the value of an optimal subset is derived by accumulating the estimated values of the selected requirements [111]. However, several studies have argued that this assumption does not hold when interdependencies exist among requirements [120, 35, 111, 2].

The reason is that software requirements affect the values of each other due to the value dependencies among them [18, 120, 24]. As a result, the requirements excluded from the optimal subset may impact the values of the selected requirements. On the other hand, value dependencies can be of various strengths in the context of real-world projects [80, 20, 5].

[1]The contents of this chapter are presented in publications **(P1)** and **(P2)**.

In other words, values of requirements can weakly, moderately, or strongly depend on each other [27]. Therefore, it is important to consider both the existence and the strengths of the value dependencies [71, 111] when considering the impacts of the requirements on values of each other during a requirement selection.

However, the existing requirement selection methods either assume that requirements are independent [82, 83, 84, 85, 87] or they do not consider the strengths of those dependencies [16, 33, 89, 90, 91, 92, 93, 97, 100, 101, 102, 108, 106] thus either ignoring value dependencies or treating them as binary (0/1) relations by formulating the value dependencies as precedence constraints of the BKP formula.

It is clear that ignoring value dependencies is not acceptable as it may result in a value loss as discussed earlier. Hence, we focus on the possibility of treating value dependencies as binary relations to demonstrate the problems caused by such an approach.

When value dependencies are treated as binary relations and formulated as precedence constraints, excluding a requirement from the optimal subset may result in ignoring all requirements whose values depend on the excluded requirement even if the budget allows for their implementation [22]. This problem is referred to as the *Selection Deficiency Problem* (SDP) [2].

As a result of the SDP, any increase in the number of the dependencies results in a significant reduction in the accumulated value of the optimal subset of the requirements [22]. Hence, the SDP can severely impact the efficiency of the selection methods that ignore the strengths of value dependencies.

This chapter presents an *Integer Programming* (IP) method, presented in publication (**P1**), for dependency-aware requirement selection. The proposed method, referred to as DARS-IP, focuses on considering the impacts of value dependencies on the value of an optimal subset of the requirements during a selection process. We have achieved this through integrating the existence and the strengths of value dependencies into requirement selection. In doing so, we have made three main contributions. First, we have demonstrated the use of the algebraic structure of fuzzy graphs [43, 121, 42, 122] to model value dependencies and their strengths.

Second, we have presented an *Integer Programming* (IP) model for requirement selection which maximizes the *Overall Value* (OV) of an optimal subset while mitigating the selection deficiency problem [2]. The proposed model not only considers value dependencies among requirements but, more importantly, explicitly factors in the strengths of those dependencies. We have proposed overall value as an alternative to accumulated value to be used as the measure of optimality. The overall value of an optimal subset (selected requirements) considers the impacts of value dependencies on the value of that subset.

Finally, we have proposed mining preferences of (potential) users, as presented in (**P1**), to identify both the existence and the strengths of explicit value dependencies among requirements of a software project. Explicit value dependencies are used to infer implicit value dependencies based on the algebraic structure of fuzzy graphs.

The validity and practicality of the work are verified through carrying out several simulations and studying a real-world software. The results of our simulations as well as a real-world case study have consistently shown that: (a) the IP method of DARS (DARS-IP) can properly capture the strengths of value dependencies among requirements during a selection process while mitigating the selection deficiency problem (SDP); (b) DARS-IP always maximizes the overall value of an optimal subset; (c) maximizing the overall value and the accumulated value of an optimal subset can be conflicting objectives [123] as maximizing one may depreciate the other.

The remainder of this chapter is organized as follows. Section 3.2 gives the details of modeling value dependencies by fuzzy graphs. Section 3.3 introduces our proposed formulation of overall value of an optimal subset as well as our proposed integer programming model.

The results of our simulations as well as the studying of a real-world software project are discussed in Section 3.4. Section 3.5 then, presents a promising technique for the automated identification of value dependencies among software requirements. Finally, Section 3.6 concludes the chapter with a summary of the main contributions and future improvements/extensions.

3.2 Modeling Value Dependencies using Fuzzy Graphs

This section highlights our main reasons for choosing fuzzy graphs and then gives the details of employing fuzzy graphs for modeling value dependencies among software requirements.

3.2.1 Why Fuzzy Graphs?

Since their introduction in 1973 [124], fuzzy graphs have been widely adopted in decision making and expert systems [42] as they contribute to more accurate models by taking into account imprecision in real-world problems [124].

Fuzzy graphs have been demonstrated to be particularly useful in capturing the imprecision of dependency relations in software projects [47, 26]. Ngo-The *et al.*, exploited fuzzy graphs for modeling dependency satisfaction in release planning [47] and capturing imprecision of coupling dependencies among requirements [26].

Moreover, Wang *et al.* [27] adopted linguistic fuzzy terms to capture the variances of strengths of dependencies among software requirements. Hence, we use fuzzy graphs and their algebraic structure for modeling value dependencies among requirements and computing the influences of the requirements on the values of each other.

3.2.2 Fuzzy Requirement Interdependency Graphs

Based on the definition of fuzzy graphs [42], a *Fuzzy Requirement Interdependency Graph* (FRIG) is defined as a directed fuzzy graph $G = (R, D, \mu, \rho)$ in which a non-empty set of the identified requirements $R = \{r_1, ..., r_n\}$ constitute the graph nodes and a set of the (explicit) value dependencies $D = R \times R$ among the software requirements form the edges of the graph.

A dependency $(r_i, r_j) \in D$ means that the value of r_i explicitly depends on the selection of r_j. The membership function $\rho : R \times R \rightarrow [0, 1]$ denotes the strengths of explicit value dependencies (membership degrees of edges) in D. $\rho(x, y) = 0$ denotes the absence of an explicit dependency from x to y.

The fuzzy membership function μ specifies the membership degree of requirements in R. Requirements of a software are either identified and listed in its requirement set R or they are unidentified. Hence, we have $\forall r_i \in R : \mu(r_i) = 1$. Therefore, $G = (R, D, \mu, \rho)$ can be abbreviated as $G = (R, D, \rho)$.

On the other hand, for $G = (R, D, \mu, \rho)$ to be a fuzzy graph, the following condition must hold at all times. $\forall (x, y) \in D : \rho(x, y) \leq \mu(x) \wedge \mu(y)$, where $\wedge$ denotes fuzzy AND operator (taking infimum). In other words, for $\rho(x, y)$ to denote a fuzzy relation, the membership degree of a relation, also referred to as the strength of the relation (dependence) must not exceed the membership degree of either of the two elements. Theorem (3.1) shows that a FRIG always satisfies the condition of fuzzy graphs.

Proposition 3.1. If $G = (R, D, \rho)$ is a FRIG, $(\forall r_i \in R : \mu(r_i) = 1)$ then G always satisfies the condition $\forall (r_i, r_j) \in D : \rho(r_i, r_j) \leq \mu(r_i) \wedge \mu(r_j)$.

Proof. $G = (R, D, \rho)$ is a FRIG $\Rightarrow$

$a)$ $\forall r_i \in R : \mu(r_i) = 1 \Rightarrow \forall (r_i, r_j) \in D : \mu(r_i) \wedge \mu(r_j) = infimum(\mu(r_i), \mu(r_j)) = 1$,
$b)$ $\rho : R \times R \rightarrow [0, 1] \Rightarrow \forall r_i, r_j \in R, \rho(r_i, r_j) \leq 1$.

Therefore, $\forall (r_i, r_j) \in D : \rho(r_i, r_j) \leq \mu(r_i) \wedge \mu(r_j)$. □

Example 3.1. Consider the FRIG $E_1 = (R, D, \rho)$ in Figure 3.1 with $R = \{r_1, r_2, r_3, r_4\}$ and $D = \{(r_1, r_2), (r_2, r_3), (r_3, r_4), (r_4, r_2)\}$. The membership function ρ specifies the strengths of explicit dependencies in D as $\rho(r_1, r_2) = 0.6$, $\rho(r_2, r_3) = 0.4$, $\rho(r_3, r_4) = 0.8$, $\rho(r_4, r_2) = 0.2$. The dependency (r_1, r_2) specifies that the value of r_1 explicitly depends on r_2 and $\rho(r_1, r_2)$ gives the strength of the dependency (0.6).

Value dependencies in a FRIG can be either explicit or implicit. Explicit dependencies are identified by the edges of the graph whereas implicit dependencies are inferred from explicit dependencies. For instance, an implicit dependency (r_1, r_2, r_3) from r_1 to r_3 in Figure 3.1 is inferred from explicit dependencies (r_1, r_2) and (r_2, r_3).

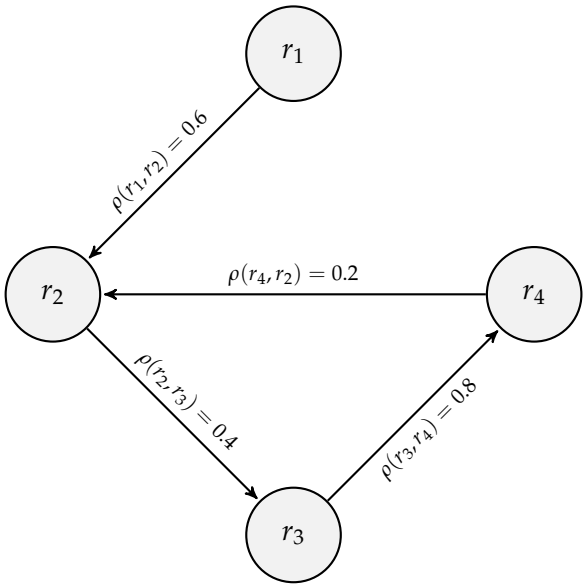


FIGURE 3.1: FRIG of Example 3.1

Definition 3.1. *Value Dependencies, and their Strengths.* Let $P = \{p_1, p_2, ..., p_m\}$ be the set of all value dependencies from node r_0 to node r_n in a FRIG $G = (R, D, \rho)$. A value dependency $p_i \in P$ is defined as a sequence of distinct nodes $(r_0, ..., r_n)$ such that $\rho(r_{i-1}, r_i) > 0$, where $1 \leq i \leq n$ and $n \geq 1$ is the length of the dependency.

The strength of a value dependency $p_i \in P$ is derived by (3.1) that is the strength of the p_i equals the strength of the weakest explicit value dependency (edge) in p_i.

$$\forall p_i = (r_0, ..., r_n) \in P,\ \rho(p_i) = \bigwedge_{j=1}^{n} \rho(r_{j-1}, r_j) \tag{3.1}$$

In a FRIG $G = (R, D, \rho)$ with m dependencies from a requirement r_0 to r_n, the overall strength of all dependencies from r_0 to r_n is denoted as $\rho^{\infty}(r_0, r_n)$ and calculated by (3.2). Based on (3.2), the overall strength of all dependencies from r_0 to r_n equals the strength of the strongest dependency among all the m dependencies from r_0 to r_n. It is clear that we have $\rho^{\infty}(r_0, r_n) = \rho(r_0, r_n)$ when there is no implicit value dependency from r_0 to r_n.

$$\rho^{\infty}(r_0, r_n) = \bigvee_{i=1}^{m} \rho(p_i) \tag{3.2}$$

To measure the level of value dependencies in a FRIG $G = (R, D, \rho)$ with n requirements ($|R| = n$) and k explicit value dependencies among those requirements ($|D| = k$), we define the *Level Of Interdependency* (LOI) as given in (3.3).

$$LOI(G) = \frac{k}{{}^{n}P_2}, {}^{n}P_2 = \frac{n!}{(n-2)!} \tag{3.3}$$

Example 3.2. For the FRIG E_1 in Example 3.1, with $n = 4$, $k = 4$ we have $LOI(E_1) = \frac{4}{{}^{4}P_2} = \frac{4}{12} = 0.33$.

It is also worth mentioning that requirements of software projects may negatively influence the values of each other. For instance, a negative dependency from a requirement r_i to a requirement r_j means that the value of r_i will be depreciated if r_j is selected with r_i in an optimal subset. Such negative dependency nonetheless can be modeled as a positive dependency from r_i to $\bar{r}_j$ where $\bar{r}_j$ denotes ignoring r_j (excluding r_j from an optimal subset). Hence, FRIGs can capture both positive and negative value dependencies. Nevertheless, in this chapter we only focus on positive dependencies for the sake of simplicity.

3.3 Integrating Value Dependencies into Selection

This section gives the details of integrating value dependencies into requirement selection in the IP method of DARS (DARS-IP). To achieve this we present a measure of value, referred to as *Overall Value*, which accounts for the impacts of value dependencies on the economic value of a requirement subset. We further present the integer programming model of the DARS-IP method, which optimizes the overall value of a selected subset of requirements.

3.3.1 Overall Value of an Optimal Subset

During a selection process some of the requirements of a software project may be excluded from the optimal subset. Due to the value dependencies among the requirements however, the excluded requirements may impact the values of the selected requirements that depend on them. Equation (3.4) captures these impacts. For a FRIG $G = (R, D, \rho)$, $O = \{o_1, ..., o_m\}$ and $\tilde{O} = \{\tilde{o}_1, ..., \tilde{o}_k\}$ denote the selected and excluded requirements respectively such that $O \subseteq R, \tilde{O} \subseteq R : O \cap \tilde{O} = \emptyset, O \cup \tilde{O} = R$.

For all $o_i \in O$, I_i denotes the impact of the excluded requirements $\tilde{o}_j \in \tilde{O}$ on the value of o_i. This impact is computed by taking supremum (fuzzy OR operator $\vee$) over the strengths of all dependencies from o_i to the excluded requirements in $\tilde{O}$. For an excluded requirement $\tilde{o}_j \in \tilde{O}$ then, the overall strength of all dependencies from o_i to $\tilde{o}_j$ is given by $\rho^{\infty}(o_i, \tilde{o}_j)$, which specifies the extent to which the value of o_i relies on the selection of $\tilde{o}_j$ (through all dependency paths from o_i to $\tilde{o}_j$) as derived by (3.2).

$$\forall o_i \in O,\ \forall \tilde{o}_j \in \tilde{O} : I_i = \bigvee_{j=1}^{k} (\rho^{\infty}(o_i, \tilde{o}_j)) \tag{3.4}$$

$$v'_i = v_i(1 - I_i) \tag{3.5}$$

$$OV = \sum_{o_i \in O} v'_i = \sum_{o_i \in O} v_i(1 - I_i) \tag{3.6}$$

As discussed earlier, the accumulated value (AV) of an optimal subset O is derived by accumulating the estimated values of the selected requirements : $\sum_{o_i \in O} v_i$, where v_i denotes the estimated value of o_i. The overall value of O, denoted by OV, on the other hand, is derived by accumulating the overall values of the selected requirements as computed by (3.6). The overall value of O ($\sum_{o_i \in O} v'_i$) captures the impacts of value dependencies as the overall value of each requirement $o_i \in O$ (v'_i) captures the impacts of value dependencies on the value of that requirement as given by (3.5).

3.3.2 The Integer Programming Model of the DAR-IP Method

As discussed earlier, the optimization models of the BK and PCBK methods aim to maximize the accumulated value of a requirement subset while ignoring value dependencies. In contrast, the optimization model of the DARS-IP method maximizes the overall value of a requirement subset by factoring in the impacts of requirements on the values of the selected requirements. The IP model of the DARS-IP method is given by (3.7)-(3.10). The model is single-objective.

In these equations, v_i and c_i denote the estimated value and cost of a requirement r_i respectively. Also, the boolean decision variable x_i specifies whether r_i is selected ($x_i = 1$) or ignored ($x_i = 0$). Moreover, $0 \leq I_i \leq 1$ is a real number specifying the impact of the ignored requirements ($x_j = 0$) on the value of a selected requirement. Constraint (3.8) in the optimization model of DARS-IP ensures that the total cost of the requirements does not exceed the budget limit b.

$$\text{Maximize} \sum_{i=1}^{n} v_i x_i (1 - I_i) \tag{3.7}$$

$$\text{Subject to} \sum_{i=1}^{n} c_i x_i < b \tag{3.8}$$

$$0 \leq I_i \leq 1, \qquad i = 1, ..., n \tag{3.9}$$

$$x_i = \{0, 1\}, \qquad i = 1, ..., n \tag{3.10}$$

The optimization model of DARS-IP, as given by (3.7)-(3.10), is convex [39] and therefore can be efficiently solved by the existing commercial solvers such as *IBM CPLEX* [40]. We have implemented, solved, and tested the optimization model of the DARS-IP method using the *Concert Technology* and the *JAVA API of IBM CPLEX* [40]. The code for this model is available in *JAVA* and *OPL* languages and can be obtained from the website of DARS[9].

[9] http://bcert.org/projects/dars

3.3.3 Examples of Requirement Selection

This section provides examples of requirement selection using the BK, PCBK, and DARS-IP methods.

Example 3.3. Let $G_p = (R, D, \rho)$ be a FRIG of a software project (Figure 3.2) with requirement set $R = \{r_1, r_2, r_3, r_4\}$ and explicit value dependencies D as in Figure 3.2 with strengths of $\rho(r_1, r_2) = 0.4$, $\rho(r_1, r_3) = 0.8$, $\rho(r_2, r_4) = 0.3$, $\rho(r_3, r_1) = 0.8$, $\rho(r_3, r_2) = 0.6$, $\rho(r_3, r_4) = 0.8$, and $\rho(r_4, r_3) = 0.2$. The costs and values of the requirements are specified by $C = \{c_1 = 10, c_2 = 10, c_3 = 15, c_4 = 10\}$ and $V = \{v_1 = 20, v_2 = 10, v_3 = 50, v_4 = 10\}$ respectively.

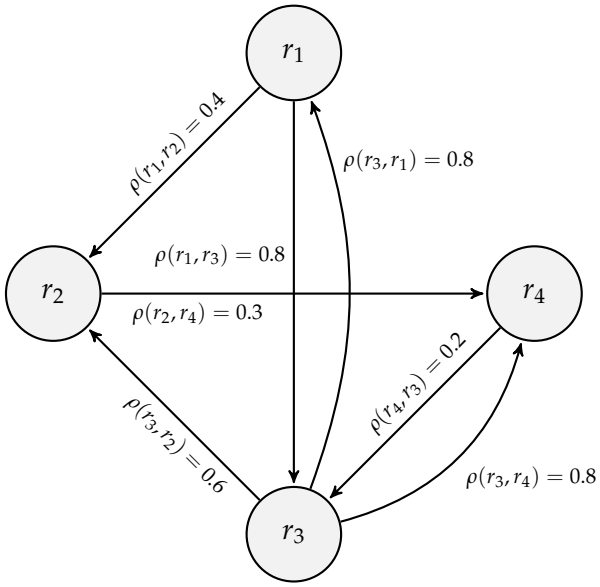


FIGURE 3.2: FRIG of Example 3.3 (numbers are hypothetical).

TABLE 3.1: Overall strengths of the value dependencies in Example 3.3

$\rho^\infty(x, y)$	r_1	r_2	r_3	r_4
r_1	1.0	0.6	0.8	0.8
r_2	0.2	1.0	0.2	0.3
r_3	0.8	0.6	1.0	0.8
r_4	0.2	0.2	0.2	1.0

TABLE 3.2: Accumulated values, overall values, and accumulated costs of the requirement subsets of Example 3.3

Subset	*AC*	*AV*	*OV*	*Subset*	*AC*	*AV*	*OV*
$s_0 = \{\}$	0	0	0	$s_8 = \{r_2, r_3\}$	25	60	17
$s_1 = \{r_1\}$	10	20	4	$s_9 = \{r_2, r_4\}$	20	20	16
$s_2 = \{r_2\}$	10	10	7	$s_{10} = \{r_3, r_4\}$	25	60	18
$s_3 = \{r_3\}$	15	50	10	$s_{11} = \{r_1, r_2, r_3\}$	35	80	21
$s_4 = \{r_4\}$	10	10	8	$s_{12} = \{r_1, r_2, r_4\}$	30	40	20
$s_5 = \{r_1, r_2\}$	20	30	11	$s_{13} = \{r_1, r_3, r_4\}$	35	80	36
$s_6 = \{r_1, r_3\}$	25	70	14	$s_{14} = \{r_2, r_3, r_4\}$	35	70	26
$s_7 = \{r_1, r_4\}$	20	30	12	$s_{15} = \{r_1, r_2, r_3, r_4\}$	45	90	90

Example 3.4. Consider finding the optimal subset of requirements by the BK method. Among all subsets of R in Table 3.2, the BK method recommends $s_6 = \{r_1, r_3\}$ as the optimal subset with the highest accumulated value of $AV = 70$ and the accumulated cost of $AC = 25$. Therefore we have $O = \{r_1, r_3\}$, $\tilde{O} = \{r_2, r_4\}$. In order to compute the overall value of the optimal subset, we first calculate the impacts of excluded requirements on the values of selected requirements based on (3.4). The impacts are calculated based on the overall strengths of the value dependencies in Table 3.1.

The overall strengths of dependencies are calculated by (3.2) as explained earlier. For instance, to compute the overall strength of the dependency from r_4 to r_2, dependencies $p_1 = (r_4, r_3, r_2)$ and $p_2 = (r_4, r_3, r_1, r_2)$ need to be considered. Based on (3.2), the overall strength of the dependency is computed as: $\rho^{\infty}(r_4, r_2) = \vee((\rho(r_4, r_3) \wedge \rho(r_3, r_2)), (\rho(r_4, r_3) \wedge \rho(r_3, r_1) \wedge \rho(r_1, r_2))) = \vee((0.2 \wedge 0.6), (0.2 \wedge 0.8 \wedge 0.4)) = \vee(0.2, 0.2) = 0.2$. The impacts then can be computed as $I_1 = \vee(\rho^{\infty}(r_1, r_2), \rho^{\infty}(r_1, r_4)) = 0.8$, $I_3 = \vee(\rho^{\infty}(r_3, r_2), \rho^{\infty}(r_3, r_4)) = 0.8$. Finally, the overall value of the optimal subset $O = \{r_1, r_3\}$ is calculated as $OV = v_1 \times (1 - I_1) + v_3 \times (1 - I_3) = 20 \times 0.2 + 50 \times 0.2 = 14$ which is less than the overall value of s_{10}. Therefore, the BK method does not necessarily maximize the overall value of an optimal subset.

Example 3.5. Consider finding the optimal subset of requirements in Example 3.4 using the PCBK method, which finds a subset of requirements with the highest AV respecting the budget ($AC \leq 25$) and the precedence constraints among the requirements. We can derive a precedence constraints set $PCS = \{x_1 \leq x_2, x_1 \leq x_3, x_2 \leq x_4, x_3 \leq x_1, x_3 \leq x_2, x_3 \leq x_4, x_4 \leq x_3\}$ from the dependency set D of G_p. Obviously there

is only one case that simultaneously satisfies both PCS and $AC \leq 25$, which is the empty set $s_0 = \{\}$ ($x_1 = x_2 = x_3 = x_4 = 0$) in Table 3.2 with $AV = OV = 0$. Hence, none of the requirements can be implemented even though the budget is available for implementing some of them. This is due to the selection deficiency problem (SDP) as discussed before.

Example 3.6. Consider finding the optimal subset of requirements in Example 3.4 using the DARS-IP, which finds a subset of requirements with the highest overall value while respecting $AC \leq 25$. To do so, we first calculate the OV of all subsets of R (steps of calculation were demonstrated for s_6 in Example 3.4) as listed in Table 3.2. Among all subsets of the requirements, $s_{10} = \{r_3, r_4\}$ gives the highest overall value of $OV = 18$ while the accumulated cost is within the budget, that is $(AC \leq 25)$. Therefore, $s_{10} = \{r_3, r_4\}$ will be selected as the optimal subset. s_{10} however, is not giving the maximum accumulated value. s_6 for instance, provides a higher AV.

3.4 Validation

Validity and practicality of the integer programming (IP) method of DARS are verified through carrying out several simulations (numerical studies) and studying a real-world software project. In this regard the following research questions have been answered about the proposed IP method.

(**RQ1**) What is the impact of using DARS-IP on the overall value of software products?

(**RQ2**) What is the relationship between maximizing the accumulated value and overall value of software products?

(**RQ3**) How effective is DARS-IP in mitigating the selection deficiency problem?

(**RQ4**) What is the impact of value dependencies on the performance of DARS-IP?

(**RQ5**) How practical is DARS-IP for software projects?

3.4.1 Simulations (Numerical Studies)

Simulation Design

We compared the performance of DARS-IP against those of the BK and PCBK methods through carrying out simulations on requirements from two classic requirement sets [125, 82, 83] from real-world projects of Ericssons *Radio Access Network* (RAN) and *Performance Management Recording* (PMR) with 14 and 11 requirements respectively. The estimated values and costs of the requirements of the RAN and PMR projects are listed in Table 3.3.

TABLE 3.3: Estimated values and costs of requirements for RAN and PMR

Requirements	RAN		PMR	
	Value	**Cost**	**Value**	**Cost**
r_1	12	1	0	6
r_2	6	2	6	5
r_3	5	3	3	6
r_4	7	4	11	19
r_5	12	6	32	28
r_6	16	11	20	4
r_7	3	4	9	5
r_8	3	6	4	7
r_9	4	7	25	10
r_{10}	5	12	9	3
r_{11}	1	4	3	8
r_{12}	1	6	–	–
r_{13}	21	23	–	–
r_{14}	3	10	–	–

Simulation has been widely used for the purpose of evaluation in system analysis [126] as well as the studies concerning requirement dependencies [27, 22]. Chen et al. [22] for instance, proposed simulating requirement dependencies for analyzing the performance of requirement selection methods. However, to the best of our knowledge, there is no work in the existing literature which has studied the distribution of the strengths of dependencies among software requirements. Hence, we simulate the strengths of explicit value dependencies with uniformly distributed random numbers in $[0, 1]$ generated by the *nextDouble() Method* of the *Class Random* in Java [127].

The simulation process starts with construction of a fuzzy requirement interdependency graph (FRIG) with randomly generated strengths of explicit value dependencies (edges of the graph) for a given level of interdependency (LOI$\in [0,1]$). A range of budgets (Budget$= \{1,2,...,120\}$) will then be specified to examine the performance of the selection methods in the presence of various budget constraints.

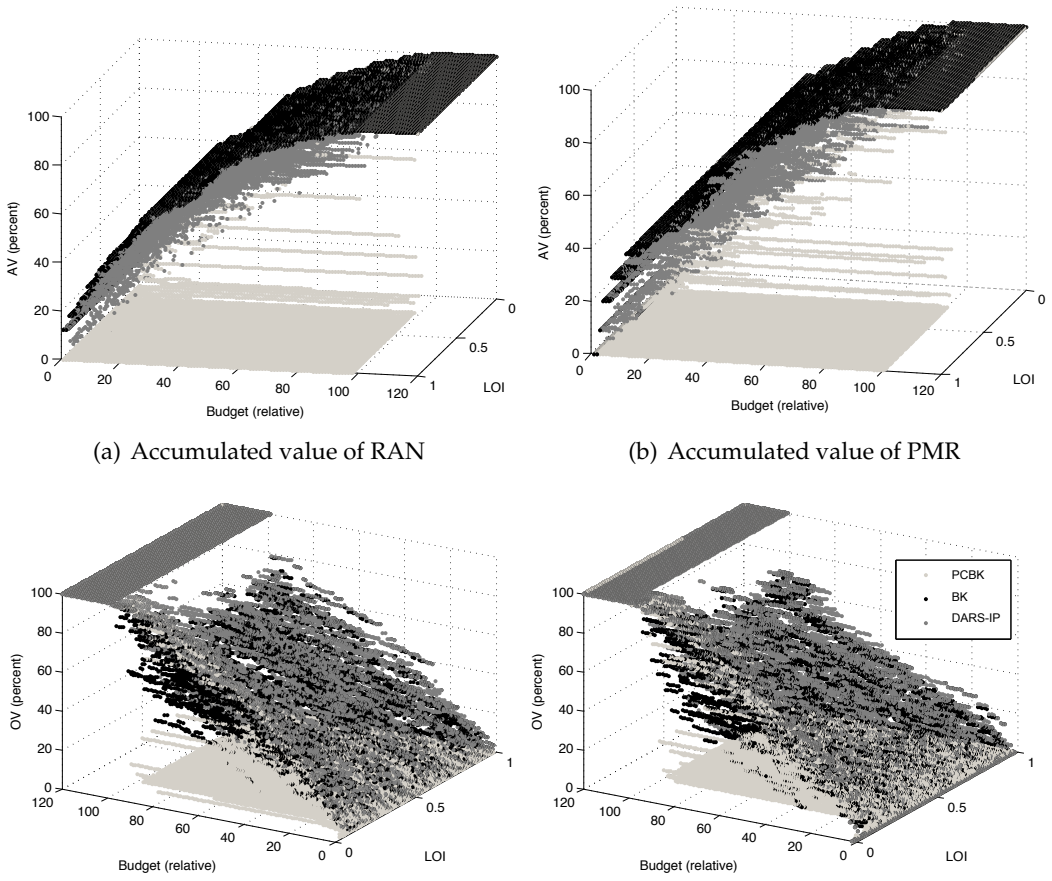
At the end of each simulation, an optimal subset of requirements will be generated by each of the selection methods. Then the accumulated value and the overall value of each optimal subset will be calculated and compared against those of the other selection methods. The simulation will be repeated for different levels of interdependency (LOIs) among requirements. Simulations are carried out using the callable library ILOG CPLEX 12.6.2 on a Windows machine with Core i7-2600 3.4 GHz processor and 16 GB of RAM.

Simulation Results

Figure 3.3 shows the results of our simulations. The x and y axes show the available budget (Budget$\in \{1,...,120\}$) and the level of interdependency (LOI $\in \{0,0.1,...,1\}$) respectively. The z axis shows the percentage of the accumulated value (overall value) of the optimal subset, which is the ratio of AV (OV) to the total accumulated value of the requirements multiplied by 100.

Our simulation results consistently showed that the BK method maximized the accumulated value (AV) while DARS-IP maximized the overall value (OV) of optimal subsets. Nevertheless, none of these methods simultaneously maximized both AV and OV for an optimal subset. In other words, maximizing AV and OV demonstrated to be conflicting objectives. This answers **(RQ1)** and **(RQ2)**.

The results of our simulations also showed (Figure 3.3) that the effectiveness of the PCBK method was severely impacted by the selection deficiency problem (SDP). In other words, the PCBK method generated the lowest AV/OV unless in the presence of a sufficient budget (Budget $\rightarrow$ 120) and/or a negligible level of interdependency (LOI $\rightarrow$ 0). For LOI $>$ 0.25 in both RAN and PMR requirement sets, almost no AV/OV was



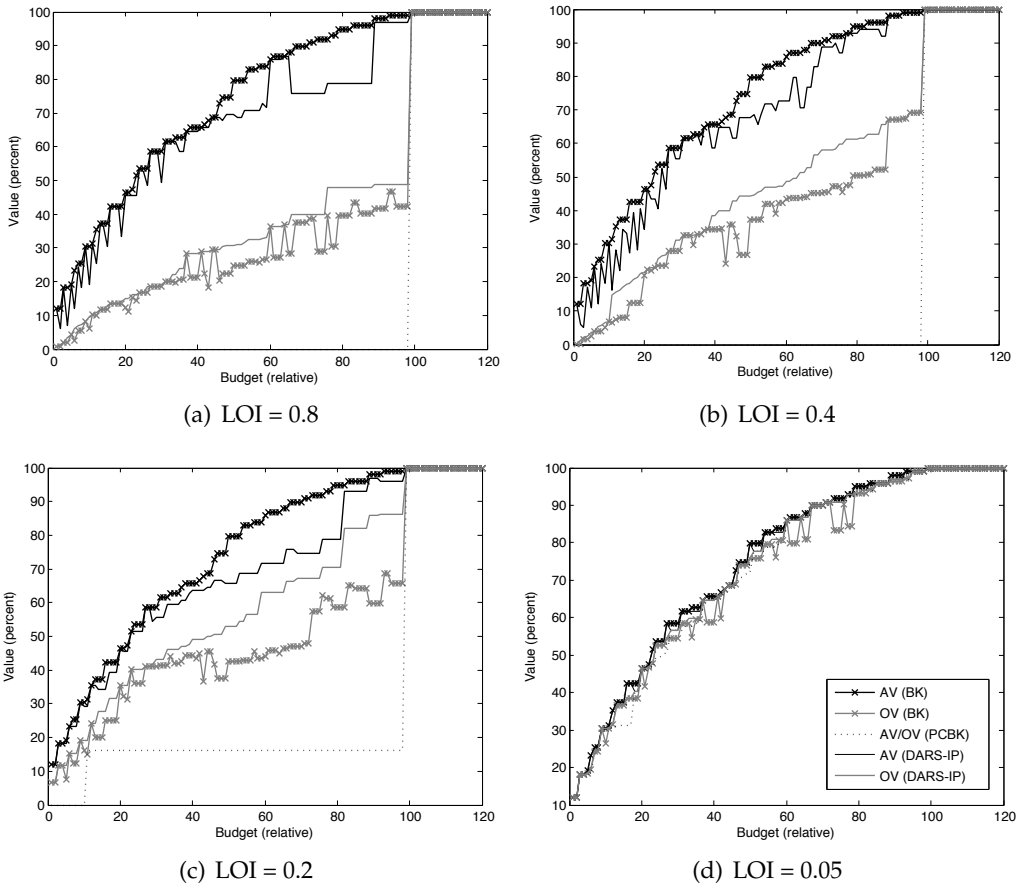
(a) Accumulated value of RAN

(b) Accumulated value of PMR

(c) Overall Value of RAN

(d) Overall value of PMR

FIGURE 3.3: Accumulated and overall values of RAN and PMR.



(a) LOI = 0.8 (b) LOI = 0.4

(c) LOI = 0.2 (d) LOI = 0.05

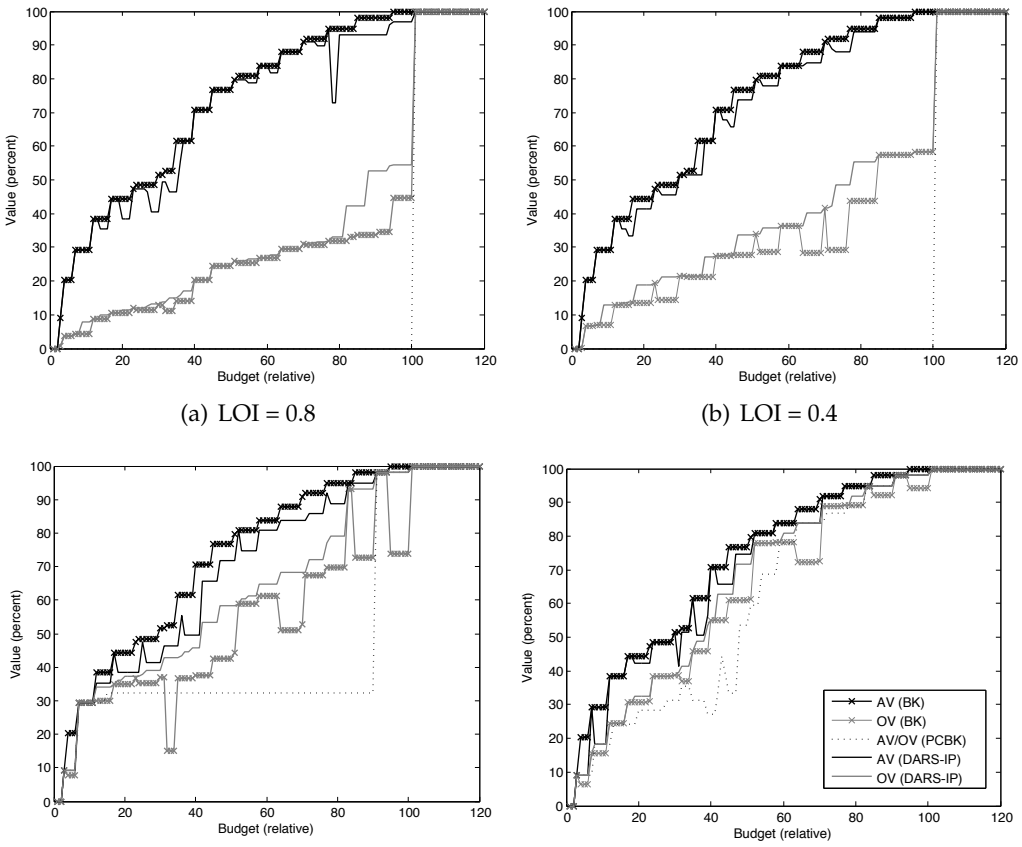
FIGURE 3.4: Sample simulation results for RAN requirements.

achieved by the PCBK method unless budget was available for all of the requirements ($b = \sum_{i=1}^{14} c_i = 99$ for RAN).

It was, moreover, observed (Figure 3.3) that the DARS-IP mitigated the impact of the SDP through considering the strengths of value dependencies. This answers **(RQ3)**. The BK method however was not subject to the SDP as it completely ignored dependencies among requirements.

We further observed (Figure 3.3) that all of the selection methods performed equally well when budget was available for all of the requirements to be implemented (Budget$\geq$ 99 for RAN and Budget $\geq$ 101 for PMR) or requirements were mutually independent.

Figure 3.4 and Figure 3.5 compare AV/OV achieved by the simulated selection methods for various levels of interdependencies among requirements of RAN and PMR respectively. A dependency level of LOI $=$ 0.8 implies that 80% of the explicit value



(a) LOI = 0.8 (b) LOI = 0.4

(c) LOI = 0.2 (d) LOI = 0.05

FIGURE 3.5: Sample simulation results for PMR requirements.

dependencies have non-zero strengths. The horizontal axis shows the available budget and the vertical axis shows the percentage of the achieved AV/OV.

In almost every simulation, it was observed that for a given optimal subset O, AV of O was smaller or equal to the OV of O. This is due to the fact that the overall value of an optimal subset considers the impacts of value dependencies on the values of requirements whereas the accumulated value of an optimal subset accumulates the estimated values of selected requirements without considering value dependencies.

It was further observed that the gap between the overall value of an optimal subset and its corresponding accumulated value (| AV-OV |) increased as the level of interdependency (LOI) grew. The reason is that increasing the LOI increases the chances that selected requirements explicitly depend on the excluded requirements which generally results in decreasing the overall value of the optimal subset (answer to **(RQ4)**). The PCBK method, however, avoids choosing a requirement without its dependencies being selected. Therefore, we have AV = OV for the PCBK method.

3.4.2 Case Study

To demonstrate the practicality of the DARS-IP and answer **(RQ5)**, we performed selection for 23 requirements of a messaging software product referred to as the *Precious Messaging System* (PMS). We employed 5 stakeholders to estimate [90] the costs and values of the requirements of the PMS. Each requirement r_i was assigned an estimated cost of $c_i \in [1, 20]$ and an estimated value of $v_i \in [1, 20]$ by different stakeholders. c_i and v_i are real numbers.

Stakeholders then performed pairwise comparisons among requirements [21] to identify explicit value dependencies and estimate the strengths of those dependencies. A dependency (r_i, r_j) was assigned a strength of $\rho(r_i, r_j) \in [0, 1]$ where $\rho(r_i, r_j) = 0$ and $\rho(r_i, r_j) = 1$ denoted no dependency and a full dependency from r_i to r_j respectively.

The median of the estimated costs/values for each requirement r_i was then computed to account for the different opinions of the stakeholders. In a similar way, for each

explicit value dependency (r_i, r_j) the median of the 5 estimated strengths of that dependency was computed to specify the strength of (r_i, r_j). Median was used as it is less affected by the extreme opinions of stakeholders compared to the arithmetic mean.

Table 3.4 lists the estimated costs and values of the requirement of the PMS as well as the strengths of explicit value dependencies among those requirements. The *Dependency Vector* of a requirement r_i in Table 3.4 denotes the strengths of explicit value dependencies from r_i to other requirements of the PMS. Based on Table 3.4 and (3.3), the level of interdependency is calculated for the requirements of the PMS as follows. $LOI(PMS) = \frac{113}{^{23}P_2} \approxeq 0.22$.

TABLE 3.4: Estimated values, costs, and strengths of explicit value dependencies.

ID	Value	Cost	Dependency Vector $\{r_1, ..., r_{23}\}$
r_1	20	10	{0.0, 0.0, 0.0, 0.5, 0.3, 0.0, 0.6, 0.4, 0.0, 0.0, 0.0, 0.7, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1, 0.0, 0.3}
r_2	20	7	{1.0, 0.0, 0.0, 0.6, 0.6, 0.0, 0.6, 0.6, 0.0, 0.3, 0.3, 0.7, 0.0, 0.3, 0.0, 0.7, 0.0, 0.0, 0.0, 0.0, 0.2, 0.0, 0.8}
r_3	6	1	{1.0, 0.0, 0.0, 0.5, 0.3, 0.0, 0.6, 0.0, 0.0, 0.0, 0.0, 0.7, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}
r_4	17	10	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9, 0.0, 0.0, 0.4, 0.0, 0.7, 0.0, 0.0, 0.0, 0.0, 0.2, 0.0, 0.0}
r_5	3	12	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}
r_6	20	20	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.6, 0.0, 0.3, 0.3, 0.4, 0.0, 0.0, 0.0, 0.0, 0.7, 0.4, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8}
r_7	15	6	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}
r_8	8	14	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}
r_9	20	15	{0.0, 0.7, 0.0, 0.0, 0.0, 0.7, 0.0, 0.3, 0.0, 0.0, 0.8, 0.2, 0.4, 0.0, 0.2, 0.7, 0.0, 0.0, 0.0, 0.0, 0.2, 0.0, 0.0}
r_{10}	16	10	{0.0, 0.7, 0.0, 0.0, 0.3, 0.7, 0.0, 0.3, 0.0, 0.0, 0.0, 0.3, 0.4, 0.0, 0.0, 0.2, 0.0, 0.0, 0.0, 0.0, 0.2, 0.6, 0.0}
r_{11}	20	4	{0.0, 0.0, 0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.4, 0.0, 0.0, 0.0, 0.2, 0.0, 0.0}
r_{12}	10	6	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.5, 0.8, 0.0, 0.0, 0.0, 0.1, 0.4, 0.0, 0.0, 0.0, 0.0, 0.0, 0.4, 0.0}
r_{13}	8	5	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.1, 0.4, 0.0, 0.0, 0.0, 0.0, 0.1, 0.6, 0.0}
r_{14}	5	12	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0}
r_{15}	8	15	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0}
r_{16}	10	3	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}
r_{17}	15	12	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0}
r_{18}	10	3	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}
r_{19}	20	20	{1.0, 0.3, 0.0, 0.7, 0.5, 1.0, 0.6, 0.5, 1.0, 0.6, 0.4, 0.0, 0.0, 0.1, 0.8, 0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8}
r_{20}	20	20	{1.0, 0.3, 0.0, 0.7, 0.5, 1.0, 0.6, 0.5, 1.0, 0.6, 0.4, 0.0, 0.0, 0.1, 0.8, 0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8}
r_{21}	15	12	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.1, 0.8, 0.2, 0.0, 0.0, 0.0, 0.0, 0.3, 0.0}
r_{22}	20	15	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}
r_{23}	20	10	{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}

Based on the estimations provided by the stakeholders, the FRIG of the PMS was constructed (Figure 3.6) and selections were performed using the DARS-IP as well as the BK and PCBK methods. Requirement selections were performed for various ranges of budgets ($Budget \in \{1, ..., 260\}$) to examine the performance of the selection methods and answer the following research questions.

TABLE 3.5: Solution vectors and their corresponding overall value (OV) provided by the different selection methods in the presence of various budget constraints. A selection variable x_i denotes whether requirement r_i is selected ($x_i = 1$) or otherwise ($x_i = 0$).

Budget	Selection Model	Overall Value (percent)	Solution Vector $\{x_1, ..., x_{23}\}$
16	BK	5.21	{0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0}
	PCBK	10.74	{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1}
	DARS-IP	12.88	{0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0}
46	BK	23.25	{1,1,1,0,0,0,1,0,0,0,1,0,0,0,0,1,0,1,0,0,0,0,1}
	PCBK	19.94	{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1}
	DARS-IP	26.63	{0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,1,0,1,0,0,0,1,1}
71	BK	31.07	{1,1,1,1,0,0,1,0,0,1,1,1,0,0,0,1,0,1,0,0,0,0,1}
	PCBK	19.94	{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1}
	DARS-IP	34.60	{1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,1,0,1,0,0,0,1,1}
76	BK	32.06	{1,1,1,1,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,0,0,1}
	PCBK	19.94	{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1}
	DARS-IP	35.74	{1,1,1,1,0,0,1,0,0,0,1,1,0,0,0,1,0,1,0,0,0,1,1}
81	BK	31.90	{1,1,1,1,0,0,1,0,0,1,1,0,1,0,0,1,1,1,0,0,0,0,1}
	PCBK	19.94	{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1}
	DARS-IP	37.98	{0,0,0,1,0,0,0,0,0,0,1,0,0,1,0,1,1,1,0,0,1,1,1}
141	BK	44.11	{1,1,1,1,0,0,1,0,1,1,1,1,1,0,0,1,1,1,0,1,0,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	59.45	{1,1,1,1,0,0,1,0,0,1,1,1,1,1,1,1,1,1,0,0,1,1,1}
146	BK	45.40	{1,1,1,1,0,0,1,0,1,1,1,1,0,0,0,1,1,1,0,1,1,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	60.43	{1,1,1,1,0,0,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1}
151	BK	46.87	{1,1,1,1,0,0,1,0,1,1,1,1,1,0,0,1,1,1,0,1,1,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	62.27	{1,1,1,1,0,1,1,0,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1}
156	BK	46.87	{1,1,1,1,0,0,1,0,1,1,1,1,1,0,0,1,1,1,0,1,1,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	62.27	{1,1,1,1,0,1,1,0,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1}
161	BK	50.12	{1,1,1,1,0,1,1,0,1,1,1,1,1,0,0,1,1,1,0,1,0,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	64.23	{1,1,1,1,0,1,1,0,0,1,1,1,1,1,1,1,1,1,0,0,1,1,1}
166	BK	51.41	{1,1,1,1,0,1,1,0,1,1,1,1,0,0,0,1,1,1,0,1,1,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	64.72	{1,1,1,1,0,1,1,0,1,0,1,1,1,1,1,1,1,1,0,0,1,1,1}
171	BK	52.88	{1,1,1,1,0,1,1,0,1,1,1,1,1,0,0,1,1,1,0,1,1,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	64.72	{1,1,1,1,0,1,1,0,1,0,1,1,1,1,1,1,1,1,0,0,1,1,1}
176	BK	52.88	{1,1,1,1,0,1,1,0,1,1,1,1,1,0,0,1,1,1,0,1,1,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	66.69	{1,1,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,0,1,1,1}
181	BK	51.35	{1,1,1,1,0,1,1,0,1,1,1,1,1,0,0,1,1,1,1,1,0,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	67.18	{1,1,1,1,0,1,1,1,1,0,1,1,1,1,1,1,1,1,0,0,1,1,1}
186	BK	52.64	{1,1,1,1,0,1,1,0,1,1,1,1,0,0,0,1,1,1,1,1,1,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	73.83	{1,1,0,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,0,1,1,1}
191	BK	54.11	{1,1,1,1,0,1,1,0,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	75.31	{1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,0,1,1,1}
196	BK	54.11	{1,1,1,1,0,1,1,0,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1}
	PCBK	53.37	{0,0,0,1,1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,1,1,1}
	DARS-IP	75.31	{1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,0,1,1,1}
246	BK	100.00	{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
	PCBK	100.00	{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
	DARS-IP	100.00	{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

Figure 3.7 summarizes the results of our experiments by comparing the accumulated values (AV) and/or overall values (OV) achieved by the selection methods. The horizontal axis shows the available budget ($Budget = \{1, ..., 260\}$) and the vertical axis shows the percentages of AV/OV. Table 3.5 lists some of the optimal subsets provided by the selection methods employed in the presence of various budget constraints.

Consistent with the simulations, the results of our case study demonstrated (Figure 3.7 and Table 3.5) that the BK method always maximized the accumulated value of the selected requirements (optimal subset) while the DARS-IP maximized the overall value of selected requirements. Moreover, maximizing the accumulated value and overall value of an optimal subset was demonstrated to be in conflict.

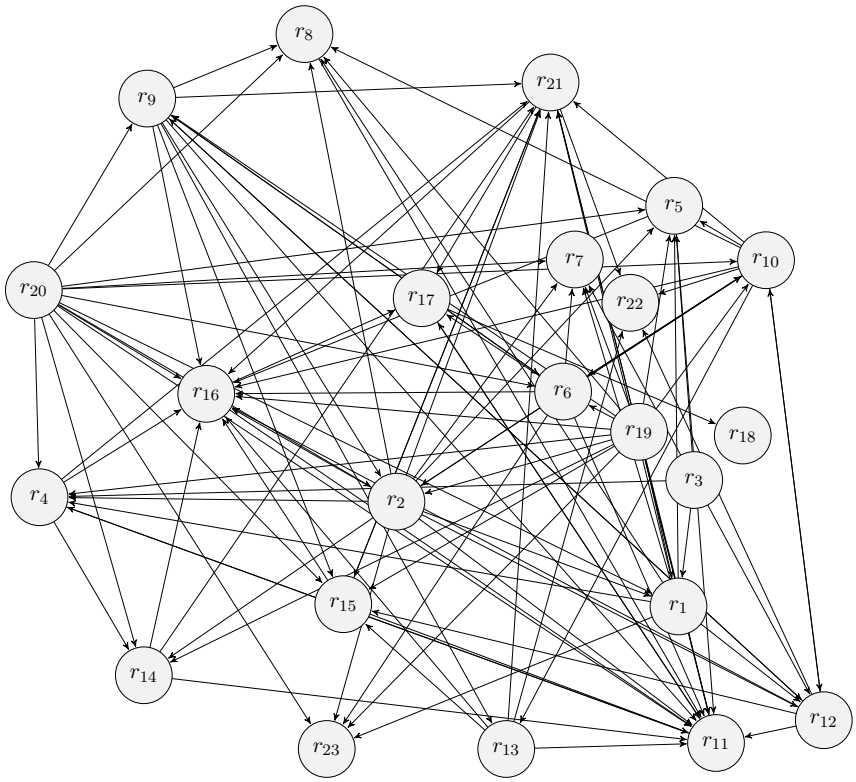


FIGURE 3.6: The FRIG of the PMS (Strengths of dependencies are not represented for the sake of readability).

Furthermore, the results of our experiments showed (Figure 3.7 and Table 3.5) that the DARS-IP mitigated the adverse impact of the selection deficiency problem (SDP) by considering the strengths of value dependencies while the efficiency of the PCBK method was negatively impacted by the SDP. For instance, we observed (Table 3.5)

that for $Budget = 81$, the overall value of the optimal subset provided by DARS-IP was almost twice as high as the overall value provided by the PCBK method. The BK method on the contrary, was not vulnerable to the SDP as it totally ignores dependencies among requirements.

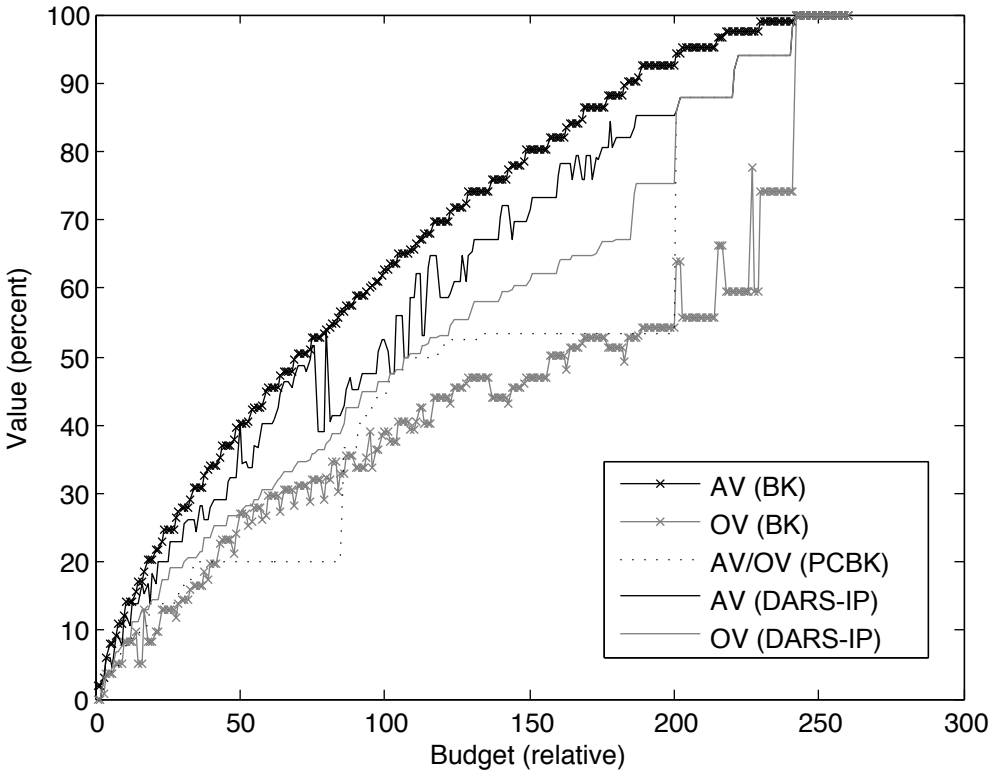


FIGURE 3.7: Selection results for the PMS (LOI $\approxeq$ 22%)

.

It is also worth mentioning that even though the DARS-IP method was applied to a real-world software, performing pairwise comparisons for identification of value dependencies demonstrated to be hard to achieve even for the relatively small requirement set used in this case study. Efficient techniques to identify value dependencies hence are essential to enhance the practicality of DARS-IP for large scale requirement sets. The following section will propose a technique for automated identification of value dependencies from user preferences.

3.5 Automated Identification of Explicit Value Dependencies

Automated identification of value dependencies and their strengths has not been discussed in the existing literature. Nonetheless, various techniques from the information retrieval and data mining domain [128] can be borrowed to assist such automation.

This section discusses one of the several possible approaches to automate the identification of value dependencies. Our proposed approach is based on mining the preferences of (potential) users of a software [129, 8] to identify both the existence and the strengths of explicit value dependencies among requirements of a software. It has been widely recognized that user preferences of software requirements can determine their values [129, 130] as highly preferred software requirements are more likely to be purchased/used by the (potential) users. In other words, users preferring a requirement r_j may also prefer a requirement r_i (with the probability $p(r_i|r_j)$). This is known as *Market Basket Analysis* or *Association Rule Mining* in the data mining domain [128]. An association from a requirement r_j to r_i (users preferring r_j will also prefer r_i) can also be interpreted as a causal relation [41] from r_j to r_i meaning that choosing r_j may cause choosing r_i by the users, thus enhancing the value made by selling r_i. As such, it is clear that a causal relation from r_j to r_i can also be interpreted as a value dependency from r_i to r_j (the value of r_i depends on preference of r_j by the users). Hence, association rule mining of user preference for requirements can be used for identification of value dependencies and the strengths of those dependencies. In this context, measures of causal strength can be used to estimate the strengths of value dependencies. One of the most commonly adopted measures of causal strength is *Pearl's Measure of Causal Strength* [41, 128, 131, 132, 45] which is denoted by $\eta_{i,j}$ in (3.11) and derived by $p(r_i|r_j)$. That is the chances that users that choose r_j also choose r_i. This can be used to estimate the strength of an explicit value dependency from r_i to r_j. Pearl's measure then can be mapped into a desired fuzzy membership function $\rho(r_i, r_j)$ (which gives the strengths of value dependencies in FRIGs) as demonstrated in Figure 3.8. Various membership functions could be explored for this mapping based on the preference of the stakeholders. For instance, the membership function of Figure 3.8(b) treats dependencies with causal strengths below 0.16 ($\eta_{i,j} < 0.16$) as not significant enough to

be considered while dependencies with $\eta_{i,j} \geq 0.83$ are treated as full dependencies of strength 1.

Such a membership function might be suitable for selection methods that formulate dependencies as precedence constraints. In such methods, it might be reasonable to consider a strong causal dependency (say $\eta_{i,j} \geq 0.95$) as a precedence relation rather than ignoring it. The PCBK method, which only captures precedence relations, hence can be improved this way by at least considering very strong value dependencies. But this may exacerbate the selection deficiency problem [2] as explained before if the threshold for strong dependencies is set too low. Figure 3.8(c) and Figure 3.8(d) depict other alternative membership functions which, unlike membership functions of Figure 3.8(a) and Figure 3.8(b), do not assume linearity for mapping $\eta_{i,j}$ to $\rho(r_i, r_j)$.

$$\eta_{i,j} = p(r_i|r_j) = \frac{p(r_i, r_j)}{p(r_j)}, \ \eta_{i,j} \in [0, 1] \tag{3.11}$$

User preferences for software requirements can be gathered in different ways [133,

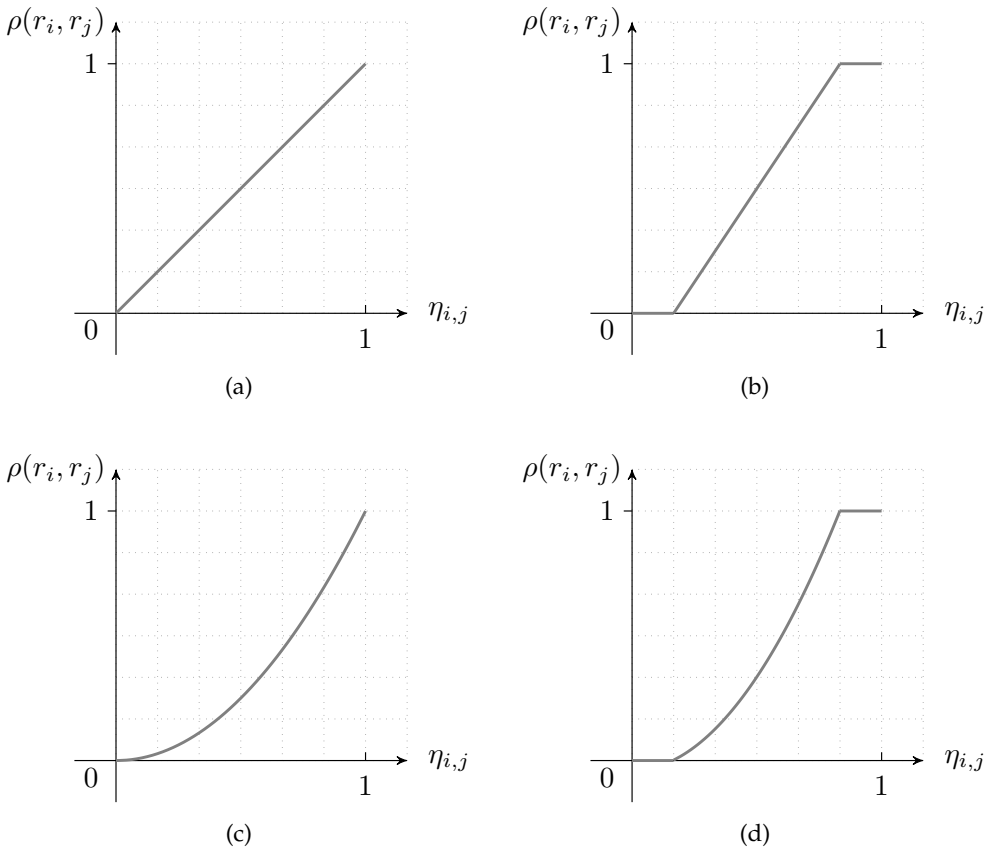


FIGURE 3.8: Sample mappings from $\eta_{i,j}$ to different membership functions $\rho(r_i, r_j)$.

134, 135] depending on the nature of a software release and the current state of a software. For the first release of a software, user preferences could be gathered by conventional market research approaches such as conducting surveys or referring to the user feedback or sales records of similar software products in the market. For future releases of a software, or when re-engineering of a software is of interest (e.g. for legacy systems) user feedback and sales records of the previous releases of the software might be used in combination with market research approaches to find user preferences. It is also worth mentioning that in cases where collecting user preferences in large quantities is difficult to achieve, re-sampling methods [136] could be used to automatically generate larger samples of user preferences from a relatively small sample while maintaining the characteristics of the initial sample [46].

$$
\mathrm{M}_{4\times 10} = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 & u_9 & u_{10} \end{array} \\
\left(\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1
\end{array}\right)
\end{array}
$$

FIGURE 3.9: A sample preference matrix

.

Definition 3.2. *Preference Matrix*. Let $R = \{r_1, ..., r_n\}$ be a requirement set and $U = \{u_1, ..., u_k\}$ be the list of users whose preference are gathered. A preference matrix $M_{n\times k}$ is a binary (0/1) matrix of size $n \times k$ where n and k denote the number of requirements and the number of users respectively. Each element $m_{i,j}$ specifies whether a user u_i has preferred a requirement r_j ($m_{i,j} = 1$) or not ($m_{i,j} = 0$). A sample preference matrix $M_{4\times 10}$ is shown in Figure 3.9.

Example 3.7. Matrix $E_{4\times 4}$ (Figure 3.10) gives Pearl's measure of causal strength computed for pairs of requirements in the preference matrix $M_{4\times 10}$ of Figure 3.9 based on (3.11). An element $\eta_{i,j}$ of $E_{4\times 4}$ denotes the causal strength of an explicit value dependence from r_i to r_j. For instance, we have $\eta_{1,3} = p(r_1|r_3) = \frac{p(r_1,r_3)}{p(r_3)} = \frac{0.2}{0.3} = 0.6667$.

$$
\mathrm{E}_{4\times4} = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}
\begin{array}{c}
\begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \end{array} \\
\begin{pmatrix}
1.0000 & 0.6000 & 0.6667 & 0.7143 \\
0.3750 & 1.0000 & 0.3333 & 0.5714 \\
0.2500 & 0.2000 & 1.0000 & 0.2857 \\
0.6250 & 0.8000 & 0.6667 & 1.0000
\end{pmatrix}
\end{array}
$$

FIGURE 3.10: Pearl measure for the preference matrix of Figure 3.9

.

3.6 Summary

This chapter presented an integer programming (IP) method, appeared in publications **(P1)** and **(P2)**, for dependency-aware requirement selection. The proposed method, i.e. DARS-IP, considers the impacts of value dependencies in software requirement selection. The method comprises three main components:

(i) *Identification of value dependencies*. We discussed the use of measures of causal strength and fuzzy membership functions to identify value dependencies and their strengths from user preferences;

(ii) *Modeling value dependencies*. We demonstrated the use of fuzzy graphs [42] and their algebraic structure [43] for modeling the strengths of value dependencies and capturing the imprecision associated with those dependencies;

(iii) *Integrating value dependencies into requirement selection*. We presented an integer programming model which maximizes the overall value (OV) of a selected subset of requirements, where the strengths of the value dependencies are taken into account.

The validity and practicality of the DARS-IP method are verified by carrying out simulations and studying a real-world software project. Our results show that: (a) our proposed integer programming method (DARS-IP) properly captures the strengths of value dependencies during a requirement selection while mitigating the selection

deficiency problem (SDP), (b) DARS-IP always maximizes the overall value of the selected requirements, and (c) maximizing the overall and the accumulated values of the selected requirements are in conflict as maximizing one may depreciate the other.

The DARS-IP method proposed in this chapter and its main components are improved by the ILP method of DARS (DARS-ILP), presented in Chapter 4, in several ways. First, the dependency identification technique in DARS-IP is enhanced by (a) considering both the strengths and qualities of value dependencies and (b) using a formal significance test to understand the accuracy of the value dependencies.

Second, the modeling technique proposed in DARS-ILP is extended to capture not only the strengths but also the qualities of value dependencies, thus allowing for reasoning about simultaneous positive and negative impacts of the explicit and implicit value dependencies among the requirements. In this regard, we have presented a modified version of the Floyd-Warshall algorithm capable of efficiently computing the positive and negative influences of the requirements on the values of each other using the algebraic structure of fuzzy graphs.

Finally, the DARS-ILP method integrates both positive and negative value dependencies into software requirement selection by taking into account the qualities of value dependencies in the optimization model of DARS-ILP. The optimization model of the DARS-ILP method is a linear model, which is scalable to software projects with large number of requirements. The computational time of the optimization model of DARS-ILP is discussed in detail in Section 4.7.

Chapter 4

The Integer Linear Programming Method (DARS-ILP) [1]

4.1 Introduction

Chapter 3 presented the integer programming method of DARS (DARS-IP), appeared in Publication **(P1)** [1], for considering value dependencies in software requirement selection. This chapter presents an integer linear programming[2] method of DARS (DARS-ILP), which extends/improves the main components of the DARS-IP method in several ways as follows.

The dependency identification is enhanced in DARS-ILP by (a) considering both the strengths and qualities of value dependencies and (b) using a formal significance test to understand the accuracy of the value dependencies. In this regard, we have contributed an automated dependency identification technique that uses the Eells measure of causal strength [45] to extract value dependencies from significant causal relations among user preferences. We have further demonstrated the use of a Latent Multivariate Gaussian model [46] to generate samples of user preferences when collecting sufficient data on user preferences is not practical [46].

Modeling value dependencies is enhanced in DARS-ILP by taking into account the qualities of value dependencies (positive or negative). The proposed modeling technique, thus, allows for reasoning about simultaneous positive and negative impacts

[1] The main results of this chapter are presented in publications **(P1)**-**(P8)**.

[2] An integer linear programming (ILP) problem is a linear program where the variables are restricted to be integers.

of the explicit and implicit value dependencies among the requirements. We have demonstrated the use of fuzzy graphs [42] for modeling the strengths and qualities of value dependencies. We have further presented a modified version of the Floyd-Warshall algorithm [44], which is capable of efficiently computing the positive and negative influences of the requirements on the values of each other using the algebraic structure of fuzzy graphs.

Finally, requirement selection is improved in DARS-ILP by integrating the qualities and the strengths of value dependencies into the ILP model of the DARS-ILP method. The ILP model of the DARS-ILP mitigates the risk of value loss posed by ignoring (selecting) the requirements with positive (negative) influences on the values of the requirements. The model is linear and scalable to projects with large number of requirements. We have further contributed a *Blind* ILP model for DARS-ILP, which aims to mitigate the risk of value loss posed by ignoring the positive influences of the requirement on the values of each other. The proposed Blind model does not require any information about value dependencies, thus it is suitable for the projects in which the identification of the value dependencies is not practical.

We show the practicality and validity of the DARS-ILP method by studying a real-world software project. We moreover, carry out extensive simulations to evaluate the effectiveness of the DARS-ILP method in providing higher overall value and mitigating the value loss in the presence of different levels of value dependencies, negative value dependencies, precedence dependencies, negative precedence dependencies, and budget. Finally, the scalability of DARS-ILP is investigated by applying the method to a real-world software project as well as carrying out simulations.

Our results show: that (a) compared to the requirement selection methods that ignore value dependencies, the ILP method of DARS provides higher overall value by mitigating the impact of ignoring (selecting) requirements with positive (negative) influence on the values of selected requirements; (b) maximizing the accumulated value and overall value of a software are conflicting objectives; and (c) DARS-ILP is scalable to software projects with large number of requirements for different levels of value dependencies and precedence dependencies among the requirements. This is demonstrated by simulating different scenarios for datasets of up to 3000 requirements.

4.2 Identification of Value Dependencies

This section presents an automated technique for identification of value dependencies based on causal relations among user preferences for requirements. We use the widely adopted Eells measure [45] of causal strength and the *Odds Ratio* [137] to identify the qualities and strengths of significant causal relations among requirements. A fuzzy membership function will then be used to estimate the strengths and qualities of value dependencies based on identified causal relations. Identified value dependencies will be used to identify implicit dependencies among requirements using the algebraic structure of fuzzy graphs and Algorithm 4.2 as will be discussed in Section 4.3.

4.2.1 Gathering User Preferences

User preferences can be gathered in different ways [133, 134, 135] depending on the nature of the release. For a new software product, preferences may be gathered by conventional market research techniques such as conducting surveys and/or mining user reviews/comments in social media and online stores [138]. User preferences may also be gathered by studying user preferences for features of similar software and/or their sales records.

When sales/usage records for the requirements of a software product are available, say from earlier versions, such information can be combined with market research results to estimate user preferences for a newer version of software. This is particularly suitable for reengineering a software or releasing different configurations in a software product line. We capture user preferences by a *Preference Matrix* as given by Definition 4.1, which is a restatement of Definition 3.2.

Definition 4.1. (restatement of Definition 3.2). *Preference Matrix*. Let $R = \{r_1, ..., r_n\}$ be a requirement set and $U = \{u_1, ..., u_k\}$ be the list of users whose preference are gathered. A preference matrix $M_{n\times k}$ is a binary (0/1) matrix of size $n \times k$ where n and k denote the number of requirements and the number of users respectively. Each element $m_{i,j}$ specifies whether a user u_i has preferred a requirement r_j ($m_{i,j} = 1$) or not ($m_{i,j} = 0$). A sample preference matrix $M_{4\times 20}$ is shown in Figure 4.1.

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}	u_{15}	u_{16}	u_{17}	u_{18}	u_{19}	u_{20}
r_1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1
r_2	1	1	0	1	0	0	1	1	0	0	1	1	0	1	0	0	1	1	1	1
r_3	0	0	1	0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	1	1
r_4	1	0	1	1	0	1	1	1	0	1	1	1	1	1	0	1	0	1	0	0

FIGURE 4.1: A sample preference matrix $M_{4\times20}$.

4.2.2 Resampling

Resampling user preferences may be required to generate samples of user preferences based on the estimated distribution of the original data (collected user preferences) to enhance the accuracy of the Eells measure. This is particularly useful when conducting a comprehensive market research is not practical.

We use a resampling technique introduced by Macke's *et al.* [139] to generate larger samples of collected user preferences using a Latent Multivariate Gaussian model. The process as given in Figure 4.2 starts with reading the preference matrix of users (Step 1) and continues with estimating the means (Step 2) and variances of user preferences (Step 3) for each requirement. Then the covariances matrix of the requirements will be computed (Step 4) to be used for generating new samples. Thereafter the number of samples will be specified (Step 5) and samples will be generated based on the Dichotomized Gaussian Distribution model discussed in [139] (Step 6).

The precision of the employed resampling technique (Figure 4.2) can be evaluated (Step 7) by comparing the means and covariance matrix of the generated samples against the covariance matrix of the initial samples gathered from users. Steps 1 to 7 may be repeated for larger numbers of samples until the means and covariance matrix of the resampled data and those of the initial sample converge.

Macke's technique has proved to be computationally efficient and feasible for a large number of variables (software requirements). Macke *et al.* [46] showed that the entropy of the Latent Multivariate Gaussian model is near theoretical maximum for a wide range of parameters.

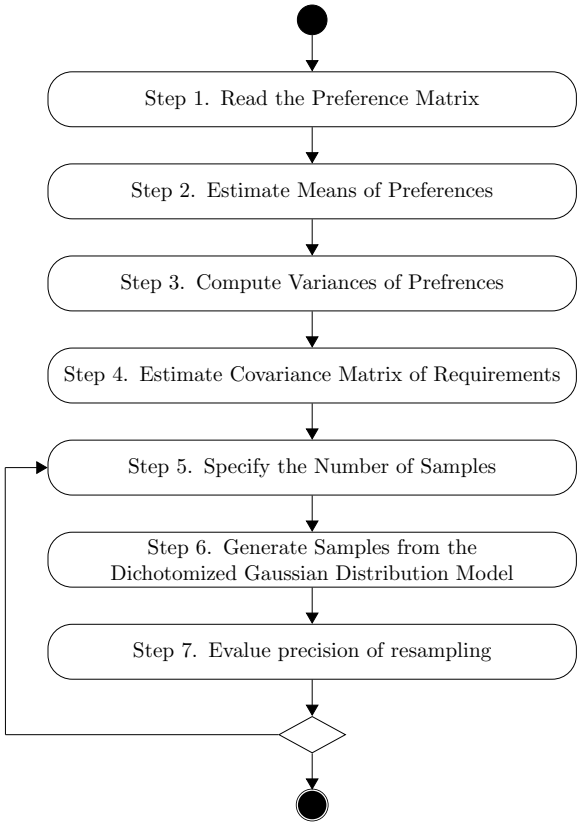


FIGURE 4.2: Steps for generating samples from user preferences.

4.2.3 Extracting Causal Relations among User Preferences

User preferences for a requirement may increase or decrease preferences for other requirements. Such causal relations can be identified using measures of causal strength [128, 131, 132]. Causal relations among user preferences can then be used to specify the strengths and qualities of value dependencies among requirements as values of software requirements are determined by user preferences for those requirements.

As such, we have adopted one of the most widely used measures of causal strength, referred to as the Eells measure [45], to estimate the strengths and qualities of explicit value dependencies among software requirements as given by (4.1). The sign (magnitude) of $\eta_{i,j}$ specifies the quality (strength) of a value dependency from a requirement r_i to r_j, where selecting (ignoring) r_j may influence, either positively or negatively, the value of r_i.

$$\eta_{i,j} = p(r_i|r_j) - p(r_i|\bar{r}_j),\ \ \eta_{i,j} \in [-1, 1] \tag{4.1}$$

For a pair of requirements (r_i, r_j), the Eells measure captures both positive and negative value dependencies from r_i to r_j by subtracting the conditional probability $p(r_i|\bar{r}_j)$ from $p(r_i|r_j)$, where conditional probabilities $p(r_i|\bar{r}_j)$ and $p(r_i|r_j)$ denote strengths of positive and negative causal relations from r_i to r_j respectively, that is selecting the requirement r_i may cause an increase or decrease in the value of r_j.

$$\begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \end{array} \\ \begin{pmatrix} 1.0000 & 0.8333 & 0.8750 & 0.7692 \\ 0.5882 & 1.0000 & 0.6250 & 0.6154 \\ 0.4118 & 0.4167 & 1.0000 & 0.3846 \\ 0.5882 & 0.6667 & 0.6250 & 1.0000 \end{pmatrix} \end{array} \quad - \quad \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \end{array} \\ \begin{pmatrix} 0.0000 & 0.8750 & 0.8333 & 1.0000 \\ 0.6667 & 0.0000 & 0.5833 & 0.5714 \\ 0.3333 & 0.3750 & 0.0000 & 0.4286 \\ 1.0000 & 0.6250 & 0.6667 & 0.0000 \end{pmatrix} \end{array} \quad = \quad \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \end{array} \\ \begin{pmatrix} +1.0000 & -0.0417 & +0.0417 & -0.2308 \\ -0.0785 & +1.0000 & +0.0417 & +0.0440 \\ +0.0785 & +0.0417 & +1.0000 & -0.0440 \\ -0.4118 & +0.0417 & -0.0417 & +1.0000 \end{pmatrix} \end{array}$$

(a) $P_{4\times4}$ (b) $\bar{P}_{4\times4}$ (c) $\boldsymbol{\eta}_{4\times4}$

FIGURE 4.3: Computing the Eells measure for the preference matrix of Figure 4.1.

Matrices $P_{4\times4}$ (Figure 4.3(a)) and $\bar{P}_{4\times4}$ (Figure 4.3(b)) show the strengths of positive and negative causal relations among user preferences for requirements in the preference matrix $M_{4\times8}$ (Figure 4.1). For a pair of requirements r_i and r_j with $i \neq j$, an off-diagonal element $p_{i,j}$ ($\bar{p}_{i,j}$) of matrix $P_{4\times4}$ ($\bar{P}_{4\times4}$) denotes the strength of a positive (negative) causal relation from r_i to r_j.

For diagonal elements of $P_{4\times4}$ ($\bar{P}_{4\times4}$) on the other hand, we have $p_{i,i} = p(r_i|r_i) = 1$ ($\bar{p}_{i,i} = p(r_i|\bar{r}_i) = 0$). Hence, subtracting each element $\bar{p}_{i,j}$ from its corresponding element $p_{i,j}$, where $i \neq j$, gives the Eells causal strength $\eta_{i,j}$ for the value dependency from r_i to r_j. Diagonal elements, however, may be ignored or set to zero as self-causation is not meaningful here.

Algorithm 4.1 specifies the steps for computing the measure of causal strength for a given preference matrix $M_{n\times k}$. In this algorithm, an element $\lambda_{i,j}$ in matrix $\lambda_{n\times2n}$

Algorithm 4.1: Computing the Eells measure of strength.

Input: *Matrix of user preferences:* $M_{n\times k}$
Output: *Matrix of Eells measure:* $\boldsymbol{\eta}_{n\times n}$

1: $P_{n\times n} \leftarrow 0$
2: $\bar{P}_{n\times n} \leftarrow 0$
3: $\boldsymbol{\eta}_{n\times n} \leftarrow 0$
4: $\boldsymbol{\lambda} n \times 2n \leftarrow 0$
5: **for each** $r_j \in R$ **do**
6: **for each** $r_i \in R$ **do**
7: **for each** $u_t \in U$ **do**
8: **if** $m_{j,t} = 1$ **then**
9: **if** $m_{i,t} = 1$ **then**
10: $\lambda_{i,j} \leftarrow (\lambda_{i,j} + 1)$
11: **else**
12: $\lambda_{i,j+n} \leftarrow (\lambda_{i,j+n} + 1)$
13: **end if**
14: **end if**
15: **end for**
16: $p_{i,j} \leftarrow (\frac{\lambda_{i,j}}{\lambda_{j,j}})$
17: $\bar{p}_{i,j} \leftarrow (\frac{\lambda_{i,j+n}}{\lambda_{j+n,j+n}})$
18: $\eta_{i,j} \leftarrow (p_{i,j} - \bar{p}_{i,j})$
19: **end for**
20: **end for**

counts the number of times that a pair of requirements (r_i,r_j) are selected together by the users. An element $\lambda_{i,j+n}$ on the other hand, gives the number of times users have selected r_i while ignoring r_j. It is clear that, $\lambda_{i,i}$ gives the number of occurrences of r_i in $M_{n\times k}$ while $\lambda_{i,i+n} = 0$.

Given a dataset of n requirements and t user preferences, lines 8 to 14 of Algorithm 4.1 will be executed for each pair of requirements and all gathered user preferences: $O(t \times n^2)$. Moreover, lines 16 to 18 need to be executed for all pairs of requirements. The computational complexity of the algorithm is therefore of $O(n^2)$. The overall complexity of the algorithm therefore is of $O(t \times n^2)$.

4.2.4 Testing the Significance of Causal Relations

Using measures of interestingness [140] is sometimes not sufficient to understand the significance of the relations found among the items of a dataset as explained in [137]. In this regard, we have employed the widely adopted measure of association referred to as the *Odds Ratio* to test if causal relations identified based on the Eells measure are

significant or not. For a positive (negative) causal relation from requirement r_j to r_i, which means the presence of r_j positively (negatively) influences the value of r_i, (4.2) computes the Odds ratio denoted by $\omega(r_i, r_j)$ in which the order of r_i and r_j does not make any difference. Also, $p(r_i, r_j)$ denotes the joint probability of r_i and r_j. Similarly, $p(r_i, \bar{r}_j)$ gives the joint probability that r_i is selected and r_j is not.

$$\omega(r_i, r_j) = \frac{p(r_i, r_j)p(\bar{r}_i, \bar{r}_j)}{p(r_i, \bar{r}_j)p(\bar{r}_i, r_j)}, \ \omega(r_i, r_j) \in (0, \infty) \tag{4.2}$$

To test the significance of a causal relation from a requirement r_j to r_i we use the technique used in [137] by computing the lower bound (ω_-) and the upper bound (ω_+) of the confidence interval of the Odds Ratio as given by (4.3)-(4.4). In these equations z' is the critical value corresponding to a desired level of confidence. Also, u denotes the total number of user preferences gathered. When we find a lower bound $\omega_- \leq 1$ AND an upper bound $\omega_+ \geq 1$ for the Odds ratio imply the absence of any significant causal relation from r_j and r_i. To exclude insignificant relations, the strengths of those relations will be set to zero.

$$\omega_-(r_i, r_j) = ln\big(\omega(r_i, r_j)\big) - \frac{z'}{\sqrt{u}}\sqrt{\frac{1}{p(r_i, r_j)} + \frac{1}{p(\bar{r}_i, \bar{r}_j)} + \frac{1}{p(\bar{r}_i, r_j)} + \frac{1}{p(r_i, \bar{r}_j)}} \tag{4.3}$$

$$\omega_+(r_i, r_j) = ln\big(\omega(r_i, r_j)\big) + \frac{z'}{\sqrt{u}}\sqrt{\frac{1}{p(r_i, r_j)} + \frac{1}{p(\bar{r}_i, \bar{r}_j)} + \frac{1}{p(\bar{r}_i, r_j)} + \frac{1}{p(r_i, \bar{r}_j)}} \tag{4.4}$$

4.2.5 Computing the Strengths and Qualities of value Dependencies

The strength of an explicit value dependency from a requirement r_i to r_j hence is computed by (4.5), which gives a mapping from the Eells measure of causal strength $\eta_{i,j}$ to the fuzzy membership function $\rho : R \times R \rightarrow [0, 1]$ as given in Figure 4.4. Only significant causal relations which pass the test in Section 4.2.4 will be considered.

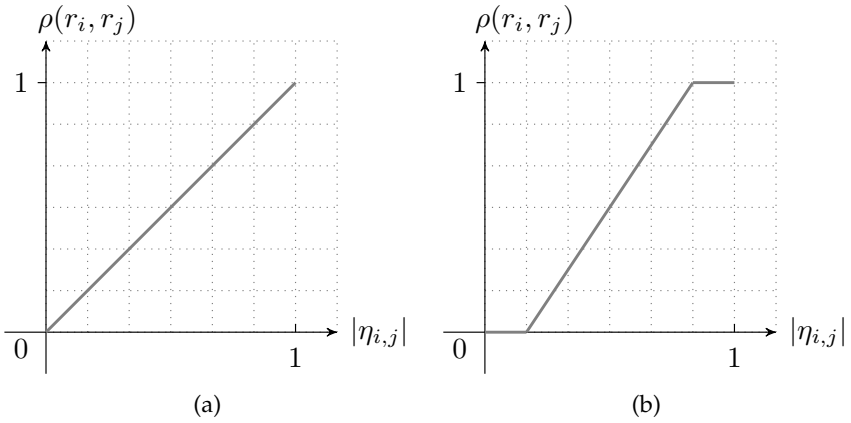


FIGURE 4.4: Sample membership functions for strengths of value dependencies.

The fuzzy membership functions however maybe adjusted to account for the imprecision of value dependencies and suit the particular needs of decision makers. For instance, the membership function of Figure 4.4(a) may be used to ignore "too weak" value dependencies while "too strong" dependencies are considered as full strength relations, $\rho(r_i, r_j) = 1$. Different membership functions and measures of causal strength may be used by decision makers resulting in a set of optimal solutions to choose from.

$$\rho(r_i, r_j) = |\eta_{i,j}| \tag{4.5}$$

$$\sigma(r_i, r_j) = \begin{cases} + & \text{if } \eta_{i,j} > 0 \\ - & \text{if } \eta_{i,j} < 0 \\ \pm & \text{if } \eta_{i,j} = 0 \end{cases} \tag{4.6}$$

As given by (4.6), $\eta_{i,j} > 0$ indicates that the strength of the positive causal relation from r_i to r_j is greater than the strength of its corresponding negative causal relation: $p(r_i|r_j) > p(r_i|\neg r_j)$ and therefore the quality of (r_i, r_j) is positive ($\sigma(r_i, r_j) = +$). Similarly, $\eta_{i,j} < 0$ indicates $p(r_i|\neg r_j) > p(r_i|r_j) \rightarrow \sigma(r_i, r_j) = -$. Also, $p(r_i|r_j) - p(r_i|\neg r_j) = 0$ specifies that the quality of the zero-strength value dependency (r_i, r_j) is non-specified ($\sigma(r_i, r_j) = \pm$).

4.2.6 Value Implications of Precedence Dependencies

As explained earlier, precedence dependencies among requirements such as *requires* and *conflicts-with* and their value implications need to be considered in requirement selection. For instance, a requirement r_i requires (conflicts-with) r_j implies that the value of r_i fully relies on selecting (ignoring) r_j. This may not be captured by value dependencies identified from user preferences.

Hence, it is important to not only consider user preferences in the identification of explicit value dependencies, but to take into account the value implications of precedence dependencies and consider them in a requirement selection. This can be achieved by modeling the precedence dependencies using a *Precedence Dependency Graph* (PDG) as introduced in Definition 4.2.

Definition 4.2. *The Precedence Dependency Graph* (PDG). A PDG is a signed directed graph $G = (R, W)$ in which $R = \{r_1, ..., r_n\}$ denotes the graph nodes (requirements) and $W(r_i, r_j) \in -1, 0, 1$ specifies the presence or absence of a precedence dependency from r_i to r_j. $W(r_i, r_j) = 1$ ($W(r_i, r_j) = -1$) specifies a positive (negative) precedence dependency from r_i to r_j meaning that r_i *requires* (*conflicts-with*) r_j. Finally $W(r_i, r_j) = 0$ specifies the absence of any precedence dependency from requirement r_i to r_j.

$$PDL(G) = \frac{k}{{}^{n}P_2} = \frac{k}{n(n-1)} \tag{4.7}$$

$$NPDL(G) = \frac{j}{k} \tag{4.8}$$

Hence, precedence dependencies of a software project can be captured by a PDG and mathematically modeled in terms of the precedence constraints of the optimization model used for a requirement selection. It is clear that increasing precedence dependencies among requirements limits the number of choices and therefore reduce the number of feasible solutions (requirement subsets). To measure the level of precedence dependencies among requirements of a PDG, we have defined the *Precedence Dependency Level* (PDL) and the *Negative Precedence Dependency Level* (NPDL) as given by (4.7) and (4.8) respectively.

The PDL of a precedence dependency graph G with n nodes (requirements) is computed by dividing the total number of the precedence dependencies (k) among the nodes of G by the maximum number of the potential precedence dependencies in G ($n(n-1)$). Also, the NPDL of G is computed by dividing the number of the negative precedence dependencies (j) by the total number of the positive and negative precedence dependencies.

4.3 Modeling Value Dependencies by Fuzzy Graphs

Since their introduction in 1973 [124], fuzzy graphs have been widely adopted in decision making and expert systems [42] as they contribute to more accurate models by taking into account imprecision in real-world problems [124].

Fuzzy graphs have, particularly, demonstrated to be useful in capturing the imprecision of dependency relations in software [47, 26]. Ngo-The *et al.*, exploited fuzzy graphs for modeling dependency satisfaction in release planning [47] and capturing the imprecision of coupling dependencies among requirements [26]. Moreover, Wang *et al.* [27] adopted linguistic fuzzy terms to capture the variances of strengths of dependencies among software requirements.

In this section we discuss modeling value dependencies by fuzzy graphs and identification of implicit value dependencies among requirements. We further use the algebraic structure of fuzzy graphs to compute the influences of requirements on the values of each other.

4.3.1 Value Dependency Graphs

To account for the imprecision of value dependencies, we have introduced *Value Dependency Graphs* (VDGs) based on fuzzy graphs for modeling value dependencies and their characteristics (quality and strength). We have specially modified the classical definition of fuzzy graphs to consider not only the strength but also the quality (positive or negative) of value dependencies as given by Definition 4.3.

Definition 4.3. *The Value Dependency Graph* (VDG) is a signed directed fuzzy graph [77] $G = (R, \sigma, \rho)$ where, requirements $R : \{r_1, ..., r_n\}$ constitutes the graph nodes. Also, the qualitative function $\sigma(r_i, r_j) \rightarrow \{+, -, \pm\}$ and the membership function $\rho : (r_i, r_j) \rightarrow [0, 1]$ denote the quality and the strength of the explicit value dependency (edge of the graph) from r_i to r_j receptively. Moreover, $\rho(r_i, r_j) = 0$ denotes the absence of any explicit value dependency from r_i to r_j. In that case we have $\sigma(r_i, r_j) = \pm$, where $\pm$ denotes the quality of the dependency is non-specified.

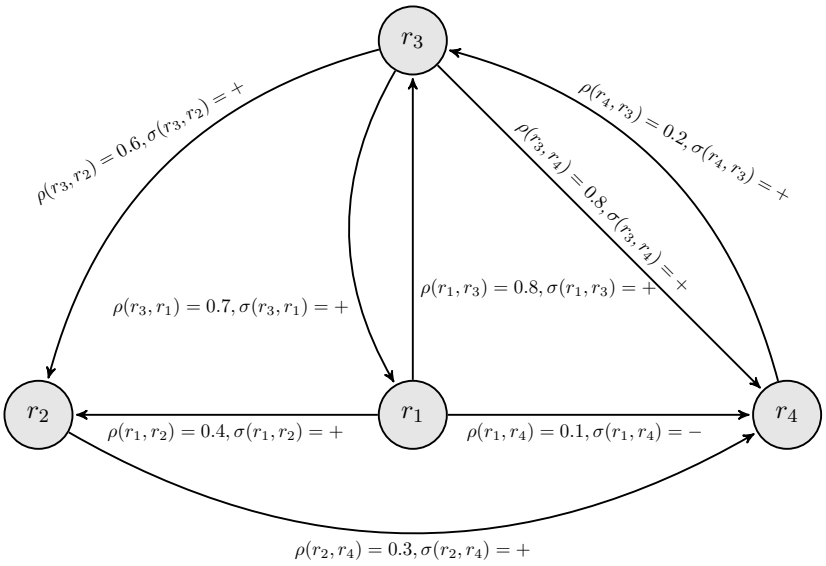


FIGURE 4.5: A sample value dependency graph.

For instance, in the value dependency graph of Figure 4.5 $\sigma(r_1, r_2) = +$ and $\rho(r_1, r_2) = 0.4$ specifies a positive value dependency from r_1 to r_2 with strength 0.4. That is selecting r_2 has an explicit positive influence on the value of r_1.

4.3.2 Value Dependencies in VDGs

In Section 4.2 we introduced an automated technique for the identification of explicit value dependencies and their characteristics (quality and strength) from user preferences. Definition 4.4 provides a more comprehensive definition of value dependencies that includes both explicit and implicit value dependencies among the requirements of a software project based on the algebraic structure of fuzzy graphs.

Definition 4.4. *Value Dependencies*. A value dependency in a value dependency graph $G = (R, \sigma, \rho)$ is defined as a sequence of requirements $d_i : \big(r(0), ..., r(k)\big)$ such that $\forall r(j) \in d_i$, $1 \leq j \leq k$ we have $\rho\big(r(j-1), r(j)\big) \neq 0$. $j \geq 0$ is the sequence of the j^{th} requirement (node) denoted as $r(j)$ on the dependency path. A consecutive pair $\big(r(j-1), r(j)\big)$ specifies an explicit value dependency.

$$\forall d_i : \big(r(0), ..., r(k)\big) : \rho(d_i) = \bigwedge_{j=1}^{k} \rho\big(r(j-1), r(j)\big) \tag{4.9}$$

$$\forall d_i : \big(r(0), ..., r(k)\big) : \sigma(d_i) = \prod_{j=1}^{k} \sigma\big(r(j-1), r(j)\big) \tag{4.10}$$

Equation (4.9) computes the strength of a value dependency $d_i : \big(r(0), ..., r(k)\big)$ by finding the strength of the weakest of the k explicit dependencies on d_i. Fuzzy operator $\wedge$ denotes Zadeh's [141] AND operation (infimum).

The quality (positive or negative) of a value dependency $d_i : \big(r(0), ..., r(k)\big)$ is calculated by qualitative serial inference [142, 143, 79] as given by (4.10) and Table 4.1. Inferences in Table 4.1 are informally proved by Wellman [143] and Kleer [142].

TABLE 4.1: Qualitative serial inference in VDGs.

$\sigma\big(r(j-1), r(j), r(j+1)\big)$		$\sigma\big(r(j), r(j+1)\big)$		
		$+$	$-$	$\pm$
$\sigma\big(r(j-1), r(j)\big)$	$+$	$+$	$-$	$\pm$
	$-$	$-$	$+$	$\pm$
	$\pm$	$\pm$	$\pm$	$\pm$

Let $D = \{d_1, d_2, ..., d_m\}$ be the set of all value dependencies from $r_i \in R$ to $r_j \in R$ in a VDG $G = (R, \sigma, \rho)$, where positive and negative dependencies can simultaneously exist from r_i to r_j. The strength of all positive value dependencies from r_i to r_j is denoted by $\rho^{+\infty}(r_i, r_j)$ and calculated by (4.11), that is to find the strength of the strongest positive dependency [42] from r_i to r_j. Fuzzy operators $\wedge$ and $\vee$ denote Zadeh's [141] fuzzy AND (minimum) and fuzzy OR (maximum) operations respectively.

In a similar way, the strength of all negative value dependencies from r_i to r_j is denoted by $\rho^{-\infty}(r_i,r_j)$ and calculated by (4.12).

$$\rho^{+\infty}(r_i,r_j) = \bigvee_{d_m \in D, \sigma(d_m)=+} \rho(d_m) \tag{4.11}$$

$$\rho^{-\infty}(r_i,r_j) = \bigvee_{d_m \in D, \sigma(d_i)=-} \rho(d_m) \tag{4.12}$$

A brute-force approach to computing $\rho^{+\infty}(r_i,r_j)$ or $\rho^{-\infty}(r_i,r_j)$ needs to calculate the strengths of all paths from r_i to r_j which is of complexity of $O(n!)$ for n requirements (VDG nodes). To avoid such complexity, we have formulated the problem of calculating $\rho^{+\infty}(r_i,r_j)$ and $\rho^{-\infty}(r_i,r_j)$ as the widest path problem (also known as the maximum capacity path problem [144]) which can be solved in polynomial time by the Floyd-Warshall algorithm [44].

For this purpose, we devised a modified version of Floyd-Warshall algorithm (Algorithm 4.2) that computes $\rho^{+\infty}(r_i,r_j)$ and $\rho^{-\infty}(r_i,r_j)$ for all pairs of requirements (r_i,r_j), $r_i,r_j \in R : \{r_1,...,r_n\}$ with the time bound of $O(n^3)$. For each pair of requirements (r_i,r_j) in a VDG $G = (R,\sigma,\rho)$, lines 18 to 35 of Algorithm 4.2 find the strength of all positive value dependencies and the strength of all negative value dependencies from r_i to r_j.

$$I_{i,j} = \rho^{+\infty}(r_i,r_j) - \rho^{-\infty}(r_i,r_j) \tag{4.13}$$

The overall strength of all positive and negative value dependencies from r_i to r_j is referred to as the *Influence* of r_j on the value of r_i and denoted by $I_{i,j}$. $I_{i,j}$ as given by (4.13) is calculated by subtracting the strength of all negative value dependencies from r_i to r_j ($\rho^{-\infty}(r_i,r_j)$) from the strength of all positive value dependencies from r_i to r_j ($\rho^{+\infty}(r_i,r_j)$). It is clear that $I_{i,j} \in [-1,1]$. $I_{i,j} > 0$ states that r_j influences the value of r_i in a positive way whereas $I_{i,j} < 0$ indicates that the ultimate influence of r_j on r_i is negative.

Algorithm 4.2: Calculating the strengths of value dependencies.

Input: VDG $G = (R, \sigma, \rho)$
Output: $\rho^{+\infty}, \rho^{-\infty}$

1: **for each** $r_i \in R$ **do**
2: **for each** $r_j \in R$ **do**
3: $\rho^{+\infty}(r_i, r_j) \leftarrow \rho^{-\infty}(r_i, r_j) \leftarrow -\infty$
4: **end for**
5: **end for**
6: **for each** $r_i \in R$ **do**
7: $\rho(r_i, r_i)^{+\infty} \leftarrow \rho(r_i, r_i)^{-\infty} \leftarrow 0$
8: **end for**
9: **for each** $r_i \in R$ **do**
10: **for each** $r_j \in R$ **do**
11: **if** $\sigma(r_i, r_j) = +$ **then**
12: $\rho^{+\infty}(r_i, r_j) \leftarrow \rho(r_i, r_j)$
13: **else if** $\sigma(r_i, r_j) = -$ **then**
14: $\rho^{-\infty}(r_i, r_j) \leftarrow \rho(r_i, r_j)$
15: **end if**
16: **end for**
17: **end for**
18: **for each** $r_k \in R$ **do**
19: **for each** $r_i \in R$ **do**
20: **for each** $r_j \in R$ **do**
21: **if** $min(\rho^{+\infty}(r_i, r_k), \rho^{+\infty}(r_k, r_j)) > \rho^{+\infty}(r_i, r_j)$ **then**
22: $\rho^{+\infty}(r_i, r_j) \leftarrow min(\rho^{+\infty}(r_i, r_k), \rho^{+\infty}(r_k, r_j))$
23: **end if**
24: **if** $min(\rho^{-\infty}(r_i, r_k), \rho^{-\infty}(r_k, r_j)) > \rho^{+\infty}(r_i, r_j)$ **then**
25: $\rho^{+\infty}(r_i, r_j) \leftarrow min(\rho^{-\infty}(r_i, r_k), \rho^{-\infty}(r_k, r_j))$
26: **end if**
27: **if** $min(\rho^{+\infty}(r_i, r_k), \rho^{-\infty}(r_k, r_j)) > \rho^{-\infty}(r_i, r_j)$ **then**
28: $\rho^{-\infty}(r_i, r_j) \leftarrow min(\rho^{+\infty}(r_i, r_k), \rho^{-\infty}(r_k, r_j))$
29: **end if**
30: **if** $min(\rho^{-\infty}(r_i, r_k), \rho^{+\infty}(r_k, r_j)) > \rho^{-\infty}(r_i, r_j)$ **then**
31: $\rho^{-\infty}(r_i, r_j) \leftarrow min(\rho^{-\infty}(r_i, r_k), \rho^{+\infty}(r_k, r_j))$
32: **end if**
33: **end for**
34: **end for**
35: **end for**

Example 4.1. Let $D = \{d_1 : (r_1, r_2, r_4), d_2 : (r_1, r_3, r_4), d_3 : (r_1, r_4)\}$ specify value dependencies from requirement r_1 to r_4 in Figure 4.5. Using (4.10), qualities of d_1 to d_3 are computed as: $\sigma(d_1) = \Pi(+,+) = +$, $\sigma(d_2) = \Pi(+,+) = +$, and $\sigma(d_3) = \Pi(-) = -$. Strengths are calculated by (4.9) as: $\rho(d_1) = \wedge\big(\rho(r_1, r_2), \rho(r_2, r_4)\big) = min(0.4, 0.3)$, $\rho(d_2) = \wedge\big(\rho(r_1, r_3), \rho(r_3, r_4)\big) = min(0.8, 0.8)$, $\rho(d_3) = min(0.1)$. Using (4.11)and (4.12) then we have $\rho(r_1, r_4)^{+\infty} = \vee(\rho(d_1), \rho(d_2)) = max(0.3, 0.8)$ and $\rho^{-\infty}(r_1, r_4) = max(\rho(d_3))$. Therefore, we have $I_{1,4} = \rho(r_1, r_4)^{+\infty} - \rho(r_1, r_4)^{-\infty} = 0.7$ which means the positive influence of r_4 on the value of r_1 prevails. Table 4.2 lists influences of requirements in the VDG of Figure 4.5 on the value of each other.

TABLE 4.2: Overall influences computed for VDG of Figure 4.5.

$I_{i,j} = \rho(r_i, r_j)^{+\infty} - \rho(r_i, r_j)^{-\infty}$	r_1	r_2	r_3	r_4
r_1	$0.0 - 0.0 = 0.0$	$0.6 - 0.1 = 0.5$	$0.8 - 0.1 = 0.7$	$0.8 - 0.1 = 0.7$
r_2	$0.2 - 0.0 = 0.2$	$0.0 - 0.0 = 0.0$	$0.2 - 0.0 = 0.2$	$0.3 - 0.0 = 0.3$
r_3	$0.7 - 0.1 = 0.6$	$0.6 - 0.1 = 0.5$	$0.0 - 0.0 = 0.0$	$0.8 - 0.1 = 0.7$
r_4	$0.2 - 0.0 = 0.2$	$0.2 - 0.0 = 0.2$	$0.2 - 0.0 = 0.2$	$0.0 - 0.0 = 0.0$

Definition 4.5. *Value Dependency Level (VDL) and Negative Value Dependency Level (NVDL).* Let $G = (R, \sigma, \rho)$ be a VDG with $R = \{r_1, ..., r_n\}$, k be the total number of explicit value dependencies in G, and m be the total number of negative explicit value dependencies. Then the VDL and NVDL of G are derived by (4.14) and (4.15) respectively.

$$VDL(G) = \frac{k}{{}^{n}P_2} = \frac{k}{n(n-1)} \tag{4.14}$$

$$NVDL(G) = \frac{m}{k} \tag{4.15}$$

Example 4.2. For the value dependency graph G of Figure 4.5 we have $n = 4$, $k = 8$, and $m = 1$. $VDL(G)$ is derived by (4.14) as: $VDL(G) = \frac{8}{4\times3} = \frac{8}{12} \approxeq 0.67$. Also we have from Equation (4.15), $NVDL(G) = \frac{1}{8} = 0.125$.

4.4 Integrating Value Dependencies into Selection

4.4.1 Overall Value of a Subset of Requirements

This section details our proposed measure for the economic worth of a selected subset of requirements (software product) i.e. overall value (OV) as an alternative to the accumulated value (AV) and the expected value (EV) of that subset. The formulation of overall value in this section takes into account user preferences for selected requirements as well as the impacts of value dependencies on the values of requirements.

Value dependencies as explained in Section 4.2 are identified based on causal relations among user preferences. Section 4.2 presented an automated technique for identification of value dependencies among requirements. Then, algorithm 4.2 was used to infer implicit value dependencies and compute the influences of requirements on the values of each other based on the algebraic structure of fuzzy graphs.

To compute the overall values of selected requirements, (4.16)-(4.17) give the penalty of ignoring (selecting) requirements with positive (negative) influence on the values of selected requirements. θ_i in this equation denotes the penalty for a requirement r_i, n denotes the number of requirements, and x_j specifies whether a requirement r_j is selected ($x_j = 1$) or not ($x_j = 0$). Also, $I_{i,j}$, as in (4.13), gives the positive or negative influence of r_j on the value of r_i.

$$\theta_i = \bigvee_{j=1}^{n} \left(\frac{x_j\big(|I_{i,j}| - I_{i,j}\big) + (1 - x_j)\big(|I_{i,j}| + I_{i,j}\big)}{2} \right) \quad = \quad \bigvee_{j=1}^{n} \left(\frac{|I_{i,j}| + (1 - 2x_j) I_{i,j}}{2} \right), \qquad i \neq j = 1, ..., n \tag{4.16}$$

$$x_j \in \{0, 1\}, \qquad j = 1, ..., n \tag{4.17}$$

We made use of the algebraic structure of fuzzy graphs for computing the influences of requirements on the values of each other as explained in Section 4.3. Accordingly, θ_i is computed using the fuzzy OR operator which is to take supremum over the strengths of all ignored positive dependencies and selected negative dependencies of r_i in its

corresponding value dependency graph. Overall values of selected requirements thus can be computed by (4.19), where v_i' denotes the overall value of a requirement r_i, $E(v_i)$ specifies the expected value of r_i, and θ_i denotes the penalty of ignoring (selecting) positive (negative) value dependencies of r_i.

Equation (4.20) derives the overall value of a software product with n requirements, where cost and expected value of a requirements r_i are denoted by c_i and $E(v_i)$ respectively. Decision variable x_i specifies whether r_i is selected ($x_i = 1$) or not ($x_i = 0$). $E(V_i)$ is computed by (4.40), where v_i denotes the estimated (nominal) value of r_i. Also $p(r_i)/p(\bar{r}_i)$ specify the probability that users select/ignore a requirement r_i.

$$E(v_i) = p(r_i) \times v_i + p(\bar{r}_i) \times 0 = p(r_i) \times v_i \tag{4.18}$$

For a requirement r_i, θ_i specifies the penalty of ignoring (selecting) requirements with positive (negative) influence on the expected value of r_i as explained earlier. $\theta_i v_i$ in (4.20) therefore, gives the value loss for a requirement r_i as a result of ignoring (selecting) requirements that positively (negatively) impact user preferences for r_i and consequently its expected value.

$$v_i' = (1 - \theta_i)E(v_i) \tag{4.19}$$

$$OV = \sum_{i=1}^{n} x_i(1 - \theta_i)E(v_i),\ x_i \in \{0, 1\} \tag{4.20}$$

Example 4.3. Consider finding penalties for requirements of Figure 4.5, where r_4 is not selected ($x_1 = x_2 = x_3 = 1, x_4 = 0$). From Table 4.2 we have $I_{1,4} = I_{3,4} = 0.7, I_{2,4} = 0.3, I_{4,4} = 0.0$. As such, based on (4.16) penalties are computed: $\theta_1 = \vee(\frac{|0.0|+(1-2(1))(0.0)}{2}, \frac{|0.45|+(1-2(1))(0.5)}{2}, \frac{|0.7|+(1-2(1))(0.7)}{2}, \frac{|0.7|+(1-2(0))(0.7)}{2}) = 0.7$. Similarly, we have $\theta_2 = 0.3, \theta_3 = 0.7$. Therefore, the overall value of the selected requirements r_1, r_2, r_3 is derived by (4.20) as: $OV(s_1) = (1 - 0.7)E(v_1) + (1 - 0.3)E(v_2) + (1 - 0.7)E(v_3)$.

4.4.2 The Integer Linear Programming Model

This section presents our proposed integer linear programming (ILP) model for optimizing the overall value of a software product. The overall value of a requirement subset, as given by (4.20), considers user preferences and the impacts of value dependencies on the expected values of the selected requirements. The proposed ILP model hence embeds user preferences and value dependencies into requirement selection by optimizing the overall value of a software product.

Equations (4.21)-(4.26) give our proposed integer programming model as a main component of the DARS-ILP method. In these equations, x_i is a selection variable denoting whether a requirement r_i is selected ($x_i = 1$) or ignored ($x_i = 0$). Also θ_i in (4.16) specifies the penalty of a requirement r_i, which is the extent to which the expected value of r_i is impacted by ignoring (selecting) requirements with positive (negative) influences on the value of r_i. Constraint (4.23) on the other hand accounts for precedence dependencies among requirements and the value implications of those dependencies.

$$\text{Maximize } \sum_{i=1}^{n} x_i(1-\theta_i)E(v_i) \tag{4.21}$$

$$\text{Subject to } \sum_{i=1}^{n} c_i x_i \leq b \tag{4.22}$$

$$\begin{cases} x_i \leq x_j & r_j \text{ precedes } r_i \\ x_i \leq 1 - x_j & r_i \text{ conflicts with } r_j,\ i \neq j = 1,...,n \end{cases} \tag{4.23}$$

$$\theta_i \geq \left(\frac{|I_{i,j}| + (1-2x_j)I_{i,j}}{2} \right), \qquad i \neq j = 1,...,n \tag{4.24}$$

$$x_i \in \{0,1\}, \qquad i = 1,...,n \tag{4.25}$$

$$0 \leq \theta_i \leq 1, \qquad i = 1,...,n \tag{4.26}$$

Moreover, for a requirement r_i, θ_i depends on the selection variable x_j and the strength of positive (negative) value dependencies as given by (4.16). Since $I_{i,j}$ is computed by (4.13) we can restate θ_i as a function of x_j: $\theta_i = f(x_j)$. The objective function (4.21),

thus, can be restated as Maximize $\sum_{i=1}^{n} x_i E(v_i) - x_i f(x_j) E(v_i)$ where $x_i f(x_j) E(v_i)$ is a quadratic non-linear expression [39]. Equations (4.21)-(4.24), on the other hand, denote a convex optimization problem as the model maximizes a concave objective function with linear constraints.

$$\text{Maximize } \sum_{i=1}^{n} x_i E(v_i) - y_i E(v_i) \tag{4.27}$$

$$\text{Subject to } \sum_{i=1}^{n} c_i x_i \leq b \tag{4.28}$$

$$\begin{cases} x_i \leq x_j & r_j \text{ precedes } r_i \\ x_i \leq 1 - x_j & r_i \text{ conflicts with } r_j,\ i \neq j = 1, ..., n \end{cases} \tag{4.29}$$

$$\theta_i \geq \left(\frac{|I_{i,j}| + (1 - 2x_j) I_{i,j}}{2} \right), \qquad i \neq j = 1, ..., n \tag{4.30}$$

$$-g_i \leq x_i \leq g_i, \qquad i = 1, ..., n \tag{4.31}$$

$$1 - (1 - g_i) \leq x_i \leq 1 + (1 - g_i), \qquad i = 1, ..., n \tag{4.32}$$

$$-g_i \leq y_i \leq g_i, \qquad i = 1, ..., n \tag{4.33}$$

$$-(1 - g_i) \leq (y_i - \theta_i) \leq (1 - g_i), \qquad i = 1, ..., n \tag{4.34}$$

$$0 \leq y_i \leq 1, \qquad i = 1, ..., n \tag{4.35}$$

$$0 \leq \theta_i \leq 1, \qquad i = 1, ..., n \tag{4.36}$$

$$x_i, g_i \in \{0, 1\}, \qquad i = 1, ..., n \tag{4.37}$$

Convex optimization problems are solvable [39]. However, for problems of moderate to large sizes, integer linear programming (ILP) models are preferred [145] as they can be efficiently solved, despite the inherent complexity of NP-hard problems, due to the advances in solving ILP models and availability of efficient tools such as ILOG CPLEX for that purpose. This motivates us to consider developing an ILP version of the model as given by (4.27).

In doing so, the non-linear expression $x_i\theta_i$ is substituted with the linear expression y_i ($y_i = x_i\theta_i$). As such, either $a : (x_i = 0, y_i = 0)$, or $b : (x_i = 1, y_i = \theta_i)$ occur. To capture the relation between θ_i and y_i in a linear form, we have made use of an auxiliary variable $g_i = \{0, 1\}$ and (4.31)-(4.35) are added to the original model. As such, we have either $(g_i = 0) \rightarrow a$, or $(g_i = 1) \rightarrow b$.

Therefore, the optimization model of DARS-ILP given by (4.27)-(4.37), is linear and therefore can be efficiently solved [39], even for large scale requirement sets, by the existing commercial solvers such as *IBM CPLEX* [40]. We have implemented, solved, and tested the optimization model of the DARS-ILP method using the *Concert Technology* and the *JAVA API of IBM CPLEX* [40]. The code for this model is available in *JAVA* and *OPL* languages and can be obtained from the website of DARS[22].

4.4.3 The Blind Integer Programming Model

The ILP model presented in Section 4.4.2 relies on the identification of value dependencies in software projects to mitigate the risk of value loss posed by ignoring the influences of the requirements on the values of each other. But there might be situations where there is little or no information available about value dependencies. This may occur, for instance, when it is not practical to collect user preferences as discussed in Section 4.2.

Without sufficient information about value dependencies, the influences of the requirements on the values of each other remain unidentified and, therefore, cannot be taken into account in the requirement selection. This may lead to ignoring (selecting) the requirements with significant positive (negative) influences on the values of the selected requirements and therefore result in value loss.

When the negative influences of the requirements on the values of each other are negligible, reducing the number of the ignored requirements for a given budget, will mitigate the risk of value loss by reducing the chances that requirements with positive influences are ignored.

[22] http://bcert.org/projects/dars

On this basis, we have proposed a *Blind* ILP model for DARS-ILP that reduces the chances that the requirements with positive influences on the values of the selected requirements are ignored. The term "Blind" is used to emphasize that the model does not rely on any information about value dependencies.

$$f_1(n, x_i, E(v_i)) = \sum_{i=1}^{n} x_i E(v_i) \tag{4.38}$$

$$f_2(n, x_i) = \sum_{i=1}^{n} x_i \tag{4.39}$$

$$E(v_i) = p(r_i) v_i \tag{4.40}$$

Equations (4.41)-(4.44) give a multi-objective (bi-objective) formulation of the blind model of DARS-ILP, which aims to simultaneously maximize the utility functions f_1 and f_2 while respecting the budget constraint (4.42) and the precedence constraints (4.43). The utility function f_1 in (4.38) concerns with the expected value of the selected requirements while the utility function f_2 specifies the number of the selected requirements as given by (4.39).

In these equations b denotes the available budget and x_i is a decision variable specifying whether a requirement r_i is selected ($x_i = 1$) or ignored ($x_i = 0$). Also, c_i and $E(v_i)$ denote the estimated cost and the expected value of r_i respectively. $E(v_i)$ is computed by (4.40) in which v_i specifies the estimated value of r_i and $p(r_i)$ denotes the probability that users purchase/use r_i.

$$\text{Maximize } \{f_1(n, x_i, E(v_i)), f_2(n, x_i)\} \tag{4.41}$$

$$\text{Subject to } \sum_{i=1}^{n} x_i c_i \leq b \tag{4.42}$$

$$\begin{cases} x_i \leq x_j & r_j \text{ precedes } r_i \\ x_i \leq 1 - x_j & r_i \text{ conflicts with } r_j,\ i \neq j = 1, ..., n \end{cases} \tag{4.43}$$

$$x_i \in \{0, 1\}, \qquad i = 1, ..., n \tag{4.44}$$

The optimization model (4.41)-(4.44), aims to find a subset of the requirements that, simultaneously, maximizes the utility functions f_1 and f_2 while keeping the cost within the budget and respecting the precedence constraints (4.43). However, maximizing the number of the selected requirements (the utility function f_2) may conflict with maximizing the expected value of the selected requirements (the utility function f_1) and vice versa.

Hence, finding an optimal subset of the requirements, without additional preference information from the stakeholders is not possible. In other words, without additional information, all *Pareto Optimal* [84] subsets found by the optimization model (4.41)-(4.44) are considered to be equally good. In a Pareto optimal (*Non-Dominated*) subset found by the model, none of the utility functions f_1 or f_2 can be improved in value without degrading the other one [49].

The optimization model (4.41)-(4.44) can be solved in different ways, as discussed in [146], depending on the viewpoints of the stakeholders and, thus, there exist different solution philosophies when solving them. The optimization model may aim to find a representative set of Pareto optimal subsets, and/or quantify the trade-offs in satisfying the utility functions f_1 and f_2, and/or finding a single subset that satisfies the preferences of the stakeholders.

In this regard, we reformulate the Blind ILP model of DARS-ILP as a single-objective optimization model given by (4.45)-(4.49). The model aims to avoid ignoring requirements (maximizing f_2) as long as the budget constraint (4.46) is respected and the utility function f_1 is partly satisfied by guaranteeing a lower-bound V for the expected value of the optimal subset. V in (4.47) will be specified by the stakeholders.

The conflict between the utility functions f_1 and f_2, thus, is reconciled by maximizing f_2 while ensuring a lower-bound for f_1. Moreover, precedence dependencies among the requirements are captured by (4.48), where $x_i \leq x_j$ states that a requirement r_i requires r_j while $x_i \leq (1 - x_j)$ means that r_i conflicts with r_j.

$$\text{Maximize } f_1(n, x_i, E(v_i)) \tag{4.45}$$

$$\text{Subject to } \sum_{i=1}^{n} x_i c_i \leq b \tag{4.46}$$

$$f_2(n, x_i) \geq V \tag{4.47}$$

$$\begin{cases} x_i \leq x_j & r_j \text{ precedes } r_i \\ x_i \leq 1 - x_j & r_i \text{ conflicts with } r_j,\ i \neq j = 1, ..., n \end{cases} \tag{4.48}$$

$$x_i \in \{0, 1\} \qquad i = 1, ..., n \tag{4.49}$$

The proposed Blind ILP model is formulated to mitigate the risk of value loss in the absence of sufficient information about value dependencies. Hence, the proposed model does not require the identification and modeling of value dependencies.

Finally, the Blind ILP model of the DARS-ILP method, as given by (4.45)-(4.49), is linear and therefore can be efficiently solved [39], even for large scale requirement sets, by the existing commercial solvers such as *IBM CPLEX* [40]. We have implemented, solved, the model using the *Concert Technology* and the *JAVA API of IBM CPLEX* [40]. The code for this model is available in *JAVA* and *OPL* languages and can be obtained from the website of DARS[26].

4.5 Case Study

This section discusses the practicality and validity of the ILP method of DARS (DARS-ILP) by studying a real-world software product. We also demonstrate why software vendors should take care with value dependencies among requirements, and how to employ DARS-ILP to assist decision makers to comprehend the results, thus raising the research questions **(RQ6)**-**(RQ8)** about the ILP method of DARS.

[26] http://bcert.org/projects/dars

(**RQ6**) How effective is DARS-ILP with respect to considering value dependencies?

(**RQ6.1**) How similar are solutions found by DARS-ILP to those found by the existing requirement selection methods?

(**RQ6.2**) What is the impact of using DARS-ILP on the overall value of software products?

(**RQ6.3**) What is the relationship between maximizing the accumulated value, expected value, and overall value of software products?

(**RQ6.4**) How effective is DARS-ILP in mitigating the value loss caused by ignoring (selecting) requirements with positive (negative) influence on the values of selected requirements?

4.5.1 Description of Study

To demonstrate practicality of the DARS-ILP method we studied a real-world software project referred to as PMS-III. As depicted in Figure 4.6 our study began with identification of value dependencies and modeling those dependencies. Then, we performed requirement selection using the PCBK, SBK, and DARS-ILP methods to find optimal configurations of PMS-III based on the sales records of different configurations of previous releases of PMS-III. The configurations found by the PCBK, SBK, and DARS-ILP methods for different price levels are shown in Figure 4.11.

For the configurations found by the PCBK, SBK, and DARS-ILP methods the accumulated value (AV), expected value (EV), and overall value (OV) were computed to compare the performance of the PCBK, SBK, and DARS-ILP methods for different price levels. This helped stakeholders of PMS-III to find, for different price levels, configurations of PMS-III with lower risk of value loss.

Table 4.4 lists the requirements of PMS-III and their estimated and expected values in $[1, 20]$. The expected value of each requirement r_i, denoted by $E(v_i)$ was computed by multiplying the frequency of the presence of r_i in the configurations sold prior to our study ($p(r_i)$) by its estimated value v_i.

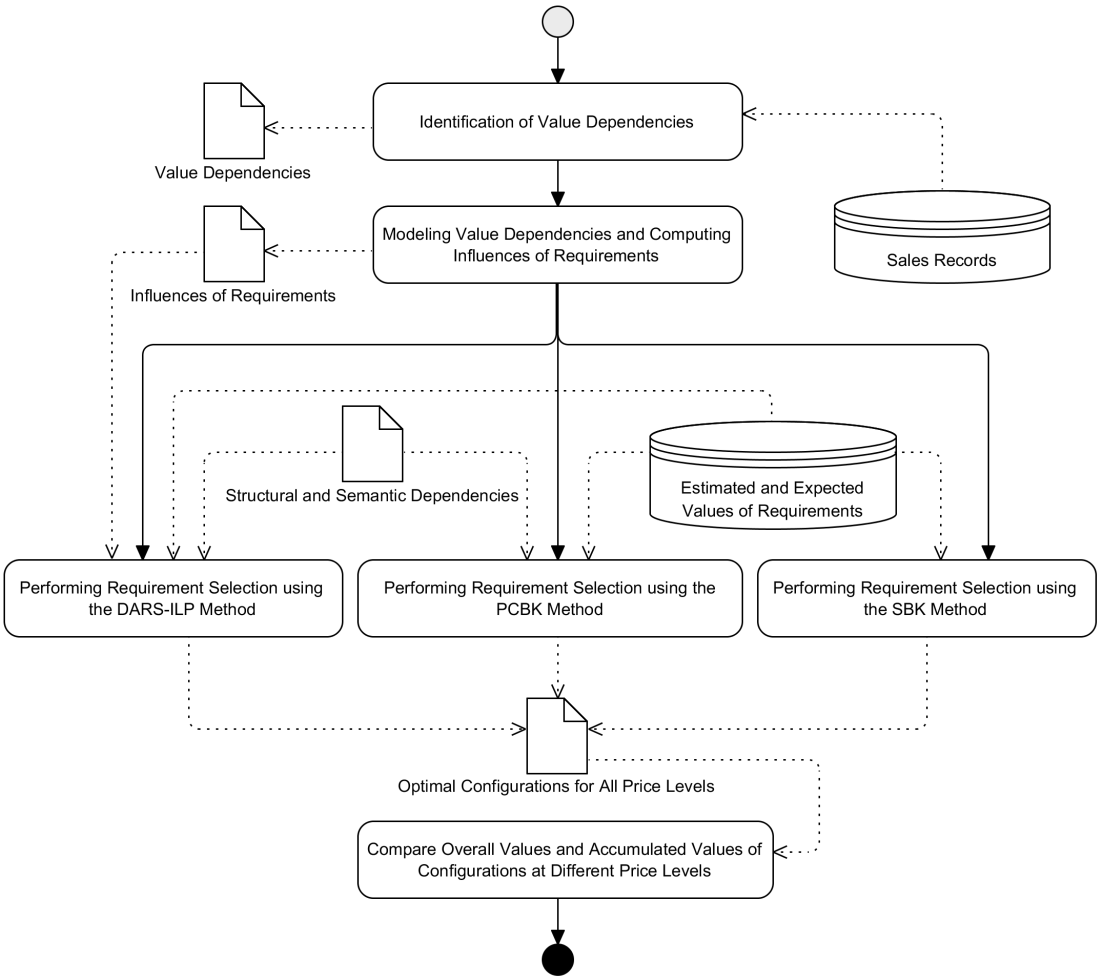


FIGURE 4.6: The case study design.

TABLE 4.3: The estimated and expected values of the requirements of PMS-III.

r_i	$p(r_i)$	v_i	$E(v_i)$	r_i	$p(r_i)$	v_i	$E(v_i)$
r_1	00.94	10.00	09.43	r_{15}	00.58	08.00	04.64
r_2	01.00	20.00	20.00	r_{16}	00.82	10.00	08.24
r_3	00.37	05.00	01.85	r_{17}	00.12	10.00	01.19
r_4	00.98	17.00	16.61	r_{18}	00.51	15.00	07.59
r_5	00.88	06.00	05.28	r_{19}	00.67	20.00	13.41
r_6	00.91	20.00	18.30	r_{20}	00.20	20.00	04.09
r_7	00.82	15.00	12.36	r_{21}	00.14	15.00	02.05
r_8	01.00	09.00	09.00	r_{22}	00.33	20.00	06.59
r_9	00.97	20.00	19.43	r_{23}	00.88	20.00	17.61
r_{10}	00.76	16.00	12.18	r_{24}	01.00	01.00	01.00
r_{11}	00.57	20.00	11.36	r_{25}	00.24	05.00	01.19
r_{12}	01.00	12.00	12.00	r_{26}	00.36	01.00	00.36
r_{13}	00.76	08.00	06.09	r_{27}	00.97	05.00	04.86
r_{14}	00.45	14.00	06.28				
Sum	-	192.00	160.17	-	-	150.00	72.82

4.5.2 Identification and Modeling of Dependencies

Precedence Dependencies in PMS-III

To account for the precedence dependencies among the requirements of PMS-III and their value implications, requirement dependencies of type *Requires* and *Conflicts-With* were extracted (Figure 4.7) from the development artifacts of PMS-III and formulated using (4.54)-(4.63), (4.71)-(4.80), and (4.95)-(4.104) in the optimization models of the PCBK, SBK, and DARS-ILP methods respectively.

Moreover, stakeholders of the PMS-III specified that the presence of either r_2 or r_6 is always essential to integrity of different configurations of PMS-III. To account for this, we introduced (4.53) to the optimization models of the PCBK, SBK, and DARS-ILP methods, where x_i denotes whether r_i is selected ($x_i = 1$) or not ($x_i = 0$).

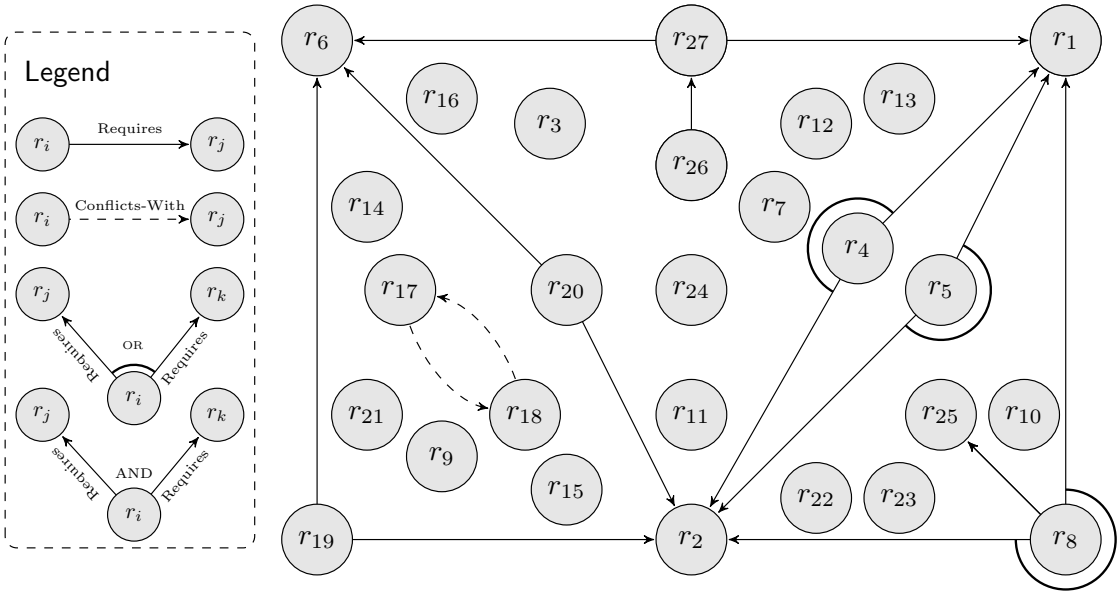


FIGURE 4.7: The precedence dependency graph of requirements of PMS-III.

Value Dependencies in PMS-III

To specify value dependencies among requirements of PMS-III, we first collected sales records of different configurations of PMS-III as explained earlier. Then the Eells measure of causal strength was computed for all pairs of requirements using Algorithm 4.1 to identify the strengths and qualities of causal relations among requirements as explained in Section 4.2.3.

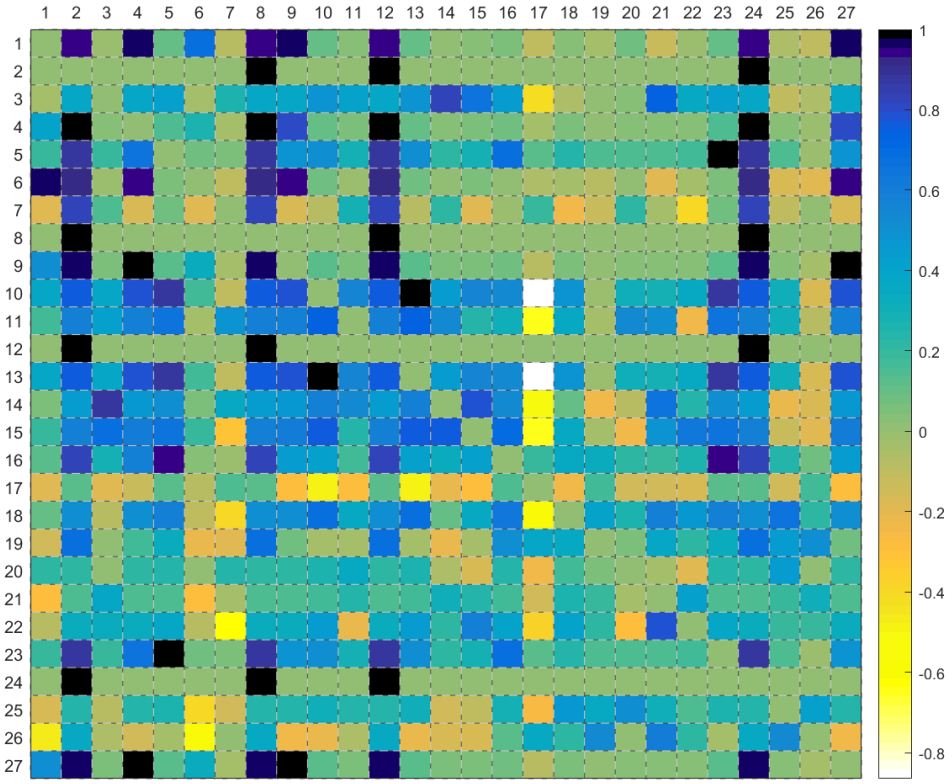


FIGURE 4.8: Explicit value dependencies among requirements of PMS-III. A cell at row i and column j denotes quality and strength of a value dependency from requirement r_i to r_j.

The significances of the identified relations were subsequently tested using the Odds Ratio at confidence level 95% as explained in Section 4.2.4. The strengths and qualities of explicit value dependencies were finally computed using the significant causal relations found and the fuzzy membership function of Figure 4.4(a) as given by (4.5)-(4.6).

Algorithm 4.2 was used to infer implicit value dependencies and compute the overall strengths of positive and negative value dependencies in the value dependency graph (VDG) of requirements. The influences of requirements on the values of each other were subsequently computed by (4.13).

Figure 4.8 shows the qualities and strengths of explicit value dependencies. The color of a cell at row i and column j specifies the quality and the strength of a value dependency from r_i to r_j. Colors associated with positive (negative) numbers denote positive (negative) dependencies. Also, zero denotes the absence of any dependency. Analogously, the positive or negative influences of requirements on the values of each other are depicted in Figure 4.9.

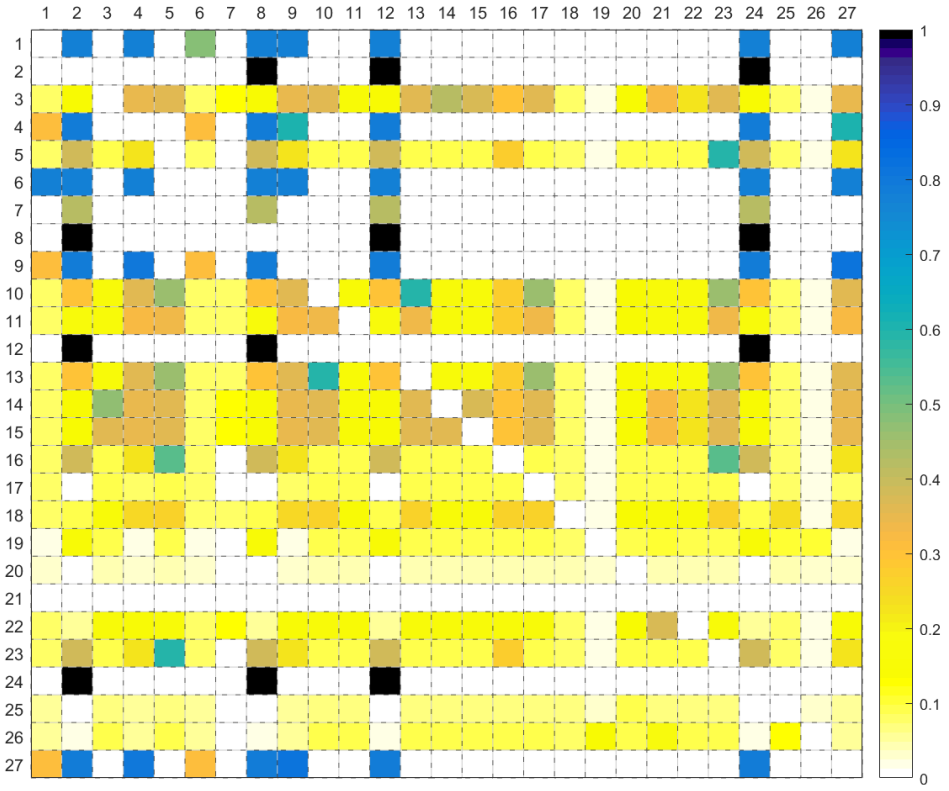


FIGURE 4.9: Influences of PMS-III requirements on the value of each other. A cell at row i and column j denotes quality and strength of the influence of r_j on r_i.

4.5.3 Performing Requirement Selection

This section demonstrates the effectiveness of DARS-ILP in considering value dependencies in software requirement selection compared to the existing requirement selection methods. The PCBK, SBK, and DARS-ILP methods, as explained earlier, find optimal subsets of requirements based on their corresponding optimality criteria. As given by (4.50) the PCBK method considers the estimated values of requirements while the SBK method, as in (4.67), accounts for user preferences for requirements by considering the expected values of requirements rather than their mere estimated values. Finally, the DARS-ILP method factors in both user preferences and value dependencies among requirements as given by (4.84)-(4.107). The expected values of requirements and value dependencies among those requirements are computed based on user preferences for requirements achieved from the Pre-TSP sales records of PMS-III as depicted in Figure 4.6.

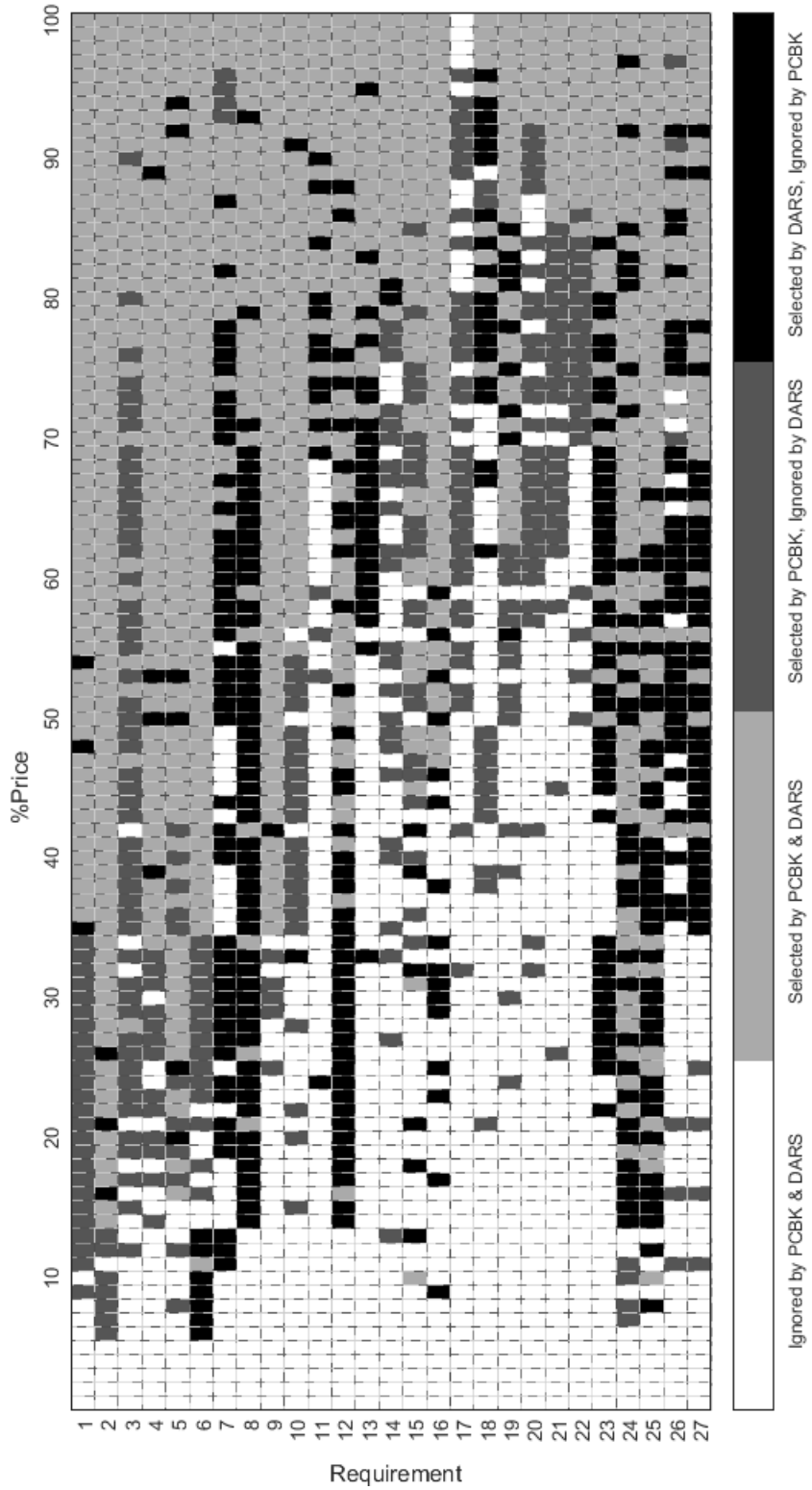


FIGURE 4.10: Comparing the requirement subsets found by the DARS-ILP and PCBK methods for different price levels.

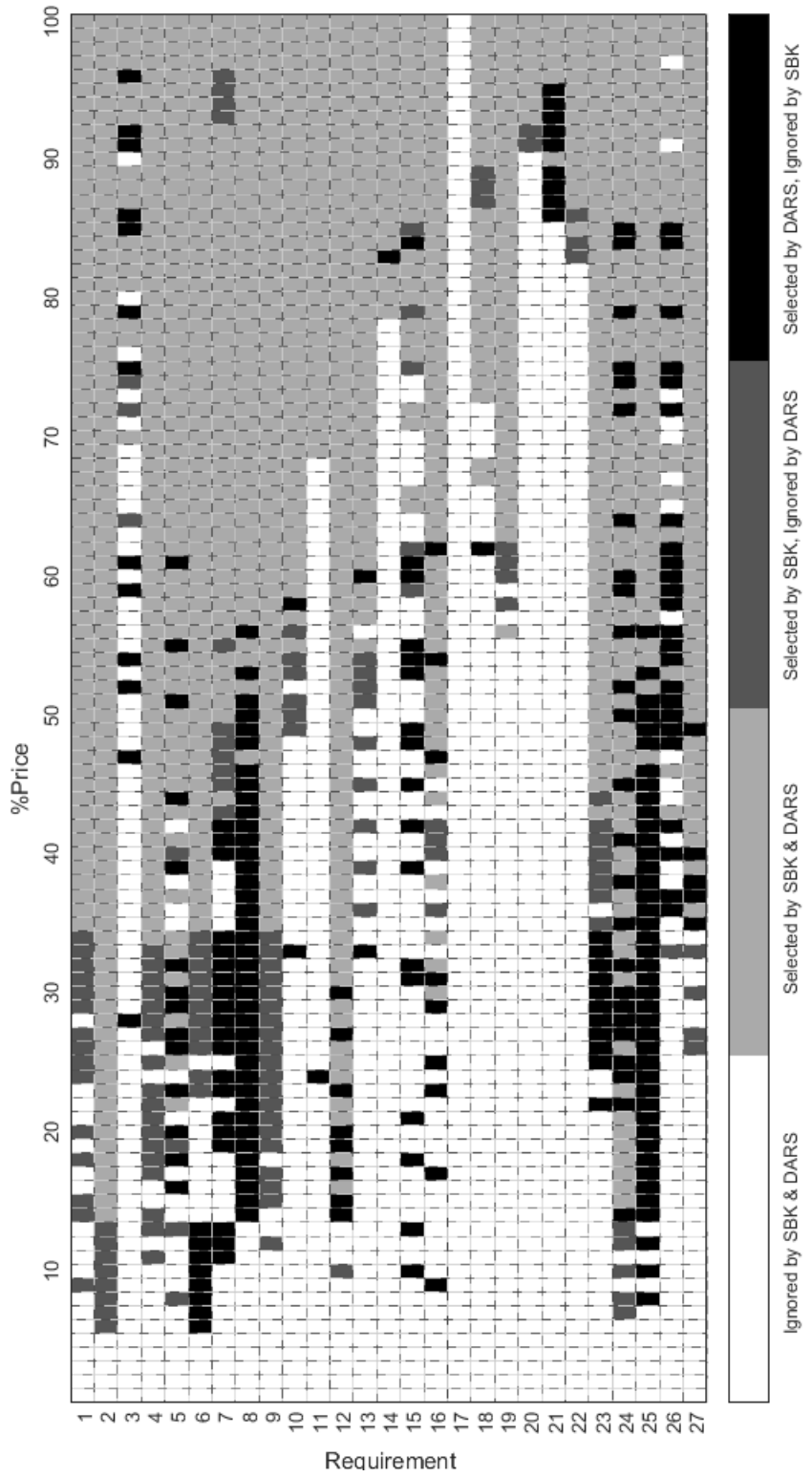


FIGURE 4.11: Comparing the requirement subsets found by the DARS-ILP and SBK methods for different price levels.

Price was determined as a major constraint for requirement selection as different configurations of PMS-III had been released at different price levels, during the Pre-TSP and TSP, to cope with the needs of different users [147]. Constraints (4.51), (4.68), and (4.85) hence were added to the optimization models of PCBK, SBK, and DARS-ILP methods respectively to contain the price of different configurations of PMS-III within their corresponding price limits. This essentially converted the problem to a variation of the Bounded Knapsack Problem. $\gamma \in \mathbb{R}^+$ denotes the price limit and v_i specifies the estimated value of a requirement r_i. Also x_i specifies whether a requirement r_i is selected ($x_i = 1$) or not ($x_i = 0$). We omitted (2.25) from the optimization model of the SBK method as considering the sales diversification is beyond the scope of this thesis. The concept of diversification was explained in detail in Section 2.2.5.

$$\text{Maximize } \sum_{i=1}^{27} x_i v_i \tag{4.50}$$

$$\text{Subject to } \sum_{i=1}^{27} v_i x_i \leq \gamma \tag{4.51}$$

$$x_i \in \{0,1\}, \qquad i = 1,...,27 \tag{4.52}$$

$$x_2 + x_6 = 1 \tag{4.53}$$

$$x_4 \leq x_1 + x_2 \tag{4.54}$$

$$x_5 \leq x_1 + x_2 \tag{4.55}$$

$$x_8 \leq x_1 + x_2 \tag{4.56}$$

$$x_8 \leq x_{25} \tag{4.57}$$

$$x_{17} \leq (1 - x_{18}) \tag{4.58}$$

$$x_{18} \leq (1 - x_{17}) \tag{4.59}$$

$$x_{19} \leq x_2 \tag{4.60}$$

$$x_{19} \leq x_6 \tag{4.61}$$

$$x_{20} \leq x_2 \tag{4.62}$$

$$x_{20} \leq x_6 \tag{4.63}$$

$$x_{26} \leq x_{27} \tag{4.64}$$

$$x_{27} \leq x_1 \tag{4.65}$$

$$x_{27} \leq x_6 \tag{4.66}$$

Moreover, (4.53),(4.70), and (4.94) were added to the optimization models of the PCBK, SBK, and DARS-ILP methods respectively to ensure that at least one of the requirements r_2 and r_6 is selected as an essential requirement of PMS-III. Also (4.54)-(4.66), (4.71)-(4.83), and (4.95)-(4.107) formulate precedence dependencies (requires and conflicts-with) of Figure 4.7 in the optimization models of the PCBK, SBK, and DARS-ILP methods respectively. Model (4.50)-(4.66) thus presents the formulation of the optimization model of the PCBK method for the requirements of PMS-III. Similarly, (4.67)-(4.83) and (4.84)-(4.107) give the optimization models of the SBK and DARS-ILP methods for the requirements of PMS-III.

$$\text{Maximize } \sum_{i=1}^{27} x_i E(v_i) \tag{4.67}$$

$$\text{Subject to } \sum_{i=1}^{27} v_i x_i \leq \gamma \tag{4.68}$$

$$x_i \in \{0,1\}, \qquad i = 1,...,27 \tag{4.69}$$

$$x_2 + x_6 = 1 \tag{4.70}$$

$$x_4 \leq x_1 + x_2 \tag{4.71}$$

$$x_5 \leq x_1 + x_2 \tag{4.72}$$

$$x_8 \leq x_1 + x_2 \tag{4.73}$$

$$x_8 \leq x_{25} \tag{4.74}$$

$$x_{17} \leq (1 - x_{18}) \tag{4.75}$$

$$x_{18} \leq (1 - x_{17}) \tag{4.76}$$

$$x_{19} \leq x_2 \tag{4.77}$$

$$x_{19} \leq x_6 \tag{4.78}$$

$$x_{20} \leq x_2 \tag{4.79}$$

$$x_{20} \leq x_6 \tag{4.80}$$

$$x_{26} \leq x_{27} \tag{4.81}$$

$$x_{27} \leq x_1 \tag{4.82}$$

$$x_{27} \leq x_6 \tag{4.83}$$

$$\text{Maximize} \sum_{i=1}^{27} x_i E(v_i) - y_i E(v_i) \tag{4.84}$$

$$\text{Subject to} \sum_{i=1}^{27} v_i x_i \leq \gamma \tag{4.85}$$

$$\theta_i \geq \left(\frac{|I_{i,j}| + (1 - 2x_j) I_{i,j}}{2} \right), \qquad i \neq j = 1, ..., 27 \tag{4.86}$$

$$-g_i \leq x_i \leq g_i, \qquad i = 1, ..., 27 \tag{4.87}$$

$$1 - (1 - g_i) \leq x_i \leq 1 + (1 - g_i), \qquad i = 1, ..., 27 \tag{4.88}$$

$$-g_i \leq y_i \leq g_i, \qquad i = 1, ..., 27 \tag{4.89}$$

$$-(1 - g_i) \leq (y_i - \theta_i) \leq (1 - g_i), \qquad i = 1, ..., 27 \tag{4.90}$$

$$0 \leq y_i \leq 1, \qquad i = 1, ..., 27 \tag{4.91}$$

$$0 \leq \theta_i \leq 1, \qquad i = 1, ..., 27 \tag{4.92}$$

$$x_i, g_i \in \{0, 1\}, \qquad i = 1, ..., 27 \tag{4.93}$$

$$x_2 + x_6 = 1 \tag{4.94}$$

$$x_4 \leq x_1 + x_2 \tag{4.95}$$

$$x_5 \leq x_1 + x_2 \tag{4.96}$$

$$x_8 \leq x_1 + x_2 \tag{4.97}$$

$$x_8 \leq x_{25} \tag{4.98}$$

$$x_{17} \leq (1 - x_{18}) \tag{4.99}$$

$$x_{18} \leq (1 - x_{17}) \tag{4.100}$$

$$x_{19} \leq x_2 \tag{4.101}$$

$$x_{19} \leq x_6 \tag{4.102}$$

$$x_{20} \leq x_2 \tag{4.103}$$

$$x_{20} \leq x_6 \tag{4.104}$$

$$x_{26} \leq x_{27} \tag{4.105}$$

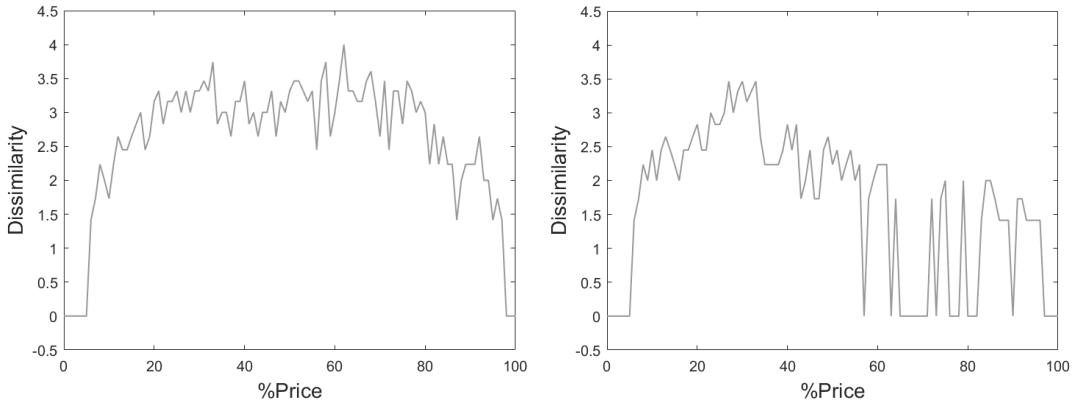
$$x_{27} \leq x_1 \tag{4.106}$$

$$x_{27} \leq x_6 \tag{4.107}$$

Selection tasks were performed for different price levels (%Price $= \{1, ..., 100\}$, Price $= \frac{\%\text{Price}}{100} \times 342$) using the PCBK, SBK, and DARS-ILP methods to find optimal subsets of requirements of PMS-III (optimal configurations of PMS-III). Optimal configurations found by the PCBK, SBK, and DARS-ILP methods were compared based on their similarities, accumulated values, expected values, and overall values to answer **(RQ6)** and its subquestions. The binary knapsack (BK) method (Section 2.2.1) and the Increase-Decrease method (Section 2.2.3) were not used in requirement selection tasks as the former ignores precedence dependencies resulting in violation of the precedence constraints and finding infeasible solutions while the latter relies on manual estimations of values of requirement subsets and does not provide any formal way to specify the amounts of the increased or decreased values of the requirement subsets as detailed in Section 2.2.3. Selections were performed using the callable library ILOG CPLEX 12.6.2 on a windows machine with a Core i7-2600 3.4 GHz processor and 16 GB of RAM.

Similarities of Solutions

In this section we compare PCBK, SBK, and DARS-ILP based on their selection patterns to answer **(RQ6.1)**. Figure 4.12 depicts dissimilarities between the requirement subsets found by the DARS-ILP and those found by PCBK/SBK based on *Euclidean Distance*. While notable at all price levels, these dissimilarities decreased for highly expensive (%Price $\rightarrow$ 100) or very cheap (%Price $\rightarrow$ 0) configurations of PMS-III.



(a) DARS-ILP vs. PCBK (b) DARS-ILP vs. SBK

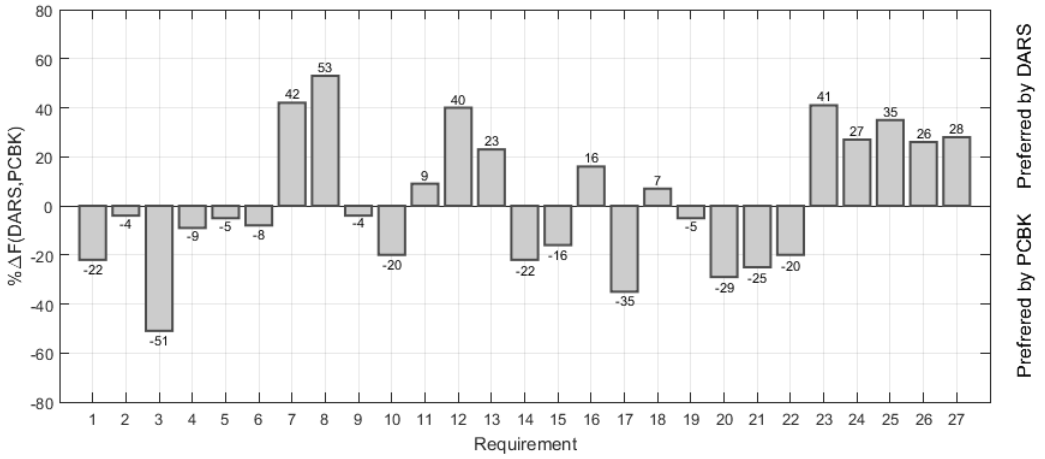
FIGURE 4.12: Dissimilarities between requirement subsets (solutions) found by DARS-ILP and those found by the PCBK/SBK methods.

The reason is expensive configurations of PMS-III comprise most requirements thus reducing the chances that requirements with positive influences on the values of the selected requirements are ignored. Moreover, as given by Figure 4.9, there are no negative influences among requirements. Hence, similarities between solutions found by the DARS-ILP method and those found by the PCBK and SBK methods increase for expensive configurations of PMS-III. For cheaper configurations, price constraint limits the solution space for the PCBK, SBK, and DARS-ILP methods especially preventing the DARS-ILP method from utilizing its advantage in considering value dependencies. This resulted in more similarities between the solutions found by the DARS-ILP method and those found by the PCBK and SBK methods for very cheap configurations of PMS-III.

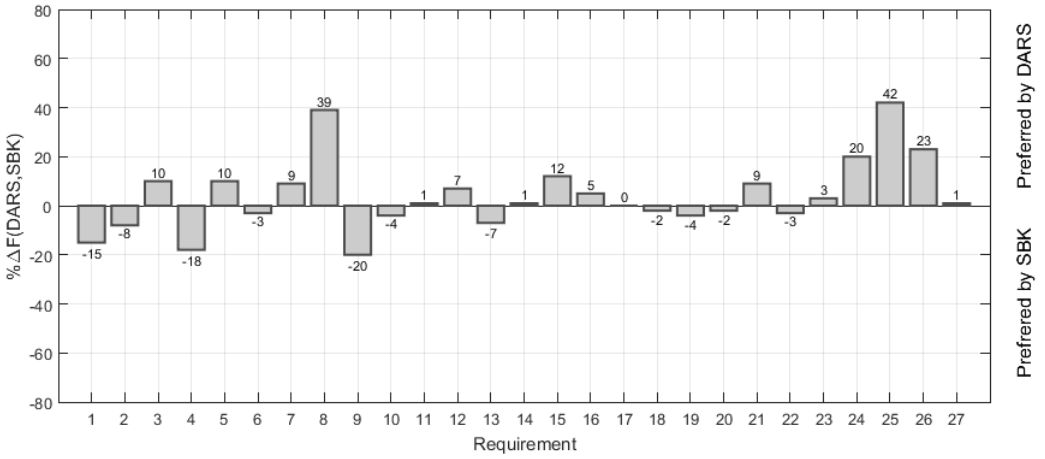
Finally we observed from Figure 4.13(a) and Figure 4.13(b) that requirement subsets (solutions) found by the DARS-ILP method were more similar to the solutions found by the SBK method than similar to the solutions found by the PCBK method. The reason is as explained before both DARS-ILP and SBK consider user preferences while the PCBK method ignores those preferences.

Figure 4.13 provides more insights into **(RQ6.1)** by comparing the selection patterns of the PCBK, SBK, and DARS-ILP methods in 100 different selection tasks performed at different price levels ($\%\text{Price} = \{1, 2, ..., 100\}$). For a given requirement r_i, $\%\text{F}_i(m_j)$ in Figure 4.13 specifies the percentages of the selection tasks in which r_i is selected by the requirement selection method m_j. Hence, $\%\Delta\text{F}_i(m_j, m_k) = \%\text{F}_i(m_j) - \%\text{F}_i(m_k) > 0$ states that the percentages of the selection tasks where r_i is selected by the selection method m_j is higher than the percentages of the selection tasks where r_i is selected by m_k. Similarly, $\%\Delta\text{F}_i(m_j, m_k) = \%\text{F}_i(m_j) - \%\text{F}_i(m_k) < 0$ states that r_i is more frequently selected by m_k compared to m_j. m_j and m_k can be any of the selection methods used in our selection tasks.

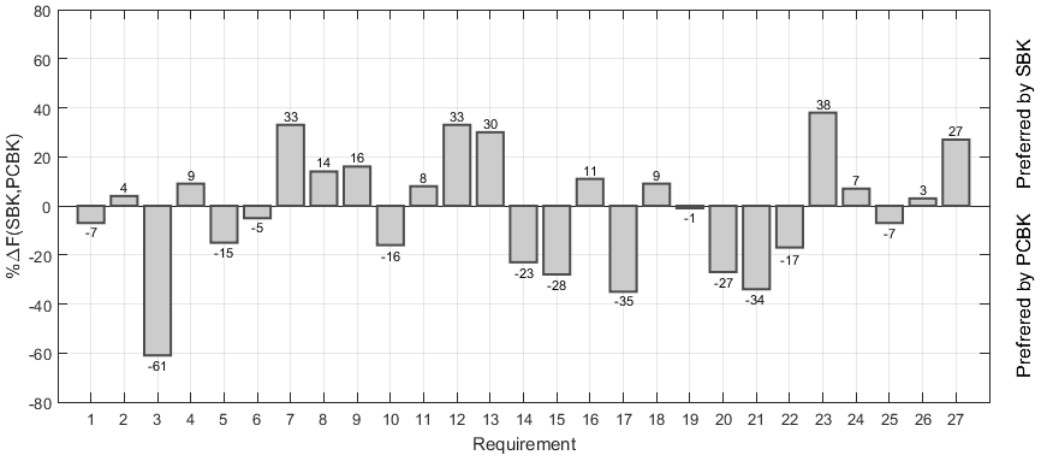
The results of our selection tasks showed (Figure 4.13) that requirements with significant influence on the values of pricey requirements were more frequently preferred by DARS-ILP compared to the PCBK and SBK methods. This was more visible for requirements r_8, r_{12}, r_{24}, and r_{27} when in Figure 4.13(a) and for requirements r_8, r_{12}, r_{24} in Figure 4.13(b).



(a) DARS-ILP vs. PCBK



(b) DARS-ILP vs. SBK



(c) SBK vs. PCBK

FIGURE 4.13: Selection patterns of PCBK, SBK, and DARS-ILP methods for requirements of PMS-III at different price levels (%Price $\in \{1, 2, ..., 100\}$). For a requirement r_i, denoted by i on the x-axis, and requirement selection methods m_j and m_k, $\%\Delta F_i(m_j, m_k) = \%F_i(m_j) - \%F_i(m_k)$, where $\%F_i(m_j)$ and $\%F_i(m_k)$ give the percentage of the selection tasks in which r_i is selected by the m_j and m_k methods respectively.

Requirement r_8 for instance was more frequently preferred by DARS-ILP compared to the PCBK and SBK methods as the optimization model of DARS-ILP considers the fact that r_8 has a significant positive influence on the values of several valuable requirements including r_2, r_4, r_6, and r_{12} (Figure 4.9).

Similarly, r_{24} has a significant (positive) influence on the values of requirements r_2, r_6, r_8, andr_{12}. r_{25} however, was more frequently selected by DARS-ILP as r_8 requires r_{25} (Figure 4.7) and r_8 is frequently selected by DARS-ILP due to its significant impact on valuable requirements. As such, selecting r_8 requires the presence of r_{25} in software. DARS-ILP and SBK however were frequently selected r_{12} and r_{27} as these two requirements are almost always preferred by users. This was not the case for PCBK as it ignores user preferences.

Selection patterns in Figure 4.13 showed that when a decision was to be made regarding the presence or absence of a requirement r_i in a configuration of PMS-III, PCBK only took into account the estimated value of r_i ignoring user preferences. SBK on the other hand, considered user preferences for r_i by evaluating the expected value of r_i rather than merely its accumulated value.

DARS-ILP, however, evaluated the expected value of r_i while considering the impact of r_i on the values of other requirements. More similarities were thus observed among configurations found by the SBK and DARS-ILP as both methods took into account user preferences. On the contrary, dissimilarities were more visible when SBK and DARS-ILP/PCBK were compared as demonstrated in Figure 4.13(a) and Figure 4.13(c).

Impact of DARS-ILP on the Overall Value

(RQ6.2) is answered by comparing the percentages of overall values ($\%OV = (OV/342) \times 100$), accumulated values ($\%AV = (AV/342) \times 100$), and estimated values ($\%EV = (EV/342) \times 100$) provided by the PCBK, SBK, and DARS-ILP methods for 100 selection tasks, each performed at a specific price level ($\%Price = \{1, 2, .., 100\}$), as shown in Figures 4.14-4.16.

Our results show (Figure 4.14) that requirement subsets found by the DARS-ILP method provided higher or equal %OV in all selection tasks compared to the PCBK method. The reason is that the optimization model of PCBK only considers estimated values (prices) of requirements while entirely ignoring user preferences and value dependencies among requirements. On the contrary, DARS-ILP not only takes into account user preferences, but considers the influences of requirements on the values of each other by integrating value dependencies into requirement selection.

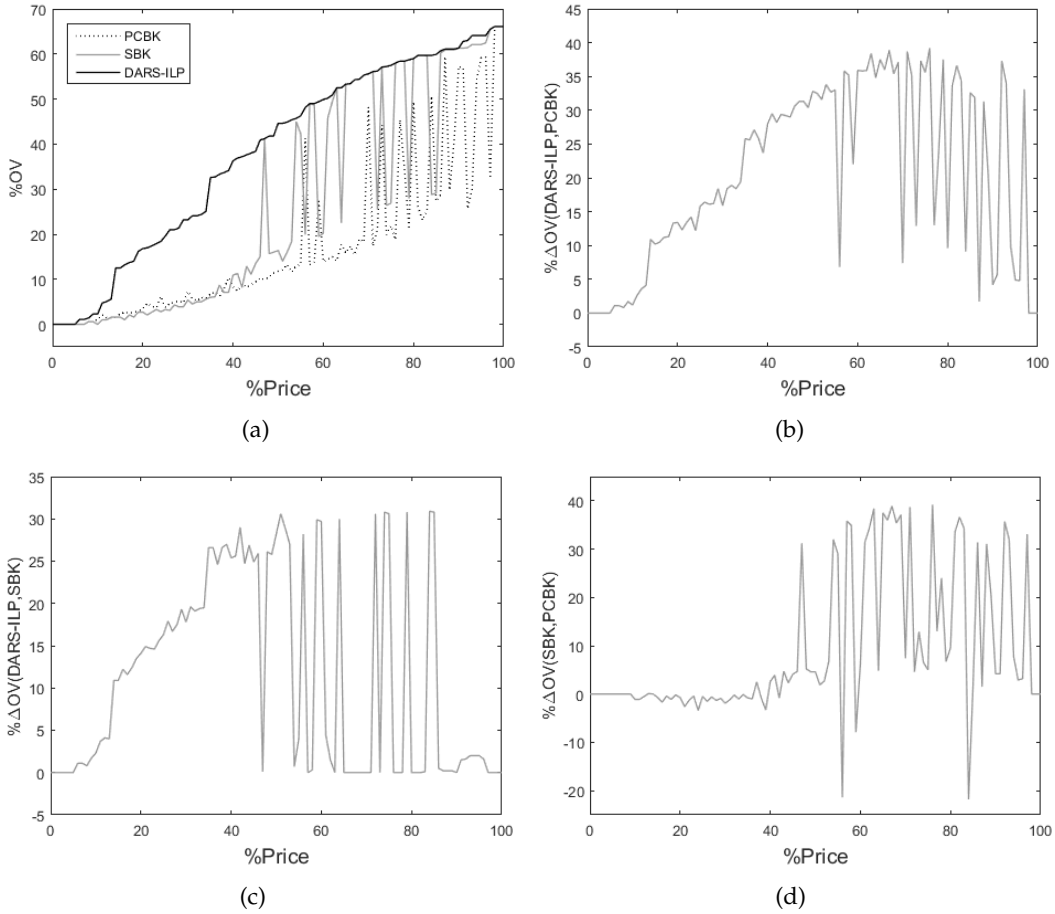


FIGURE 4.14: Comparing the overall values provided by the PCBK, SBK, and DARS-ILP methods at different price levels. $\%\Delta OV(m_j, m_k) = \%OV(m_j) - \%OV(m_k)$, where m_j and m_k denote the selection methods which are being compared against each other.

For a given price, $\%OV$ of the requirement subset (solutions) found by the SBK method was, for most price levels, higher than $\%OV$ of the solution provided by the PCBK

method but still less than or equal to the overall value of the solution found by DARS-ILP. The reason is that even though the SBK method does not consider value dependencies, it still accounts for user preferences, similar to DARS-ILP, by optimizing the expected values of selected requirements. This results in more similarities between the configurations of PMS-III found by the SBK method and those found by the DARS-ILP method as discussed in Section 4.5.3.

We observed in Figure 4.14(b) that the gap between the %OV achieved from the DARS-ILP method and the PCBK/SBK method was notable in almost all selection tasks performed at different price levels. But the gap reduced to almost negligible for highly expensive (%Price $\rightarrow$ 100) or very cheap (%Price $\rightarrow$ 0) configurations of PMS-III. The reason is, on one hand, there are no negative influences among the requirements of PMS-III (Figure 4.9) and, on the other hand, expensive configurations of PMS-III comprise most requirements, which reduces the chances that requirements with positive influence are ignored by PCBK/SBK.

That increases similarities between the expensive configurations of PMS-III found by the PCBK/SBK and DARS-ILP methods as discussed in Section 4.5.3. For cheaper configurations, the price-constraint limited the solution space in all the PCBK, SBK, and DARS-ILP methods and specially prevented the DARS-ILP method from utilizing its advantage in considering value dependencies. The price constraint further reduced the gap between %AV provided by the DARS-ILP and PCBK/SBK methods (Figures 4.15) in the selection tasks.

We further observed insignificant differences amongst the accumulated values provided by the selection methods experimented in this study as shown in Figure 4.15. The reason is that the price constraints (4.51), (4.68), and (4.85) in the optimization models of the PCBK, SBK, and DARS-ILP methods respectively contain the accumulated values of the solutions found by those models. The price constraints (4.51), (4.68), and (4.85) are needed to factor out the interplay between the price and sales as explained earlier.

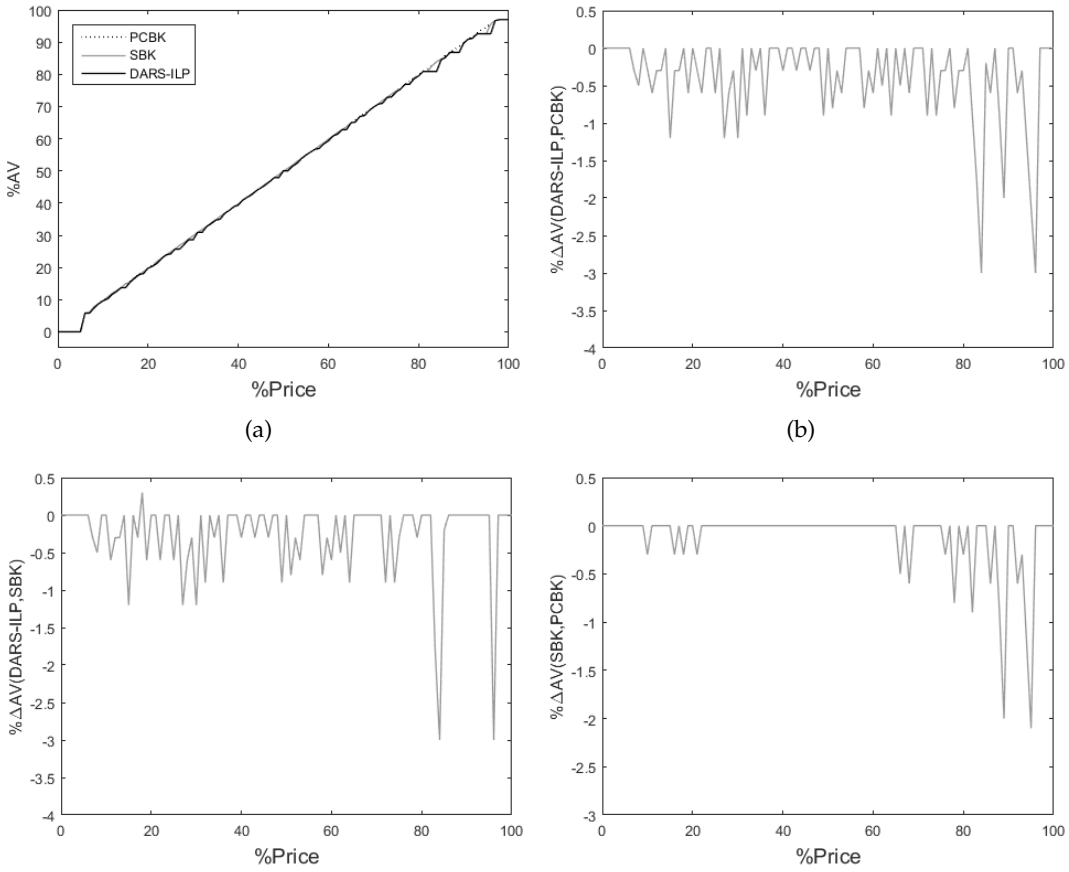


FIGURE 4.15: Comparing the accumulated values provided by the PCBK, SBK, and DARS-ILP methods at different price levels. $\%\Delta\text{AV}(m_j, m_k) = \%\text{AV}(m_j) - \%\text{AV}(m_k)$, where m_j and m_k denote the selection methods compared against each other.

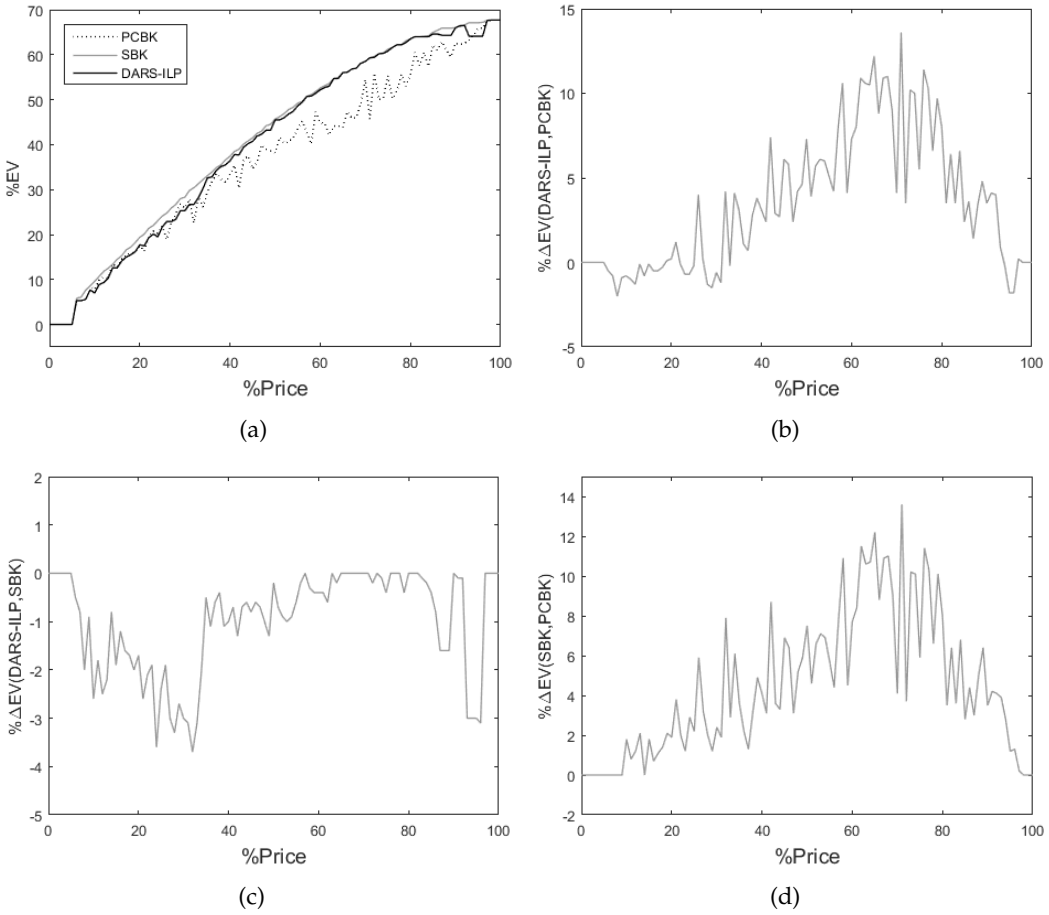


FIGURE 4.16: Comparing the expected values provided by the PCBK, SBK, and DARS-ILP methods at different price levels. $\%\Delta\text{EV}(m_j, m_k) = \%\text{EV}(m_j) - \%\text{EV}(m_k)$, where m_j and m_k denote the selection methods being compared against each other.

Moreover, the expected values of the requirement subsets found by the SBK method were higher than those found by the DARS-ILP and PCBK methods in all selection tasks (for all price levels) as shown in Figure 4.16(c) and Figure 4.16(d) respectively.

The expected values of requirement subsets found by the DARS-ILP method were higher than those of the requirement subsets found by the PCBK method in most selection tasks. This can be seen in Figure 4.16(b).

In some of the selection tasks, however, the expected values of the requirement subsets found by the PCBK method were higher than those found by the DARS-ILP method even though the PCBK method does not account for user preferences whatsoever. The reason is that the DARS-ILP method optimizes the overall value of a requirement subset (solution), which, simultaneously accounts for both user preferences and value dependencies as give by (4.84). Hence in some cases the DARS-ILP method may find solutions with lower expected values as taking into account value dependencies may be in conflict with maximizing the expected values of a requirement subset.

Understanding the Conflicting Objectives

To answer **(RQ6.3)**, we compared the overall values (Figure 4.14), accumulated values (Figure 4.15), and expected values (Figure 4.16) of the requirement subsets found by the PCBK, SBK, and DARS-ILP methods in different requirement selection tasks performed at different price levels. From Figure 4.14 and Figure 4.15 it can be seen that maximizing the accumulated value (AV) of a selected subset of requirements conflicts with maximizing the overall value (OV) of that subset. This can be specially seen in Figure 4.14(b) and Figure 4.15(b), where in several selection tasks, choosing requirement subsets with higher %AV by the PCBK method (Figure 4.15) reduced the overall value.

Maximizing the expected value of a requirement subset also conflicts with optimizing its overall value as the former may result in ignoring requirements with lower

expected values even if they have a significant influence on the values of other requirements. That will increase the penalty of ignoring requirements with positive influences on the values of selected requirements, as given by (4.16), resulting in lower overall value. This can be seen by comparing Figure 4.14(c) and Figure 4.16(c).

Mitigating the Value-Loss

Ignoring value dependencies among requirements can pose a major risk to the economic worth of PMS-III configurations and eventually result in value loss as given by (4.16). This risk can be be measured by the gap between the expected value of software and its overall value, which accounts for value dependencies, as depicted in Figure 4.17. As shown in this figure, for each selection task performed at a specific price level, the gap between the expected value and the overall value of the PMS-III configuration found by the DARS-ILP method was notably smaller than the gaps between the %EV and %OV provided by the PCBK method. Hence, using DARS-ILP contributed to a smaller risk of value loss in different configurations of PMS-III.

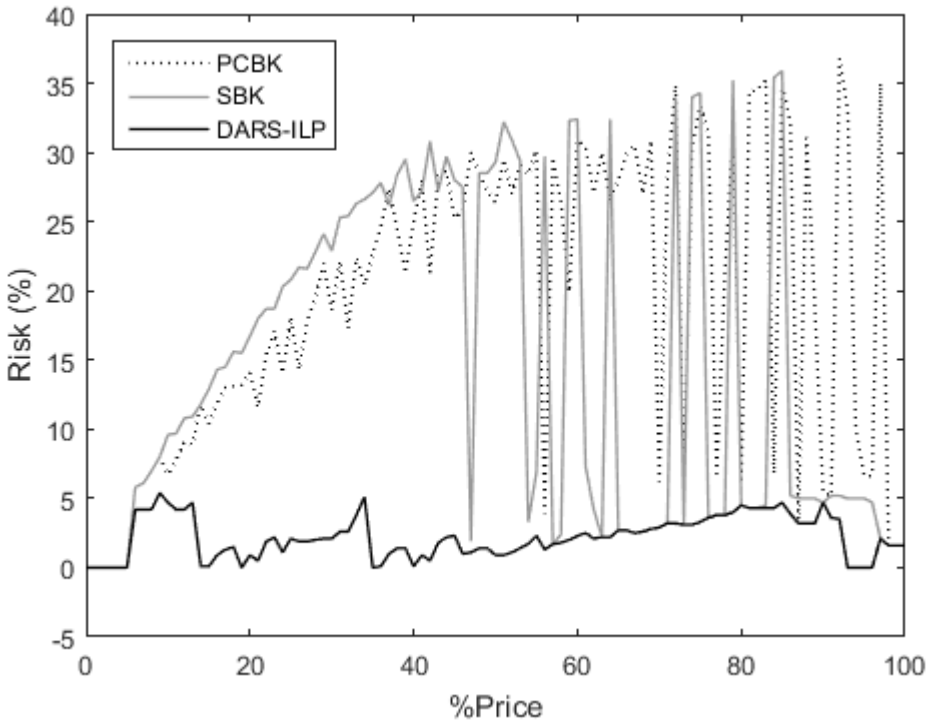


FIGURE 4.17: Risk of value loss for configurations of PMS-III found by the PCBK, SBK, and DARS-ILP methods at different price levels.

We observed (Figure 4.17) that the risk of value loss for different configurations of PMS-III found by DARS-ILP were under 5% while the risk of value loss for configurations found by the PCBK and SBK methods inconsistently changed from almost negligible for cheaper configurations to around 37% in more expensive configurations. In most configurations found by the PCBK and SBK, an inconsistent pattern of "the higher the price the higher the risk of value loss" was observed suggesting a higher risk for expensive configurations of PMS-III. The risk of value loss for the configurations found by the SBK method, however, converged to those found by the DARS-ILP method for %Price $\geq$ 88 as both methods tended to choose more similar configurations (Figure 4.12). This concludes our answer to **(RQ6.4)**.

4.6 Simulations

Studying real-world software products, as in Section 4.5, helps understand practical aspects of requirement selection methods. However, that may not be sufficient by itself to understand the impact of different levels of value dependencies and precedence dependences on requirement selection. To address this, we simulated requirement selection for different levels of value dependencies and precedence dependencies in different scenarios.

As discussed earlier, the optimization model of DARS-ILP considers value dependencies while optimization models of the BK, PCBK, and SBK methods ignore value dependencies. Also, both DARS-ILP and SBK consider user preferences by evaluating the expected values of requirements as given by (4.27) and (2.24). However, the SBK and DARS-ILP methods may find similar solutions in several cases as both methods consider user preferences. This may specially happen when value dependencies are found among frequently preferred requirements with high estimated values.

The interplay between considering user preferences and considering value dependencies thus may interfere with studying the impact of value dependencies on the effectiveness of the selection methods being investigated. To avoid this and further highlight the differences between considering and ignoring value dependencies in the simulated scenarios, we modify the optimization model of DARS-ILP, as given by

(4.108)-(4.118), by using the definition of overall value given in Section 3.3.2, which substitutes the expected values of requirements $(E(v_i))$ with their corresponding estimated values (v_i) thus factoring out user preferences from simulations.

$$\text{Maximize } \sum_{i=1}^{n} x_i v_i - y_i v_i \tag{4.108}$$

$$\text{Subject to } \sum_{i=1}^{n} c_i x_i \leq b \tag{4.109}$$

$$\begin{cases} x_i \leq x_j & r_j \text{ precedes } r_i \\ x_i \leq 1 - x_j & r_i \text{ conflicts with } r_j,\ i \neq j = 1, ..., n \end{cases} \tag{4.110}$$

$$\theta_i \geq \left(\frac{|I_{i,j}| + (1 - 2x_j) I_{i,j}}{2} \right), \qquad i \neq j = 1, ..., n \tag{4.111}$$

$$-g_i \leq x_i \leq g_i, \qquad i = 1, ..., n \tag{4.112}$$

$$1 - (1 - g_i) \leq x_i \leq 1 + (1 - g_i), \qquad i = 1, ..., n \tag{4.113}$$

$$-g_i \leq y_i \leq g_i, \qquad i = 1, ..., n \tag{4.114}$$

$$-(1 - g_i) \leq (y_i - \theta_i) \leq (1 - g_i), \qquad i = 1, ..., n \tag{4.115}$$

$$0 \leq y_i \leq 1, \qquad i = 1, ..., n \tag{4.116}$$

$$0 \leq \theta_i \leq 1, \qquad i = 1, ..., n \tag{4.117}$$

$$x_i, g_i \in \{0, 1\}, \qquad i = 1, ..., n \tag{4.118}$$

When the same substitution is performed on the optimization model of SBK, the objective function of the method becomes identical to that of the optimization model of the PCBK method as given by (4.119)-(4.122). Moreover, as given by Table 2.2, the stochastic binary knapsack method is the same as the PCBK method as far as considering aspects of value dependencies is concerned. That is the SBK method only considers the value implications of precedence dependencies. As such, we avoid repeating the simulations for SBK.

$$\text{Maximize} \sum_{i=1}^{n} x_i v_i \tag{4.119}$$

$$\text{Subject to} \sum_{i=1}^{n} c_i x_i \leq b \tag{4.120}$$

$$\begin{cases} x_i \leq x_j & r_j \text{ precedes } r_i \\ x_i \leq 1 - x_j & r_i \text{ conflicts with } r_j,\ i \neq j = 1, ..., n \end{cases} \tag{4.121}$$

$$x_i \in \{0, 1\}, \qquad i = 1, ..., n \tag{4.122}$$

Simulations were carried out for requirements of a system referred to as PMS-II with 27 requirements with the estimated values and costs given in Table 4.4. The estimated costs and values are scaled into $[0, 20]$. Value dependencies and precedence dependencies among these requirements were generated randomly as will be explained in the following.

TABLE 4.4: The estimated costs and values of the requirements of PMS-II.

r_i	c_i	v_i	r_i	c_i	v_i
r_1	05.00	10.00	r_{15}	15.00	08.00
r_2	20.00	20.00	r_{16}	13.00	10.00
r_3	00.00	04.00	r_{17}	14.00	06.00
r_4	10.00	17.00	r_{18}	03.00	10.00
r_5	01.00	03.00	r_{19}	10.00	20.00
r_6	20.00	20.00	r_{20}	07.00	20.00
r_7	06.00	15.00	r_{21}	12.00	15.00
r_8	05.00	09.00	r_{22}	15.00	20.00
r_9	16.00	20.00	r_{23}	08.00	20.00
r_{10}	10.00	16.00	r_{24}	02.00	05.00
r_{11}	04.00	20.00	r_{25}	10.00	00.00
r_{12}	03.00	10.00	r_{26}	00.00	00.00
r_{13}	05.00	06.00	r_{27}	01.00	00.00
r_{14}	07.00	08.00			
Sum	112.00	178.00	-	110.00	134.00

The Java API of IBM CPLEX was used to implement the optimization models of the BK, PCBK, and DARS-ILP methods and run them using the callable library ILOG CPLEX 12.6.2 on a windows machine with a Core i7-2600 3.4 GHz processor and 16 GB of RAM.

Requirement selection, then, was performed for different percentages of budget (%Budget $=$ $\{1, ..., 100\}$), value dependency levels (VDL $\in [0, 1]$), negative value dependency levels (NVDL $\in [0, 1]$), precedence dependency levels (PDL $\in [0, 1]$), and negative precedence dependency levels (NPDL $\in [0, 1]$). Table 4.5 lists our simulations settings. This section answers the following research questions with regard to the performance of DARS-ILP in different scenarios in Table 4.5.

(**RQ7**) How is the performance of DARS-ILP affected by changing value dependencies, precedence dependencies and project constraints?

(**RQ7.1**) What is the impact of the value dependencies on the performance of DARS-ILP in the presence of various budget constraints?

(**RQ7.2**) What is the impact of the negative value dependencies on the performance of DARS-ILP in the presence of various budget constraints?

(**RQ7.3**) What is the impact of the precedence dependencies on the performance of DARS-ILP in the presence of various budget constraints?

(**RQ7.4**) What is the impact of the negative precedence dependencies on the performance of DARS-ILP in the presence of various budget constraints?

(**RQ7.5**) What is the impact of the negative value dependencies in highly, moderately, or loosely interdependent value dependency graphs?

(**RQ7.6**) What is the impact of the negative precedence dependencies in highly or loosely interdependent precedence dependency graphs?

To simulate value dependencies for a desired VDL and NVDL, uniformly distributed random numbers in $[-1, 1]$ were generated, where the sign and magnitude of each number specified the quality and strength of its corresponding explicit value dependency respectively. In a similar way, for a desired PDL and NPDL, random numbers in $\{-1, 0, 1\}$ were generated where 1 (-1) specified a positive (negative) precedence dependency and 0 denoted the absence of any precedence dependency from a requirement r_i to a requirement r_j.

Furthermore, percentages of the overall value of selected requirements (%OV= $\frac{\text{OV}}{\sum_{i=1}^{27} v_i}$) were used to measure the performance of the simulated selection methods. The performance of the BK method was, however, arbitrary in all simulations as the BK method does not consider precedence dependencies and therefore in many cases, depending on the PDL and NPDL, violates those dependencies giving infeasible solutions with no value (%OV $= 0$). On the other hand, the PCBK method enhances the BK method by considering precedence dependencies. As such, the PCBK method always outperforms the BK method giving higher or equal %OV. Hence, we have mostly avoided discussing the results related to the BK method. Instead, we focus on comparing the performance of the PCBK and DARS-ILP methods. Moreover, Increase-Decrease methods were not simulated as they do not specify how to achieve the amount of the increased or decreased values as explained in Section 2.2.3.

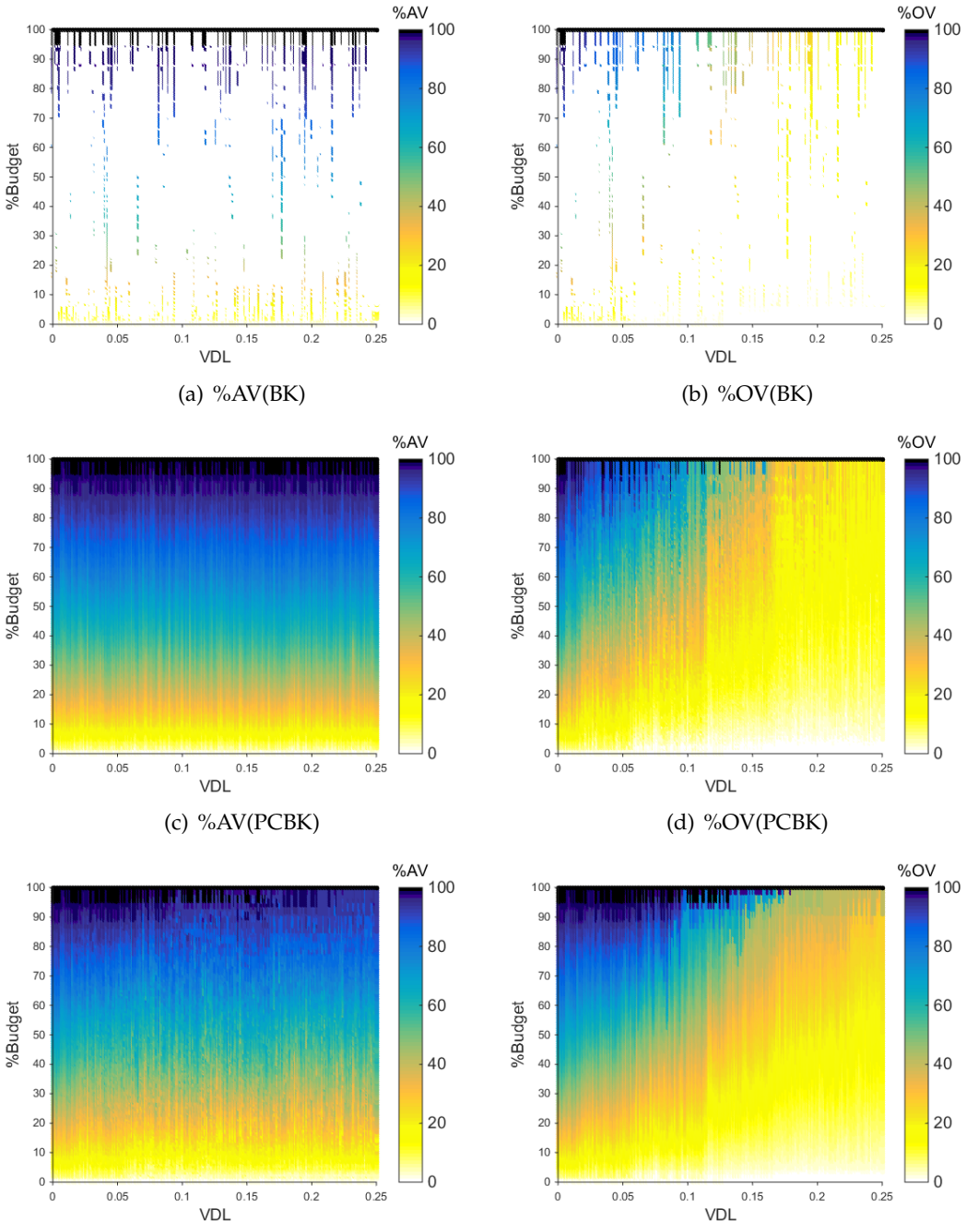
TABLE 4.5: Performance simulations for the BK, PCBK, and DARS-ILP methods.

Simulation	%Budget	VDL	NVDL	PDL	NPDL
I	[0,100]	[0,1]	0.00	0.02	0.00
II	[0,100]	0.15	[0,1]	0.02	0.00
III	[0,100]	0.15	0.00	[0,1]	0.00
IV	[0,100]	0.15	0.00	0.02	[0,1]
V	95	[0,1]	[0,1]	0.02	0.00
VI	95	0.15	0.00	[0,1]	[0,1]

4.6.1 Value Dependencies vs Budget

To answer **(RQ7.1)**, simulations were carried out for various percentages of budget (%Budget $\in [0,100]$) and value dependency levels ($VDL = [0,1]$) with settings of Simulation I in Table 4.5. Figure 4.18 shows the percentages of the accumulated value (%AV) and overall value (%OV) achieved from simulated selection methods.

As expected, the BK method violated precedence dependencies and generated infeasible solutions with %AV=%OV= 0 in most simulations (Figure 4.18(a) and Figure 4.18(b)). The reason is the BK method does not consider precedence dependencies. It is clear that when there is no precedence dependencies (PDL=0), the BK and the PCBK methods will be equal.



(a) %AV(BK)

(b) %OV(BK)

(c) %AV(PCBK)

(d) %OV(PCBK)

(e) %AV(DARS-ILP)

(f) %OV(DARS-ILP)

FIGURE 4.18: %AV and %OV achieved for Simulation I (%Budget vs. VDL).

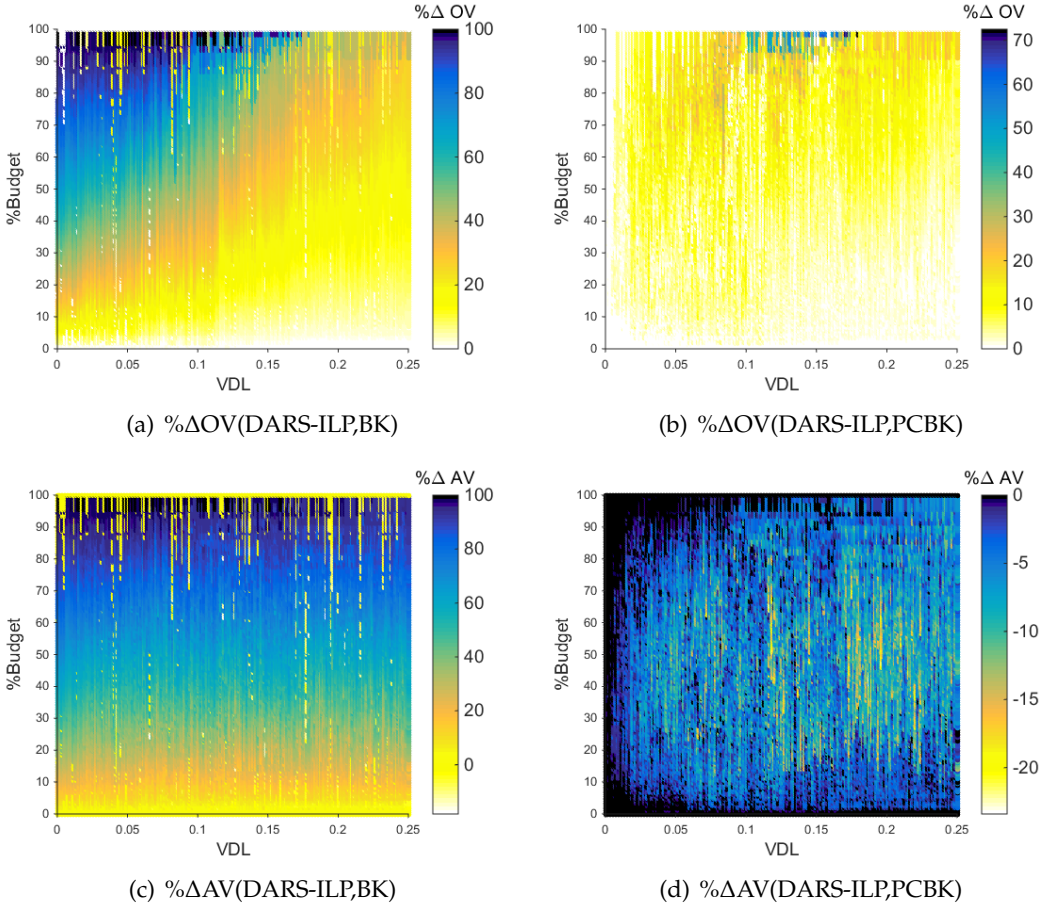
Nonetheless, our simulations showed that for %Budget=100 and NPDL=0, the BK method provided %OV=%AV=100. This, however, could not be achieved in the presence of negative precedence dependencies. The reason is in such cases the BK method had to exclude some of the requirements from the optimal subset to avoid violating negative precedence dependencies. We further observed (Figure 4.18) that for a given %Budget and NVLD=0 increasing VDL generally decreased %OV achieved by all selection methods. The reason is increasing VDL increases the chances that the positive dependencies of a requirement are excluded. This, as in (4.16), results in increasing penalties of selected requirements and consequently reducing %OV.

Figure 4.18 and Figure 4.19, also, show that DARS-ILP gives higher %OV for all VDLs and %Budget compared to the PCBK and BK methods. The reason is DARS-ILP considers the value dependencies as well as the value implications of precedence dependencies while the BK method ignores dependencies all together, and the PCBK method only considers precedence dependencies. Figure 4.19 compares %OV and %AV provided by the DARS-ILP method against those of the BK and PCBK methods for various %Budget and VDLs. We have $\%\Delta OV(m_1,m_2)=\%OV(m_1)-\%OV(m_2)$ and $\%\Delta AV(m_1,m_2)=\%AV(m_1)-\%AV\ (m_2)$ for a pair of selection methods m_1 and m_2.

Our results demonstrated that the DARS-ILP method outperformed the BK and PCBK methods by providing higher %OV as given in Figure 4.19. Moreover, we observed that finding a subset of requirements with the highest accumulated value conflicts with finding a subset with the highest overall value. In other words, to maximize OV and AV are conflicting objectives. This is demonstrated in many points in the graphs of Figure 4.19(d) and Figure 4.19(b), where for a given %Budget and VDL, $\%\Delta AV(\text{DARS-ILP,PCBK}) < 0$ while $\%\Delta OV(\text{DARS-ILP,PCBK}) > 0$.

4.6.2 Negative Value Dependencies vs Budget

To answer the research question **(RQ7.2)**, simulations for various values of budget and NVDL were performed with settings of Simulation II in Table 4.5. It was observed that increasing NVDL resulted in arbitrary change in %OV achieved from PCBK and DARS-ILP methods.



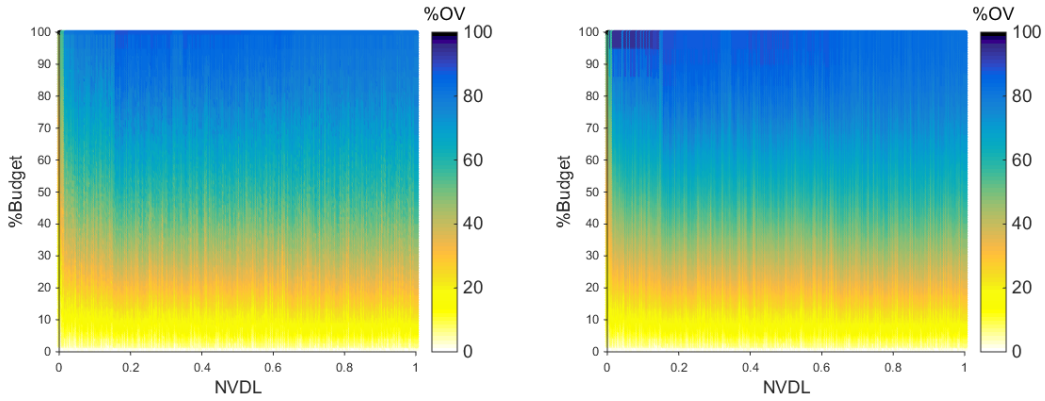
(a) %ΔOV(DARS-ILP,BK) (b) %ΔOV(DARS-ILP,PCBK)

(c) %ΔAV(DARS-ILP,BK) (d) %ΔAV(DARS-ILP,PCBK)

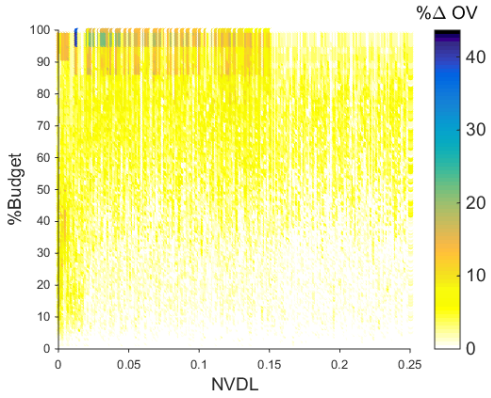
FIGURE 4.19: %OV and %AV achieved for Simulation I (%Budget vs. VDL).

This is seen in (4.13) where increasing NVDL may arbitrarily increase or decrease the penalty of selecting or ignoring requirements depending on the strengths of positive and negative dependencies and the structure of the value dependency graph of the requirements. Moreover, for %Budget=100 and $NVDL \rightarrow 0$, we observed (Figure 4.20), for both PCBK and DARS-ILP methods, that the maximum %OV can be achieved. The reason is, as expected, in such cases positive value dependencies do not matter as no requirement has to be excluded from the optimal subset of requirements due to the presence of sufficient budget.

Nonetheless, as we move to the right on the x axis in the graphs of (Figure 4.20), NVDL increases and thus, even with sufficient budget, the maximum %OV cannot be achieved. The reason is that, as explained earlier, in such cases even selecting requirements may reduce the values of other requirements due to the negative value dependencies among those requirements.



(a) %OV(PCBK) (b) %OV(DARS-ILP)



(c) %ΔOV(DARS-ILP,PCBK)

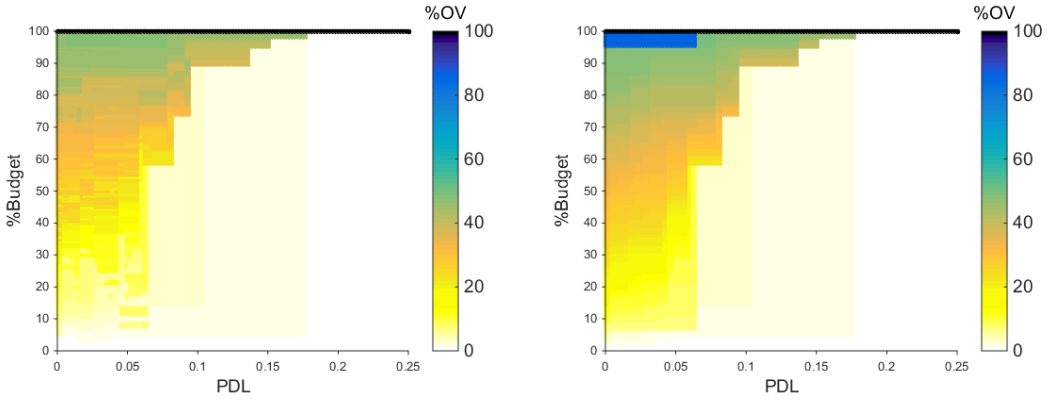
FIGURE 4.20: %OV and %ΔOV achieved for Simulation II (%Budget vs. NVDL).

We, further, observed (Figure 4.20(c)), consistent with other simulations, that the DARS-ILP method always provided higher %OV compared to the PCBK method as expected. On the other hand, it was observed that the BK method failed to find feasible solutions in most simulations due to violating precedence dependencies.

4.6.3 Precedence Dependencies vs Budget

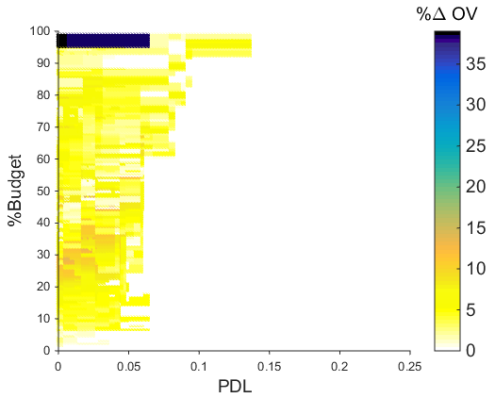
To answer **(RQ7.3)**, simulations for various %Budget and PDLs were performed with settings of Simulation III in Table 4.5. For a given VDL, we observed that increasing PDL generally resulted in decreasing the %OV achieved from the DARS-ILP method (Figure 4.21(b)). The reason is increasing PDL reduces the number of feasible solutions that maintain preceded dependencies. This is also described as the selection deficiency problem (SDP) [2] where the efficiency of selection models is limited by precedence dependencies.

A similar effect was observed for the PCBK method (Figure 4.21(a)). However, decreasing %OV with PDL increase did not monotonically occur when the PCBK method was used. In fact, increasing PDL resulted in higher %OV in some cases.



(a) %OV(PCBK)

(b) %OV(DARS-ILP)

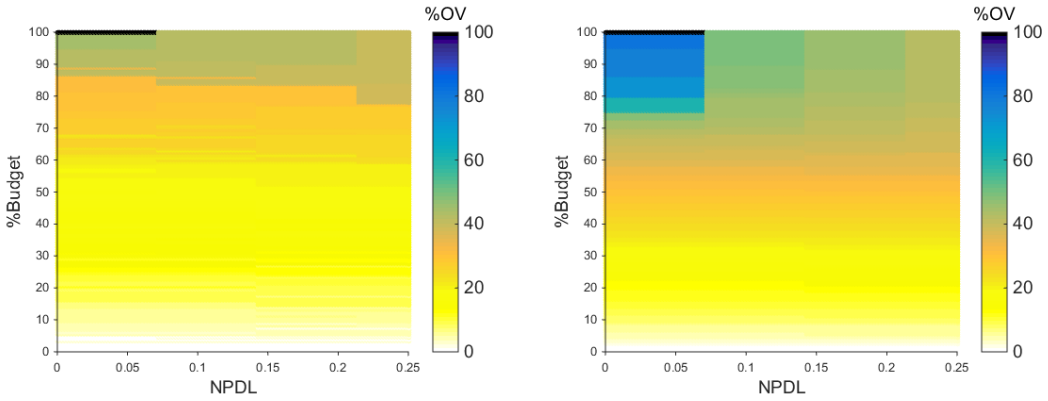


(c) %ΔOV(DARS-ILP,PCBK)

FIGURE 4.21: %OV and %ΔOV achieved for Simulation III (%Budget vs. PDL).

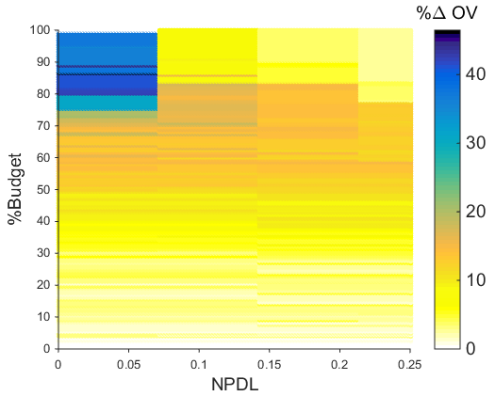
These arbitrary effects are more tangible in simulations with %Budget ≥ 70 and $vdl \leq 0.1$. The reason is the PCBK method ignores value dependencies. As such, even with fewer precedence constraints, the PCBK method may choose a solution with lower %OV. On the contrary, the DARS-ILP method managed to give higher %OV for smaller PDLs (Figure 4.21(b)).

Figure 4.21(c) shows the DARS-ILP method outperformed the PCBK method but, for PDL ≥ 0.15, the performances of the models converged. The reason is for larger PDLs the number of feasible solutions is significantly reduced in both models. That prevented the DARS-ILP method from exploiting value dependencies to provide solutions with higher %OV.



(a) %OV(PCBK)

(b) %OV(DARS-ILP)



(c) %ΔOV(DARS-ILP,PCBK)

FIGURE 4.22: %OV and %ΔOV achieved for Simulation IV (%Budget vs. NPDL).

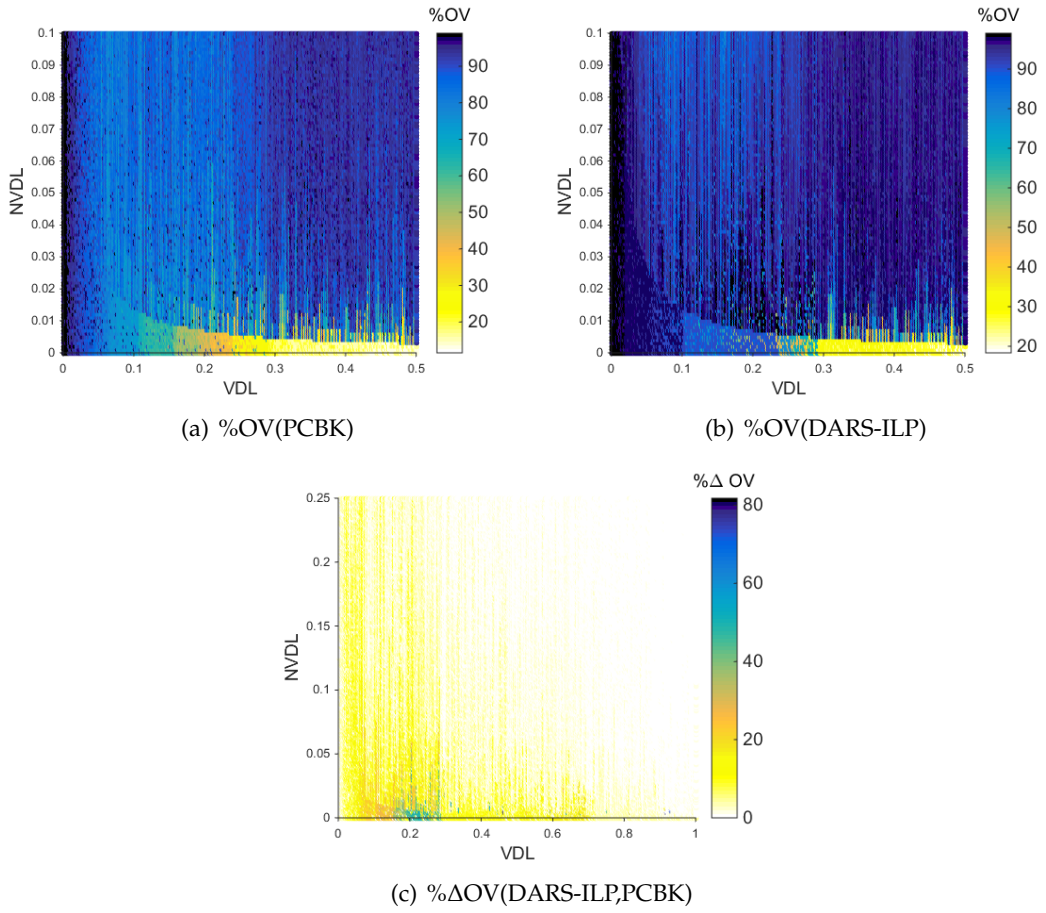
4.6.4 Negative Precedence Dependencies vs Budget

To answer **(RQ7.4)**, simulations for different %Budget and NPDLs were performed with settings of Simulation IV in Table 4.5.

For both PCBK and DARS-ILP methods increasing the NPDL simultaneously limited the number of feasible solutions and increased the chances that certain requirements are selected. The former resulted in decreasing the %OV while the latter increased the %OV (4.22). This resulted in an arbitrary change in %OV when NPDL increased.

As one example for (b), consider requirements r_1, r_2, r_3, where there are negative precede dependencies from r_1 to r_2 ($x_1 \leq (1 - x_2)$) and from r_2 to r_3, ($x_2 \leq (1 - x_3)$). Hence, selecting r_3 ($x_3 = 1$) results in ignoring r_2 ($x_2 = 0$), which increases the chances that r_1 is selected ($x_1 \leq 1$). Such effects, result in increasing the %OV even when NPDL increases. This is seen in several points in the graph of figure (4.22).

We, moreover, observed (Figure 4.22(c)) that the DARS-ILP method outperformed PCBK for up to around 46%. In addition, for $75 \leq$ %Budget < 100 and $NPDL \leq 0.15$, we observed that $\Delta(\text{PCBK}, \text{BK})$ and $\Delta(\text{DARS}, \text{PCBK})$ were the highest. The reason is the DARS-ILP method can find better solutions in the presence of fewer negative precedence constraints and more budget.



(a) %OV(PCBK)

(b) %OV(DARS-ILP)

(c) %ΔOV(DARS-ILP,PCBK)

FIGURE 4.23: %OV and %ΔOV achieved for Simulation V (NVDL vs. VDL).

4.6.5 Positive vs Negative Value Dependencies

To answer **(RQ7.5)**, simulations for various values of VDL and NVDL were performed with %Budget $=$ 95, PDL=0.02, and NPDL =0.0 as given by settings of Simulation V in Table 4.5. The impact of VDL is shown to differ for smaller and larger NVDLs. For smaller NVDLs ($NVDL \leq 0.01$) increasing VDL monotonically decreased the %OV provided by the selection methods. In other cases, however, increasing VDL was demonstrated to increase %OV, although such an increase was not monotonic.

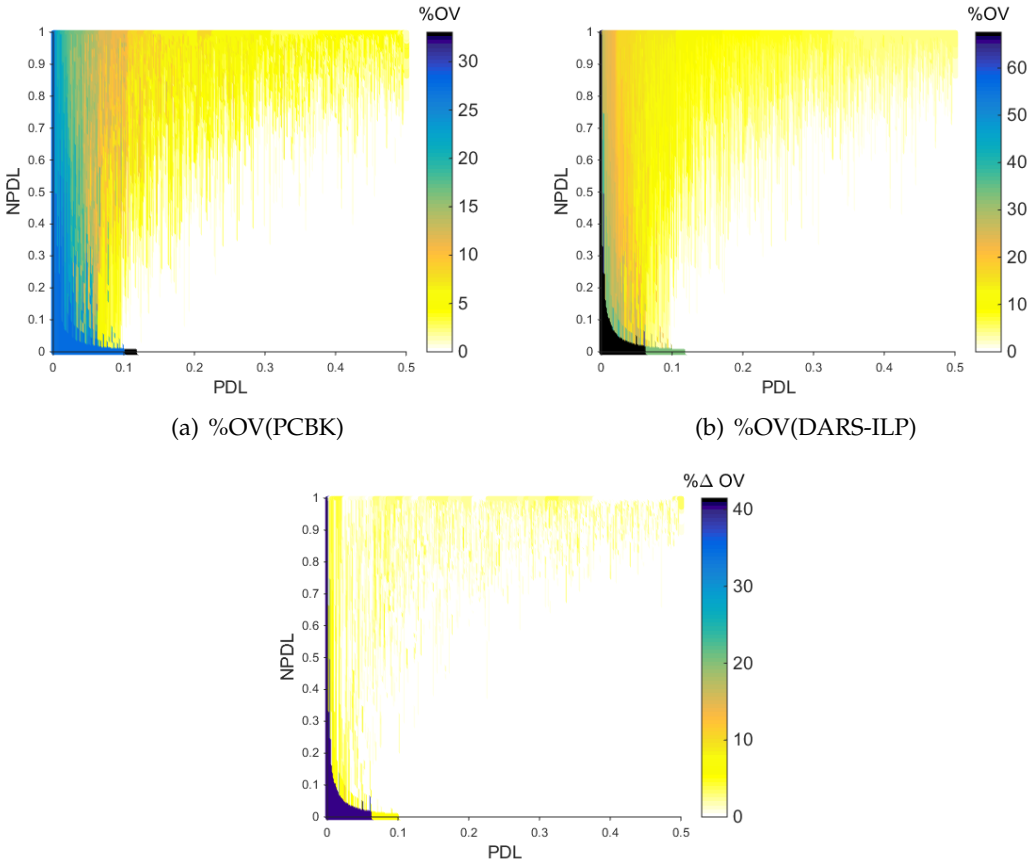
The reason is, as explained earlier, higher NVDLs increase the chances that simultaneous negative and positive value dependencies exist from a requirement r_i to r_j and therefore, negative value dependencies from r_i to $r_j : \rho(r_i, r_j)^{-\infty}$ mitigate the impact of positive dependencies from r_i to r_j: $\rho(r_i, r_j)^{+\infty}$ and vice versa. This reduces the overall influence of r_j on r_i based on (4.13). Hence, neither selecting nor ignoring r_j does not result in a significant loss in the value of r_i and the %OV of the selected subset of requirements.

4.6.6 Positive vs Negative Precedence Dependencies

To answer **(RQ7.6)**, simulations for different PDLs and NPDLs were performed with settings of Simulation VI in Table 4.5. Our simulations showed (Figure 4.24) that increasing PDL, in general, decreased the %OV provided by the PCBK and DARS-ILP methods. This decrease, nevertheless, was not monotonic in the presence of negative precedence dependencies (NPDL$\neq$ 0). The reason for this arbitrary impact of negative precedence dependencies was explained in detail in Section 4.6.4.

Our simulations showed that for any PDL there exists a threshold t, where PCBK and DARS do not give any value for NPDLs $< t$. These thresholds increased as the PDL increased. This is more visible for PDL $\geq$ 0.1. To further explain this, consider a requirement set $R = \{r_1, r_2\}$ with equal costs and equal values, where r_1 requires r_2 (positive precedence dependency from r_1 to r_2) and r_2 requires r_1. This means we have $PDL = 1$ and $NPDL = 0$. As such, for %Budget < 100, either or both of the r_1 and r_2 will have to be excluded from the optimal subset which results in violating precedence dependencies and therefore no feasible solution can be found resulting in %OV=0.

However, for $NPDl = 0.5$, one of the precedence dependencies will change to negative (conflicts-with). As this is performed randomly, we have either (a): r_1 requires r_2 AND r_2 conflicts with r_1 or (b): r_2 requires r_1 AND r_1 conflicts with r_2. It is clear that in either case (for Budget $\geq$ 50), at least one requirement (r_2 in (a) and r_1 in (b)) can be selected. For %Budget $= 100$, nevertheless, both r_1 and r_2 are selected in (a). This, clearly shows how, for a given PDL, increasing $NPDL$ can provide higher $\%OV$.



(a) %OV(PCBK)

(b) %OV(DARS-ILP)

(c) %ΔOV(DARS-ILP,PCBK)

FIGURE 4.24: %OV and %ΔOV achieved for Simulation VI (NPDL vs. PDL).

Last but not least, we observed that for simulations with PDL$\geq$ 0.1, the performance of the PCBK and DARS-ILP methods converged as both methods provided similar $\%OV$: Δ%OV(DARS,PCBK)$\rightarrow$ 0. This, however, was not the case in the presence of high levels of negative precedence dependencies.

The reason is a large number of precedence dependencies substantially reduce the number of feasible solutions impacting the performance of the DARS and PCBK methods. Nonetheless, increasing NPDL can increase the number of feasible solutions as explained above. Under such circumstances, it is clear that the DARS-ILP method can make better choices with regard to the %OV as it takes into account value dependencies in addition to the precedence dependencies.

4.7 Complexity and Scalability Analysis

This section evaluates the scalability of DARS-ILP for identification and modeling value dependencies as well as considering those dependencies in software requirement selection. We specially generate random datasets with different numbers of requirements (up to 3000) to investigate the scalability of the ILP model of DARS-ILP for different scenarios in relation to value and precedence dependencies among requirements. Simulations thus were designed to answer the following questions.

(**RQ8**) What is the overhead of identification and modeling of value dependencies in DARS-ILP?

(**RQ9**) How scalable is the ILP model of DARS-ILP?

(**RQ9.1**) Is the ILP model scalable to large scale requirement sets?

(**RQ9.2**) What is the impact of budget on runtime?

(**RQ9.3**) What is the impact of precedence dependencies on runtime?

(**RQ9.4**) What is the impact of value dependencies on runtime?

4.7.1 The Overhead of using DARS-ILP

Our proposed DARS-ILP method relies on the identification and modeling of value dependencies. These two processes hence constitute the main overhead of DARS. In the ILP method of DARS presented in this chapter, the identification of value dependencies from causal relations among user preferences is automated as explained in Section 4.2. The process, nevertheless, relies on computing the Eells measure [45] for pairs of the requirements. Algorithm 4.2 computes the Eells measure in $O(t \times n^2)$ for n requirements and t records of user preferences.

Precedence dependencies among requirements (e.g. requires, conflicts-with, AND, OR) on the other hand, are identified as part of the requirement analysis and inferred from the structure and/or semantic of a software product using automated or semi-automated techniques [148, 18]. This, is an inevitable aspect of software requirement analysis and is not specific to DARS-ILP.

Moreover, construction of a value dependency graph of requirements, inferring implicit value dependencies, and computing the influences of requirements using Algorithm 4.2 is of computational complexity of $O(n^3)$ as discussed earlier in Section 4.3. This concluded our answer to **(RQ8)**.

4.7.2 Scalability of the Optimization Model of DARS-ILP

The optimization model of the DARS-ILP method as given by (4.27)-(4.35) is scalable to datasets with a large number of requirements, different budget constraints, and various degrees of precedence/value dependencies. To demonstrate this, runtime simulations in Table 4.6 were carried out.

TABLE 4.6: Runtime Simulations for the optimization model of DARS-ILP

Simulation	Size	%Budget	VDL	NVDL	PDL	NPDL
1	[0,3000]	50	0.15	0.00	0.02	0.00
2	200	[0,100]	0.15	0.00	0.02	0.00
3	200	50	0.15	0	[0,1]	0.00
4	200	50	0.15	0.00	0.02	[0,1]
5	200	50	[0,1]	0.00	0.02	0.00
6	200	50	0.15	[0,1]	0.02	0.00

To simulate value dependencies for a desired VDL and NVDL, uniformly distributed random numbers in $[-1, 1]$ were generated, where the sign and magnitude of each number specified the quality and the strength of its corresponding explicit value dependency.

We used *Precedence Dependency Level* (PDL) and *Negative Precedence Dependency Level* (NPDL) as given by (4.123) and (4.124) to specify the degree of precedence dependencies in a precedence graph G with n nodes (requirements). k gives the total number of precedence dependencies while j denotes the number of negative precedence dependencies in (4.123) and (4.124) respectively.

$$PDL(G) = \frac{k}{{}^{n}P_2} = \frac{k}{n(n-1)} \tag{4.123}$$

$$NPDL(G) = \frac{j}{k} \tag{4.124}$$

For a given PDL and NPDL, random numbers in $\{-1, 0, 1\}$ were generated where 1 (-1) specified a positive (negative) precedence dependency and 0 denoted the absence of any precedence dependency from a requirement r_i to r_j. Simulations were carried out using the callable library ILOG CPLEX 12.6.2 on a windows machine with a Core i7-2600 3.4 GHz processor and 16 GB of RAM.

(RQ9.1) is answered by runtime simulation 1, which evaluates the runtime of the optimization model of the DARS-ILP method for different numbers of requirements (Figure 4.25). We observed that increasing the number of requirements increased, as expected, the runtime of the optimization model of the DARS-ILP method. Nonetheless, for requirement sets with up to 750 ($n \leq 750$) requirements, the model managed to find the optimal solution in less than a minute. For $750 < n \leq 2000$ the runtime was above one minute but did not exceed two hours. Finally, for $2000 < n \leq 3000$ it took hours before selection was completed.

On the other hand, our results for Simulation 2 demonstrated (Figure 4.26) that the runtime of the optimization model of the DARS-ILP method increased with budget increase. The reason is with more budget, more requirements can be selected which results in a larger solution space. As such, it may take longer for the optimization model of the DARS-ILP method to find an optimal subset of requirements. This answers **(RQ9.2)**.

To answer **(RQ9.3)** we simulated the requirement selection for various precedence dependency levels (PDLs). Our results (Figure 4.27) demonstrated, in general, that the runtime of the optimization model of the DARS-ILP method increased when PDL increased. The reason is increasing PDL limits the number of choices for the optimization model of the DARS-ILP method as the model needs to respect precedence dependencies. Hence, it takes longer for the optimization to complete.

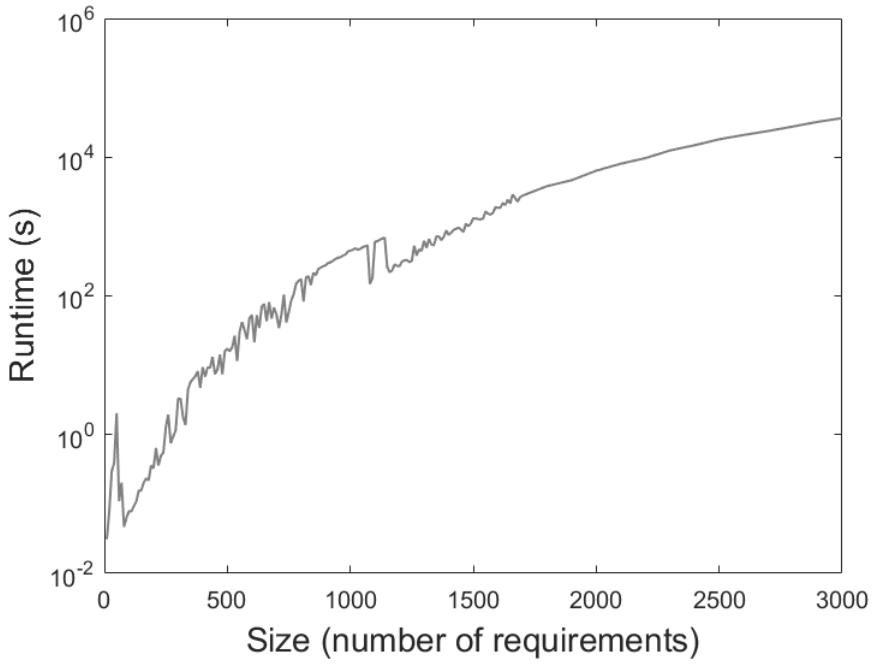


FIGURE 4.25: Runtime of DARS-ILP for different Sizes (Simulation 1).

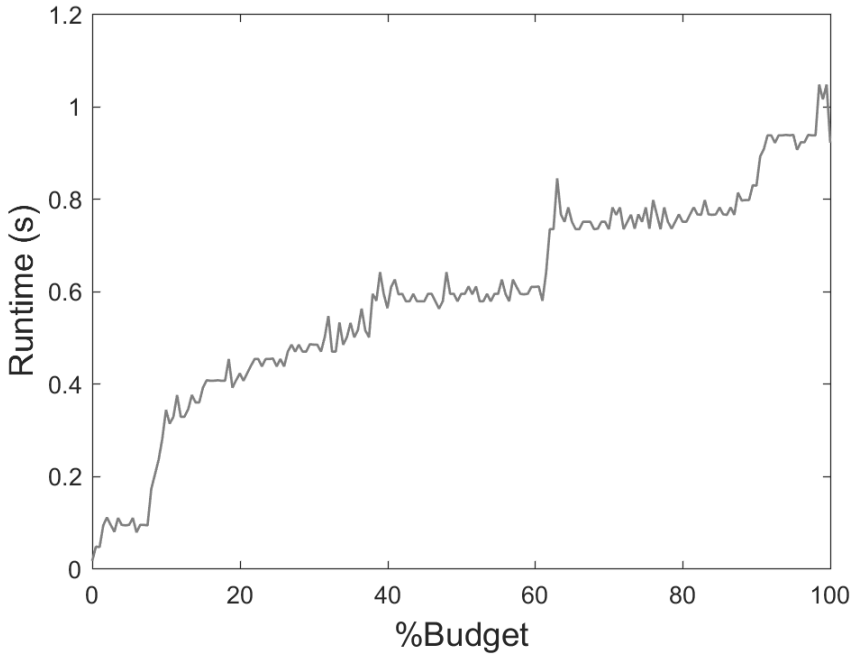


FIGURE 4.26: Runtime of DARS-ILP for different %Budget (Simulation 2).

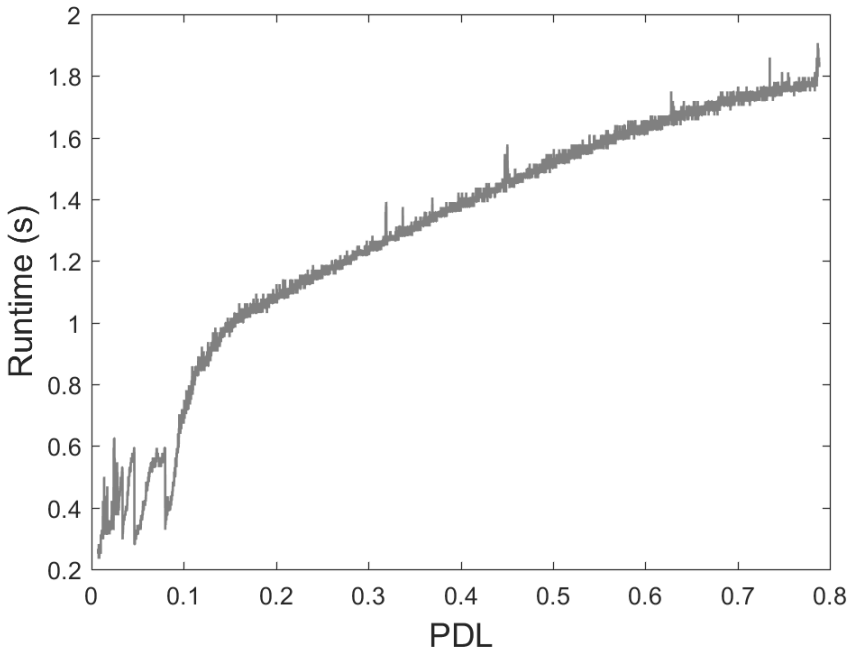


FIGURE 4.27: Runtime of DARS-ILP for different PDLs (Simulation 3).

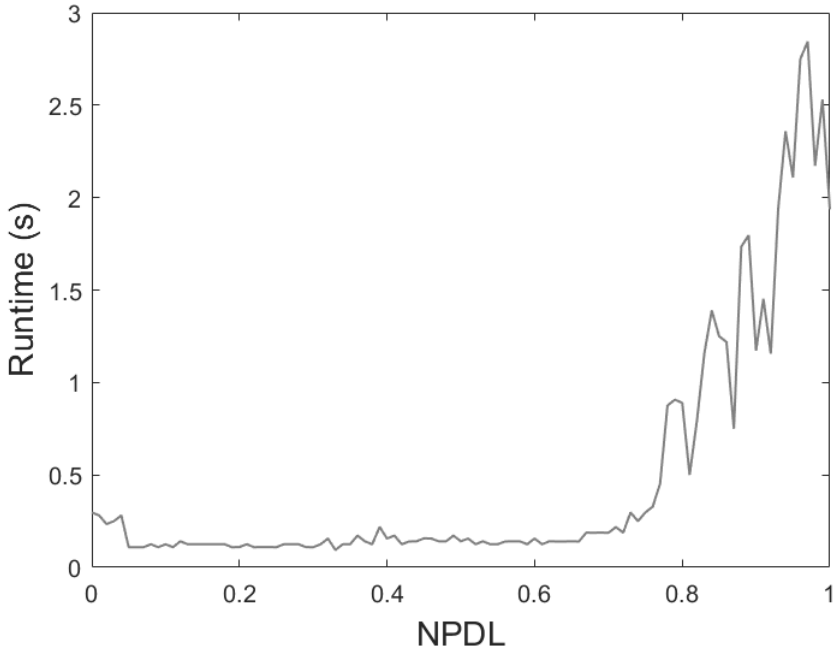


FIGURE 4.28: Runtime of DARS-ILP for different NPDLs (Simulation 4).

Increasing NPDL on the other hand, had no significant impact on the runtime of the optimization model of the DARS-ILP method in most places. Nonetheless, for larger NPDLs ($NPDL \rightarrow 1$), runtime was increased. The reason is at such high NPDL, the optimization model of the DARS-ILP method cannot find a feasible solution with some values as each requirement conflicts with almost every other requirement. Hence, it takes longer for the optimization to complete and return the null set (%OV=0) as the only feasible solution.

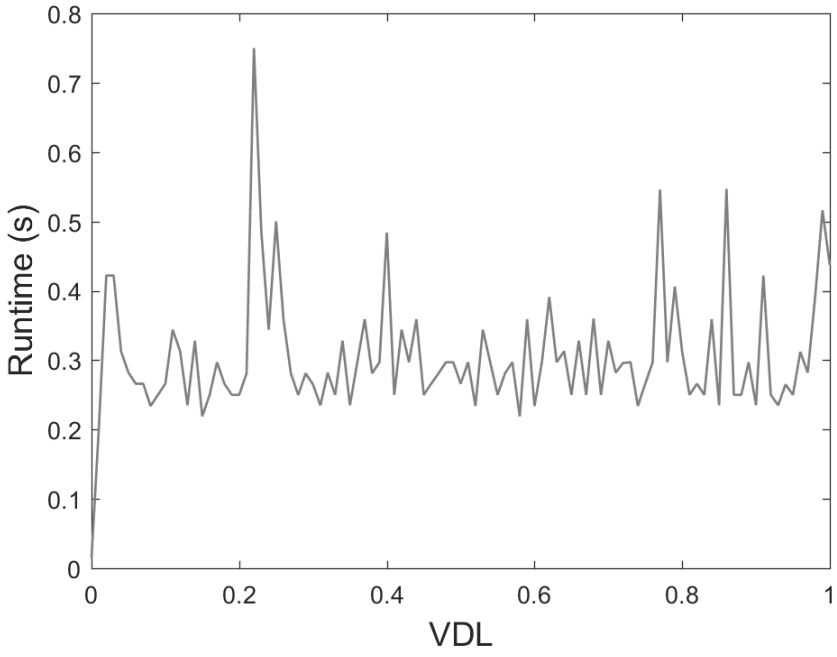


FIGURE 4.29: Runtime of DARS-ILP for different VDLs (Simulation 5).

Simulation 5 was carried out to answer **(RQ9.4)** by measuring the runtime of the selection models in the presence of various value dependency levels (VDLs). Our results demonstrate (Figure 4.29) that increasing (decreasing) VDL has an inconsistent impact of negligible magnitude on the runtime of the optimization model of the DARS-ILP method. In a similar way, our simulations for various negative value dependency levels (NVDLs) showed (Figure 4.30) that the impact of increasing (decreasing) NVDL on the runtime of the optimization model of the DARS-ILP method was unpredictable.

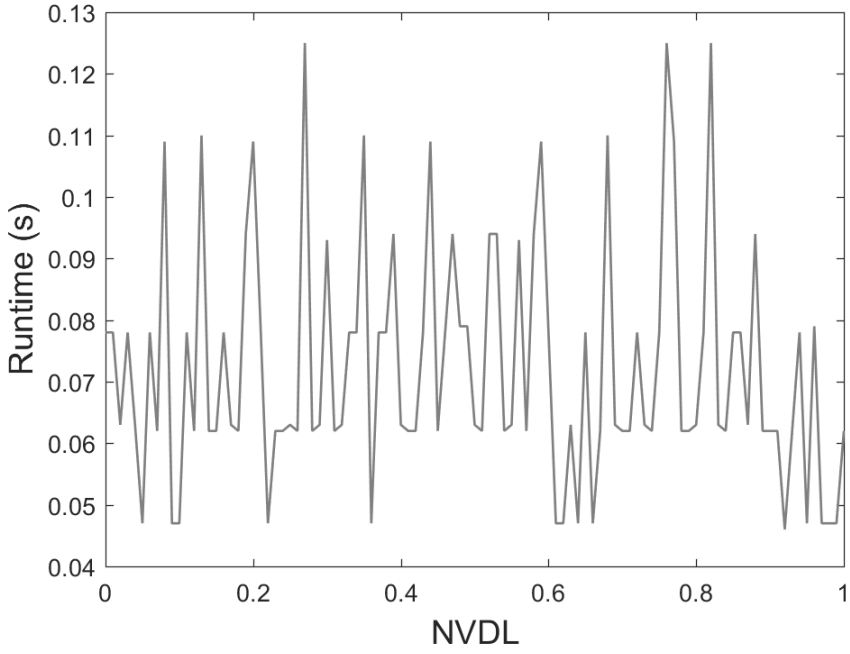


FIGURE 4.30: Runtime of DARS-ILP for different NVDLs (Simulation 6).

4.8 Summary

In this chapter we presented an integer linear programming (ILP) method, referred to as the DARS-ILP method, for dependency-aware requirement selection, which mitigates the risk of value loss posed by ignoring (selecting) the requirements with positive (negative) influences on the values of the selected requirements. The proposed method allows for the identification and modeling of value dependencies as well as the integration of those dependencies into software requirement selection. The main results of this chapter are presented in publications **(P1)**-**(P8)**.

The DARS-ILP method enhances the main components of the DARS-IP method, presented in Chapter 3, in several ways. First, the dependency identification technique in DARS-IP is enhanced by (a) considering the qualities of value dependencies and (b) using a formal significance test to understand the accuracy of the value dependencies. Second, the modeling technique proposed in DARS-ILP considers the qualities of value dependencies, thus allowing for reasoning about simultaneous positive and negative impacts of the explicit and implicit value dependencies among the requirements.

In this regard, we have presented a modified version of the Floyd-Warshall algorithm [44], which efficiently computes the positive and negative influences of the requirements on the values of each other based on the algebraic structure of fuzzy graphs. Fourth, the ILP model of DARS-ILP integrates both positive and negative value dependencies into software requirement selection. Finally, the optimization model of DARS-ILP method is linear, and, thus, is scalable to software projects with large number of requirements.

We have further contributed a *Blind* ILP model for the DARS-ILP method, which mitigates the risk of value loss posed by ignoring the positive influences of the requirement on the values of each other. The model does not rely on the identification of value dependencies and, thus, it is suitable for the projects in which the identification of value dependencies is not practical.

We demonstrated the practicality, effectiveness, and scalability of the proposed ILP method by studying a real-world software product and carrying out simulations. Our results show that (a) compared to the requirement selection methods that ignore value dependencies, the ILP method of DARS provides higher overall value by mitigating the impact of ignoring (selecting) requirements with positive (negative) influence on the values of selected requirements, (b) maximizing the accumulated value and overall value of a software are in conflict, and (c) DARS-ILP is scalable to large scale requirement sets for different levels of value dependencies and precedence dependencies among requirements. This is demonstrated by simulating different scenarios for datasets of up to 3000 requirements.

Chapter 5

The Mixed Integer Programming Method (DARS-MIP) [1]

5.1 Introduction

We demonstrated in Chapter 4 that the DARS-ILP method, which is an enhanced version of the DARS-IP method, mitigates the value loss by taking into account the influences of the requirements on the values of each other. We further observed in Section 4.6 that the the effectiveness of the DARS-ILP method in mitigating the value loss reduces in the presence of a tight budget or a high level of precedence dependencies (PDL) among the requirements. The reason is that requirements with significant positive influences on the values of the selected requirements may have to be ignored due to their conflicts with other requirements or the lack of sufficient budget. Analogously, requirements with negative influences on the values of the requirements may need to be selected when they are required by other selected requirements. Hence, a value loss caused by ignoring (selecting) requirements with positive (negative) influences on the values of the selected requirements is foreseeable in DARS-ILP. To mitigate this value loss, we have proposed allowing for partial selection (satisfaction) of the requirements when that can be tolerated. When partial selection is allowed in DARS, requirements with positive (negative) influences can be partly satisfied rather than being completely ignored (satisfied). This can mitigate the risk of value loss caused by ignoring (selecting) requirements with positive (negative) influences.

[1] The results of this chapter are presented in publications **(P3)**, **(P4)**, and **(P6)**-**(P12)**.

Section 5.2 of this chapter demonstrates the use of a fuzzy method, referred to as *Prioritization and Partial Selection* (PAPS), for partial selection of software requirements. The PAPS method was presented in Publication (**P10**) to account for partial satisfaction of security requirements. The method was also used in Publication (**P9**) to allow for partial selection of Agile requirements. We further presented an enhanced version of PAPS in Publication (**P12**).

To allow for partial selection of the requirements in DARS, we have presented, in Section 5.3, a mixed integer programming[2] (MIP) method referred to as the MIP method of DARS, i.e. DARS-MIP. The optimization model of DARS-MIP aims to find an optimal investment policy that accounts for the value dependencies among the requirements. Such a policy mitigates the value loss by allowing for increasing (decreasing) the investment in the requirements with significant positive (negative) influences on the values of the partially/fully selected requirements.

The investment in each requirement r_i is bounded by the lower-bound cost and the upper-bound cost of r_i. The upper-bound cost of r_i may be estimated by the stakeholders and then be RELAX-ed using the RELAX-ation technique proposed as part of the PAPS method presented in Section 5.2 to determine the lower bound cost of r_i. When r_i cannot be tolerated to be partially selected (satisfied), however, the lower-bound cost of r_i will be identical to its upper-bound cost.

The dependency identification technique presented Section 4.2 is used in DARS-MIP for the identification of value dependencies among requirements. The DARS-MIP method further uses the modeling technique presented in Section 4.3 for modeling value dependencies and computing the influences of the requirements on the values of each other. Hence, this chapter only focuses on the optimization model of DARS-MIP. The model is linear and scalable to large scale requirement sets.

It is also worth mentioning that the DARS-MIP method assumes that the budget is efficiently invested in the selected requirements, thus the expected value of a requirement will be adjusted based on the proportion of the budget invested in that requirement.

[2] A mixed integer programming (MIP) problem is a linear program where some of the variables are restricted to have integer values only.

The amount of the investment in each requirement is, however, bounded by the upper-bound cost and the lower-bound cost of that requirement. The DARS-MIP method, thus, accounts for the uncertainty associated with estimating the costs of the requirements by considering a range for those costs.

5.2 Partial Selection of Requirements

Due to the resource limitations it is hardly, if ever, possible to implement the entire set of identified requirements for a software project [149, 2]. Hence, prioritization and selection is required to find an optimal subset of requirements [150, 107] with the highest value. Requirement selection, however, may result in ignoring some of the requirements. But ignoring requirements even if they are of lower values may result in value loss due to the existence of value dependencies among requirements.

To mitigate this we proposed in (**P9**) and (**P10**) a fuzzy method referred to as *Prioritization and Partial Selection* (PAPS) which reduces the risk of value loss caused by ignoring requirements through accounting for partial selection (satisfaction) [9] of requirements, when that can be tolerated, rather than ignoring requirements or postponing them to the future releases. The proposed method helps reduce the value loss by reducing the chances that requirements with positive influences on the values of other requirements are ignored. The PAPS method is scalable to software projects with large number of requirements and allows for prioritization and selection of requirements with respect to different goals [9, 10].

The method is composed of two major processes as shown in Figure 5.1. The first process is referred to as *Pre Prioritization and Selection* (Pre-PAS), which comprises modeling and description of requirements as well as preprocessing the data for the prioritization and selection process. The Pre-PAS uses our previously developed modeling technique in [76, 78, 151] to capture partiality of requirements (goals) [152, 149] in a *Software Requirement Model* (SRM) of a software project will be used to construct the *Software Requirement List* (SRL) of that software project.

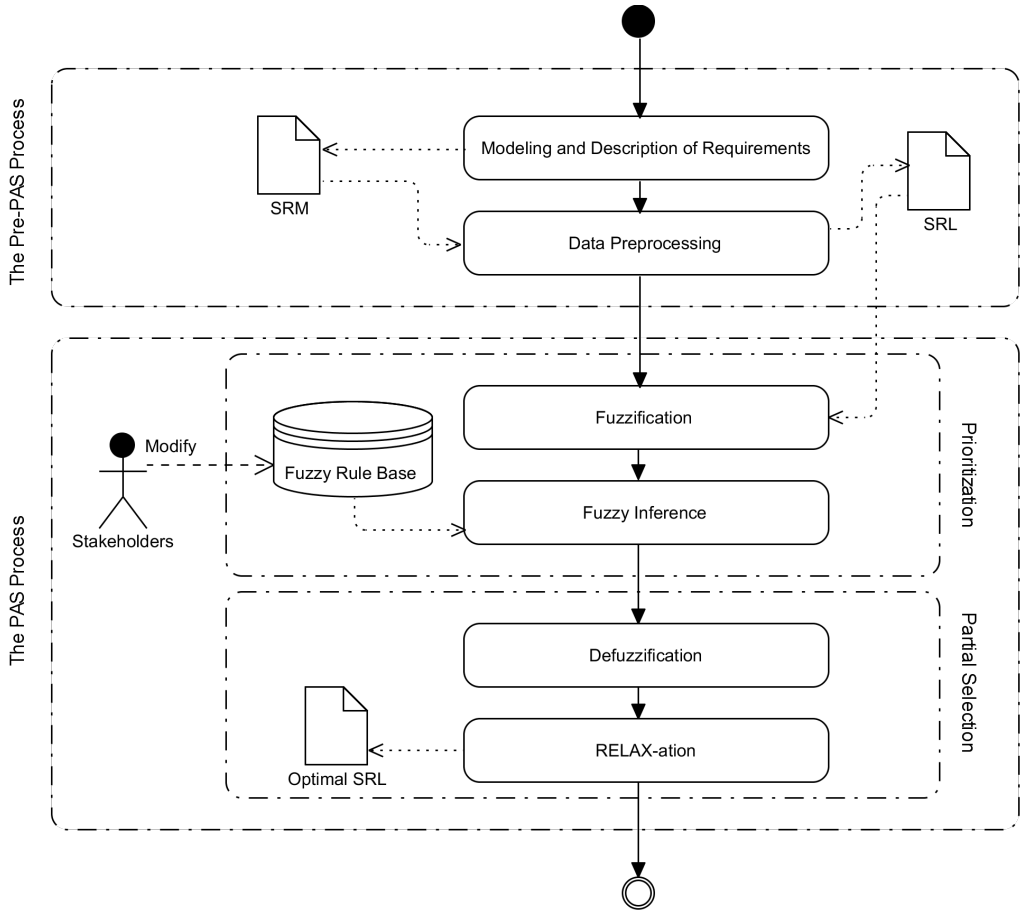


FIGURE 5.1: Architecture of PAPS.

Then requirements in the SRL will be prioritized using a *Fuzzy Inference System* (FIS) [153]. Each requirement contributes to satisfaction of at least one goal. The FIS, infers the linguistic priorities of the requirements with respect to their selection criteria. We account for partiality in the optimal SRL through RELAX-ing [154, 155] the satisfaction criteria of requirements when that can be tolerated. Requirements then will be RELAX-ed and partially included in the optimal SRL.

5.2.1 The Pre-PAS Process

The Pre-PAS process, as depicted in Figure 5.1, starts with the modeling and description of software requirements. First, the SRM of a software product will be developed and then a *Goal-based Fuzzy Grammar* (GFG) [156] of the SRM will be constructed to formally describe the SRM. The Pre-PAS process ends with preparing the data for the prioritization and selection process.

Modeling and Description of Requirements

An efficient model is required to capture the partiality of software requirements [78]. This section gives an overview of a goal-based modeling technique developed in our prior work [78, 76, 152, 157, 152], which captures partiality of requirements in software products.

Due to its inherent support for partial satisfaction [9] of requirements, SRM is employed to serve as the input of the prioritization and selection process in the PAPS method.

TABLE 5.1: The KAOS description of the requirements (goals) of OBS.

Goal	Description	Requirement	Description
s	*maintain OBS security*	r_1	*achieve request transaction code*
g_1	*avoid transfer money out of account*	r_2	*achieve latency examination*
g_2	*avoid unauthorized online transfer*	r_3	*achieve one-time pad*
g_3	*avoid stealing id and password*	r_4	*achieve SSL*
g_4	*avoid man in the middle*	r_5	*achieve password trial limitation*
g_5	*avoid guessing id and password*	r_6	*achieve password policy*
g_6	*avoid dictionary attack*	r_7	*achieve password encryption*
g_7	*avoid guess password*	r_8	*achieve random id*
g_8	*avoid guess id*	r_9	*achieve CAPTCHA*
g_9	*avoid brute forcing*	r_{10}	*achieve complex pin*
g_{10}	*avoid unauthorized transfer via debit card*	r_{11}	*achieve access control*
g_{11}	*maintain transfer network security*	r_{12}	*achieve redundant server*
g_{12}	*avoid hijack server*		
g_{13}	*maintain service availability*		

The SRM of a software product is developed through a goal-based modeling process presented in our prior work [76]. The process starts with the identification of the main goals of the software system. Then goals will be developed into lower level goals and eventually the requirements.

Our goal-based modeling process uses a combination of the RELAX [154, 155] and KAOS [158] description languages to describe software goals (requirements). The requirement model of an *Online Banking System* (OBS) is illustrated in Figure 5.2 and the SRM nodes (goals/requirements) are described in Table 5.1.

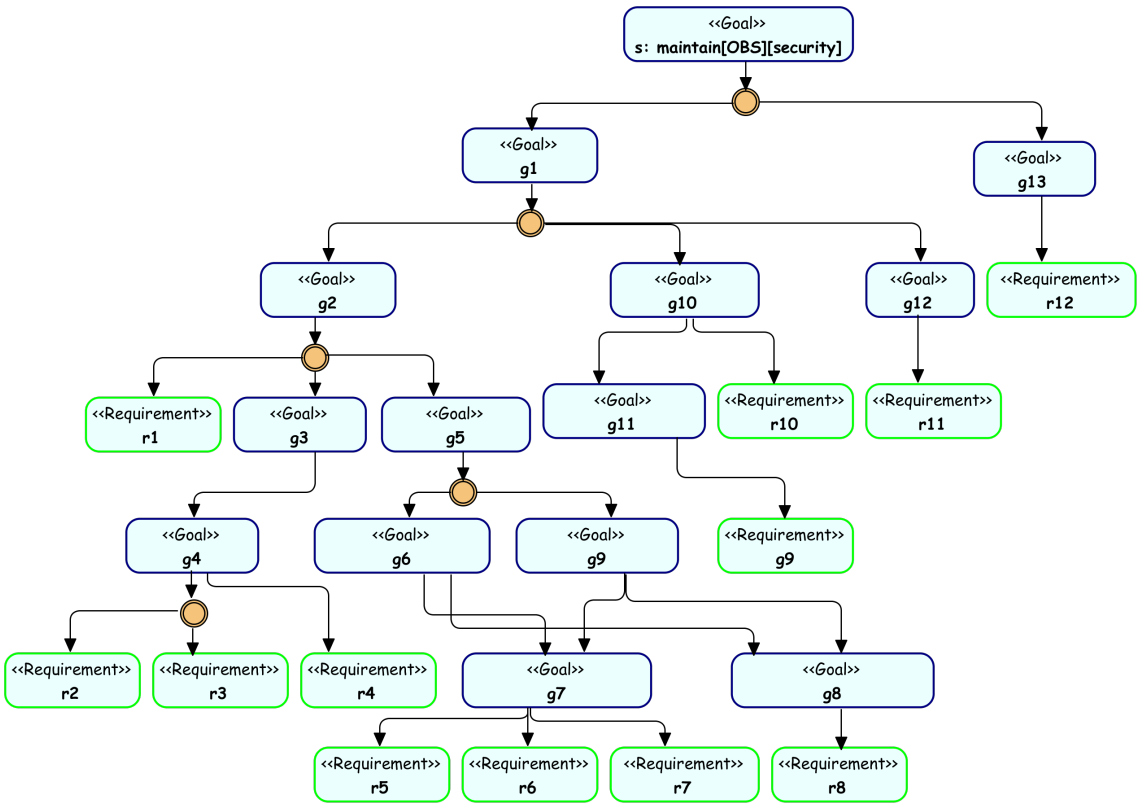


FIGURE 5.2: The SRM of OBS. Junction points and their absence represent logical AND and logical OR respectively.

We have made use of a fuzzy-based technique presented in our earlier work [156, 159, 160, 161] to formally describe the requirement model of a software project. The technique employed, allows for the description of the partiality in the SRM of a software project [9, 11]. The description process includes the construction of the goal-based fuzzy grammar (Definition 5.1) of a software product and extracting the derivation rules (Definition 5.2) among goals/requirements.

Definition 5.1. *Goal-Based Fuzzy Grammar*. Is a quintuple $GR = (G, R, P, s, \tau)$ in which G is a set of goals, R is a set of requirements, P is a set of fuzzy derivation rules and τ denotes the membership function of derivation. s represents the top-level goal of the system.

For OBS, $G = \{g_1, ..., g_{13}\}$, $R = \{r_1, ..., r_{12}\}$, $P = \{p_1, ..., p_{20}\}$ and $s =$ "maintain [OBS] [security]". Due to its fuzziness, GFG is able to properly capture partiality in the SRM of the system. The elements of $P \in GR$ are expressions of the form given in (5.1).

In this equation, d is the degree to which a sub-goal g_j contributes to the satisfaction of a goal or subgoal g_i. If $r_1, ..., r_n$ are fuzzy statements in $(G \cup R)^*$ and $r_1 \rightarrow r_2 \rightarrow ... \rightarrow r_n$, then we call this chain a goal derivation chain under the GFG employed.

$$\tau(g_i \rightarrow g_j) = d, d \in [0,1] \text{ or } \tau(g_i, g_j) = d \tag{5.1}$$

TABLE 5.2: The derivation rules of the SRM of OBS.

Rule	Membership Value	Rule	Membership Value
$p_1 : s \rightarrow g_1 g_{13}$	0.95	$p_{11} : g_4 \rightarrow r_2 r_3$	0.75
$p_2 : g_1 \rightarrow g_2 g_{10} g_{12}$	0.95	$p_{12} : g_4 \rightarrow r_4$	0.90
$p_3 : g_{13} \rightarrow r_{12}$	0.90	$p_{13} : g_6 \rightarrow g_7$	0.60
$p_4 : g_2 \rightarrow r_1 g_3 g_5$	0.85	$p_{14} : g_6 \rightarrow g_8$	0.60
$p_5 : g_{10} \rightarrow g_{11}$	0.90	$p_{15} : g_9 \rightarrow g_7$	0.65
$p_6 : g_{10} \rightarrow r_{10}$	0.40	$p_{16} : g_9 \rightarrow g_8$	0.60
$p_7 : g_{12} \rightarrow r_{11}$	0.90	$p_{17} : g_7 \rightarrow r_5$	0.70
$p_8 : g_3 \rightarrow g_4$	0.85	$p_{18} : g_7 \rightarrow r_6$	0.80
$p_9 : g_5 \rightarrow g_6 g_9$	0.90	$p_{19} : g_7 \rightarrow r_7$	0.90
$p_{10} : g_{11} \rightarrow r_9$	0.80	$p_{20} : g_8 \rightarrow r_8$	0.60

Definition 5.2. *Extracting Derivation Rules*. The description technique employed constructs a GFG for a given SRM and identifies the derivation rules [156, 161, 159]. The degree to which the success of a rule contributes to the satisfaction of its predecessor will specify its membership value. This value will be determined by the membership function τ of the GFG [161].

The extracted derivation rules for the SRM of the OBS and their corresponding membership values are listed in Table 5.2. The derivation rule $p_{11}(g_1 \rightarrow r_2 r_3)$ in Table 5.2, states that r_2 AND r_3 contribute to the satisfaction of the goal g_1.

Data Preprocessing

Data preprocessing includes estimation of the criteria used for prioritization and selection. Our proposed PAPS method as presented in (**P9**) and (**P10**) uses *Cost*, *Technical-Ability*, and *Impact* of requirements on the satisfaction of different goals as its main selection criteria.

TABLE 5.3: Cost and Technical-ability of the OBS Requirements.

Requirement	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}
Cost	0.50	0.70	0.70	0.30	0.05	0.50	0.20	0.01	0.60	0.10	0.70	1.00
Technical-ability	1.00	0.20	0.10	0.90	1.00	0.30	0.20	1.00	0.10	1.00	0.20	0.20

The cost of implementation is a real number in $[0,1]$ as given in Table 5.3. Also, the technical-ability is defined as a real number in $[0,1]$, which reflects the ease of implementation for each requirement. Technical-ability of requirements in the SRM of the OBS are listed in Table 5.3. To compute the impacts ($\in [0,1]$) of requirements we first construct the SRL of those requirements as depicted in Figure 5.3.

Let $GR = (G, R, P, s, \tau)$ be the GFG of a SRM. For each goal $g \in G$, $SRL[g] \subseteq R^*$ whose members contribute to the satisfaction of the goal g. A requirement x is said to be in $SRL[g]$ if and only if x can be derived from g. $SRL[g]$ can be constructed for the goal g in the SRM of the system. This allows for the goal-based prioritization of requirements with focus on the satisfaction of different goals. Function "buildSRL" in Figure 5.3 constructs the SRL of the system. We have implemented the SRL as a two dimensional list (list of lists) in which a list of $SRL[g]$, contains requirements which contribute to the satisfaction of the goal g.

the impact of each requirement x in $SRL[g]$ of the system is denoted by $DC_g(x)$ as given by 5.2, which is the degree of contribution of x to the satisfaction of the goal g. To calculate the impact of x firstly the membership values of the derivation rules on the derivation chain of x will be evaluated based on (5.2). Subsequently the fuzzy membership values will be calculated for each derivation chain through taking minimum of all membership values of the derivation rules on the derivation chain of x. Finally, the impact will be calculated through taking supremum over all membership values of the derivation chains which can generate x.

Equation (5.2) computes the impact of a requirement x on a goal g in the SRM of a software project. The t-norm sign $\oplus$ and a t-conorm sign $\otimes$ denote fuzzy OR (maximum) and AND (minimum) operators respectively based on Zadeh's definition in [162]. $\tau_g(x)$ specifies the strongest degree for contribution of x to the satisfaction of the goal g.

```
public class prePAS {
    public static SRM srm;
    public static SRL <String,ArrayList> srl;
    public static GFG gfg;
    // The constructor takes the list of requirements and
    // their corresponding costs and technical-ability as the input
    // and generates the SRL of the software project
    public prePAS(List<ArrayList<Requirement>> requirements){
        // Build the software requirement model of the projects
        // and formally describe the model by constructing the
        // GFG of the project
        srm = buildSRM(requirements);
        //calculate the impacts of requirements on
        //the satisfaction of the goals in the SRM
        // and update the SRM
        srm = calculateImpacts(srm);
        // build the software requirement list
        srl = buildSRL(srm);
    }
    // building the software requirement list
    public void buildSRL(){
        for(each Requirement r in gfg.R){
            for(each Goal g in gfg.G){
                if(r.impact[g] != 0){
                    // if r contributes to the satisfaction of g
                    SRL[g].add(r);
                }
            }
        }
    }
    // calculating the impact of a Requirement r
    // on satisfaction of the Goal g
    public void calculateImpact(Requirement r, Goal g){
        for(each Goal g in gfg.G){
            for(each Requirement r in gfg.R){
                double impact = 0;
                List <ArrayList<derivationRule>> derivationChains = parse(r);
                // the minimum impact of the rules on the derivation chain
                double minimumImpact = 1;
                // the maximum impact of the rules on the derivation chain
                double maximumImpact = 0;
                // taking supremum over derivation chains
                for(each ArrayList<derivationRule> chain in derivationChains){
                    for(each rule in derivationChain){
                        // Mu denotes the fuzzy memebership function
                        minimumImpact = min(minimumImpact,GFG.Mu(rule));
                    }
                    r.impact[g] = max(maximumImpact,minimumImpact);
                }
            }
        }
    }
}
```

FIGURE 5.3: The Pre-PAS process in PAPS.

For instance, $\tau_s(r_7)$ for r_7 in the SRM of OBS is calculated for two derivation chains: i) $s \rightarrow g_1 \rightarrow g_2 \rightarrow g_5 \rightarrow g_9 \rightarrow g_7 \rightarrow r_7$ and ii) $s \rightarrow g_1 \rightarrow g_2 \rightarrow g_5 \rightarrow g_6 \rightarrow g_7 \rightarrow r_7$, as follows: $\tau_s(X = r_7) = (0.95 \otimes 0.95 \otimes 0.85 \otimes 0.9 \otimes 0.65 \otimes 0.9) \oplus (0.95 \otimes 0.95 \otimes 0.85 \otimes 0.9 \otimes 0.6 \otimes 0.9) = 0.65$. The impacts of requirements in the SRM of OBS are calculated by the function "calculateImpacts" in Figure 5.3 as listed in Table 5.4.

$$DC_g(x) = \tau_g(x) = \oplus(\tau(g, r_1) \otimes \tau(r_1, r_2) \otimes ... \otimes \tau(r_n, x)) \tag{5.2}$$

TABLE 5.4: Impact of Requirements in the SRM of OBS.

Goal	$\tau(r_1)$	$\tau(r_2)$	$\tau(r_3)$	$\tau(r_4)$	$\tau(r_5)$	$\tau(r_6)$	$\tau(r_7)$	$\tau(r_8)$	$\tau(r_9)$	$\tau(r_{10})$	$\tau(r_{11})$	$\tau(r_{12})$
s	0.85	0.75	0.75	0.85	0.65	0.65	0.65	0.60	0.80	0.40	0.90	0.90
g_1	0.85	0.75	0.75	0.85	0.65	0.65	0.65	0.60	0.80	0.40	0.90	0.00
g_2	0.85	0.75	0.75	0.85	0.65	0.65	0.65	0.60	0.00	0.00	0.00	0.00
g_3	0.00	0.75	0.75	0.85	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
g_4	0.00	0.75	0.75	0.90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
g_5	0.00	0.00	0.00	0.00	0.65	0.65	0.65	0.60	0.00	0.00	0.00	0.00
g_6	0.00	0.00	0.00	0.00	0.60	0.60	0.60	0.60	0.00	0.00	0.00	0.00
g_7	0.00	0.00	0.00	0.00	0.70	0.80	0.90	0.00	0.00	0.00	0.00	0.00
g_8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00	0.00	0.00	0.00
g_9	0.00	0.00	0.00	0.00	0.65	0.65	0.65	0.65	0.00	0.00	0.00	0.00
g_{10}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.80	0.40	0.00	0.00
g_{11}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.80	0.00	0.00	0.00
g_{12}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.90	0.00
g_{13}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.90

5.2.2 Prioritization and Selection Process

The PAS process starts with preprocessing the FIS inputs. For a software project preprocessing includes construction of the SRL of that project and calculation of the impacts for requirements in the SRL. Subsequently, the impact, cost and technical-ability values will be fuzzified [163] to serve as the input of the FIS. We have employed a Mamdani-type [164] fuzzy inference system to specify the priorities of requirements with respect to their impacts, costs and technical-abilities. Prioritization can be performed with focus on satisfaction of any of the goals in the SRL of a software project. This is important especially when satisfaction of a particular goal is emphasized by the stakeholders. Finally, prioritized requirements will be partially selected by RELAX-ation of their satisfaction criteria. To perform RELAX-ation, we need to obtain *Required Degree of Satisfaction* (RDS) for each requirement through deffuzification of its priority.

Prioritization

Prioritization starts with fuzzification of the FIS inputs. These inputs include the impact, cost and technical-ability of the requirements. As depicted in Figure 5.1, the fuzzified values will serve as the inputs for the FIS. The FIS then infers the fuzzified priority values of the requirements based on the fuzzy rule-base of PAPS. Priority of each requirement specifies the extent to which it needs to be satisfied.

```
public class PAS {
    // PAS: Prioritization and Selection
    public PAS(SRL <String,ArrayList> srl, GFG gfg){
        for(each Goal g in gfg.G){
            for(each Requirement r in SRL[g]){
                // fuzzify prioritization criteria
                r.fuzzifiedImpact=fuzzify(r.impact);
                r.fuzzifiedCost=fuzzify(r.cost);
                r.fuzzifiedTechnicalAbility=fuzzify(r.technical_ability);
                // infer priorities based on the fuzzy rule-base
                r.priority=FIS(r);
                // defuzzify linguistic prioritization
                r.RDS=defuzzify(r.priority);
                // RELAX-ing the effort needed for
                // implementation/satisfaction of r
                r.relaxed_effort= r.RDS * r.effort;
            }
        }
    }
}
```

FIGURE 5.4: Prioritization and selection of requirements in PAPS.

Fuzzification

All of the FIS inputs (impact, cost and technical-ability) are categorized under three fuzzy categories of Low (L), Medium (M), and High (H). Three membership functions are defined for each input and its corresponding categories. We have employed a semi-trapezoids shape for membership functions. Consequently, four diverse points are required to define each membership function. The membership functions are calculated based on (5.3) and listed in Table 5.5. Each membership function will be calculated based on four specific points as given by equation (5.3) and depicted in Figure 5.3. We have employed a combination of a *Fuzzy Control Language* (FCL) [162] and jFuzzyLogic [165] to implement the membership functions.

TABLE 5.5: Membership functions for FIS inputs/output.

FIS Input/ Output	Membership Function ($\tau_{iv}(x)$)
Impact (i)	$\tau_{il}(i) = max(min(\frac{i+0.35}{0.35}, 1, \frac{0.45-i}{0.35}), 0)$
	$\tau_{im}(i) = max(min(\frac{i-0.2}{0.25}, 1, \frac{0.55-i}{0.25}), 0)$
	$\tau_{ih}(i) = max(min(\frac{i-0.5}{0.1}, 1, \frac{1-i}{0.1}), 0)$
Cost(c)	$\tau_{cl}(c) = max(min(\frac{i+0.35}{0.35}, 1, \frac{0.45-i}{0.35}), 0)$
	$\tau_{cm}(c) = max(min(\frac{i-0.2}{0.25}, 1, \frac{0.55-i}{0.25}), 0)$
	$\tau_{ch}(c) = max(min(\frac{i-0.5}{0.1}, 1, \frac{1-i}{0.1}), 0)$
Technical-ability (t)	$\tau_{tl}(t) = max(min(\frac{i+0.35}{0.35}, 1, \frac{0.45-i}{0.35}), 0)$
	$\tau_{tm}(t) = max(min(\frac{i-0.2}{0.25}, 1, \frac{0.55-i}{0.25}), 0)$
	$\tau_{th}(t) = max(min(\frac{i-0.5}{0.1}, 1, \frac{1-i}{0.1}), 0)$
Priority (p)	$\tau_{po}(p) = max(min(\frac{p+0.1}{0.1}, 1, \frac{0.3-p}{0.1}), 0)$
	$\tau_{pw}(p) = max(min(\frac{p-0.2}{0.1}, 1, \frac{0.5-p}{0.1}), 0)$
	$\tau_{pn}(p) = max(min(\frac{p-0.3}{0.15}, 1, \frac{0.8-p}{0.15}), 0)$
	$\tau_{ps}(p) = max(min(\frac{p-0.65}{0.1}, 1, \frac{1-p}{0.1}), 0)$

$$\forall i \in \{impact, cost, technical\text{-}ability\}, \forall v \in \{low, medium, high\} : \tag{5.3}$$
$$\exists (x_0, x_1, x_2, x_3), \tau(x_0) = \tau(x_3) = 0, \tau(x_1) = \tau(x_2) = 1 \rightarrow$$
$$\tau_{iv}(x) = \max\big(\min(\frac{x - x_0}{x_1 - x_0}, 1, \frac{x_3 - x}{x_3 - x_2}), 0\big).$$

For each goal g and requirement r Figure 5.4 shows fuzzification of $SRL[g][r].impact$, $SRL[g][r].cost$, and $SRL[g][r].technicalAbility$ using the membership functions of Table 5.5.

Fuzzy Inference

The PAPS method employs a Mamdani-type [164] fuzzy inference system to specify the linguistic priorities of software requirements based on four main categories of Optional (O), Weak (W), Normal (N) and Strong (S).

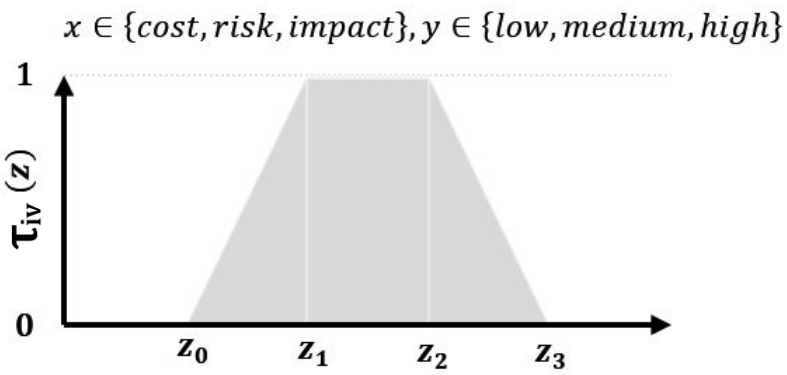


FIGURE 5.5: A sample membership function τ_{iv}

Priorities are inferred, for each requirement, with respect to its fuzzified prioritization criteria (impact, cost, and technical-ability as presented in (**P9**), (**P10**). Our employed FIS relies on the inference rules in the fuzzy rule base of the software project. The fuzzy rules in the rule-base of the FIS may be modified, as depicted in Figure 5.1, by the stakeholders to reflect their preferences.

We have implemented the fuzzy rules using FCL statements. As shown in Figure 5.4, 27 fuzzy rules are listed in the rule-base of the project. Software requirements of OBS are prioritized and listed in Table 5.6. Each requirement has 14 different priority values each computed with regard to a specific goal in the SRM of OBS.

We demonstrated in (**P9**), (**P10**) that goal-based prioritization and selection provides structured arguments regarding requirements. For instance requirement "r_7: achieve password encryption" in the SRL of OBS is strongly needed for the satisfaction of goal "g_2: avoid [unauthorized online transfer]" while it is weakly recommended with respect to the satisfaction of goal "g_6: avoid guess password". Consequently, other requirements (e.g. r_5) might need to be selected if satisfaction of g_6 is concerned.

Figure 5.5 demonstrates the membership functions of the FIS inputs (impact, cost, technical-ability) as well as the membership function of the fuzzy priority of r_7. r_7 is prioritized with respect to the top-level goal s. Membership values for FIS variables are depicted by vertical lines in their corresponding membership functions.

```
RULEBLOCK FuzzyRuleBase
  RULE 1 : IF impact IS l AND cost IS l AND technicalAbility IS l THEN priority IS o ;
  RULE 2 : IF impact IS l AND cost IS l AND technicalAbility IS m THEN priority IS w ;
  RULE 3 : IF impact IS l AND cost IS l AND technicalAbility IS h THEN priority IS n ;
  RULE 4 : IF impact IS l AND cost IS m AND technicalAbility IS l THEN priority IS o ;
  RULE 5 : IF impact IS l AND cost IS m AND technicalAbility IS m THEN priority IS o ;
  RULE 6 : IF impact IS l AND cost IS m AND technicalAbility IS h THEN priority IS w ;
  RULE 7 : IF impact IS l AND cost IS h AND technicalAbility IS l THEN priority IS o ;
  RULE 8 : IF impact IS l AND cost IS h AND technicalAbility IS m THEN priority IS o ;
  RULE 9 : IF impact IS l AND cost IS h AND technicalAbility IS h THEN priority IS o ;
  RULE 10: IF impact IS m AND cost IS l AND technicalAbility IS l THEN priority IS w ;
  RULE 11: IF impact IS m AND cost IS l AND technicalAbility IS m THEN priority IS n ;
  RULE 12: IF impact IS m AND cost IS l AND technicalAbility IS h THEN priority IS s ;
  RULE 13: IF impact IS m AND cost IS m AND technicalAbility IS l THEN priority IS o ;
  RULE 14: IF impact IS m AND cost IS m AND technicalAbility IS m THEN priority IS w ;
  RULE 15: IF impact IS m AND cost IS m AND technicalAbility IS h THEN priority IS n ;
  RULE 16: IF impact IS m AND cost IS h AND technicalAbility IS l THEN priority IS o ;
  RULE 17: IF impact IS m AND cost IS h AND technicalAbility IS m THEN priority IS w ;
  RULE 18: IF impact IS m AND cost IS h AND technicalAbility IS h THEN priority IS w ;
  RULE 19: IF impact IS h AND cost IS l AND technicalAbility IS l THEN priority IS n ;
  RULE 20: IF impact IS h AND cost IS l AND technicalAbility IS m THEN priority IS s ;
  RULE 21: IF impact IS h AND cost IS l AND technicalAbility IS h THEN priority IS s ;
  RULE 22: IF impact IS h AND cost IS m AND technicalAbility IS l THEN priority IS w ;
  RULE 23: IF impact IS h AND cost IS m AND technicalAbility IS m THEN priority IS s ;
  RULE 24: IF impact IS h AND cost IS m AND technicalAbility IS h THEN priority IS s ;
  RULE 25: IF impact IS h AND cost IS h AND technicalAbility IS l THEN priority IS o ;
  RULE 26: IF impact IS h AND cost IS h AND technicalAbility IS m THEN priority IS w ;
  RULE 27: IF impact IS h AND cost IS h AND technicalAbility IS h THEN priority IS s ;
END_RULEBLOCK
```

FIGURE 5.6: Fuzzy rules implemented in FCL

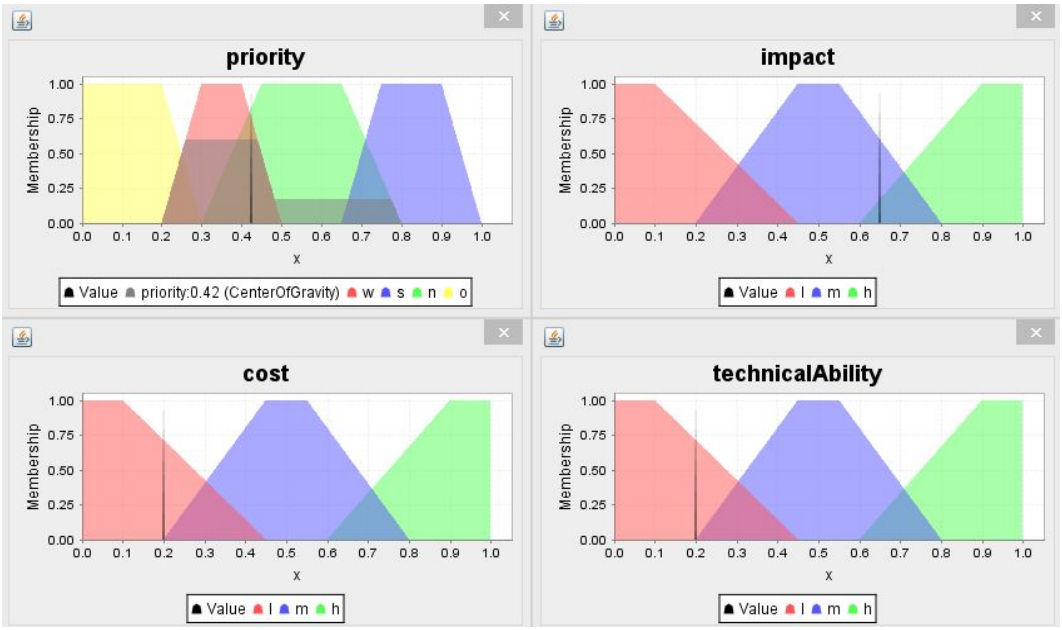


FIGURE 5.7: Fuzzy Inference for r_7 with respect to the top-level goal

Partial Selection

Partial selection (satisfaction) of requirements, when tolerated, can be explicitly addressed by RELAX-ing [76, 166] the satisfaction criteria of those requirements. As one example, implementation of a complex password policy in a software product increases the level of security on one hand and reduces the usability of the system [167] on the other hand.

On the other hand, implementation of a less complex password policy may be tolerated to maintain the usability of the system. In this case, the satisfaction criterion of the requirement "password policy" can be relaxed to be partially included into the optimal SRL of a software product. In other words, "password policy" can be partially selected. Partial selection in PAPS, includes defuzzification and RELAX-ation of requirements.

We need crisp values in order to perform the RELAX-ation operation on the requirements [78, 9]. Therefore, defuzzification is required to map the linguistic priority values into their corresponding crisp representations. To do so, we need to use the membership functions of the priority (the FIS output) values. We have listed these membership functions in Table 5.5.

TABLE 5.6: Priority values inferred by FIS for requirements of OBS.

Goal	Linguistic Priority Values											
	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}
s	*S*	*W*	*W*	*S*	*N*	*W*	*N*	*N*	*W*	*N*	*W*	*O*
g_1	*S*	*W*	*W*	*S*	*N*	*W*	*N*	*N*	*W*	*N*	*W*	-
g_2	*S*	*W*	*W*	*S*	*N*	*W*	*N*	*N*	-	-	-	-
g_3	-	*W*	*W*	*S*	-	-	-	-	-	-	-	-
g_4	-	*W*	*W*	*S*	-	-	-	-	-	-	-	-
g_5	-	-	-	-	*N*	*W*	*N*	*N*	-	-	-	-
g_6	-	-	-	-	*N*	*O*	*W*	*N*	-	-	-	-
g_7	-	-	-	-	*N*	*N*	*N*	-	-	-	-	-
g_8	-	-	-	-	-	-	-	*N*	-	-	-	-
g_9	-	-	-	-	*N*	*W*	*N*	*N*	-	-	-	-
g_{10}	-	-	-	-	-	-	-	-	*W*	*N*	-	-
g_{11}	-	-	-	-	-	-	-	-	*W*	-	-	-
g_{12}	-	-	-	-	-	-	-	-	-	-	*W*	-
g_{13}	-	-	-	-	-	-	-	-	-	-	-	*O*

Four different linguistic categories of optional, weak, normal and strong specify the priority of requirements. Defuzzification, as in Figure 5.4, updates the RDS attributes of the requirements in the SRL based on their corresponding defuzzified priorities. We have employed the Center of Gravity (COG) [168] formula in order to defuzzify the priority values.

The linguistic prioritizes of the requirements will fall into one of the four major categories of *optional*, *weak*, *normal* or *strong* as explained earlier. A requirement in the optional category has the lowest priority. Requirements in the *weak* and *normal* categories however can be weakly and normally satisfied respectively while requirements in the *strong* category must be strongly satisfied. To explicitly address this partiality, we need to specify the extent to which each requirement is expected to be satisfied. For this purpose we use RDS values obtained from defuzzification of the linguistic priorities.

For some of the requirements of software projects such as *availability of a system*, for instance, it may be rather easy to measure the satisfaction of requirements. But, this is not easy to achieve for several types of requirements due to the lack of proper measures to evaluate the satisfaction of those requirements. Samples criteria for evaluating the satisfaction of requirements of OBS are listed in Table 5.8.

TABLE 5.7: The RDS values of the requirements of OBS.

Goal	RDS Values											
	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}
s	0.82	0.25	0.25	0.82	0.59	0.36	0.42	0.55	0.35	0.55	0.25	0.13
g_1	0.82	0.25	0.25	0.82	0.59	0.36	0.42	0.55	0.35	0.55	0.25	-
g_2	0.82	0.25	0.25	0.82	0.59	0.36	0.42	0.55	-	-	-	-
g_3	-	0.25	0.25	0.82	-	-	-	-	-	-	-	-
g_4	-	0.25	0.25	0.82	-	-	-	-	-	-	-	-
g_5	-	-	-	-	0.59	0.36	0.42	0.55	-	-	-	-
g_6	-	-	-	-	0.55	0.24	0.35	0.55	-	-	-	-
g_7	-	-	-	-	0.64	0.6	0.55	-	-	-	-	-
g_8	-	-	-	-	-	-	-	0.55	-	-	-	-
g_9	-	-	-	-	0.59	0.36	0.42	0.59	-	-	-	-
g_{10}	-	-	-	-	-	-	-	-	0.35	0.55	-	-
g_{11}	-	-	-	-	-	-	-	-	0.35	-	-	-
g_{12}	-	-	-	-	-	-	-	-	-	-	0.25	-
g_{13}	-	-	-	-	-	-	-	-	-	-	-	0.13

TABLE 5.8: RELAX-ed requirements of OBS.

Requirement	Sample Satisfaction Criterion	ζ'_i
r_1: achieve request transaction code	expiry rate	$0.82\zeta_1$
r_2: achieve latency examination	examination delay	$0.25\zeta_2$
r_3: achieve one-time pad	randomness	$0.25\zeta_3$
r_4: achieve SSL	entropy	$0.82\zeta_4$
r_5: achieve password trial limitation	trial delay	$0.59\zeta_5$
r_6: achieve password policy	complexity	$0.36\zeta_6$
r_7: achieve password encryption	length of the encryption key	$0.42\zeta_7$
r_8: achieve random id	randomness	$0.55\zeta_8$
r_9: achieve CAPTCHA	level of distortion	$0.35\zeta_9$
r_{10}: achieve complex pin	complexity	$0.55\zeta_{10}$
r_{11}: achieve access control	complexity	$0.25\zeta_{11}$
r_{12}: achieve redundant server	number of servers	$0.13\zeta_{12}$

Hence, it would be hard, if possible at all, to explicitly RELAX the satisfaction criteria of requirements. To tackle this, we RELAX the effort needed for complete satisfaction of requirements to indirectly RELAX their satisfaction criteria. There are well established measures of effort [169], which can be used in this regard. We demonstrated in (**P9**) using story points to RELAX the amount of the effort put on satisfaction of Agile requirements. Equation (5.4) computes the RELAX-ed efforts for a requirement r_i, where ζ_i denotes the estimated effort needed to fully satisfy (implement) r_i while ζ'_i specifies the RELAX-ed effort put on r_i.

Nevertheless, RELAX-ing the satisfaction criteria of requirements by RELAX-ing their required effort is based on the assumption that the effort is put efficiently on satisfaction of requirements thus the more effort is put on satisfaction of a requirement the more the requirement is satisfied. In other words, the amount of the effort put on a requirement determines the satisfaction criteria of that requirements. Hence, RELAX-ing the amount of the effort needed to satisfy a requirement will also RELAX the satisfaction criterion of that requirement.

$$\zeta_i' = \zeta_i r_i.RDS \tag{5.4}$$

As one example for RELAX-ation, consider the requirement "r_6: achieve password policy". Based on Table 5.7, SRL$[s][r_6] = weak$. In this case r_6 is neither required to be fully included in the $SRL[s]$ nor fully omitted. Instead, r_6 can be partially included (selected) in the SRL[s] through RELAX-ing the effort needed for its satisfaction. If a less complex password encryption can be tolerated in $SRL[s]$, less effort can be put on satisfaction of r_6 by implementing a less complex password policy. This will save some effort to be put on other requirements while still providing users with a password policy feature of an acceptable quality. Hence we have $\zeta_6' = r_6.RDS \times \zeta_6$. It is worth mentioning again that this RELAX-ation can only be achieved when partial satisfaction of r_6 is tolerated. Table 5.8 lists the RELAX-ed effort values for requirements of OBS.

5.3 The MIP Model of DARS-MIP

We demonstrated in Chapter 4 that the DARS-ILP method, which is an enhanced version of the DARS-IP method, mitigates the value loss by taking into account the influences of the requirements on the values of each other. We further observed in Section 4.6 that the precedence relations among the requirements and budget constraint reduce the effectiveness of DARS-ILP in mitigating the value loss.

The reasons is that requirements with significant positive influences on the values of the selected requirements may have to be ignored due to their conflicts with other requirements or the lack of sufficient budget. Analogously, requirements with negative influences on the values of the requirements may need to be selected when they are required by other selected requirements.

To further mitigate the value loss, we propose partial selection (satisfaction) of requirements, when that can be tolerated, in dependency-aware requirement selection. Partial selection of the requirements of a software project mitigates the value loss by (a) increasing the investment in the requirements that positively influence the values of the requirements; and (b) reducing the investment in the requirements that negatively influence the values of the requirements.

A minimum investment amount (lower-bound cost) and a maximum investment amount (upper-bound cost) are specified for the satisfaction of each requirement r_i. When r_i cannot be tolerated to be partially selected (satisfied), however, the lower-bound cost of r_i will be the same as its upper-bound cost.

One approach to specifying these boundaries is to use the estimated costs of the requirements as the upper-bound costs and then RELAX the effort needed for the satisfaction of the requirements using the fuzzy method, i.e. PAPS, presented in Section 5.2 to determine the lower bound costs of the requirements.

To allow for partial selection of requirements in DARS, we have contributed a mixed integer programming (MIP) method referred to as the MIP method of DARS, i.e. DARS-MIP, for integrating value dependencies into requirement selection while allowing for partial selection of the requirements when that can be tolerated.

Hence, the proposed DARS-MIP method aims to find an optimal investment policy for the satisfaction of the requirements of a software project, where such a policy increases (decreases) the investment in the requirements that positively (negatively) influence the values of the requirements.

Model (5.6)-(5.14) give the optimization model of the DARS-MIP method. In these equations, the relaxed (real) variable $0 \leq x_i \leq 1$ specifies the investment ratio of a

requirement r_i, which is the proportion of the budget that is invested in the satisfaction of r_i based on the optimal investment policy found by the optimization model of DARS-MIP. As given by (5.10), the investment ratio of r_i is contained within its lower-bound $z_i(\frac{c_{i,l}}{b})$ and upper-bound $z_i(\frac{c_{i,u}}{b})$, which are determined by the lower-bound cost and the upper-bound cost of the satisfaction of r_i denoted by $c_{i,l}$, and $c_{i,u}$ respectively.

$$E(v_i) = p(r_i)v_i \tag{5.5}$$

When no money is invested in r_i ($z_i = 0$), we have $x_i = 0$. But when some of the budget is invested in the satisfaction of r_i ($z_i = 1$), (5.10) determines the lower-bound $(\frac{c_{i,l}}{b})$ and the upper-bound $(\frac{c_{i,u}}{b})$ for the investment ratio of r_i. When r_i cannot be tolerated to be partially selected (satisfied), however, we have $c_{i,l} = c_{i,u}$, which implies $x_i = \frac{c_{i,u}}{b}$ if $z_i = 1$, and $x_i = 0$ if $z_i = 0$.

$$\text{Maximize } \sum_{i=1}^{n} (\frac{x_i b}{c_{i,u}}) E(v_i) - y_i E(v_i) \tag{5.6}$$

$$\text{Subject to} \sum_{i=1}^{n} x_i \leq 1 \tag{5.7}$$

$$\begin{cases} z_i \leq z_j & r_j \text{ precedes } r_i \\ z_i \leq 1 - z_j & r_i \text{ conflicts with } r_j,\ i \neq j = 1, ..., n \end{cases} \tag{5.8}$$

$$\theta_i \geq \bigvee_{j=1}^{n} \left(\frac{(\frac{x_j b}{c_{j,u}})(|I_{i,j}| - I_{i,j}) + (1 - \frac{x_j b}{c_{j,u}})(|I_{i,j}| + I_{i,j})}{2} \right), \qquad i \neq j = 1, ..., n \tag{5.9}$$

$$z_i(\frac{c_{i,l}}{b}) \leq x_i \leq z_i(\frac{c_{i,u}}{b}), \qquad i = 1, ..., n \tag{5.10}$$

$$-z_i \leq y_i \leq z_i, \qquad i = 1, ..., n \tag{5.11}$$

$$-(1 - z_i) \leq y_i - \theta_i \leq (1 - z_i), \qquad i = 1, ..., n \tag{5.12}$$

$$z_i \in \{0, 1\}, \qquad i = 1, ..., n \tag{5.13}$$

$$0 \leq \theta_i \leq 1, \qquad i = 1, ..., n \tag{5.14}$$

The optimization model of the DARS-MIP method finds an optimal investment policy, which specifies the optimal subset of the requirements in which requirements are fully ($x_i = \frac{c_{i,u}}{b}$) or partly ($\frac{c_{i,l}}{b} \leq x_i < \frac{c_{i,u}}{b}$) included.

The expression $(\frac{x_i b}{c_{i,u}})E(v_i)$ in (5.6) states that the estimated value (v_i) of a (partially) selected requirement r_i and subsequently its expected value ($E(v_i)$) are influenced by the proportion of the budget invested in the satisfaction of r_i. $E(v_i)$ is computed by (6.29), where $p(r_i)$ denotes the probability that users purchase or use r_i.

Moreover, θ_i, as given by (5.9), specifies the penalty of a requirement r_i, which is the extent to which the expected value of r_i is negatively impacted by ignoring (selecting) requirements with positive (negative) influences on the value of r_i. We use the value dependency graphs (VDGs) presented in Section 4.3 for modeling value dependencies and reasoning about the qualities and strengths of those dependencies based on the algebraic structure of fuzzy graphs. Hence θ_i is computed by the fuzzy operator $\bigvee$, which finds the maximum of the influences of the ignored (selected) requirements with positive (negative) influence on the value of r_i.

When a requirement r_j with a positive influence on the value of r_i is ignored ($x_j = 0$), $I_{i,j}$ will be considered as the negative influence of ignoring r_j on the value of r_i as given by (5.9). But when r_j is partially selected ($\frac{c_{j,l}}{b} \leq x_j < \frac{c_{j,u}}{b}$), this implies that r_j is only partially ignored and therefore the negative influence of the ignoring r_j on the value of r_i is adjusted to $(1 - \frac{x_j b}{c_{j,u}})I_{i,j}$.

Analogously, when a requirement r_j with a negative influence on the value of r_i is fully selected, this influence is considered as $I_{i,j}$ by (5.9). But when r_j is partially selected, the negative influence of the selecting r_j on the value of r_i is adjusted to $\frac{x_j b}{c_{j,u}} I_{i,j}$ in (5.9). In other words, the lower the investment in r_j, the lower the value loss caused by selecting r_j will be.

In addition, (5.8) accounts for the precedence constraints and their value implications while (5.7) ensures that the total amount of the money invested in the requirements does not exceed the available budget b. The expression $z_i \leq z_j$ in (5.8) states that a requirement r_i requires r_j while $z_i \leq (1 - z_j)$ means that r_i conflicts with r_j.

It is also worth mentioning that the proposed DARS-MIP method relies on the dependency identification technique presented in Section 4.2. The method also uses value dependency graphs (VDGs) presented in Section 4.3 for modeling value dependencies and reasoning about the strengths and qualities of those dependencies. Hence, we avoid repeating those components of the DARS-ILP method in this chapter.

The optimization model of DARS-MIP, as given by (5.6)-(5.14), is linear and therefore can be efficiently solved [39], even for large scale requirement sets, by the existing commercial solvers such as *IBM CPLEX* [40]. We have implemented, solved, and tested the optimization model of the DARS-MIP method using the *Concert Technology* and the *JAVA API of IBM CPLEX* [40]. The code for this model is available in *JAVA* and *OPL* languages and can be obtained from the website of DARS[23].

5.4 Summary

Budget limitations and precedence dependencies among requirements may result in ignoring (selecting) the requirements with significant positive (negative) influences on the values of the selected requirements in DARS-ILP. This may result in value loss. To further mitigate the risk of value loss posed by ignoring (selecting) the requirements with positive (negative) influences, we proposed allowing for partial selection (satisfaction) of the requirements when that can be tolerated.

When partial selection is allowed in DARS, requirements with positive (negative) influences on the values of the selected requirements can be partly satisfied rather than being completely ignored (satisfied). This can mitigate the risk of value loss caused by ignoring (selecting) the requirements with positive (negative) influences. To achieve this, we presented a mixed integer programming (MIP) method referred to as the MIP method of DARS, i.e. DARS-MIP.

The method takes into account value dependencies while allowing for partial selection of requirements. The optimization model of the DARS-MIP method finds an optimal investment policy that mitigates the value loss by allowing for increasing (decreasing)

[23] http://bcert.org/projects/dars

the investment in the requirements with significant positive (negative) influences on the values of the partially/fully selected requirements.

The investment in a requirement r_i is bounded by the lower-bound cost and the upper-bound cost of r_i. The upper-bound cost of r_i is estimated by the stakeholders and then RELAX-ed, using a RELAX-ation technique presented in this chapter, to determine the lower bound cost of r_i.

The MIP model of the DARS-MIP method is linear and scalable to software projects with large number of requirement. The main contributions of Chapter 5 are presented in publications **(P3)**, **(P4)**, and **(P6)**-**(P12)** and application to real-world software projects is now under way as part of our ongoing research to further investigate the effectiveness of the method in mitigating the value loss.

Chapter 6

The Society-Oriented DARS Method (DARS-SOC) [1]

6.1 Introduction

Software requirement selection, also known as *Software Release Planning* [16, 17], aims to find a subset of requirements with the highest economic value for a release of software while respecting the project constraints [18]. However, there are several types of human values [38] i.e. *Social Values*, as depicted in Figure 6.1, with long term impacts on the society [37] that are also important and need to be considered in software requirement selection.

In fact the social values of certain software requirements may be even more important than their economic value. Software requirements that help people with their illnesses and disabilities or software requirements used for promoting social values [170] in the society are examples of such requirements in which social values are at least as important as the economic value. Social values, therefore, need to be embedded into software requirement selection.

Moreover, software is increasingly seen as a way of promoting positive social changes in the society. This includes initiatives which "strive to build innovative software solutions with a social conscience" as stated by Ferrario *et al.* [170].

Embedding social values into the software engineering activities, including the requirement selection is also referred to as "software engineering for social good" [170].

[1] The contents of this chapter are presented in publications **(P3)**, **(P4)**, **(P6)**, **(P8)**, **(P10)**, **(P12)**, and **(P13)**.

This area is receiving growing interest in recent years. But the existing requirement selection works (Table 2.2) ignore social values of software requirements and their impacts on the society. To consider the social values in software requirement selection, these values need to be integrated into the optimization models of the requirement selection methods. A sample map of the social values in software domain is demonstrated[2] in Figure 6.1. Social values, however, may change across different societies.

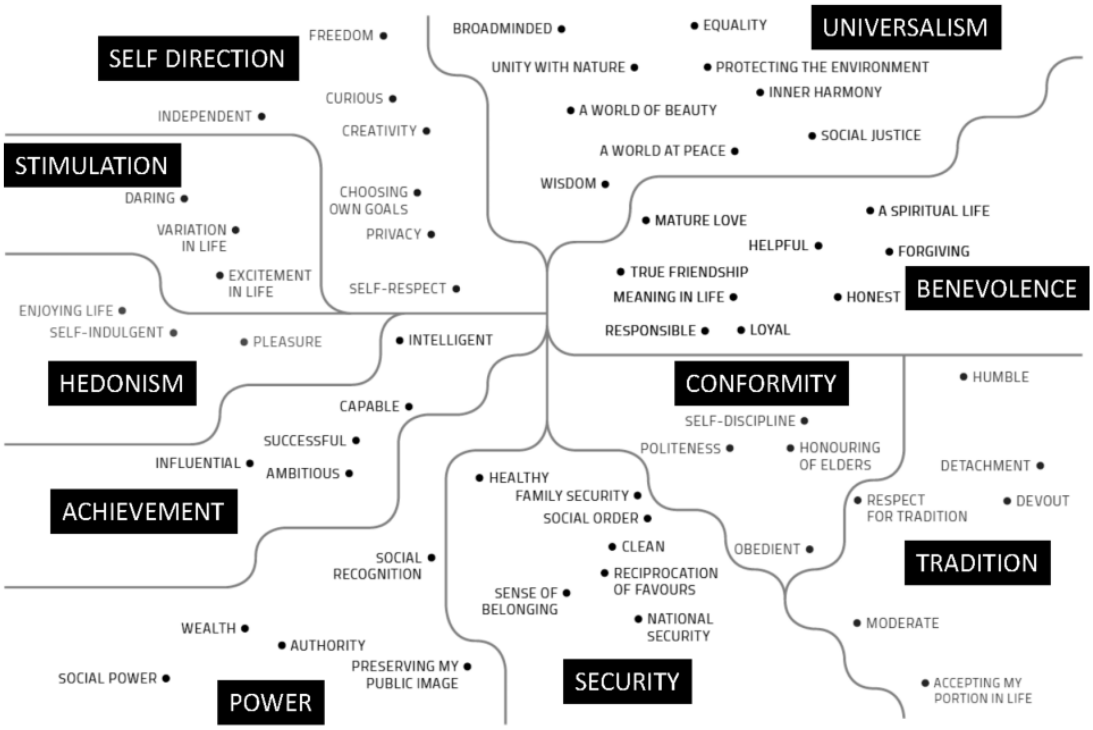


FIGURE 6.1: A sample map of the social values in software projects.

It is widely known that in a software project the economic values of the selected requirements may positively or negatively depend on the presence or absence of other requirements [19, 20] in the selected subset of the requirements, i.e. *Optimal Subset*. Analogously, there are also dependencies among social values of the requirements in the sense that the presence or absence of the certain requirements may impact the social values of other requirements. Moreover, there might be relationships and conflicts [171] among different types of social values.

[2]Source: [37]

Hence, it is important that we take into account the dependencies among the social values as well as the dependencies among the economic values of the requirements in the optimization models of the software requirement selection methods. Dependencies among the social values and dependencies among the economic values are all referred to as value dependencies for the ease of reference in this chapter.

Moreover, as observed by Carlshamre *et al.* [21], requirement dependencies in general and value dependencies in particular are *fuzzy* [21] in the sense that the strengths of those dependencies are imprecise and vary [18, 25, 26, 21] from large to insignificant [27] in real-world projects. Hence, it is important to consider not only the existence but the strengths of value dependencies and the imprecision of those dependencies in software projects.

This chapter extends the DARS-ILP and DARS-MIP methods presented in Chapter 4 and Chapter 5 to account for the social values in dependency-aware software requirement selection. The proposed method, referred to as the *Society-Oriented* method of DARS (DARS-SOC), accounts for the economic and social values as well as the dependencies among those values in software requirement selection.

The proposed DARS-SOC method comprises two main optimization models, with different characteristics, that allow for embedding the social values and the dependencies among those values into software requirement selection. We have further extended the definitions of the *Value Dependency Graphs* (VDGs) and value dependencies presented in Section 4.3.2 to capture different types of social values in modeling value dependencies.

Our proposed DARS-SOC method relies on the technique presented in Section 4.2 for the identification of the economic value dependencies. But, to the best of our knowledge, there are not any techniques in the present literature for identification of social value dependencies. These dependencies may be identified manually for small requirement sets. But development of more sophisticated techniques is needed for automated identification of social value dependencies in medium to large scale requirement sets. this is, however, beyond the scope of this thesis.

6.2 Modeling The Economic and Social Value Dependencies

This section presents a technique for modeling the economic and social value dependencies among software requirements. The proposed technique extends the definition and the use of value dependency graphs (VDGs) presented in Section 4.3 to account for different types of value dependencies among the requirements of software projects. The algebraic structure of fuzzy graphs is used for computing the influences of the requirements on the values of each other.

6.2.1 Value Dependency Graphs

In this section we use value dependency graphs (VDGs) for modeling the economic and social value dependencies and their characteristics (qualities and strengths) in software projects. The definition of a type t VDG of a software project is provided by Definition 6.1. A type t VDG of a software project captures the dependencies among the type t values (e.g. economic values) of the requirements of that project.

Definition 6.1. *Value Dependency Graph* (VDG). A type t VDG is a signed directed fuzzy graph $G_t = (R, \sigma_t, \rho_t)$, where the requirement set $R : \{r_1, ..., r_n\}$ constitutes the graph nodes. Also, the qualitative function $\sigma_t(r_i,r_j) \rightarrow \{+, -, \pm\}$ and the membership function ρ_t: $(r_i, r_j) \rightarrow [0, 1]$ specify the quality and the strength of an explicit type t value dependency from r_i to r_j respectively. $\rho_t(r_i, r_j) = 0$ and $\sigma_t(r_i, r_j) = \pm$ specify the absence of any explicit value dependency of type t from r_i to r_j.

It is clear that the extended definition of the VDGs, as given by Definition 6.1, allows for the construction of multiple value dependency graphs for software projects, where each graph captures the value dependencies related to a specific aspect of social values.

6.2.2 The Economic and Social Value Dependencies in VDGs

Definition 6.2 provides a comprehensive definition of value dependencies that includes explicit and implicit value dependencies of different types. These value dependencies are defined based on the algebraic structure of fuzzy graphs.

Definition 6.2. *Value Dependencies*. A type t value dependency in a VDG $G_t = (R, \sigma_t, \rho_t)$ is defined as a sequence of the requirements $d_i : \big(r(0), ..., r(k)\big)$ such that $\forall r(j) \in d_i$, $1 \leq j \leq k$ we have $\rho_t\big(r(j-1), r(j)\big) \neq 0$. $j \geq 0$ is the sequence of the j^{th} requirement (node) denoted as $r(j)$ on the dependency path. A consecutive pair $\big(r(j-1), r(j)\big)$ specifies an explicit value dependency.

$$\forall d_i : \big(r(0), ..., r(k)\big) : \rho_t(d_i) = \bigwedge_{j=1}^{k} \rho_t\big(r(j-1), r(j)\big) \tag{6.1}$$

$$\forall d_i : \big(r(0), ..., r(k)\big) : \sigma_t(d_i) = \prod_{j=1}^{k} \sigma_t\big(r(j-1), r(j)\big) \tag{6.2}$$

Equation (6.1) computes the strength of a type t value dependency $d_i : \big(r(0), ..., r(k)\big)$ by finding the strength of the weakest of the k explicit type t dependencies on d_i. The fuzzy operator $\wedge$ denotes Zadeh's [141] AND operation (infimum). Also, the quality (positive or negative) of a type t value dependency $d_i : \big(r(0), ..., r(k)\big)$ is calculated by the qualitative serial inference [142, 143, 79] of (6.2) as shown in Table 6.1.

TABLE 6.1: Qualitative serial inference in a type t VDG.

$\sigma_t\big(r(j-1), r(j), r(j+1)\big)$		$\sigma_t\big(r(j), r(j+1)\big)$		
		$+$	$-$	$\pm$
$\sigma_t\big(r(j-1), r(j)\big)$	$+$	$+$	$-$	$\pm$
	$-$	$-$	$+$	$\pm$
	$\pm$	$\pm$	$\pm$	$\pm$

Let $D_t = \{d_1, d_2, ..., d_m\}$ be the set of all type t value dependencies from $r_i \in R$ to $r_j \in R$ in a type t VDG $G_t = (R, \sigma_t, \rho_t)$, where the positive and negative dependencies can simultaneously exist from r_i to r_j. The strength of all positive value dependencies of type t from r_i to r_j is denoted by $\rho_t^{+\infty}(r_i, r_j)$ and calculated by (6.3), that is to find the strength of the strongest positive dependency [42] from r_i to r_j.

Fuzzy operators $\wedge$ and $\vee$ denote Zadeh's [141] fuzzy AND (minimum) and fuzzy OR (maximum) operators respectively. In a similar way, the strength of all negative value dependencies of type t from r_i to r_j is denoted by $\rho_t^{-\infty}(r_i, r_j)$ and calculated by (6.4).

$$\rho_t^{+\infty}(r_i, r_j) = \bigvee_{d_m \in D_t, \sigma_t(d_m)=+} \rho_t(d_m) \tag{6.3}$$

$$\rho_t^{-\infty}(r_i, r_j) = \bigvee_{d_m \in D_t, \sigma_t(d_i)=-} \rho_t(d_m) \tag{6.4}$$

A brute-force approach to computing $\rho_t^{+\infty}(r_i, r_j)$ or $\rho_t^{-\infty}(r_i, r_j)$ in a type t VDG needs to calculate the strengths of all paths from r_i to r_j, which is of complexity of $O(n!)$ for n requirements (the VDG nodes). To avoid such a complexity, we have formulated the problem of calculating $\rho_t^{+\infty}(r_i, r_j)$ and $\rho_t^{-\infty}(r_i, r_j)$ as the widest path problem, also known as the maximum capacity path problem [144], which can be solved in polynomial time using the Floyd-Warshall algorithm [44].

In doing so, we devised a modified version of the Floyd-Warshall algorithm (Algorithm 6.1) that computes $\rho_t^{+\infty}(r_i, r_j)$ and $\rho_t^{-\infty}(r_i, r_j)$ for all pairs of the requirements (r_i, r_j), $r_i, r_j \in R : \{r_1, ..., r_n\}$ with the time bound of $O(n^3)$. For each pair of the requirements (r_i, r_j) in a type t VDG $G_t = (R, \sigma_t, \rho_t)$, lines 18 to 35 of Algorithm 6.1 find the strength of all positive value dependencies and the strength of all negative value dependencies from r_i to r_j.

$$I_{i,j,t} = \rho_t^{+\infty}(r_i, r_j) - \rho_t^{-\infty}(r_i, r_j) \tag{6.5}$$

The overall strength of all positive and negative value dependencies of type t from r_i to r_j is referred to as the *Influence* of r_j on the type t value of r_i and denoted by $I_{i,j,t}$. $I_{i,j,t} \in [-1, 1]$, as given by (6.5), is calculated by subtracting $\rho_t^{-\infty}(r_i, r_j)$ from $\rho_t^{+\infty}(r_i, r_j)$. $I_{i,j,t} > 0$ states that r_j influences the type t value of r_i in a positive way and $I_{i,j,t} < 0$ states that the ultimate influence of r_j on the type t value of r_i is negative.

Algorithm 6.1: Calculating the strengths of the type t value dependencies.

Input: VDG $G_t = (R, \sigma_t, \rho_t)$
Output: $\rho_t^{+\infty}, \rho_t^{-\infty}$

1: **for each** $r_i \in R$ **do**
2: **for each** $r_j \in R$ **do**
3: $\rho_t^{+\infty}(r_i, r_j) \leftarrow \rho_t^{-\infty}(r_i, r_j) \leftarrow -\infty$
4: **end for**
5: **end for**
6: **for each** $r_i \in R$ **do**
7: $\rho_t(r_i, r_i)^{+\infty} \leftarrow \rho_t(r_i, r_i)^{-\infty} \leftarrow 0$
8: **end for**
9: **for each** $r_i \in R$ **do**
10: **for each** $r_j \in R$ **do**
11: **if** $\sigma_t(r_i, r_j) = +$ **then**
12: $\rho_t^{+\infty}(r_i, r_j) \leftarrow \rho_t(r_i, r_j)$
13: **else if** $\sigma_t(r_i, r_j) = -$ **then**
14: $\rho_t^{-\infty}(r_i, r_j) \leftarrow \rho_t(r_i, r_j)$
15: **end if**
16: **end for**
17: **end for**
18: **for each** $r_k \in R$ **do**
19: **for each** $r_i \in R$ **do**
20: **for each** $r_j \in R$ **do**
21: **if** $min\left(\rho_t^{+\infty}(r_i, r_k), \rho_t^{+\infty}(r_k, r_j)\right) > \rho_t^{+\infty}(r_i, r_j)$ **then**
22: $\rho_t^{+\infty}(r_i, r_j) \leftarrow min(\rho_t^{+\infty}(r_i, r_k), \rho_t^{+\infty}(r_k, r_j))$
23: **end if**
24: **if** $min\left(\rho_t^{-\infty}(r_i, r_k), \rho_t^{-\infty}(r_k, r_j)\right) > \rho_t^{+\infty}(r_i, r_j)$ **then**
25: $\rho_t^{+\infty}(r_i, r_j) \leftarrow min(\rho_t^{-\infty}(r_i, r_k), \rho_t^{-\infty}(r_k, r_j))$
26: **end if**
27: **if** $min\left(\rho_t^{+\infty}(r_i, r_k), \rho_t^{-\infty}(r_k, r_j)\right) > \rho_t^{-\infty}(r_i, r_j)$ **then**
28: $\rho_t^{-\infty}(r_i, r_j) \leftarrow min(\rho_t^{+\infty}(r_i, r_k), \rho_t^{-\infty}(r_k, r_j))$
29: **end if**
30: **if** $min\left(\rho_t^{-\infty}(r_i, r_k), \rho_t^{+\infty}(r_k, r_j)\right) > \rho_t^{-\infty}(r_i, r_j)$ **then**
31: $\rho_t^{-\infty}(r_i, r_j) \leftarrow min(\rho_t^{-\infty}(r_i, r_k), \rho_t^{+\infty}(r_k, r_j))$
32: **end if**
33: **end for**
34: **end for**
35: **end for**

6.3 The Proposed Optimization Models for DARS-SOC

We have proposed two different optimization models for taking into account the economic and social values in DARS-SOC. The first optimization model is an integer linear programming (ILP) model that extends the ILP model of the DARS-ILP method (Section 4.4.2) by taking into account the social values of the requirements. The second optimization model is a mixed integer programming (MIP) model, which is based on the optimization model of the DARS-MIP method (Section 5.3). The proposed MIP model of DARS-SOC considers the social values of the requirements while allowing for partial selection of those requirements when that can be tolerated.

We consider two types of values in the optimization models of the DARS-SOC method. First is the economic value, which is manifested in terms of revenue/profit and second is the social values such as the values depicted in the value map of Figure 6.1. For the sake of the notational convenience, we specify the economic value of a software requirement r_i by $v_{i,1}$ while the social values of r_i are specified by $v_{i,2}, ..., v_{i,T}$. T denotes the total number of the values including the economic value.

In order to account for the impact of the value dependencies on different types of values, we compute the penalties of ignoring (selecting) the requirements with positive (negative) influences on the economic/social values of the requirements based on the algebraic structure of fuzzy graphs.

6.3.1 The Integer Linear Programming Model

Model (6.7)-(6.18) give our proposed ILP model for DARS-SOC. In these equations, x_i is a selection variable denoting whether a requirement r_i is selected ($x_i = 1$) or ignored ($x_i = 0$). Also $\theta_{i,t}$ in (6.11) specifies the penalty for the type t value of a requirement r_i, which is the extent to which the type t value of r_i is impacted by ignoring (selecting) requirements with positive (negative) influences on the type t value of r_i. Also, T in (6.7)-(6.18) specifies the total number of the value types including the economic value.

Moreover, $\theta_{i,t}$ as appeared in (6.11) gives the simplified algebraic expression of the penalty given in Equation (6.6). The simplified expression of $\theta_{i,t}$ in (6.11) is, of course,

equivalent of its original expression in (6.6). Moreover, (6.5) computes $I_{i,j,t}$, which is the influence of a requirement r_j on the type t value of r_i as explained in Section 6.2.

$$\theta_{i,t} = \bigvee_{j=1}^{n} \left(\frac{x_j\big(|I_{i,j,t}| - I_{i,j,t}\big) + (1 - x_j)\big(|I_{i,j,t}| + I_{i,j,t}\big)}{2} \right) = \bigvee_{j=1}^{n} \left(\frac{|I_{i,j,t}| + (1 - 2x_j) I_{i,j,t}}{2} \right), \quad i \neq j = 1, ..., n,\ t = 1, ..., T \tag{6.6}$$

As stated earlier, $v_{i,1}$ in (6.7)-(6.18) denotes the economic value of a requirement r_i and $E(v_{i,t})$, $t \in \{2, ..., n\}$ denotes the expected type t value of a requirement r_i. Similarly, in all other variables/parameters in (6.7)-(6.18), $t = 1$ denotes the variables/parameters related to the economic value while $t = \{2, ..., T\}$ specify the variables/parameters related to the social values. The expected values of the requirements are used in the proposed ILP model to account for the uncertainties associated with the economic/social values of those requirements.

The objective function (6.7) aims to optimize the economic value of a selected subset of the requirements subject to (6.8)-(6.18). Constraint (6.8) ensures that the total cost of the requirements will not exceed the project budget b. Also, (6.9) in the proposed model accounts for the precedence dependencies among the requirements and the value implications of those dependencies, which may impact all value types. Precedence dependencies mainly include the requirement dependencies of type *Requires* (*Conflicts-with*), where one requirement intrinsically requires (conflicts with) the other one.

The set of the *Social Constraints* (6.10) ensures that the minimum amounts (lower-bounds) required for the expected type t values of the requirements are provided. β_t in (6.10) denotes the required lower-bound for the expected type t value of the selected requirements.

Assigning proper values to β_t is specially useful for the reconciliation of the potential conflicts among the social values when the satisfaction of one value type conflicts with

the satisfaction of another value type. β_t can be modified in such cases to suit the value preferences of the stakeholders.

$$\text{Maximize } \sum_{i=1}^{n} x_i E(v_{i,1}) - y_{i,1} E(v_{i,1}) \tag{6.7}$$

$$\text{Subject to } \sum_{i=1}^{n} c_i x_i \leq b \tag{6.8}$$

$$\begin{cases} x_i \leq x_j & r_j \text{ precedes } r_i \\ x_i \leq 1 - x_j & r_i \text{ conflicts with } r_j,\ i \neq j = 1, ..., n \end{cases} \tag{6.9}$$

$$\sum_{i=1}^{n} x_i E(v_{i,t}) - y_{i,t} E(v_{i,t}) \geq \beta_t, \qquad t = 2, ..., T \tag{6.10}$$

$$\theta_{i,t} \geq \left(\frac{|I_{i,j,t}| + (1 - 2x_j) I_{i,j,t}}{2} \right), \qquad i \neq j = 1, ..., n,\ t = 1, ..., T \tag{6.11}$$

$$-g_i \leq x_i \leq g_i, \qquad i = 1, ..., n \tag{6.12}$$

$$1 - (1 - g_i) \leq x_i \leq 1 + (1 - g_i), \qquad i = 1, ..., n \tag{6.13}$$

$$-g_i \leq y_{i,t} \leq g_i, \qquad i = 1, ..., n,\ t = 1, ..., T \tag{6.14}$$

$$-(1 - g_i) \leq (y_{i,t} - \theta_{i,t}) \leq (1 - g_i), \qquad i = 1, ..., n,\ t = 1, ..., T \tag{6.15}$$

$$0 \leq y_{i,t} \leq 1, \qquad i = 1, ..., n,\ t = 1, ..., T \tag{6.16}$$

$$0 \leq \theta_{i,t} \leq 1, \qquad i = 1, ..., n,\ t = 1, ..., T \tag{6.17}$$

$$x_i, g_i \in \{0, 1\}, \qquad i = 1, ..., n \tag{6.18}$$

For a given requirement r_i, in (6.7)-(6.18) we have either $a : (x_i = 0, y_{i,t} = 0), t = 1, ..., T$, or $b : (x_i = 1, y_{i,t} = \theta_{i,t}), t = 1, ..., T$ occur. To capture the relation between $\theta_{i,t}$ and $y_{i,t}$ in a linear form, we have made use of an auxiliary variable $g_i = \{0, 1\}$ and (6.12)-(6.18). As such, we have either $(g_i = 0) \rightarrow a$, or $(g_i = 1) \rightarrow b$. The selection model (6.7)-(6.18) therefore is a linear model as it has a linear objective function with linear inequality constraints.

Moreover, the dependency identification technique presented in Section 4.2 can be used for automated identification of the economic value dependencies from user preferences. But, to the best of our knowledge, there are not any techniques in the present literature for identification of social value dependencies. These dependencies may be identified manually for small requirement sets. But development of more sophisticated techniques is needed for automated identification of social value dependencies in medium to large scale requirement sets. We have implemented and solved the ILP model of the DARS-SOC method using *IBM CPLEX* [40]. The *OPL* code for this model can be obtained from the website of DARS[11].

6.3.2 The Mixed Integer Programming Model

As discussed in Section 4.6 the precedence and budget constraints impact the effectiveness of the ILP model of DARS-ILP in mitigating the value loss. The reason is that the requirements with significant positive influences on the values of the selected requirements may have to be ignored due to their conflicts with other requirements or the lack of sufficient budget. Analogously, requirements with negative influences on the values of the requirements may need to be selected when they are required by other selected requirements. Hence, a value loss caused by ignoring (selecting) the requirements with positive (negative) influences may occur when the ILP model of DARS-ILP is used.

The ILP model of the DARS-SOC method is based on the ILP model of DARS-ILP and, thus, suffers from the same problem. To further mitigate the risk of value loss in DARS-SOC, we adopt the approach proposed in Chapter 5 by allowing for partial selection (satisfaction) of the requirements when that can be tolerated. For a type $t \in \{1, ..., T\}$ value, partial selection reduces the risk of value loss by increasing (decreasing) the investment in the satisfaction of the requirements that positively (negatively) influence the type t values of the requirements. A lower-bound cost and an upper-bound cost will be specified for the satisfaction of each requirement r_i. When r_i cannot be tolerated to be partially selected (satisfied), however, the lower-bound cost of r_i equals its upper-bound cost.

[11] http://bcert.org/projects/dars

One approach to specifying these boundaries is to use the estimated costs of the requirements as the upper-bound costs and then RELAX the effort needed for the satisfaction of those requirements, using the RELAX-ation technique proposed as part of the PAPS method presented in Section 5.2, to determine the lower bound costs of the requirements.

To allow for partial selection of requirements in DARS-SOC, we have contributed a mixed integer programming (MIP) model for integrating the economic/social value dependencies into requirement selection while allowing for partial selection of the requirements when that can be tolerated. Hence, the proposed DARS-MIP method finds an optimal investment policy for the satisfaction of the requirements, where such a policy increases (decreases) the investment in the requirements that positively (negatively) influence the economic/social values of the requirements.

Equations (6.19)-(6.28) give our proposed MIP model of DARS-SORC, which aims to find an optimal investment policy that maximizes the economic value of an optimal subset of the requirements while respecting the *Social Constraints* in (6.22) and keeping the cost within the budget b. The proposed model also mitigates the value loss caused by ignoring value dependencies in two ways: (i) by taking into account the influences of the requirements on the values of each other and (ii) by allowing for partial selection of the requirements when that can be tolerated.

In (6.19)-(6.28), the relaxed (real) variable $0 \leq x_i \leq 1$ specifies the investment ratio of a requirement r_i, which is the proportion of the budget invested in the satisfaction of r_i based on the optimal investment policy found by the MIP model of DARS-SOC. As given by (6.24), the investment ratio of r_i is contained within its lower-bound $g_i(\frac{c_{i,l}}{b})$ and upper-bound $g_i(\frac{c_{i,u}}{b})$, which are given by the lower-bound cost and the upper-bound cost of r_i specified by $c_{i,l}$, and $c_{i,u}$ respectively.

When there is no investment in r_i ($g_i = 0$), we have $x_i = 0$. But when there is some investment in r_i ($g_i = 1$), (6.24) determines the lower-bound $(\frac{c_{i,l}}{b})$ and the upper-bound $(\frac{c_{i,u}}{b})$ for the investment ratio of r_i. When r_i cannot be tolerated to be partially selected (satisfied), however, we have $c_{i,l} = c_{i,u}$, which means $x_i = \frac{c_{i,u}}{b}$ for $g_i = 1$, and $x_i = 0$ for $g_i = 0$. The optimization model of the DARS-MIP method builds an optimal

investment policy, which specifies the optimal subset of the requirements in which requirements are fully ($x_i = \frac{c_{i,u}}{b}$) or partially ($\frac{c_{i,l}}{b} \leq x_i < \frac{c_{i,u}}{b}$) included.

The expression $(\frac{x_i b}{c_{i,u}})E(v_{i,t})$ in (6.19) captures the impact of the investment ratio of a requirement r_i on the expected type t value ($E(v_{i,t})$) of r_i. Equation (6.29) gives $E(v_i)$, where $v_{i,t}$ denotes the estimated type t value of r_i and $p(r_i)$ denotes the probability that users purchase or use r_i.

We specify the economic value of a software requirement r_i by $v_{i,1}$ while the social values of r_i are specified by $v_{i,2}, ..., v_{i,T}$. T denotes the total number of the values including the economic value. Similarly, in all other variables/parameters in (6.19)-(6.28), $t = 1$ denotes the variables/parameters related to the economic value while $t = \{2, ..., T\}$ specify the variables/parameters related to the social values.

$$\text{Maximize } \sum_{i=1}^{n} (\frac{x_i b}{c_{i,u}})E(v_{i,1}) - y_{i,1}E(v_{i,1}) \tag{6.19}$$

$$\text{Subject to } \sum_{i=1}^{n} x_i \leq 1 \tag{6.20}$$

$$\begin{cases} g_i \leq g_j & r_j \text{ precedes } r_i \\ g_i \leq 1 - g_j & r_i \text{ conflicts with } r_j,\ i \neq j = 1, ..., n \end{cases} \tag{6.21}$$

$$\sum_{i=1}^{n} (\frac{x_i b}{c_{i,u}})E(v_{i,t}) - y_{i,t}E(v_{i,t}) \geq \beta_t, \qquad t = 2, ..., T \tag{6.22}$$

$$\theta_{i,t} \geq \left(\frac{|I_{i,j,t}| + \left(1 - 2(\frac{x_j b}{c_{j,u}})\right) I_{i,j,t}}{2}\right), \qquad i \neq j = 1, ..., n,\ t = 1, ..., T \tag{6.23}$$

$$g_i(\frac{c_{i,l}}{b}) \leq x_i \leq g_i(\frac{c_{i,u}}{b}), \qquad i = 1, ..., n \tag{6.24}$$

$$-g_i \leq y_{i,t} \leq g_i, \qquad i = 1, ..., n,\ t = 1, ..., T \tag{6.25}$$

$$-(1 - g_i) \leq y_{i,t} - \theta_{i,t} \leq (1 - g_i), \qquad i = 1, ..., n,\ t = 1, ..., T \tag{6.26}$$

$$g_i \in \{0, 1\}, \qquad i = 1, ..., n \tag{6.27}$$

$$0 \leq \theta_{i,t} \leq 1, \qquad i = 1, ..., n,\ t = 1, ..., T \tag{6.28}$$

$$E(v_i) = p(r_i)v_i \tag{6.29}$$

Moreover, $\theta_{i,t}$, as given by (6.23), specifies the type t penalty of a requirement r_i, which is the extent to which the type t expected value of r_i is negatively impacted by ignoring (selecting) requirements with positive (negative) influences on the type t value of r_i. We use the value dependency graphs (VDGs) presented in Section 6.2 for modeling value dependencies and reasoning about the qualities and strengths of those dependencies based on the algebraic structure of fuzzy graphs. Hence $\theta_{i,t}$ is computed by the fuzzy operator $\bigvee$, which finds the maximum of the influences of the ignored (selected) requirements with positive (negative) influence on the type t value of r_i.

When a requirement r_j with a positive influence on the type t value of r_i is ignored ($x_j = 0$), $I_{i,j,t}$ will be considered as the negative influence of ignoring r_j on the type t value of r_i as given by (6.23). But when r_j is partially selected ($\frac{c_{j,l}}{b} \leq x_j < \frac{c_{j,u}}{b}$), this implies that r_j is only partially ignored and therefore the negative influence of the ignoring r_j on the type t value of r_i is adjusted to $(1 - \frac{x_j b}{c_{j,u}})I_{i,j,t}$. Equation (6.5) computes $I_{i,j,t}$ as explained in Section 6.2.

Analogously, when a requirement r_j with a negative influence on the type t value of r_i is fully selected, this influence is computed as $I_{i,j,t}$ in (6.23). But when r_j is partially selected, the negative influence of the selecting r_j on the type t value of r_i is adjusted to $\frac{x_j b}{c_{j,u}} I_{i,j,t}$ in (6.23). In other words, the lower the investment in r_j, the lower the type t value loss caused by selecting r_j will be.

The set of the social constraints (6.22) ensures that the optimal policy found by the optimization model will provide an expected type t value, which is at least as large as β_t. Assigning proper values to β_t further helps reconcile the potential conflicts among the social values when the satisfaction of one value conflicts with the satisfaction of another one. β_t can be modified in such cases to suit the preferences of the stakeholders of the software projects.

It is also worth mentioning that the proposed MIP model of DARS-SOC uses value dependency graphs (VDGs) presented in Section 6.2 for modeling different types of value dependencies and reasoning about their characteristics.

Moreover, the dependency identification technique presented in Section 4.2 can be used for automated identification of the economic value dependencies from user preferences. But, to the best of our knowledge, there are not any techniques in the present literature for identification of social value dependencies. These dependencies may be identified manually for small requirement sets. But development of more sophisticated techniques is needed for automated identification of social value dependencies in medium to large scale requirement sets.

The MIP model of the DARS-SOC, as given by (6.19)-(6.28), is linear and therefore can be efficiently solved [39], even for large scale requirement sets, by the existing commercial solvers such as *IBM CPLEX* [40]. We have implemented and solved the MIP model of the DARS-SOC method using the *IBM CPLEX* [40]. The *OPL* code for this model can be obtained from the website of DARS[17].

6.4 Summary

In this chapter we proposed a method for dependency-aware software requirement selection that not only takes into account the economic values of the requirements but also accounts for social values of those requirements. The proposed method, referred to as the Society-Oriented method of DARS (DARS-SOC) helps embed social values into software requirement selection.

The proposed DARS-SOC method comprises two main optimization models with different characteristics, that allow for embedding the social values and dependencies among those values into software requirement selection. The first optimization model is an integer linear programming (ILP) model that aims to optimize the economic value of a selected subset of the requirements subject to a set of social constraints. The set of the social constraints ensures that the lower-bound of the social values are provided by the optimal subset of the requirements.

[17] http://bcert.org/projects/dars

The second optimization model presented for DARS-SOC is a mixed integer programming (MIP) model that enhances the ILP model of DARS-SOC by further mitigating the risk of value loss posed by ignoring (selecting) requirements with positive (negative) influences on the values of the selected requirements.

This is achieved in the proposed MIP model by allowing for partial selection (satisfaction) of the requirements when that can be tolerated. The model, thus, aims to find an optimal budget investment policy that maximizes the economic value of an optimal subset of the requirements while respecting the social constraints.

We have further extended the definitions of the value dependency graphs (VDGs) and value dependencies presented in Section 4.3.2 to capture different types of social values in modeling value dependencies.

Finally, the dependency identification component of the DARS-ILP method is used in DARS-SOC for automated identification of the economic value dependencies. But, devising more sophisticated techniques is needed for automated identification of social value dependencies in requirement sets. This is, however, beyond the scope of this thesis[19]. The contents of this chapter are presented in publications **(P3)**, **(P4)**, **(P6)**, **(P8)**, **(P10)**, **(P12)**, and **(P13)**.

[19]The author of the thesis has joined the Society-Oriented Software Design project at the Faculty of Information Technology, Monash University.

Chapter 7

Conclusions

7.1 Summary of the main contributions

Software requirement selection aims to find an optimal subset of requirements with the highest value while respecting the project constraints. Values of requirements, however, may positively or negatively depend on the presence or absence of other requirements in the optimal subset. Moreover, value dependencies are imprecise and hard to specify in software projects. Hence, it is important to consider *Value Dependencies* and the imprecision associated with those dependencies in software requirement selection. This thesis focused on considering value dependencies and their different aspects in software requirement selection.

To effectively consider value dependencies in software requirement selection this thesis presented a mathematical programming approach, referred to as *Dependency-Aware Requirement Selection* (DARS). The proposed approach comprises an *Integer Programing* (IP) method i.e. DARS-IP, an *Integer Linear Programming* (ILP) method i.e. DARS-ILP, a *Mixed Integer Programing* MIP method i.e. DARS-MIP, and a *Society-Oriented* method i.e. DARS-SOC with different characteristics for considering value dependencies in requirement selection. Each method is comprised of three major components: (i) a technique for identification of value dependencies; (ii) a technique for modeling value dependencies and computing influences of requirements on the values of each other; and (iii) a mathematical programming model for integrating value dependencies into finding optimal subsets of requirements. In this regard, the following contributions were presented.

7.1.1 The DARS-IP Method

Chapter 3 focused on considering the impacts of value dependencies on the value of an optimal subset of the requirements during a selection process. In this regard, an integer programming (IP) method of DARS, referred to as DARS-IP, was presented. The IP method includes; (i) identification of value dependencies, (ii) modeling value dependencies, and (iii) integrating value dependencies into software requirement selection.

We showed the practicality and validity of the DARS-IP by studying a real-world software project and carrying out simulations. Our results, as presented in publications **(P1)** and **(P2)**, answered research questions **(RQ1)**-**(RQ5)** indicating that (a) the IP method of DARS (DARS-IP) can properly capture the strengths of value dependencies among requirements during a selection process while mitigating the selection deficiency problem (SDP); (b) the DARS-IP method always maximizes the overall value of an optimal subset; (c) maximizing the overall value and the accumulated value of an optimal subset can be conflicting objectives [123] as maximizing one may depreciate the other.

7.1.2 The DARS-ILP Method

Chapter 4 focused on enhancing the main components of the DARS-IP method in different ways as presented in publications **(P1)**-**(P8)**. An integer linear programming (ILP) method referred to as DARS-ILP was presented for dependency-aware requirement selection, which is scalable to software projects with large number of requirements. The proposed ILP method of DARS (DARS-ILP) further enhances the IP method (DARS-IP) by considering not only the strengths but also qualities of value dependencies in the optimization model.

Moreover, the dependency identification technique in DARS-IP was improved by DARS-ILP through (a) considering both strengths and qualities of value dependencies and (b) using a formal significance test to prune value dependencies (which may be spurious rather than reflecting real relationships amongst values of requirements). The ILP method of DARS also allows for modeling negative value dependencies.

In this regard, a modified version of the Floyd-Warshall algorithm [44] was contributed that computes the positive and negative influences of requirements on the values of each other using the algebraic structure of fuzzy graphs.

Chapter 4 answered research questions **(RQ6)**-**(RQ9)** by studying a real-world software project and carrying out extensive simulations to study the impact of value dependencies on the effectiveness of DARS-ILP in mitigating the value loss caused by ignoring value dependences.

Our results show: that (a) compared to the requirement selection methods that ignore value dependencies, the ILP method of DARS provides higher overall value by mitigating the impact of ignoring (selecting) requirements with positive (negative) influence on the values of selected requirements; (b) maximizing the accumulated value and overall value of a software are conflicting objectives; and (c) DARS-ILP is scalable to software projects with large number of requirements for different levels of value dependencies and precedence dependencies. This was demonstrated by simulating different scenarios for datasets of up to 3000 requirements.

7.1.3 The DARS-MIP Method

Chapter 5 presented a mixed integer programming (MIP) method that integrates value dependencies into requirement selection while allowing for partial selection (satisfaction) of requirements when that can be tolerated. The MIP method of DARS (DARS-MIP) pursues the policy of partially selecting requirements rather than ignoring them or postponing them to the future releases. The main contributions of this chapter are presented in publications **(P3)**, **(P4)**, and **(P6)**-**(P12)**.

The optimization model of the DARS-MIP method finds an optimal investment policy that mitigates the value loss by allowing for increasing (decreasing) the investment in the requirements with significant positive (negative) influences on the values of the partially/fully selected requirements. The investment in each requirement r_i is bounded by the lower-bound cost and the upper-bound cost of r_i.

The upper-bound cost of r_i is estimated by the stakeholders and then RELAX-ed, using a RELAX-ation technique proposed as part of a fuzzy method referred to as *Prioritization and Partial selection* (PAPS), to determine the lower bound cost of r_i.

The optimization model of the DARS-MIP method is linear and scalable to software projects with large number of requirement. Application of the DARS-MIP method to real-world software projects is now under way as part of our ongoing research to further investigate the effectiveness of the method in mitigating the value loss in real-world settings.

7.1.4 The Society-Oriented DARS Method (DARS-SOC)

The DARS-IP, DARS-ILP, and DARS-MIP methods focus on the economic values of software requirements and the dependencies among those values. However, there are other types of human values i.e. *Social Values* with long term impacts on the society that are also important and need to be considered in software requirement selection.

To address this, we presented in Chapter 6 a society-oriented method for DARS, i.e. DARS-SOC, that accounts for social values in dependency-aware requirement selection. The proposed DARS-SOC method comprises two main optimization models, with different characteristics, that allow for embedding the social values and the dependencies among those values into software requirement selection. We further extended the value dependency graphs, presented as the modeling component of DARS-ILP, for capturing different types of social values in modeling value dependencies.

Our proposed DARS-SOC method uses the dependency identification component of DARS-ILP for identification of the economic value dependencies. But, to the best of our knowledge, there are not any techniques in the present literature for identification of social value dependencies. These dependencies may be identified manually for small requirement sets. But development of more sophisticated techniques is needed for automated identification of social value dependencies in medium to large scale requirement sets.

The contents of Chapter 6 are mainly presented in publications **(P3)**, **(P4)**, **(P6)**, **(P8)**, **(P10)**, **(P12)**, and **(P13)**.

7.2 Current Limitations

We discuss the limitations of this thesis to contain it within a complete and full understanding of its limitations. Typical of experimental studies [172], the limitations of the thesis belong to the classes of internal, external, conclusion, and construct validity.

7.2.1 Internal, External, and Conclusion Validity

Limitations concerning the internal, external, and conclusion validity of the proposed selection methods are mainly about evaluation of the impact of the selection methods on the actual value (manifested in terms of revenue, profit, or return) of software products and generalization of the results to industrial practice. Such limitations are typical of the software requirement selection (release planning) works regardless of the methods used. All the existing software requirement selection works including the main works reviewed in Section 2.2 (Table 2.2) share the same limitation due to the following reasons.

An ultimate evaluation of a software requirement selection method can be achieved by studying its impact on the actual value (manifested in terms of revenue, profit, or return) of software products as optimizing the value is the ultimate purpose of the requirement selection methods. Such study however, requires software vendors to invest in developing and/or maintaining different configurations of a software product based on different selection methods even if such investments are not economically worthwhile. This is difficult, if possible at all, to achieve in real-world software projects as software vendors hesitate to invest their money on developing different configurations of a software product proposed by different selection methods when the profitability of using those methods is under investigation.

To avoid developing multiple configurations of the same software product, one may suggest releasing different configurations of an already existing software product based on the requirement subsets found by different selection methods and then study the value achieved from each configuration to understand the impact of using different selection methods on the actual value.

However, there are several complexities concerning the technical and financial aspects of releasing and maintaining different configurations of a software products, which makes it less appealing to most investors unless they can be assured that the released configurations are likely to be economically worthwhile.

Moreover, the prices of different configurations of a software product can also be a determining factor in choosing certain configurations of a software product by the users. In other words, the higher/lower price of a configuration may encourage the users to choose another configuration. Widely recognized in finance [147], the interplay between the sales and price is hard to exclude from analysis of value. This further exacerbates the difficulty of evaluating the impacts of requirement selection methods on the actual value.

Furthermore, users may compare a software product with its alternatives in the market, whether it is from the same vendor or other competitors. Hence, users satisfaction or dissatisfaction with a certain product may impact their preferences for its alternative products in the market. The dynamics of the software market and the role of the competitors in increasing or decreasing the sales and subsequently the value of a software product however, is hard to predict.

For the these reasons and other factors such as marketing and familiarity of the users with the features of software products, that also impact the sales and value, it is hard to specifically evaluate the impact of the requirement selection methods on the actual value. Hence, unless the above-mentioned factors are controlled somehow, the impacts of the existing requirement selection works, including the main works reviewed in Section 2.2 (Table 2.2), on the actual value is unknown. Unfortunately, there are no studies in the present literature, to the best of our knowledge, that has addressed this.

7.2.2 Construct Validity

Limitations concerning the construct validity are mainly about measures and tools used in the experiment design. We used different measures of causal strength [45] in Chapters 3-5 in this thesis to estimate qualities and strengths of value dependencies.

However, further studies are needed to investigate the accuracy of such estimations for different datasets as, to the best of our knowledge, no previous work has measured value dependencies among software requirements.

Moreover, in Section 4.5 of this thesis we used the sales record of a software product to extract the frequencies of the user preferences for different requirements and identify the value dependencies among those requirements. It is clear that, in this way, preferences of the users are impacted by the prices of the requirements. In other words, a higher price might have encouraged or discouraged users to purchase a certain feature of a software product. Analogously, a lower price might also have encouraged or discouraged users to choose certain features. This was not an issue in the case study of Section 4.5 as a price change was not intended by the stakeholders.

But, in other cases, the interplay between the price and user preferences may interfere with reflecting the actual preference of the users. In other words, when the prices of the requirements of a new release of a software product are significantly different from their original prices in the earlier version(s), the value dependencies found from the sales records might be misguiding. In such cases tracking the usage patterns of the users in the earlier version(s) of a software product or similar products in the market may help extract the user preferences and estimate the value dependencies among the requirements. Hence, more sophisticated techniques are needed to allow for capturing the preferences of the users from different sources.

Finally, in this thesis we did not cover automated identification of social value dependencies among requirements, which is required by the optimization model of the society-oriented method of DARS (DARS-SOC). These dependencies maybe identified manually for small requirement sets. But for medium to large requirement sets, developing more sophisticated techniques for automated identification of social value dependencies is necessary. To achieve this, we further need to develop proper measures for evaluating social values in software projects.

7.3 Ongoing and Future Work[1]

To address the limitations of the thesis discussed in Section 7.2, the following research directions can be explored. Some of these directions are part of our ongoing research while the rest can be pursued as future work.

7.3.1 Enhancing the Accuracy of the Dependency Identification

Enhancing the Quality of the Collected Data

Practicality of the proposed requirement selection methods, namely DARS-IP, DARS-ILP, DARS-MIP, and DARS-SOC, can be enhanced by improving the techniques used for the automated collection of user preferences, which serve as the input to the dependency identification components of the proposed selection methods.

Conducting surveys is a conventional way of collecting user preferences. But complementary techniques are also required to be used in combination with surveys to overcome the limitations of the surveys (Section 7.2.2) and enhance the quality of the collected user preferences.

In Section 4.5 we demonstrated the use of the sales records of a software product to extract the frequencies of the user preferences for different requirements and identify the value dependencies. This was reasonable to be used in the case study of Section 4.5 as a price change was not intended by the stakeholders. But, in other cases, preferences of the users may be impacted by the prices of the requirements as explained in Section 7.2.2.

In this regard, mining user opinions in online stores (repositories) and information gathered from tracking user behavior on similar software seem to be promising approaches. This, particularly, helps factor out the impact of the price on user preferences as discussed in Section 7.2.2. limited by the accuracy of its text-mining technique, the work [138] is an interesting example for gathering user opinions from online sources.

[1]The author of the thesis has recently joined the Society-Oriented Software Design project at the Faculty of Information Technology, Monash University.

Establishing a Shared Repository for User Preferences

Gathering user preferences for different requirements of software products may be resource consuming and, therefore, not affordable to smaller software vendors. Larger software vendors on the other hand, have access to more resources and most likely to benefit from more efficient ways of gathering user preferences. However, such information is rarely available to smaller software vendors for a variety of reasons including the resource limitations and confidentiality matters.

As such, smaller software vendors (e.g. developers of the mobile applications), which constitute the majority of the active players in the software market, cannot properly take into account the actual preferences of the users. This can be mitigated by establishing an open access shared repository of user preferences, which can be efficiently accessed by the developers as well as the software development tools. This lays a solid foundation for the automated integration of the user preferences into software development activities including the requirement selection. Such a repository will also allow users to monitor how their preferences are taken into account by different vendors.

The Classification of Software Requirements

There might be cases, where identification of value dependencies among individual requirements is not feasible (say due to the lack of time or sufficient data) while value dependencies among classes of requirements can be found and taken into account. For instance, choosing requirements belonging to the class of security may increase users satisfaction with the requirements belonging to the class of financial transactions and therefore the presence of security features may improve the values of the financial transaction requirements.

Hence, considering value dependencies among classes of software requirements at different levels of abstraction may still help prevent value loss caused by ignoring those dependencies. Value dependencies identified at higher levels of abstraction may be used to identify value dependencies among lower level classes of requirements and individual requirements.

This provides a structured way of integrating value dependencies into requirement selection and consequently mitigating the value loss caused by ignoring value dependencies even if identification of those dependencies among individual requirements is not feasible.

The Classification of Users

Identification of value dependencies for different classes of users helps enhance the accuracy of the identified value dependencies and therefore improve the effectiveness of the requirement selection methods. This way, different configurations of a software product can be released for different classes of the users based on the value dependencies identified for those classes. This further increases users satisfaction and mitigates the value loss for different configurations of a software targeted for different classes of users.

Enhancing the Accuracy of the Dependency Identification

The accuracy of the dependency identification technique used in this thesis can be enhanced by exploring different measures of casual strength on a variety of the real-world datasets. Also different fuzzy membership functions can be explored, based on the feedback from software practitioners, for enhancing the accuracy of the strengths and qualities of the value dependencies among software requirements.

7.3.2 Embedding Social Values into the Requirement Selection

Software requirement selection is traditionally guided by the economic worth of software, i.e. value. Social values in software and the dependencies among those values, however, are also important [173] and need to be considered in dependency-aware requirement selection. In this regard, Chapter 6 of this thesis presented a Society-Oriented method of DARS, referred to as DARS-SOC, which integrates social values into requirement selection. The DARS-SOC method can be improved in several ways.

The Identification of Social Value Dependencies

In this thesis we did not cover automated identification of social value dependencies among requirements, which is required by the optimization model of the society-oriented method of DARS (DARS-SOC). These dependencies maybe identified manually for small requirement sets. But for medium to large size requirement sets, developing more sophisticated techniques for automated identification of social value dependencies is necessary. To achieve this, we need to develop proper measures for evaluating social values in software projects.

Lack of proper criteria for measuring social values has been characterized in th literature [174] as a major challenge to the effectiveness of the methods used for embedding social/sustainability values into software products. The first step to the automated identification of social value dependencies, thus, is to address this challenge by introducing practical measures for social values. Such measures must be able to properly capture the imprecision of social values. Moreover, it is important to develop the measures of social values in full participation with stakeholders of software projects as social values may change across different classes of stakeholders.

Hence, establishing a collaborative platform, in which a participatory development of the standards and guidelines concerning the identification and measuring social values is supported, can be of significant benefit. For instance, in partnership with the stakeholders, customizable measures of social values can be developed using a *Goal Question Metric* (GQM) approach [175], where the stakeholders specify the goals based on their particular needs.

Embedding Social Values into the Requirement Modeling

A proper modeling technique is required for modeling social values and considering the imprecision of those values in software projects. Such a model will serve as the input for a requirement selection process. Social values are treated as non-functional requirements or soft-goals [37]. Hence, adopting goal-oriented techniques such as [78, 174] for construction of Social Value Model (SVM) of software products help capture these soft-goals and their relations with functional requirements of software products.

Moreover, participatory [176] construction of the social value model (SVM) of software products helps take into account opinions of different stakeholders when specifying social values as those values may differ across different classes of the stakeholders.

SVMs may use *Fuzzy Reasoning* [161, 1] for capturing the uncertainty of relations and conflicts among social goals and requirements. The SVM of a software product can also allow, through impact analysis, for investigating the consequences of breaching social values in software to give more visibility [37] to social values and their direct and indirect [1] impacts.

7.3.3 Applications to Other Problems

Consider a general form of the dependency-aware requirement selection, which is a *Binary Knapsack Problem with Dependent Item Values* (BKP-DIV) as described in Publication **(P3)** [3]. The problem of concern is to find an optimal subset of the items with the highest value without the assumption of independence among values of the requirements. In other words, the values of the selected items may change in the presence or absence of other items in the knapsack.

In other words, the value of an item is not known until the decision is made about the presence or absence of the other items in the knapsack.This SKP is one of the most commonly found problems in Software Engineering activities as well as other real-world problems. In fact it can be rarely assumed that the values of the items of a real-world selection problem are independent. We have also formulated a similar problem, referred to as the *Synergistic Knapsack Problem* (SKP) in publication **(P4)** [4].

Among the many applications of the BKP-DIV and SKP, are the software requirement selection problem [5, 2], software test-case selection problem, and medical treatment selection problem, where the value (effectiveness) of a certain treatment may change in the presence or absence of other treatments. We have discussed the relations among medical treatments and the uncertainty of those relations in publication **(P14)** [14]. This publication lays a foundation for a dependency-aware selection of medical treatments.

Bibliography

[1] D. Mougouei and D. M. W. Powers. Modeling and selection of interdependent software requirements using fuzzy graphs. *International Journal of Fuzzy Systems*, 19(6):1812–1828, Dec 2017.

[2] D. Mougouei. Factoring requirement dependencies in software requirement selection using graphs and integer programming. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 884–887. ACM, 2016.

[3] D. Mougouei, D. M. W. Powers, and A. Moeini. An integer linear programming model for binary knapsack problem with dependent item values. In W. Peng, D. Alahakoon, and X. Li, editors, *AI 2017: Advances in Artificial Intelligence: 30th Australasian Joint Conference, Melbourne, VIC, Australia, August 19–20, 2017, Proceedings*, pages 144–154. Springer International Publishing, Cham, 2017.

[4] D. Mougouei and D. M. W. Powers. The synergistic knapsack problem. *Fuzzy Optimization and Decision Making*, Under Review.

[5] D. Mougouei, D. M. Powers, and A. Moeini. Dependency-aware software release planning. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 198–200. ACM, 2017.

[6] D. Mougouei and D. M. W. Powers. An integer programming method for considering value-related dependencies in software requirement selection. *Information and Software Technology*, Under Review.

[7] D. Mougouei and D. M. W. Powers. Dependency-aware software release planning using fuzzy graphs and integer programming. *Engineering Applications of Artificial Intelligence*, Under Review.

[8] D. Mougouei and D. M. W. Powers. Dependency-aware software release planning through mining user preferences. *Expert Systems with Applications*, Under Review.

[9] D. Mougouei, H. Shen, and A. Babar. Partial selection of agile software requirements. *International Journal of Software Engineering & Its Applications*, 9(1):113–126, 2015.

[10] D. Mougouei and D. M. W. Powers. Paps: A scalable framework for prioritization and partial selection of security requirements. *International Journal of Approximate Reasoning*, Under Review.

[11] D. Mougouei and M. K. Yeung. Visibility requirements engineering for commercial websites. *International Journal of Software Engineering & Its Applications*, 8(8):11–18, 2014.

[12] D. Mougouei and D. M. W. Powers. Partial selection of software requirements. *International Conference on Computer Science, Engineering and Applications*, Accepted.

[13] D. Mougouei and D. M. W. Powers. An integer programming model for embedding social values into software requirement selection. *International Conference on Computer Science, Engineering and Applications*, Accepted.

[14] D. Mougouei and D. M. W. Powers. Gotm: a goal-oriented framework for capturing uncertainty of medical treatments. *Intelligent Systems Conference (IntelliSys) 2018*, Accepted.

[15] D. Mougouei. Considering value-related dependencies among requirements in software release planning: An integer programming approach. Faculty of Information Technology, Monash University, July 2017.

[16] A. J. Bagnall, V. J. RaywardSmith, and I. M. Whittley. The next release problem. *Information and Software Technology*, 43(14):883–890, Dec. 2001.

[17] X. Franch and G. Ruhe. Software release planning. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 894–895. ACM, 2016.

[18] Å. G. Dahlstedt and A. Persson. Requirements interdependencies: state of the art and future challenges. In *Engineering and managing software requirements*, pages 95–116. Springer, 2005.

[19] Y. Zhang, M. Harman, and S. L. Lim. Empirical evaluation of search based requirements interaction management. *Information and Software Technology*, 55(1):126 – 152, 2013. Special section: Best papers from the 2nd International Symposium on Search Based Software Engineering 2010.

[20] W. N. Robinson, S. D. Pawlowski, and V. Volkov. Requirements interaction management. *ACM Comput. Surv.*, 35(2):132-190, June 2003.

[21] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. Natt och Dag. An industrial survey of requirements interdependencies in software product release planning. In *Fifth IEEE International Symposium on Requirements Engineering, 2001. Proceedings*, pages 84–91, 2001.

[22] C. Li, M. v. d. Akker, S. Brinkkemper, and G. Diepen. An integrated approach for requirement selection and scheduling in software release planning. *Requirements Engineering*, 15(4):375–396, Nov. 2010.

[23] H. Zhang, J. Li, L. Zhu, R. Jeffery, Y. Liu, Q. Wang, and M. Li. Investigating dependencies in software requirements for change propagation analysis. *Information and Software Technology*, 56(1):40–53, 2014.

[24] J. Karlsson, S. Olsson, and K. Ryan. Improved practical support for largescale requirements prioritising. *Requirements Engineering*, 2(1):51–60, Mar. 1997.

[25] A. Ngo-The and G. Ruhe. A systematic approach for solving the wicked problem of software release planning. *Soft Computing*, 12(1):95–108, 2008.

[26] A. Ngo-The and M. O. Saliu. Measuring dependency constraint satisfaction in software release planning using dissimilarity of fuzzy graphs. In *Cognitive Informatics, 2005.(ICCI 2005). Fourth IEEE Conference on*, pages 301–307. IEEE, 2005.

[27] J. Wang, J. Li, Q. Wang, H. Zhang, and H. Wang. A simulation approach for impact analysis of requirement volatility considering dependency change. *Requirements Engineering: Foundation for Software Quality*, pages 59–76, 2012.

[28] P. Baker, M. Harman, K. Steinhofel, and A. Skaliotis. Search based approaches to component selection and prioritization for the next release problem. In *Proceedings of the 22Nd IEEE International Conference on Software Maintenance*, pages 176–185. IEEE, 2006.

[29] M. A. Boschetti, M. Golfarelli, S. Rizzi, and E. Turricchia. A lagrangian heuristic for sprint planning in agile software development. *Computers & Operations Research*, 43:116–128, Mar. 2014.

[30] A. A. Araújo, M. Paixao, I. Yeltsin, A. Dantas, and J. Souza. An architecture based on interactive optimization and machine learning applied to the next release problem. *Automated Software Engineering*, pages 1–49, 2016.

[31] D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243 – 253, 2004.

[32] A. Pitangueira, P. Tonella, A. Susi, R. Maciel, and M. Barros. Minimizing the stakeholder dissatisfaction risk in requirement selection for next release planning. *Information and Software Technology*, 2017.

[33] L. Li, M. Harman, F. Wu, and Y. Zhang. The value of exact analysis in requirements selection. *IEEE Transactions on Software Engineering*, 43(6):580–596, 2017.

[34] L. Li, M. Harman, E. Letier, and Y. Zhang. Robust next release problem: handling uncertainty during optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 1247–1254. ACM, 2014.

[35] M. van den Akker, S. Brinkkemper, G. van Diepen, and J. Versendaal. Flexible release planning using integer linear programming. *REFSQ'05*, 2005.

[36] J. d. Sagrado, I. M. d. Águila, and F. J. Orellana. Multiobjective ant colony optimization for requirements selection. *Empirical Software Engineering*, pages 1–34, Nov. 2013.

[37] M. A. Ferrario, W. Simm, S. Forshaw, A. Gradinar, M. T. Smith, and I. Smith. Values-first se: research principles in practice. In *Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on*, pages 553–562. IEEE, 2016.

[38] S. H. Schwartz. Basic human values: An overview. *Recuperado de http://www. yourmorals. org/schwartz*, 2006.

[39] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[40] I. I. CPLEX. V12.7: Cplex user's manual. *International Business Machines Corporation*, 2016.

[41] B. Fitelson and C. Hitchcock. *Probabilistic measures of causal strength*. na, 2011.

[42] A. Rosenfeld. Fuzzy graphs. *Fuzzy Sets and Their Applications*, 77:95, 1975.

[43] A. Kalampakas, S. Spartalis, L. Iliadis, and E. Pimenidis. Fuzzy graphs: algebraic structure and syntactic recognition. *Artificial Intelligence Review*, pages 1–12, July 2013.

[44] R. W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[45] E. Eells. *Probabilistic causality*, volume 1. Cambridge University Press, 1991.

[46] J. H. Macke, P. Berens, A. S. Ecker, A. S. Tolias, and M. Bethge. Generating spike trains with specified correlation coefficients. *Neural Computation*, 21(2):397–423, 2009.

[47] A. Ngo The and M. O. Saliu. Fuzzy structural dependency constraints in software release planning. In *The 14th IEEE International Conference on Fuzzy Systems, 2005. FUZZ'05.*, pages 442–447. IEEE, 2005.

[48] X. F. Liu and J. Yen. An analytic framework for specifying and analyzing imprecise requirements. In *Proceedings of the 18th international conference on Software engineering*, pages 60–69. IEEE Computer Society, 1996.

[49] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[50] S. Bradley, A. Hax, and T. Magnanti. Applied mathematical programming. 1977.

[51] W. Cui, Y. Wu, S. Liu, F. Wei, M. X. Zhou, and H. Qu. Context preserving dynamic word cloud visualization. In *Visualization Symposium (PacificVis), 2010 IEEE Pacific*, pages 121–128. IEEE, 2010.

[52] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[53] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[54] R. Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952.

[55] P. Festa. A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems. In *Transparent Optical Networks (ICTON), 2014 16th International Conference on*, pages 1–20. IEEE, 2014.

[56] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[57] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. In *Spin Glass Theory and Beyond: An Introduction to the Replica Method and Its Applications*, pages 339–348. World Scientific, 1987.

[58] D. E. Goldberg. Genetic algorithms in search, optimization, and machine learning, 1989. *Reading: Addison-Wesley*, 1989.

[59] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[60] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 29(3):653–684, 2000.

[61] E. H. Aarts and J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 1997.

[62] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In *Meta-heuristics*, pages 433–458. Springer, 1999.

[63] T. A. Feo and M. G. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71, 1989.

[64] S. Voß, S. Martello, I. H. Osman, and C. Roucairol. *Meta-heuristics: Advances and trends in local search paradigms for optimization*. Springer Science & Business Media, 2012.

[65] I. H. Osman and G. Laporte. Metaheuristics: A bibliography, 1996.

[66] M. Harman and B. F. Jones. Search-based software engineering. *Information and software Technology*, 43(14):833–839, 2001.

[67] B. A. McCarl and T. H. Spreen. Applied mathematical programming using algebraic systems. *Cambridge, MA*, 1997.

[68] D. G. Luenberger and Y. Ye. The simplex method. In *Linear and Nonlinear Programming*, pages 33–82. Springer, 2016.

[69] S. Sinha. *Mathematical Programming: Theory and Methods*. Elsevier, 2005.

[70] D.-S. Chen, R. G. Batson, and Y. Dang. *Applied integer programming: modeling and solution*. John Wiley & Sons, 2011.

[71] M. M. A. Brasil, T. G. N. d. Silva, F. G. d. Freitas, J. T. d. Souza, and M. I. Cortés. A multiobjective optimization approach to the software release planning with undefined number of releases and interdependent requirements. In R. Zhang, J. Zhang, Z. Zhang, J. Filipe, and J. Cordeiro, editors, *Enterprise Information Systems*, number 102, pages 300–314. Springer Berlin Heidelberg, Jan. 2012.

[72] A. M. Pitangueira, R. S. P. Maciel, and M. Barros. Software requirements selection and prioritization using sbse approaches: A systematic review and mapping of the literature. *Journal of Systems and Software*, 103:267–280, 2015.

[73] P. K. *Process-Centered Requirements Engineering*. Research Studies Pre, Taunton, Somerset, England New York, Oct. 1996.

[74] J. N. och Dag, B. Regnell, P. Carlshamre, M. Andersson, and J. Karlsson. A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7(1):20–33, 2002.

[75] J. Li, R. Jeffery, K. H. Fung, L. Zhu, Q. Wang, H. Zhang, and X. Xu. *A Business Process-Driven Approach for Requirements Dependency Analysis*, pages 200–215. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[76] D. Mougouei, M. Moghtadaei, and S. Moradmand. A goal-based modeling approach to develop security requirements of fault tolerant security-critical systems. In *Computer and Communication Engineering (ICCCE), 2012 International Conference on*, pages 200–205. IEEE, 2012.

[77] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge University Press, 1994.

[78] D. Mougouei. Goal-based requirement engineering for fault tolerant security-critical systems. *International Journal of Software Engineering and Its Applications*, 7(5):1–14, 2013.

[79] A. Kusiak and J. Wang. Dependency analysis in constraint negotiation. *Systems, Man and Cybernetics, IEEE Transactions on*, 25(9):1301–1313, 1995.

[80] A. G. Dahlstedt and A. Persson. Requirements interdependencies - moulding the state of research into a research agenda. In *Ninth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2003)*, pages 71–80, 2003.

[81] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, Jan. 2001.

[82] J. Karlsson and K. Ryan. A costvalue approach for prioritizing requirements. *IEEE Software*, 14(5):67–74, Sept. 1997.

[83] H.-W. Jung. Optimizing value and cost in requirements analysis. *IEEE Software*, 15(4):74–78, July 1998.

[84] Y. Zhang, M. Harman, and S. A. Mansouri. The multi-objective next release problem. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, page 1129–1137, New York, NY, USA, 2007. ACM.

[85] A. Finkelstein, M. Harman, S. A. Mansouri, J. Ren, and Y. Zhang. A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. *Requirements Engineering*, 14(4):231–245, 2009.

[86] Y. Zhang, M. Harman, A. Finkelstein, and S. A. Mansouri. Comparing the performance of metaheuristics for the analysis of multi-stakeholder tradeoffs in requirements optimisation. *Information and Software Technology*, 53(7):761–773, 2011.

[87] J. del Sagrado, I. M. del Aguila, and F. J. Orellana. Ant colony optimization for the next release problem: A comparative study. In *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on*, pages 67–76. IEEE, 2010.

[88] A. C. Kumari, K. Srinivas, and M. Gupta. Software requirements selection using quantum-inspired elitist multi-objective evolutionary algorithm. In *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on*, pages 782–787. IEEE, 2012.

[89] N. Veerapen, G. Ochoa, M. Harman, and E. K. Burke. An integer linear programming approach to the single and bi-objective next release problem. *Information and Software Technology*, 65:1–13, 2015.

[90] D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. 46(4):243–253, 2004.

[91] G. Ruhe and D. Greer. Quantitative studies in software release planning under risk and resource constraints. In *Proceedings of the 2003 International Symposium on Empirical Software Engineering*, pages 262–270, Sept 2003.

[92] G. van Valkenhoef, T. Tervonen, B. de Brock, and D. Postmus. Quantitative release planning in extreme programming. *Information and software technology*, 53(11):1227–1235, 2011.

[93] Y. Zhang and M. Harman. Search based optimization of requirements interaction management. In *Search Based Software Engineering (SSBSE), 2010 Second*

International Symposium on, pages 47–56. IEEE, 2010.

[94] P. Tonella, A. Susi, and F. Palma. Using interactive ga for requirements prioritization. In *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on*, pages 57–66. IEEE, 2010.

[95] F. G. Freitas, D. P. Coutinho, and J. T. Souza. Software next release planning approach through exact optimization. *Int. J. Comput. Appl*, 22(8):1–8, 2011.

[96] F. Colares, J. Souza, R. Carmo, C. Padua, and G. Mateus. A new approach to the software release planning. In *Software Engineering, 2009. SBES '09. XXIII Brazilian Symposium on*, pages 207–215, Oct 2009.

[97] M. O. Saliu and G. Ruhe. Bi-objective release planning for evolving software systems. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 105–114. ACM, 2007.

[98] O. Saliu and G. Ruhe. Supporting software release planning decisions for evolving systems. In *29th Annual IEEE/NASA Software Engineering Workshop*, pages 14–26. IEEE, 2005.

[99] H. Jiang, J. Zhang, J. Xuan, Z. Ren, and Y. Hu. A hybrid aco algorithm for the next release problem. In *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on*, pages 166–171. IEEE, 2010.

[100] M. van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal. Determination of the next release of a software product: an approach using integer linear programming. In *CAiSE Short Paper Proceedings*, 2005.

[101] A. Ngo-The and G. Ruhe. Optimized resource allocation for software release planning. *IEEE Transactions on Software Engineering*, 35(1):109–123, 2009.

[102] W.-N. Chen and J. Zhang. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *IEEE Transactions on Software Engineering*, 39(1):1–17, 2013.

[103] J. del Sagrado, I. M. ÁAguila, and F. J. Orellana. Requirements interaction in the next release problem. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 241–242. ACM, 2011.

[104] F. Colares, J. Souza, R. Carmo, C. Pádua, and G. R. Mateus. A new approach to the software release planning. In *Software Engineering, 2009. SBES'09. XXIII Brazilian Symposium on*, pages 207–215. IEEE, 2009.

[105] A. M. Pitangueira, P. Tonella, A. Susi, R. S. Maciel, and M. Barros. Risk-aware multi-stakeholder next release planning using multi-objective optimization. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 3–18. Springer, 2016.

[106] M. van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal. Software product release planning through optimization and what-if analysis. *Information and Software Technology*, 50(1):101–111, 2008.

[107] P. Tonella, A. Susi, and F. Palma. Interactive requirements prioritization using a genetic algorithm. *Information and software technology*, 55(1):173–187, 2013.

[108] J. Xuan, H. Jiang, Z. Ren, and Z. Luo. Solving the large scale next release problem with a backbone-based multilevel algorithm. *IEEE Transactions on Software Engineering*, 38(5):1195–1212, 2012.

[109] O. Saliu and G. Ruhe. Software release planning for evolving systems. *Innovations in Systems and Software Engineering*, 1(2):189–204, 2005.

[110] J. J. Burg, J. Ainsworth, B. Casto, and S.-D. Lang. Experiments with the "oregon trail knapsack problem". *Electronic Notes in Discrete Mathematics*, 1:26–35, 1999.

[111] M. Harman, J. Krinke, I. MedinaBulo, F. PalomoLozano, J. Ren, and S. Yoo. Exact scalable sensitivity analysis for the next release problem. *ACM Trans. Softw. Eng. Methodol.*, 23(2):19:1–19:31, Apr. 2014.

[112] M. I. Henig. Risk criteria in a stochastic knapsack problem. *Operations research*, 38(5):820–825, 1990.

[113] T. Doganoglu, C. Hartz, and S. Mittnik. Portfolio optimization when risk factors are conditionally varying and heavy tailed. *Computational Economics*, 29(3):333–354, 2007.

[114] J. Suárez-Lledó. The black swan: the impact of the highly improbable. *The Academy of Management Perspectives*, 25(2):87–90, 2011.

[115] The pitfalls of modern portfolio theory - assumptions.

[116] S. Barney, A. Aurum, and C. Wohlin. A product management challenge: Creating software product value through requirements selection. *Journal of Systems Architecture*, 54(6):576–593, June 2008.

[117] G. Ruhe. *Product Release Planning: Methods, Tools and Applications*. Taylor & Francis, June 2010.

[118] P. Achimugu, A. Selamat, R. Ibrahim, and M. N. Mahrin. A systematic literature review of software requirements prioritization research. *Information and Software Technology*, 56(6):568–585, June 2014.

[119] N. Kukreja, S. S. Payyavula, B. Boehm, and S. Padmanabhuni. Value-based requirements prioritization: Usage experiences. *Procedia Computer Science*, 16:806–813, 2013.

[120] P. Carlshamre. Release planning in market-driven software product development: Provoking an understanding. *Requirements Engineering*, 7(3):139–151, Sept. 2002.

[121] H.-J. Zimmermann. Fuzzy relations and fuzzy graphs. In *Fuzzy Set Theory and Its Applications*, pages 69–89. Springer Netherlands, Jan. 1996.

[122] J. N. Mordeson. Fuzzy mathematics. In L. S. Davis, editor, *Foundations of Image Understanding*, number 628, pages 95–125. Springer US, Jan. 2001.

[123] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2006.

[124] S. Mathew and M. Sunitha. Strongest strong cycles and theta fuzzy graphs. *IEEE Transactions on Fuzzy Systems*, 21(6):1096-1104, Dec 2013.

[125] J. Karlsson and K. Ryan. Supporting the selection of software requirements. In *Proceedings of the 8th International Workshop on Software Specification and Design, 1996.*, pages 146–149, Mar 1996.

[126] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 2nd edition, 1997.

[127] Random (Java Platform SE 7).

[128] J. Y. Halpern and C. Hitchcock. Graded causation and defaults. *The British Journal for the Philosophy of Science*, 66(2):413–457, 2015.

[129] T. do Nascimento Ferreira, A. A. Araújo, A. D. B. Neto, and J. T. de Souza. Incorporating user preferences in ant colony optimization for the next release problem. *Applied Soft Computing*, 49:1283–1296, 2016.

[130] Z. Racheva, M. Daneva, K. Sikkel, and L. Buglione. Business value is not only dollars – results from case study research on agile software projects. In M. A. Babar, M. Vierimaa, and M. Oivo, editors, *ProductFocused Software Process Improvement*, number 6156 in Lecture Notes in Computer Science, pages 131–145. Springer Berlin Heidelberg, Jan. 2010.

[131] J. Pearl. *Causality*. Cambridge university press, 2009.

[132] D. Janzing, D. Balduzzi, M. Grosse-Wentrup, B. Schölkopf, et al. Quantifying causal influences. *The Annals of Statistics*, 41(5):2324–2358, 2013.

[133] C. W.-K. Leung, S. C.-F. Chan, F.-L. Chung, and G. Ngai. A probabilistic rating inference framework for mining user preferences from reviews. *World Wide Web*, 14(2):187–215, 2011.

[134] S. Holland, M. Ester, and W. Kießling. Preference mining: A novel approach on mining user preferences for personalized applications. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 204–216. Springer, 2003.

[135] A. S. Sayyad, T. Menzies, and H. Ammar. On the value of user preferences in search-based software engineering: a case study in software product lines. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 492–501. IEEE, 2013.

[136] C.-F. J. Wu. Jackknife, bootstrap and other resampling methods in regression analysis. *the Annals of Statistics*, pages 1261–1295, 1986.

[137] J. Li, T. D. Le, L. Liu, J. Liu, Z. Jin, B. Sun, and S. Ma. From observational studies to causal rule mining. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 7(2):14, 2016.

[138] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering*, pages 14–24. ACM, 2016.

[139] D. P. Kroese, J. C. Chan, et al. *Statistical modeling and computation*. Springer, 2014.

[140] T.-D. B. Le and D. Lo. Beyond support and confidence: Exploring interestingness measures for rule-based specification mining. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 331–340. IEEE, 2015.

[141] L. A. Zadeh. Fyzzy sets. *Inf. Comput.*, 8:338-353, Dec 1965.

[142] J. De Kleer and J. S. Brown. A qualitative physics based on confluences. *Artificial intelligence*, 24(1):7–83, 1984.

[143] M. P. Wellman and M. Derthick. *Formulation of tradeoffs in planning under uncertainty*. Pitman London, 1990.

[144] V. Vassilevska, R. Williams, and R. Yuster. All-pairs bottleneck paths for general graphs in truly sub-cubic time. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 585–589. ACM, 2007.

[145] D. G. Luenberger and Y. Ye. *Linear and nonlinear programming*, volume 228. Springer, 2015.

[146] R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[147] J. M. Karpoff. The relation between price changes and trading volume: A survey. *Journal of Financial and quantitative Analysis*, 22(1):109–126, 1987.

[148] W. Zhang, H. Mei, and H. Zhao. A feature-oriented approach to modeling requirements dependencies. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 273–282. IEEE, 2005.

[149] K. Loer and M. D. Harrison. An integrated framework for the analysis of dependable interactive systems (ifadis): Its tool support and evaluation. *Automated Software Engineering*, 13(4):469–496, 2006.

[150] A. Roy, D. S. Kim, and K. S. Trivedi. Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12. IEEE, 2012.

[151] D. Mougouei and M. K. Yeung. *Requirement Engineering for Intrusion Tolerant Systems*. LAP LAMBERT Academic Publishing, Aug. 2013.

[152] D. Mougouei, W. N. W. A. Rahman, and M. M. Almasi. Measuring security of web services in requirement engineering phase. *International Journal of Cyber-Security and Digital Forensics (IJCSDF)*, 1(2):89–98, 2012.

[153] G. Klir and B. Yuan. *Fuzzy sets and fuzzy logic*, volume 4. Prentice hall New Jersey, 1995.

[154] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel. Relax: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, 15(2):177–196, 2010.

[155] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *2009 17th IEEE International Requirements Engineering Conference*, pages 79–88. IEEE, 2009.

[156] D. Mougouei and W. N. W. A. Rahman. Fuzzy description of security requirements for intrusion tolerant web-services. In *The Second International Conference on Cyber Security, Cyber Peacefare and Digital Forensic (CyberSec2013)*, pages 141–147. The Society of Digital Information and Wireless Communication, 2013.

[157] D. Mougouei, W. N. W. A. Rahman, and M. Moein Almasi. Evaluating fault tolerance in security requirements of web services. In *Cyber Security, Cyber Warfare*

and Digital Forensic (CyberSec), 2012 International Conference on, pages 111–116. IEEE, 2012.

[158] A. Van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proceedings of the 26th International Conference on Software Engineering*, pages 148–157. IEEE Computer Society, 2004.

[159] D. Mougouei, W. Nurhayati, and M. Eshraghi Eavri. Fuzzy-based intrusion tolerance for web-services. pages 83–88, 2013.

[160] D. Mougouei, W. Nurhayati, and M. Eshraghi Eavri. Fuzzy-based intrusion tolerance for web-services. *International Journal of Advances in Computer Science and its Applications*, 4(1):83–88, 2014.

[161] D. Mougouei and W. Nurhayati. A fuzzy-based technique for describing security requirements of intrusion tolerant systems. *International Journal of Software Engineering and its Applications*, 7(2):99–112, 2013.

[162] L. A. Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

[163] B. Bede. Fuzzy inference. In *Mathematics of Fuzzy Sets and Fuzzy Logic*, pages 79–103. Springer, 2013.

[164] E. H. Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. *Electrical Engineers, Proceedings of the Institution of*, 121(12):1585–1588, 1974.

[165] R. W. Lewis. *Programming industrial control systems using IEC 1131-3*. Number 50. Iet, 1998.

[166] B. H. Cheng, P. Sawyer, N. Bencomo, and J. Whittle. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *International Conference on Model Driven Engineering Languages and Systems*, pages 468–483. Springer, 2009.

[167] A. Adams and M. A. Sasse. Users are not the enemy. *Communications of the ACM*, 42(12):40–46, 1999.

[168] E. Van Broekhoven and B. De Baets. Fast and accurate center of gravity defuzzification of fuzzy system outputs defined on trapezoidal fuzzy partitions. *Fuzzy Sets and Systems*, 157(7):904–918, 2006.

[169] A. J. Albrecht and J. E. Gaffney. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE transactions on software engineering*, (6):639–648, 1983.

[170] M. A. Ferrario, W. Simm, P. Newman, S. Forshaw, and J. Whittle. Software engineering for'social good': integrating action research, participatory design, and agile development. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 520–523. ACM, 2014.

[171] X. Shi, L. Wu, and X. Meng. A new optimization model for the sustainable development: Quadratic knapsack problem with conflict graphs. *Sustainability*, 9(2):236, 2017.

[172] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[173] B. Friedman. Value-sensitive design. *interactions*, 3(6):16–23, 1996.

[174] J. Cabot, S. Easterbrook, J. Horkoff, L. Lessard, S. Liaskos, and J.-N. Mazón. Integrating sustainability in decision-making processes: A modelling strategy. In *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 207–210. IEEE, 2009.

[175] V. Caldiera and H. D. Rombach. The goal question metric approach. *Encyclopedia of software engineering*, 2(1994):528–532, 1994.

[176] M. A. Ferrario, W. Simm, P. Newman, S. Forshaw, and J. Whittle. Software engineering for 'social good': Integrating action research, participatory design, and agile development. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 520–523, New York, NY, USA, 2014. ACM.