

Dependency-Aware Software Requirements Selection using Fuzzy Graphs and Integer Programming

Davoud Mougouei *

*School of Computing and IT, University of Wollongong, NSW, Australia
dmougouei@gmail.com*

David M. W. Powers

*College of Science and Engineering, Flinders University, SA, Australia
David.Powers@flinders.edu.au*

Abstract

Software requirements selection aims to find an optimal subset of the requirements with the highest *value* while respecting the project constraints. But the value of a requirement may depend on the presence or absence of other requirements in the optimal subset. Such *Value Dependencies*, however, are imprecise and hard to capture. In this paper, we propose a method based on integer programming and fuzzy graphs to account for value dependencies and their imprecision in software requirements selection. The proposed method, referred to as *Dependency-Aware Software Requirements Selection* (DARS), is comprised of three components: (i) an automated technique for the identification of value dependencies from user preferences, (ii) a modeling technique based on fuzzy graphs that allows for capturing the imprecision of value dependencies, and (iii) an *Integer Linear Programming* (ILP) model that takes into account user preferences and value dependencies identified from those preferences to reduce the risk of value loss in software projects. Our work is verified by studying a real-world software project. The results show that our proposed method reduces the value loss in software projects and is scalable to large requirement sets.

Keywords: Fuzzy; Integer Programming; Value; Dependencies; Software

1. Introduction

Software requirement selection, also known as *Release Planning*, is to find a subset of requirements that delivers the highest value for a release of software while respecting the project constraints, e.g. budget (Zhang et al., 2018a,b; de Melo França et al., 2018; Bagnall et al., 2001; Franch & Ruhe, 2016). Selecting (ignoring) a requirement, however, may influence the values of other requirements (Aydemir et al., 2018; Mougouei, 2016; Mougouei et al., 2017a; Mougouei & Powers, 2020; Zhang et al., 2013; Robinson et al., 2003); it is important to

consider value dependencies in requirement selection (Carlshamre et al., 2001; Li et al., 2010; Zhang et al., 2014; Karlsson et al., 1997; Mougouei & Powers, 2019). This is further emphasized by the fact that value dependencies widely exist in software projects Carlshamre et al. (2001); Carlshamre (2002); Pitangueira et al. (2015). Moreover, as observed by Carlshamre et al. (Carlshamre et al., 2001), requirement dependencies and value dependencies in particular are *fuzzy* (Carlshamre et al., 2001) as the strengths of those dependencies are imprecise and vary (Dahlstedt & Persson, 2005; Ngo-The & Ruhe, 2008; Ngo-The & Saliu, 2005; Carlshamre et al., 2001) from large to insignificant (Wang et al., 2012).

Although the need for considering value dependencies and their strengths was observed as early as in 2001 (Carlshamre et al., 2001), existing requirements selection methods have mainly ignored value dependencies by simply optimizing either the *Accumulated Value* (AV) (Baker et al., 2006; Li et al., 2010; Boschetti et al., 2014; Araújo et al., 2016; Greer & Ruhe, 2004a) or the *Expected Value* (EV) of the selected requirements (Pitangueira et al., 2017; Li et al., 2017, 2014). Some methods have attempted considering value dependencies by manually estimating the values of requirement subsets, that may require up to 2^n comparisons for n requirements (van den Akker et al., 2005b). When estimations are limited to pairs, still $O(n^2)$ estimations are needed (Li et al., 2010; Sagrado et al., 2013; Zhang et al., 2013). Such complexity severely impacts the practicality of these methods, not to mention the issues around the accuracy of manual estimations.

Finally, requirements selection methods based on manual estimations of values of requirement subsets do not capture the direction of an influence. In other words, these methods do not distinguish among three scenarios: (i) requirement r_i influences the value of requirement r_j and not the other way, (ii) r_j influences the value of r_i and not the other way, and (iii) both r_i and r_j influence the values of each other but to different extents. To effectively consider value dependencies in software requirements selection, we have proposed a method based on fuzzy graphs and integer programming (Gkioulekas & Papageorgiou, 2019; Tavana et al., 2015) with three major components:

(i) *Identification of value dependencies*. We have contributed an automated technique that uses Eells measure of casual strength (Eells, 1991) to extract value dependencies from significant causal relations among user preferences. *Odds Ratio* (Li et al., 2016) is used to specify the significance of such relations. We have further, demonstrated the use of a Latent Multivariate Gaussian model (Macke et al., 2009) to generate samples of user preferences when needed (Macke et al., 2009).

(ii) *Modeling value dependencies*. We have demonstrated the use of fuzzy graphs (Zahedi et al., 2016; Rosenfeld, 1975) and their algebraic structure (Kalampakas et al., 2013) for modeling strengths and qualities of value dependencies and capturing the imprecision of those dependencies. On this basis, value dependencies and their strengths are modeled by fuzzy relations (Carlshamre et al., 2001; Ngo The & Saliu, 2005; Ngo-The & Saliu, 2005; Liu & Yen, 1996) and their fuzzy membership functions respectively.

(iii) *Considering value dependencies in requirements selection.* At the heart of DARS is an integer linear programming (ILP) model, which maximizes the *Overall Value* (OV) of a selected subset of the requirements, where user preferences and the value dependencies identified from those preferences are considered. The model further, considers structural and semantic dependencies (e.g. Requires and Conflicts-With) among software requirements by formulating them as the precedence constraints of the optimization model.

We have demonstrated practicality and scalability of DARS by studying a real-world software. Our results show that (a) compared to existing requirements selection methods, that ignore value dependencies, DARS provides higher overall value by mitigating the risk of value loss caused by ignoring (selecting) requirements with positive (negative) influences on the values of the selected requirements, (b) maximizing the accumulated value or the estimated value of a software conflicts with maximizing its overall value, and (c) DARS is scalable to datasets with large number of requirements 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.

2. Background and Related Work

It is widely recognized that software requirements influence the values of each other (Zhang et al., 2013; Brasil et al., 2012; Robinson et al., 2003; Dahlstedt & Persson, 2005). Such influences are described in the literature as value dependencies (Carlshamre et al., 2001; Li et al., 2010; Zhang et al., 2014; Karlsson et al., 1997). Value dependencies are fuzzy relations (Carlshamre et al., 2001) with varying strengths (e.g. weak, moderate, strong) and qualities (positive or negative) which are imprecise, and hard to specify (Carlshamre et al., 2001; Mougouei, 2016). Requirement selection methods hence, should consider qualities and strengths of explicit and implicit value dependencies among software requirements.

Moreover, *Precedence Dependencies* such as *Requires* (Dahlstedt & Persson, 2005), *Conflicts-With* (K, 1996), *AND*, and *OR* 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 value implications of precedence dependencies in software requirements selection. On this basis, we have characterized properties (P1)-(P7) for requirements selection methods in relation to how they treat value dependencies.

Table 1 categorizes existing requirements selection methods into four groups based on (P1)-(P7).

- (P1) Considering explicit value dependencies.
- (P2) Considering implicit value dependencies.
- (P3) Considering qualities (positive or negative) of value dependencies.
- (P4) Considering strengths of value dependencies.
- (P5) Considering directions of value dependencies.

- (P6) Considering precedence dependencies and their value implications.
- (P7) Relying on manual estimations of values of requirement subsets.

Table 1: Considering aspects of value dependencies in existing works.

Technique	Employed by	P1	P2	P3	P4	P5	P6	P7
BK	Harman et al. (1997); Jung (1998); Zhang et al. (2007); Baker et al. (2006); Faldutova et al. (2009); Zhang et al. (2011); del Sotillo et al. (2010); Krumm et al. (2012); Horowitz et al. (2013); Bouda et al. (2013); Sagrado et al. (2013); Sagami et al. (2013); Geor & Rabe (2014); Bouda et al. (2014); Rabe & Geor (2014); van Veldhuizen et al. (2011); Zhang & Heurum (2010); Bouda et al. (2010); Fritsch et al. (2011); Calabuig et al. (2008); Galati & Rabe (2007); 2010); Jiang et al. (2010); van der Aalst et al. (2005); Sun, Tian & Rabe (2009); Chen & Zhang (2010); del Sotillo et al. (2011); Krumm et al. (2010); Calabuig et al. (2008); Pflanzger et al. (2009)	NO	NO	NO	NO	NO	YES	NO
PCBK	van der Aalst et al. (2008); Sun, Tian & Rabe (2009); Toulha et al. (2013); Xuan et al. (2012); Sotillo & Rabe (2005)	YES	NO	YES	NO	NO	YES	YES
Increase/Decrease	Li et al. (2010); Sagrado et al. (2013); Zhang et al. (2011) (subset of size 2) van der Aalst et al. (2008) (subset of size 100)	YES	YES	YES	NO	NO	YES	YES
MR	Pflanzger et al. (2017); Li et al. (2017, 2014)	NO	NO	NO	NO	NO	YES	YES

2.1. Binary Knapsack Methods

The first group of selection methods (Table 1), i.e. *Binary Knapsack* (BK) methods, are solely based on the classical formulation of binary knapsack problem (Harman et al., 2014; Carlshamre et al., 2001) as given by (1)-(3). Let $R = \{r_1, \dots, r_n\}$ be a set of identified requirements, where $\forall r_i \in R$ ($1 \leq i \leq n$), v_i and c_i in (1)-(3) denote the value and the cost of r_i respectively. Also, b in (2) denotes the available budget. A decision variable x_i specifies whether requirement r_i is selected ($x_i = 1$) or not ($x_i = 0$). The objective of *BK* methods as given by (1) is to find a subset of R that maximizes the accumulated value of the selected requirements ($\sum_{i=1}^n v_i x_i$) while entirely ignoring value dependencies as well as precedence dependencies among the requirements (Karlsson & Ryan, 1997; Jung, 1998; Zhang et al., 2007).

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

$$\text{Subject to } \sum_{i=1}^n c_i x_i \leq b \tag{2}$$

$$x_i \in \{0, 1\}, \quad i = 1, \dots, n \tag{3}$$

2.2. Precedence-Constrained Binary Knapsack Methods

The *Precedence-Constrained Binary Knapsack* (PCBK) methods, enhance the BK methods by adding (4) to the optimization model of BK methods to consider precedence dependencies (*requires* (Dahlstedt & Persson, 2005) and *conflicts-with* (K, 1996)) and their value implications. 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 (4). Also, decision variable x_i denotes whether a requirement r_i is selected ($x_i = 1$) or not.

$$\begin{cases} x_j \leq x_k & \text{if } r_j \text{ positively depends on } r_k \\ x_j \leq 1 - x_k & \text{if } r_j \text{ negatively depends on } r_k, \quad j \neq k = 1, \dots, n \end{cases} \tag{4}$$

2.3. Increase-Decrease Methods

The third group of requirement selection methods i.e. *Increase-Decrease* methods consider value dependencies among requirements through estimating the amount of the increased (decreased) values resulted by selecting different subsets of requirements. The optimization model for an increases-Decreases technique proposed by (van den Akker et al., 2005b) is given in (5)-(8). For a subset $s_j \in S : \{s_1, \dots, s_m\}, m \leq 2^n$, 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 (5) specifies whether a subset s_j is realized ($y_j = 1$) or not ($y_j = 0$). Also, constraint (7) ensures that $y_j = 1$ only if $\forall r_k \in s_j, x_k = 1$.

$$\text{Maximize } \sum_{i=1}^n v_i x_i + \sum_{j=1}^m (w_j - \sum_{r_k \in s_j} v_k) y_j \quad (5)$$

$$\text{Subject to } n_j y_j \leq \sum_{r_k \in s_j} x_k \quad (6)$$

$$\sum_{i=1}^n c_i x_i \leq b \quad (7)$$

$$x_i, y_j \in \{0, 1\}, \quad i = 1, \dots, n, \quad j = 1, \dots, m \quad (8)$$

Increase-Decrease methods are complex and prone to human error as they rely on manual estimations for requirement subsets (Mougouei, 2016). For n requirements, these estimations are at least as complex as $O(n^2)$, when only values of pairs are estimated (Li et al., 2010; Sagrado et al., 2013; Zhang et al., 2013), and can get as complex as $O(2^n)$ in worst case.

$$\text{Maximize } \sum_{i=1}^n v_i x_i + \sum_{i=1}^n \sum_{j=1}^n x_i x_j w_{i,j} \quad (9)$$

$$\text{Subject to } \sum_{i=1}^n c_i x_i \leq b \quad (10)$$

$$y_{ij} \leq x_i \quad (11)$$

$$y_{ij} \leq x_j \quad (12)$$

$$y_{i,j} \geq x_i + x_j - 1 \quad (13)$$

$$x_i, y_{i,j} \in \{0, 1\} \quad i, j = 1, \dots, n \quad (14)$$

Moreover, relying on pairwise estimations results in ignoring implicit value dependencies as the direction of dependencies are not specified. For instance, consider requirements $R : \{r_1, r_2, r_3\}$ with 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 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). If no explicit value dependency is found between r_1 and r_3 hence the influence of r_3 on the value of r_1 will be ignored.

2.4. Stochastic Binary Knapsack Methods

Stochastic Binary Knapsack (SBK) requirements selection methods maximize the expected value of a requirement subset based on the formulation of stochastic knapsack problem (Henig, 1990) as given by (15). In this equation, $E(v_i)$ denotes the expected value of a requirement r_i . The work (Pitangueira et al., 2017) for instance, optimizes the expected value of a software at different risk levels, where risk is formulated in terms of summation of covariances of values of requirements as given by (16), where $\sigma_{i,j}$ specifies the covariance of v_i and v_j and l specifies the risk level.

$$\text{Maximize } \sum_{i=1}^n x_i E(v_i) \quad (15)$$

$$\text{Subject to } \sum_{i=1}^n \sum_{j=1}^n x_i x_j \sigma_{i,j} \leq l \quad (16)$$

$$\sum_{i=1}^n x_i c_i \leq b \quad (17)$$

$$x_i \in \{0, 1\} \quad i = 1, \dots, n \quad (18)$$

One may suggest that the covariance of values of requirements (risk) in (16) may somehow capture value dependencies. But, there are four major problems with this. First, covariance is a measure of correlation and does not capture causality. In other words, by using covariance one is assuming that all value dependencies are bidirectional and the strengths of dependencies in either direction are equal. Such an assumption, however, may not be realistic. Second, optimization models based on covariance can only capture linear relations ignoring non-linear relations even if they are significant.

Also, limiting or minimizing the risk may contradict with choosing requirements that positively influence each other as SBK methods avoid choosing positively correlated (with respect to value) requirements. On the contrary, SBK methods tend to choose negatively correlated requirements or independent ones as such combinations satisfy (16). In other words, risk, which is defined based on covariances, and value dependencies, which are, by nature, causal relations, are different and cannot be used interchangeably.

It is also worth mentioning that Value (of the requirement/software) is a broad concept and has several manifestations as discussed in Mougouei et al. (2018); Perera et al. (2019a); Hussain et al. (2018); Perera et al. (2019b). In this paper, however, we only focus on the monetary interpretation of Value.

	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	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 1: A sample preference matrix $M_{4 \times 20}$.

3. 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 (Eells, 1991) of causal strength and the *Odds Ratio* (Li et al., 2016) 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.

3.1. Gathering User Preferences

User preferences can be gathered in different ways (Leung et al., 2011; Holland et al., 2003; Sayyad et al., 2013) 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 mining user reviews in social media and online stores (Villarroel et al., 2016). User preferences may also be gathered by studying similar software and 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 the software. This is particularly suitable for reengineering software or releasing different configurations of a software product line. We capture user preferences by a *Preference Matrix* as given by Definition 1.

Proposition 1. *Preference Matrix.* Let $R = \{r_1, \dots, r_n\}$ be a requirement set and $U = \{u_1, \dots, u_k\}$ be the list of users whose preference are gathered. A preference matrix M 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 1.

3.2. Resampling

Resampling 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 Eells measure. This is particularly useful when conducting comprehensive market research is not practical.

We use a resampling technique introduced by (Kroese et al., 2014) to generate larger samples of collected user preferences using a Latent Multivariate Gaussian model. The process as given in Figure 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 covariance 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 (Kroese et al., 2014), Step 6.

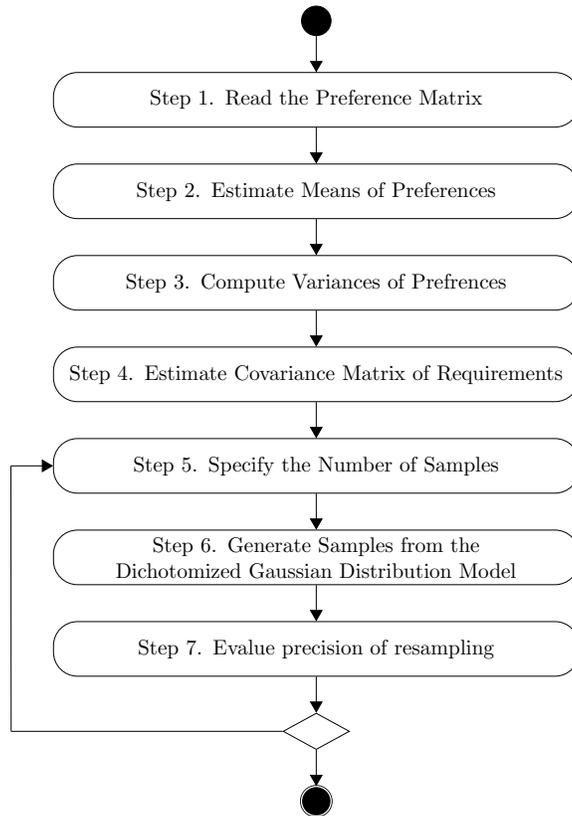


Figure 2: Steps for generating samples from user preferences.

The precision of the employed resampling technique (Figure 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). The entropy of the Latent Multivariate Gaussian model is near theoretical maximum for a wide range of parameters (Macke et al., 2009).

3.3. Extracting Causal Relations among User Preferences

User preferences for a requirement may positively or negatively influence the preferences of the users for other requirements. Such causal relations can be identified using measures of causal strength (Halpern & Hitchcock, 2015; Pearl, 2009; Janzing et al., 2013). 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 Eells measure (Eells, 1991), to estimate the strengths and qualities of explicit value dependencies among software requirements as given by (19). 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 .

$$\eta_{i,j} = p(r_i|r_j) - p(r_i|\bar{r}_j), \quad \eta_{i,j} \in [-1, 1] \quad (19)$$

For a pair of requirements (r_i, r_j) , 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 r_i may increase or decrease the value of r_j .

Matrix $P_{4 \times 4}$ (Figure 3(a)) and Matrix $\bar{P}_{4 \times 4}$ (Figure 3(b)) show the strengths of positive and negative causal relations among user preferences for requirements in the preference matrix $M_{4 \times 8}$ (Figure 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 $P_{4 \times 4}$ ($\bar{P}_{4 \times 4}$) denotes the strength of a positive (negative) causal relation from r_i to r_j .

For diagonal elements of $P_{4 \times 4}$ ($\bar{P}_{4 \times 4}$) 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 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 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 $\lambda_{n \times 2n}$ counts the number of times that a pair of requirements (r_i, r_j) are selected

$$\begin{array}{ccc}
\begin{array}{c} r_1 \quad r_2 \quad r_3 \quad r_4 \\
\begin{pmatrix} 1.0000 & 0.8333 & 0.8750 & 0.7692 \\
0.5882 & 1.0000 & 0.6250 & 0.6154 \\
0.4118 & 0.4167 & 1.0000 & 0.3846 \\
0.5882 & 0.6667 & 0.6250 & 1.0000 \end{pmatrix} \\
\text{(a) } P_{4 \times 4}
\end{array} & - & \begin{array}{c} r_1 \quad r_2 \quad r_3 \quad r_4 \\
\begin{pmatrix} 0.0000 & 0.8750 & 0.8333 & 1.0000 \\
0.6667 & 0.0000 & 0.5833 & 0.5714 \\
0.3333 & 0.3750 & 0.0000 & 0.4286 \\
1.0000 & 0.6250 & 0.6667 & 0.0000 \end{pmatrix} \\
\text{(b) } \bar{P}_{4 \times 4}
\end{array} \\
= & & \begin{array}{c} r_1 \quad r_2 \quad r_3 \quad r_4 \\
\begin{pmatrix} +1.0000 & -0.0417 & +0.0417 & -0.2308 \\
-0.0785 & +1.0000 & +0.0417 & +0.0440 \\
+0.0785 & +0.0417 & +1.0000 & -0.0440 \\
-0.4118 & +0.0417 & -0.0417 & +1.0000 \end{pmatrix} \\
\text{(c) } \boldsymbol{\eta}_{4 \times 4}
\end{array}
\end{array}$$

Figure 3: Computing Eells measure for the preference matrix of Figure 1.

Algorithm 1: Computing 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:     $p_{i,j} \leftarrow \left( \frac{\lambda_{i,j}}{\lambda_{j,j}} \right)$ 
17:     $\bar{p}_{i,j} \leftarrow \left( \frac{\lambda_{i,j+n}}{\lambda_{j+n,j+n}} \right)$ 
18:     $\eta_{i,j} \leftarrow (p_{i,j} - \bar{p}_{i,j})$ 
19:  end for
20: end for

```

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 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 $O(t \times n^2)$.

3.4. Testing the Significance of Causal Relations

Using measures of interestingness (Geng & Hamilton, 2006) is sometimes not sufficient to understand the significance of the relations found among the items of a dataset as explained in (Li et al., 2016). 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 , (20) 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.

$$\omega(r_i, r_j) = \frac{p(r_i, r_j)p(\bar{r}_i, \bar{r}_j)}{p(r_i, \bar{r}_j)p(\bar{r}_i, r_j)}, \quad \omega(r_i, r_j) \in (0, \infty) \quad (20)$$

To test the significance of a causal relation from a requirement r_j to r_i , we use the technique used in (Li et al., 2016) by computing the lower bound (ω_-) and the upper bound (ω_+) of the confidence interval of the Odds Ratio as given by (21)-(22). In these equations z' is the critical value corresponding to a desired level of confidence. Also, u denotes the total number of user preferences. 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.

$$\omega_-(r_i, r_j) = \ln(\omega(r_i, r_j)) - \frac{z'}{\sqrt{u}} \sqrt{\frac{1}{p(r_i, r_j)} + \frac{1}{p(\bar{r}_i, \bar{r}_j)} + \frac{1}{p(\bar{r}_i, r_j)} + \frac{1}{p(r_i, \bar{r}_j)}} \quad (21)$$

$$\omega_+(r_i, r_j) = \ln(\omega(r_i, r_j)) + \frac{z'}{\sqrt{u}} \sqrt{\frac{1}{p(r_i, r_j)} + \frac{1}{p(\bar{r}_i, \bar{r}_j)} + \frac{1}{p(\bar{r}_i, r_j)} + \frac{1}{p(r_i, \bar{r}_j)}} \quad (22)$$

3.5. Computing the Strengths and Qualities of value Dependencies

The strength of an explicit value dependency from a requirement r_i to r_j is computed by (23), which gives a mapping from 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. Significant causal relations which pass the test in Section 3.4 will be considered.

The fuzzy membership functions, however, may be adjusted to account for the imprecision of value dependencies and suit a particular needs of decision

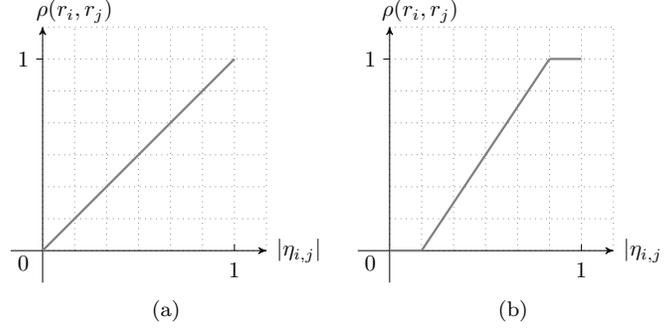


Figure 4: Sample membership functions for strengths of value dependencies.

makers. For instance, the membership function of Figure 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.

$$\rho(r_i, r_j) = |\eta_{i,j}| \quad (23)$$

$$\sigma(r_i, r_j) = \begin{cases} + & \text{if } \eta_{i,j} > 0 \\ - & \text{if } \eta_{i,j} < 0 \\ \pm & \text{if } \eta_{i,j} = 0 \end{cases} \quad (24)$$

As given by (24), $\eta_{i,j} > 0$ 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 ($\sigma(r_i, r_j) = \pm$).

3.6. 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 requirements 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 requirements selection.

This can be achieved by modeling the precedence dependencies using a *Precedence Dependency Graph* (PDG) as introduced in Definition 2.

Proposition 2. *The Precedence Dependency Graph* (PDG). A PDG is a signed directed graph $G = (R, W)$ in which $R = \{r_1, \dots, r_n\}$ 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 .

$$PDL(G) = \frac{k}{{}^n P_2} = \frac{k}{n(n-1)} \quad (25)$$

$$NPDL(G) = \frac{j}{k} \quad (26)$$

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 requirements selection. It is clear that increasing the 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 (25) and (26) 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 negative precedence dependencies (j) by the total number of the positive and negative precedence dependencies.

4. Modeling Value Dependencies by Fuzzy Graphs

Fuzzy logic and Fuzzy graphs Mathew & Sunitha (2013) have been widely adopted in decision making and expert systems (Rosenfeld, 1975) as they contribute to more accurate models by taking into account the imprecision of real-world problems (Mathew & Sunitha, 2013; Mougouei et al., 2012a,b,c; Mougouei, 2013; Mougouei & Nurhayati, 2013). Fuzzy logic has been adopted in requirement selection for capturing the partiality of requirements Mougouei et al. (2019, 2015). Also, Fuzzy graphs have, particularly, demonstrated useful in capturing the imprecision of dependency relations in software (Ngo The & Saliu, 2005; Ngo-The & Saliu, 2005; Mougouei & Powers, 2017; Mougouei, 2018). Ngo-The *et al.*, exploited fuzzy graphs for modeling dependency satisfaction in release planning (Ngo The & Saliu, 2005) and capturing the imprecision of coupling dependencies among requirements (Ngo-The & Saliu, 2005).

Moreover, Wang *et al.* (Wang et al., 2012) 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.

4.1. 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. 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 3.

Proposition 3. *The Value Dependency Graph (VDG) is a signed directed fuzzy graph (Wasserman & Faust, 1994) $G = (R, \sigma, \rho)$ where, requirements $R : \{r_1, \dots, r_n\}$ 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.*

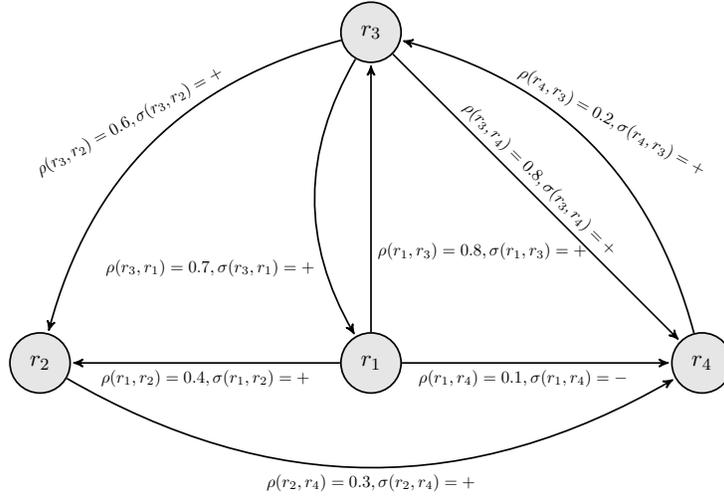


Figure 5: A sample value dependency graph.

For instance, in the value dependency graph of Figure 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 .

4.2. Value Dependencies in VDGs

In Section 3 we introduced an automated technique for identification of explicit value dependencies and their characteristics (quality and strength) from user preferences. Definition 4 provides a more comprehensive definition of value dependencies that includes both explicit and implicit value dependencies among the requirements of software based on the algebraic structure of fuzzy graphs.

Proposition 4. *Value Dependencies.* A value dependency in a value dependency graph $G = (R, \sigma, \rho)$ is defined as a sequence of requirements $d_i : (r(0), \dots, r(k))$ such that $\forall r(j) \in d_i, 1 \leq j \leq k$ we have $\rho(r(j-1), r(j)) \neq 0$. $j \geq 0$ is the sequence of the j^{th} requirement (node) denoted as $r(j)$ on the dependency path. A consecutive pair $(r(j-1), r(j))$ specifies an explicit value dependency.

$$\forall d_i : (r(0), \dots, r(k)) : \rho(d_i) = \bigwedge_{j=1}^k \rho(r(j-1), r(j)) \quad (27)$$

$$\forall d_i : (r(0), \dots, r(k)) : \sigma(d_i) = \prod_{j=1}^k \sigma(r(j-1), r(j)) \quad (28)$$

Equation (27) computes the strength of a value dependency $d_i : (r(0), \dots, r(k))$ by finding the strength of the weakest of the k explicit dependencies on d_i . Fuzzy operator \wedge denotes Zadeh's (Zadeh, 1965) AND operation (infimum). The quality (positive or negative) of a value dependency $d_i : (r(0), \dots, r(k))$ is calculated by qualitative serial inference (De Kleer & Brown, 1984; Wellman & Derthick, 1990; Kusiak & Wang, 1995) as given by (28) and Table 2. Inferences in Table 2 are informally proved by Wellman (Wellman & Derthick, 1990) and Kleer (De Kleer & Brown, 1984).

Table 2: Qualitative serial inference in VDGs.

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

Let $D = \{d_1, d_2, \dots, d_m\}$ 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 , denoted by $\rho^{+\infty}(r_i, r_j)$, is calculated by (29), that is to find the strength of the strongest positive dependency (Rosenfeld, 1975) from r_i to r_j . Fuzzy operators \wedge and \vee denote Zadeh's (Zadeh, 1965) fuzzy AND and OR operations respectively. Analogously, 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 (30).

$$\rho^{+\infty}(r_i, r_j) = \bigvee_{d_m \in D, \sigma(d_m)=+} \rho(d_m) \quad (29)$$

$$\rho^{-\infty}(r_i, r_j) = \bigvee_{d_m \in D, \sigma(d_m)=-} \rho(d_m) \quad (30)$$

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 devised a modified version of Floyd-Warshall (Floyd, 1962) algorithm (Algorithm 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) , $r_i, r_j \in R : \{r_1, \dots, r_n\}$ in polynomial time: $O(n^3)$. For each pair of requirements (r_i, r_j) in a VDG $G = (R, \sigma, \rho)$, lines 18 to 35 of Algorithm 2 find the strength of all positive (negative) value dependencies from r_i to r_j .

$$I_{i,j} = \rho^{+\infty}(r_i, r_j) - \rho^{-\infty}(r_i, r_j) \quad (31)$$

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 (31) 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 positively influences the value of r_i whereas $I_{i,j} < 0$ indicates a negative influence from r_j on r_i .

Example 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 5. Using (28), 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 (27) as: $\rho(d_1) = \wedge(\rho(r_1, r_2), \rho(r_2, r_4)) = \min(0.4, 0.3)$, $\rho(d_2) = \wedge(\rho(r_1, r_3), \rho(r_3, r_4)) = \min(0.8, 0.8)$, $\rho(d_3) = \min(0.1)$. Using (29) and (30) then we have $\rho(r_1, r_4)^{+\infty} = \vee(\rho(d_1), \rho(d_2)) = \max(0.3, 0.8)$ and $\rho(r_1, r_4)^{-\infty} = \max(\rho(d_3))$. 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 3 lists influences of requirements in the VDG of Figure 5 on the value of each other.

Proposition 5. *Value Dependency Level (VDL) and Negative Value Dependency Level (NVDL).* Let $G = (R, \sigma, \rho)$ be a VDG with $R = \{r_1, \dots, r_n\}$, 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 (32) and (33) respectively.

$$VDL(G) = \frac{k}{{}^n P_2} = \frac{k}{n(n-1)} \quad (32)$$

$$NVDL(G) = \frac{m}{k} \quad (33)$$

Algorithm 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**

Table 3: Overall influences computed for VDG of Figure 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	$0.2 - 0.0 = 0.2$	$0.2 - 0.0 = 0.2$	$0.2 - 0.0 = 0.2$	$0.0 - 0.0 = 0.0$

Example 2. For the value dependency graph G of Figure 5 we have $n = 4$, $k = 8$, and $m = 1$. $VDL(G)$ is derived by (32) as: $VDL(G) = \frac{8}{4 \times 3} = \frac{8}{12} \cong 0.67$. Also we have from Equation (33), $NVDL(G) = \frac{1}{8} = 0.125$.

5. Integrating Value Dependencies into Requirements Selection

5.1. 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 3 are identified based on causal relations among user preferences. Section 3 presented an automated technique for identification of value dependencies among requirements. Then, algorithm 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, (34)-(35) 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 (31), gives the positive or negative influence of r_j on the value of r_i .

$$\theta_i = \bigvee_{j=1}^n \left(\frac{x_j(|I_{i,j}| - I_{i,j}) + (1 - x_j)(|I_{i,j}| + I_{i,j})}{2} \right) =$$

$$\bigvee_{j=1}^n \left(\frac{|I_{i,j}| + (1 - 2x_j)I_{i,j}}{2} \right), \quad i \neq j = 1, \dots, n \quad (34)$$

$$x_j \in \{0, 1\}, \quad j = 1, \dots, n \quad (35)$$

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.

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 (37), 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 (38) 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 (36), 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 .

$$E(v_i) = p(r_i) \times v_i + p(\bar{r}_i) \times 0 = p(r_i) \times v_i \quad (36)$$

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 (38) 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.

$$v'_i = (1 - \theta_i)E(v_i) \quad (37)$$

$$OV = \sum_{i=1}^n x_i(1 - \theta_i)E(v_i), \quad x_i \in \{0, 1\} \quad (38)$$

Example 3. Consider finding penalties for requirements of Figure 5, where r_4 is not selected ($x_1 = x_2 = x_3 = 1, x_4 = 0$). From Table 3 we have $I_{1,4} = I_{3,4} = 0.7, I_{2,4} = 0.3, I_{4,4} = 0.0$. As such, based on (34) penalties are computed: $\theta_1 = \sqrt{\frac{|0.0|+(1-2(1))(0.0)}{2}, \frac{|0.45|+(1-2(1))(0.5)}{2}, \frac{|0.7|+(1-2(1))(0.7)}{2}, \frac{|0.7|+(1-2(0))(0.7)}{2}} = 0.7$. 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 (38) as: $OV(s_1) = (1 - 0.7)E(v_1) + (1 - 0.3)E(v_2) + (1 - 0.7)E(v_3)$.

5.2. 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 (38), 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 requirements selection by optimizing the overall value of a software product.

Equations (39)-(44) give our proposed integer programming model as a main component of DARS. 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 (34) 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 (41) on the other hand accounts for precedence dependencies among requirements and the value implications of those dependencies.

$$\text{Maximize } \sum_{i=1}^n x_i(1 - \theta_i)E(v_i) \quad (39)$$

$$\text{Subject to } \sum_{i=1}^n c_i x_i \leq b \quad (40)$$

$$\begin{cases} x_i \leq x_j & r_j \text{ precedes } r_i \\ x_i \leq 1 - x_j & r_i \text{ conflicts with } r_j, \quad i \neq j = 1, \dots, n \end{cases} \quad (41)$$

$$\theta_i \geq \left(\frac{|I_{i,j}| + (1 - 2x_j)I_{i,j}}{2} \right), \quad i \neq j = 1, \dots, n \quad (42)$$

$$x_i \in \{0, 1\}, \quad i = 1, \dots, n \quad (43)$$

$$0 \leq \theta_i \leq 1, \quad i = 1, \dots, n \quad (44)$$

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 (34).

Since $I_{i,j}$ is computed by (31) we can restate θ_i as a function of x_j : $\theta_i = f(x_j)$. The objective function (39), 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 (Boyd & Vandenberghe, 2004). Equations (39)-(42), on the other hand, denote a convex optimization problem as the model maximizes a concave objective function with linear constraints.

$$\text{Maximize } \sum_{i=1}^n x_i E(v_i) - y_i E(v_i) \quad (45)$$

$$\text{Subject to } \sum_{i=1}^n c_i x_i \leq b \quad (46)$$

$$\begin{cases} x_i \leq x_j & r_j \text{ precedes } r_i \\ x_i \leq 1 - x_j & r_i \text{ conflicts with } r_j, \quad i \neq j = 1, \dots, n \end{cases} \quad (47)$$

$$\theta_i \geq \left(\frac{|I_{i,j}| + (1 - 2x_j)I_{i,j}}{2} \right), \quad i \neq j = 1, \dots, n \quad (48)$$

$$-g_i \leq x_i \leq g_i, \quad i = 1, \dots, n \quad (49)$$

$$1 - (1 - g_i) \leq x_i \leq 1 + (1 - g_i), \quad i = 1, \dots, n \quad (50)$$

$$-g_i \leq y_i \leq g_i, \quad i = 1, \dots, n \quad (51)$$

$$-(1 - g_i) \leq (y_i - \theta_i) \leq (1 - g_i), \quad i = 1, \dots, n \quad (52)$$

$$0 \leq y_i \leq 1, \quad i = 1, \dots, n \quad (53)$$

$$0 \leq \theta_i \leq 1, \quad i = 1, \dots, n \quad (54)$$

$$x_i, g_i \in \{0, 1\}, \quad i = 1, \dots, n \quad (55)$$

Convex optimization problems are solvable (Boyd & Vandenberghe, 2004). However, for problems of moderate to large sizes, integer linear programming (ILP) models are preferred (Luenberger & Ye, 2015; Mougouei et al., 2017b) 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 (45).

In doing so, non-linear expression $x_i \theta_i$ is substituted by 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 (49)-(53) are added to the original model. As such, we have either $(g_i = 0) \rightarrow a$, or $(g_i = 1) \rightarrow b$. Therefore, (45)-(55) is linear and can be efficiently solved (Boyd & Vandenberghe, 2004), even for large scale requirement sets, by existing commercial solvers such as *IBM CPLEX*.

6. Case Study

This section discusses the practicality and validity of DARS by studying a real-world software project. We demonstrate why software vendors should take care with value dependencies among requirements, and how to employ DARS

to assist decision makers to comprehend the results, thus raising the following research questions.

- (RQ1) How effective is DARS in considering value dependencies?
- (RQ1.1) How similar are the solutions found by DARS to those found by other selection methods?
- (RQ1.2) What is the impact of using DARS on the overall value of software?
- (RQ1.3) What is the relationship between maximizing the accumulated value, expected value, and overall value of a software product?
- (RQ1.4) How effective is DARS in mitigating value loss?

6.1. Description of Study

To demonstrate the practicality of DARS, we studied a real-world software project. Table 4 lists the requirements of the project 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 of the project sold in the earlier versions of software ($p(r_i)$) by its estimated value v_i .

Table 4: The estimated and expected values of the requirements.

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

Our study began with the identification of value dependencies and modeling those dependencies as depicted in Figure 6. Then we performed requirements selection using PCBK, SBK, and DARS based on the sales records of the previously released configurations of software. For the configurations found by PCBK, SBK, and DARS (Figure 10 and Figure 11), the accumulated value (AV), expected value (EV), and overall value (OV) were computed to compare the performance of those methods for different price levels. This helped stakeholders to find, for different price levels, configurations with lower risk of value loss.

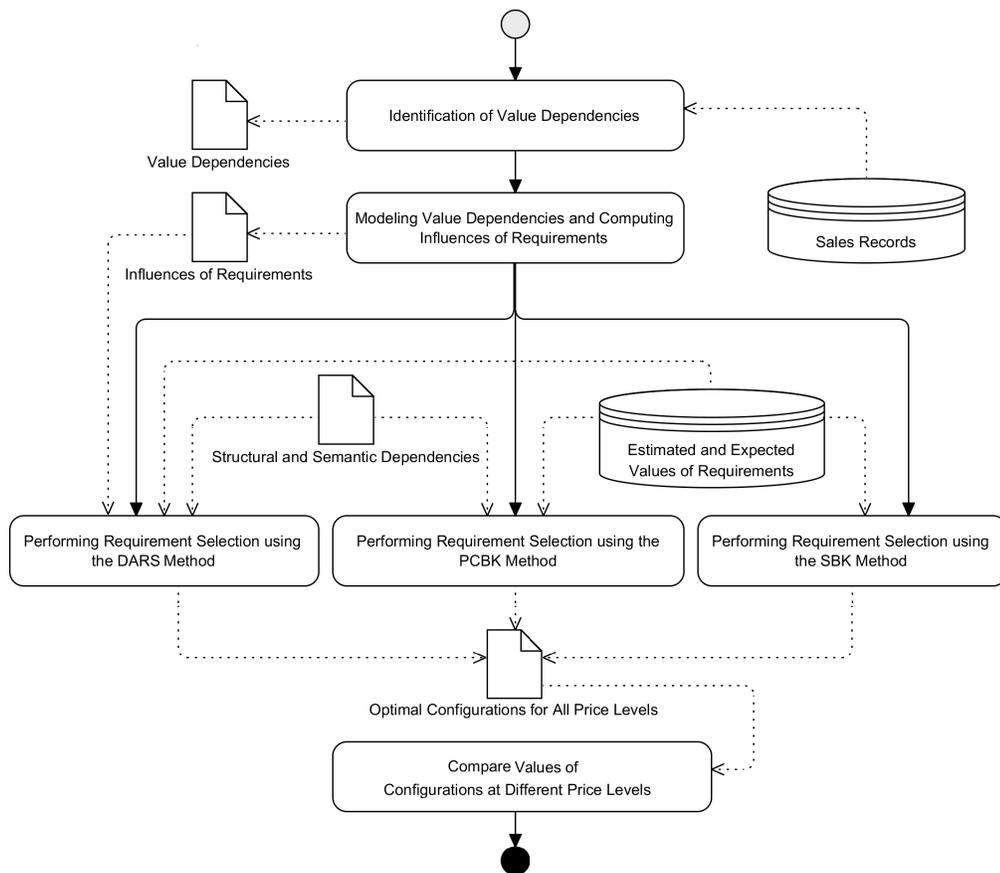


Figure 6: The case study design.

earlier. Then the Eells measure of causal strength was computed for all pairs of requirements using Algorithm 1 to identify the strengths and qualities of causal relations among the requirements as explained in Section 3.3. The significances of the identified relations were subsequently tested using the Odds Ratio at confidence level 95% as explained in Section 3.4. 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(a) as given by (23)-(24). Algorithm 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 the requirements. The influences of the requirements on the values of each other were then computed by (31).

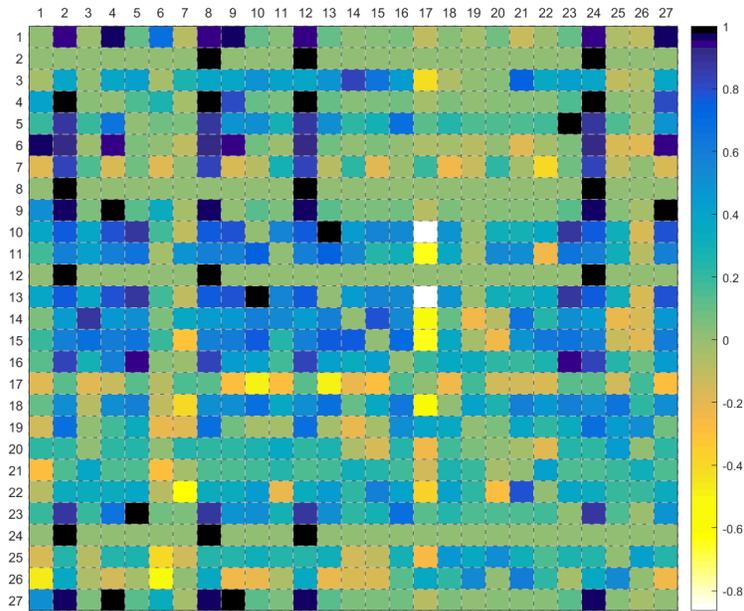


Figure 8: Explicit value dependencies among the requirements. Row i and column j denotes quality and strength of a value dependency from requirement r_i to r_j .

Figure 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 value dependency. Similarly, the positive or negative influences of the requirements on the values of each other are depicted in Figure 9.

6.3. Performing Requirements Selection

This section demonstrates the effectiveness of DARS in considering value dependencies compared to BK, PCBK and SBK methods. As discussed in Section 2.2, PCBK considers the estimated values of requirements while SBK, as

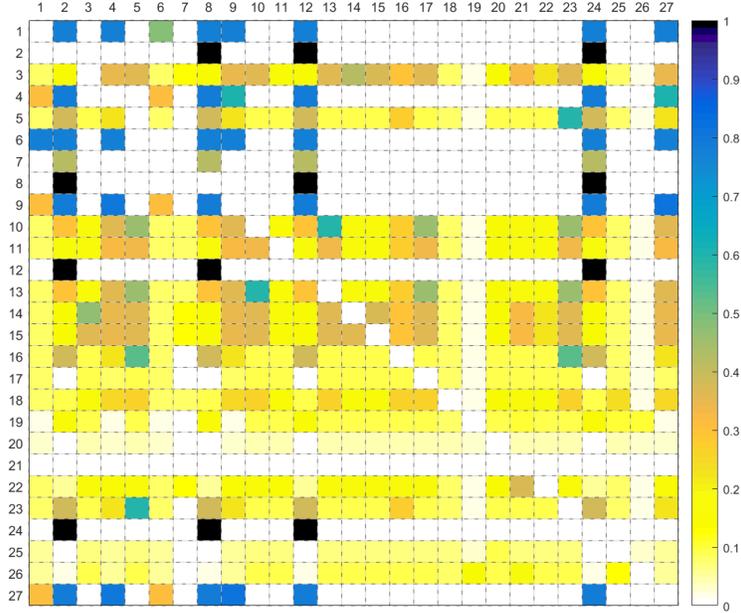


Figure 9: Influences among the requirements: cell i, j gives the influence of r_j on r_i .

in Section 2.4, accounts for user preferences for requirements by considering the expected values of requirements rather than their estimated values. Finally, DARS method factors in both user preferences and value dependencies among requirements as explained in Section 5.2. The expected values of the requirements and value dependencies among them are computed based on the user preferences achieved from the sales records of the project.

$$\sum_{i=1}^{27} v_i x_i \leq \gamma \quad (70)$$

Price was determined as a major constraint for requirements selection as different configurations of the project had been released at different price levels earlier to cope with the needs of different users (Karpoff, 1987). Constraints (70) hence was added to the optimization models of PCBK, SBK, and DARS respectively to contain the price of different configurations of software within their corresponding price limits. This converted the problem to a variation of 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 (16) from the optimization model of SBK as considering the sales diversification is beyond the scope of this paper. The concept of diversification was explained in detail in Section 2.

Selection tasks were performed for different price levels ($\%Price = \{1, \dots, 100\}$, $Price = \frac{\%Price}{100} \times 342$) using the optimization models of PCBK, SBK, and DARS

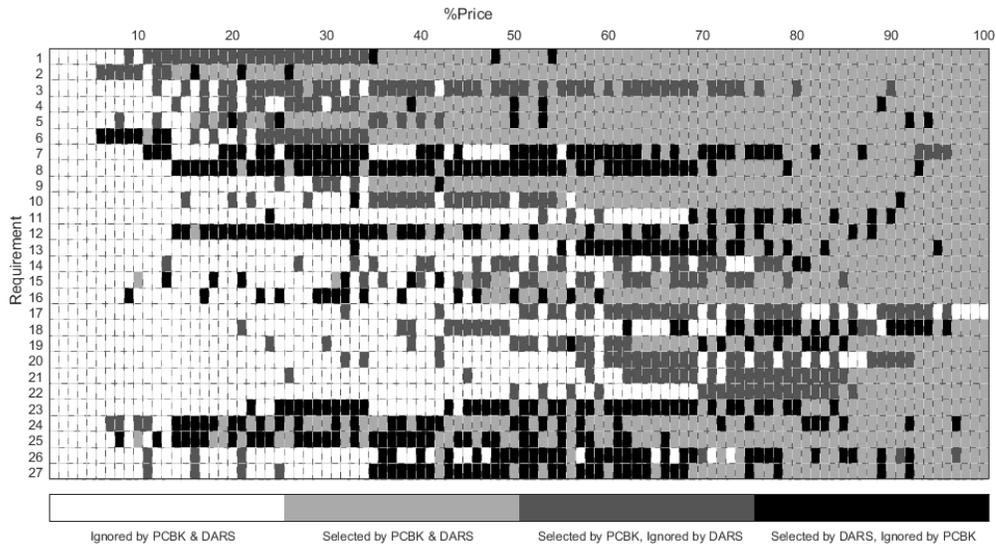


Figure 10: Requirement subsets found by DARS and PCBK for different price levels.

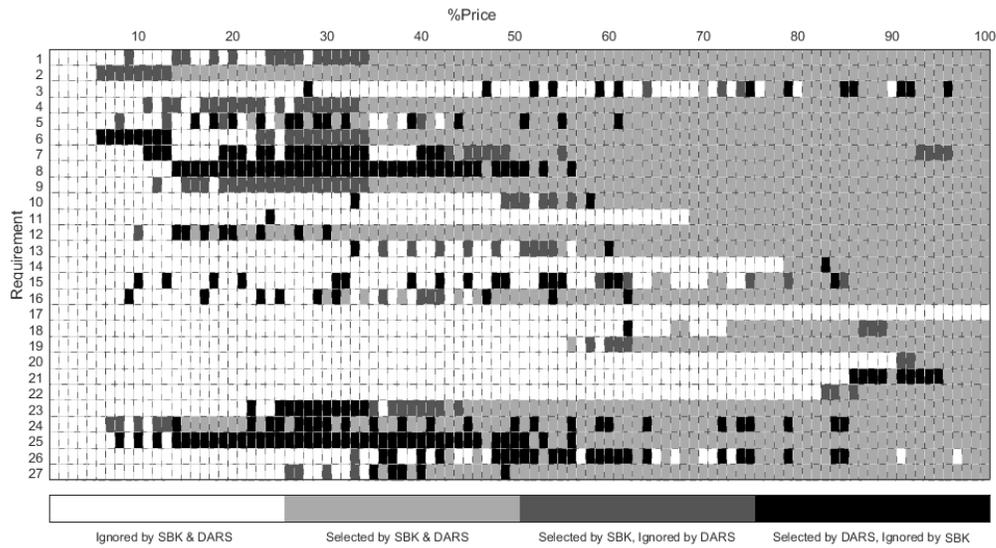


Figure 11: Requirement subsets found by DARS and SBK for different price levels.

with (56)-(69) to find optimal subsets of the requirements (optimal configurations). Optimal configurations found by PCBK, SBK, and DARS were compared based on their similarities, accumulated values, expected values, and overall values to answer **(RQ1)** and its subquestions. Binary knapsack (BK) method (Section 2.1) and the Increase-Decrease method (Section 2.3) were not used in the requirements selection tasks as the former ignores precedence dependencies resulting in violation of the precedence constraints while the latter does not provide any formal way to specify the amounts of the increased or decreased values of the requirement subsets as detailed in Section 2.3. 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.

6.3.1. Similarities of the Solutions

In this section, we compare PCBK, SBK, and DARS based on their selection patterns to answer **(RQ1.1)**. Figure 12 depicts dissimilarities between the requirement subsets found by the DARS 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 the project. The reason is expensive configurations of software comprise most requirements thus reducing the chances that requirements with positive influences on the values of the selected requirements are ignored.

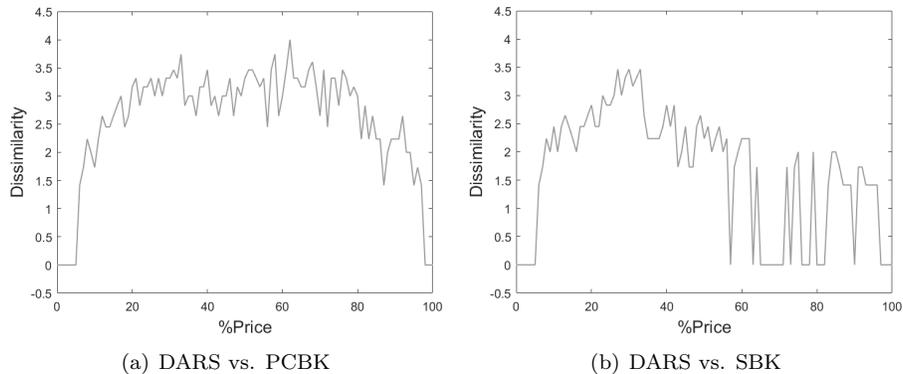


Figure 12: Dissimilarities of the solutions found by DARS and PCBK/SBK.

Moreover, there are no negative influences among requirements (Figure 9). Hence, similarities between solutions found by the DARS method and those found by PCBK and SBK increase for expensive configurations of software. For cheaper configurations, price constraint limits the solution space for PCBK, SBK, and DARS especially preventing the DARS method from utilizing its advantage in considering value dependencies. This resulted in more similarities between the solutions found by the DARS method and those found by PCBK and SBK for very cheap configurations of software.

Finally, we observed from Figure 13(a) and Figure 13(b) that the requirement subsets (solutions) found by DARS were more similar to the solutions found by SBK than similar to the solutions found by PCBK. The reason is as explained before both DARS and SBK consider user preferences while PCBK ignores those preferences.

Figure 13 provides more insights into **(RQ1.1)** by comparing the selection patterns of PCBK, SBK, and DARS in 100 different selection tasks performed at different price levels ($\%Price = \{1, 2, \dots, 100\}$). For a given requirement r_i , $\%F_i(m_j)$ specifies the percentages of the selection tasks in which r_i is selected by requirements selection method m_j . Hence, $\%\Delta F_i(m_j, m_k) = \%F_i(m_j) - \%F_i(m_k) > 0$ states that the percentages of the selection tasks where r_i is selected by m_j is higher than the percentages of the selection tasks where r_i is selected by m_k . Similarly, $\%\Delta F_i(m_j, m_k) = \%F_i(m_j) - \%F_i(m_k) < 0$ 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.

We observed (Figure 13) that requirements with significant influence on the values of pricey requirements were more frequently preferred by DARS compared to PCBK and SBK. This was more visible for requirements r_8 , r_{12} , r_{24} , and r_{27} when in Figure 13(a) and for requirements r_8 , r_{12} , r_{24} in Figure 13(b).

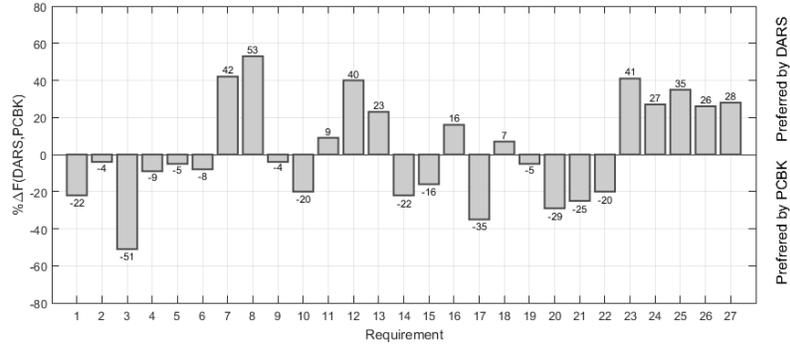
Requirement r_8 , for instance, was more frequently preferred by DARS compared to PCBK and SBK as the optimization model of DARS considers the fact that r_8 has a significant positive influence on the values of several valuable requirements including r_2 , r_4 , r_6 , and r_{12} (Figure 9). Similarly, r_{24} has a significant (positive) influence on the values of requirements r_2 , r_6 , r_8 , and r_{12} . r_{25} however, was more frequently selected by DARS as r_8 requires r_{25} (Figure 7) and r_8 is frequently selected by DARS due to its significant impact on valuable requirements. As such, selecting r_8 requires the presence of r_{25} in software. DARS 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 13 showed that when a decision was to be made regarding the presence or absence of a requirement r_i in a configuration of software, 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.

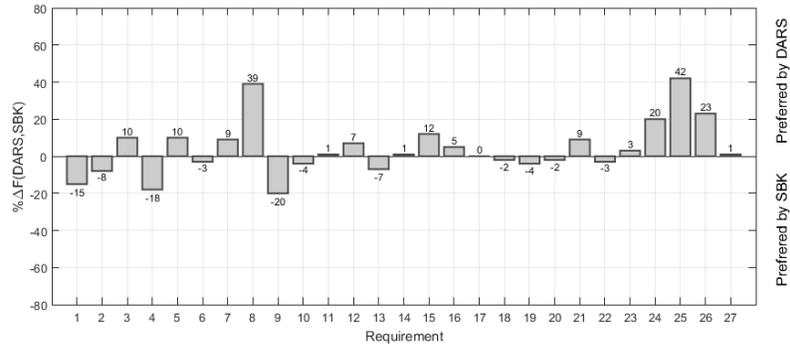
DARS, however, considered the expected value of r_i and the impact of r_i on the values of other requirements. More similarities were, therefore, observed among the configurations found by SBK and DARS as both methods took into account user preferences. On the contrary, dissimilarities were more visible when SBK and DARS/PCBK were compared as demonstrated in Figure 13(a) and Figure 13(c).

6.3.2. Impact of DARS on the Overall Value

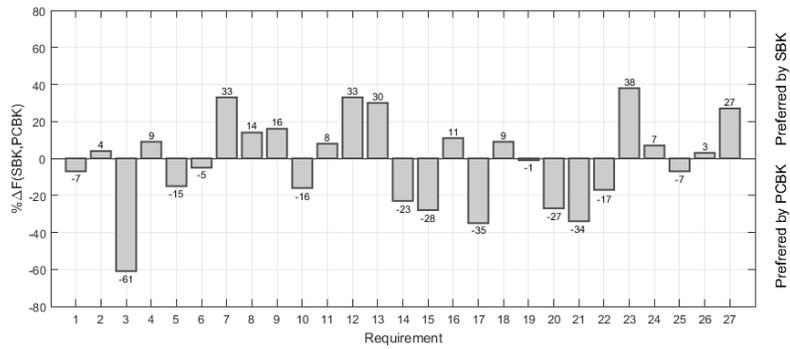
(RQ1.2) is answered by comparing the percentages of overall values ($\%OV = (OV/342) \times 100$), accumulated values ($\%AV = (AV/342) \times 100$), and estimated values ($\%EV = (EV/342) \times 100$) provided by PCBK, SBK, and DARS for 100



(a) DARS vs. PCBK



(b) DARS vs. SBK



(c) SBK vs. PCBK

Figure 13: Selection patterns of PCBK, SBK, and DARS at different price levels. For a requirement r_i , denoted by i on the x-axis, and requirements 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 m_j and m_k respectively.

selection tasks, each performed at a specific price level ($\%Price = \{1, 2, \dots, 100\}$), as shown in Figures 14-18. Our results show (Figure 14) that requirement subsets found by DARS provided higher or equal $\%OV$ in all selection tasks compared to the PCBK method. The reason is that the optimization model of PCBK ignores user preferences and value dependencies among the requirements.

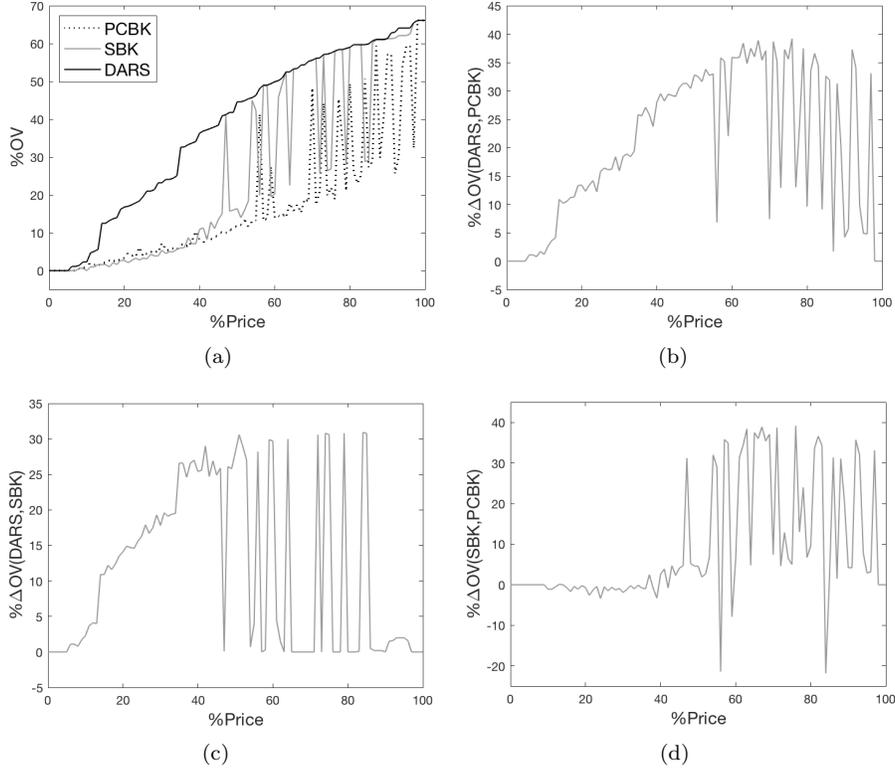


Figure 14: Comparing the overall values at different price levels. $\% \Delta OV(m_j, m_k) = \%OV(m_j) - \%OV(m_k)$, where m_j denotes a selection method (DARS, PCBK, or SBK).

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. The reason is that even though the SBK method does not consider value dependencies, it still accounts for user preferences, similar to DARS, by optimizing the expected values of selected requirements. This results in more similarities between the configurations found by the SBK method and those found by DARS as discussed in Section 6.3.1.

We observed in Figure 14(b) that the gap between the $\%OV$ achieved from DARS 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.

The reason is, on one hand, there are no negative influences among the requirements (Figure 9) and, on the other hand, expensive configurations of software comprise most requirements, which reduces the chances that requirements with positive influence are ignored by PCBK/SBK.

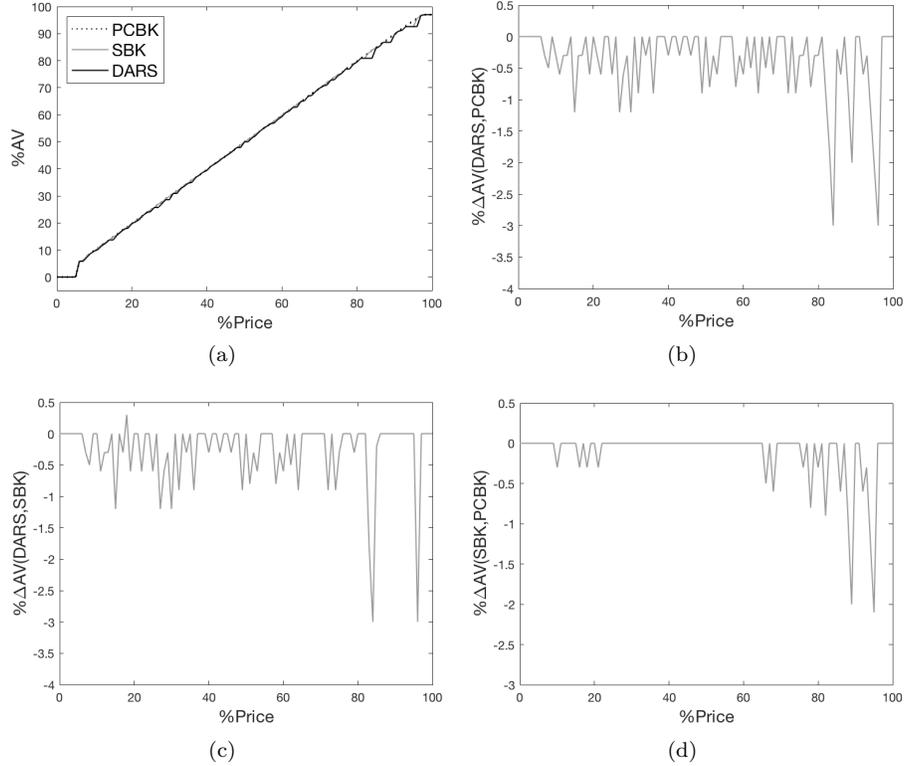


Figure 15: Comparing the accumulated values at different price levels. $\% \Delta AV(m_j, m_k) = \% AV(m_j) - \% AV(m_k)$, where m_j denotes a selection method (DARS, PCBK, or SBK).

That increases similarities between the expensive configurations found by PCBK/SBK and DARS as discussed in Section 6.3.1. For cheaper configurations, the price constraint limited the solution space in all the PCBK, SBK, and DARS and specially prevented DARS from utilizing its advantage in considering value dependencies. The price constraint further reduced the gap between %AV provided by DARS and PCBK/SBK (Figures 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 15. The reason is that the price constraint in the optimization models of the PCBK, SBK, and DARS contain the accumulated values of the solutions found by those models. The price constraint is needed to factor out the interplay between the price and sales as explained earlier. Moreover, the expected

values of the requirement subsets found by the SBK method were higher than those found by DARS and PCBK in all selection tasks (for all price levels) as shown in Figure 16(c) and Figure 16(d) respectively.

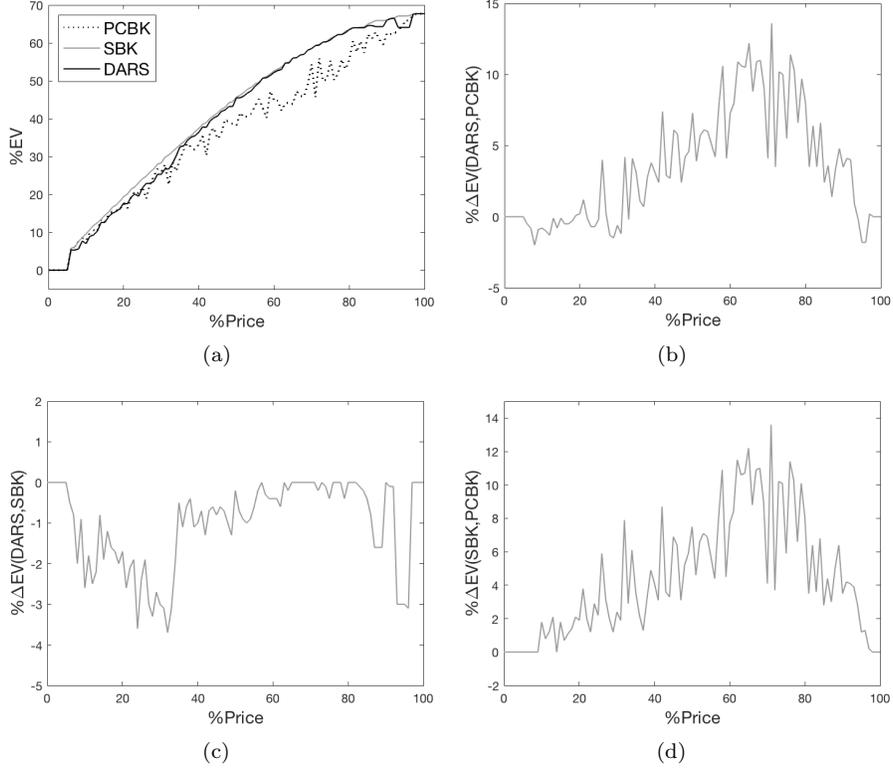


Figure 16: Comparing the accumulated values at different price levels. $\% \Delta EV(m_j, m_k) = \%EV(m_j) - \%EV(m_k)$, where m_j denotes a selection method (DARS, PCBK, or SBK).

The expected values of requirement subsets found by DARS were higher than those of the requirement subsets found by the PCBK method in most selection tasks (Figure 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 DARS even though the PCBK method does not account for user preferences. The reason is that DARS optimizes the overall value of a requirement subset (solution), which accounts for both user preferences and value dependencies. Hence in some cases DARS 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.

6.3.3. Understanding the Conflicting Objectives

To answer **(RQ1.3)**, we compared the overall values (Figure 14), accumulated values (Figure 15), and expected values (Figure 18) of the requirement subsets found by the PCBK, SBK, and DARS in different requirements selection tasks performed at different price levels. From Figure 14 and Figure 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 14(b) and Figure 15(b), where in several selection tasks, choosing requirement subsets with higher %AV by the PCBK method (Figure 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 (34), resulting in lower overall value. This can be seen by comparing Figure 14(c) and Figure 16(c).

6.3.4. Mitigating the Value Loss

Ignoring value dependencies can pose a risk to the economic worth of software configurations and eventually result in value loss as given by (34). This risk can be measured by the gap between the expected value of software and its overall value, which accounts for value dependencies, as depicted in Figure 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 software configuration found by DARS was notably smaller than the gaps between the %EV and %OV provided by the PCBK method. Hence, using DARS contributed to a smaller risk of value loss in different configurations of software.

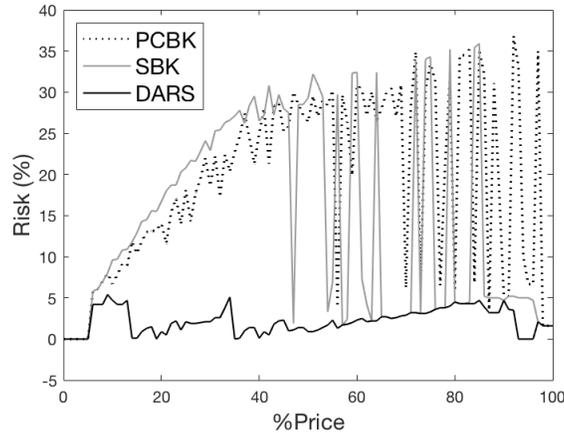


Figure 17: Risk of value loss at different price levels.

We observed (Figure 17) that the risk of value loss for the configurations found by DARS was under 5% while this fluctuated from almost negligible to around 37% in PCBK and SBK. In most configurations found by 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 software. The risk of value loss for the configurations found by the SBK method, however, converged to those found by DARS for %Price ≥ 88 as both methods chose more similar configurations ((**RQ1.4**)).

7. Complexity and Scalability Analysis

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

(**RQ2**) What is the overhead of identifying and modeling dependencies?

(**RQ3**) How scalable is the ILP model of DARS?

(**RQ3.1**) Is the ILP model scalable to large scale requirement sets?

(**RQ3.2**) What is the impact of budget on runtime?

(**RQ3.3**) What is the impact of precedence dependencies on runtime?

(**RQ3.4**) What is the impact of value dependencies on runtime?

7.1. The Overhead of using DARS

Our proposed DARS method relies on the identification and modeling of value dependencies – that constitutes the main overhead of DARS. Identification of value dependencies from causal relations among user preferences is automated in DARS as explained in Section 3. The process, nevertheless, relies on computing the Eells measure (Eells, 1991) for pairs of the requirements. Algorithm 2 computes the Eells measure in $O(t \times n^2)$ for n requirements and t records of user preferences. Precedence dependencies among requirements (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 (Zhang et al., 2005; Dahlstedt & Persson, 2005). This is an inevitable aspect of software requirement analysis and is not specific to DARS. Moreover, construction of a value dependency graph of requirements, inferring implicit value dependencies, and computing the influences of requirements using Algorithm 2 is of the computational complexity of $O(n^3)$ as discussed earlier in Section 4. This concluded our answer to (**RQ2**).

7.2. Scalability of the Optimization Model of DARS

The optimization model of the DARS method as given by (45)-(53) 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 5 were carried out. 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 (71) and (72) 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 (71) and (72) respectively.

Table 5: Runtime Simulations for the optimization model of DARS

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

$$PDL(G) = \frac{k}{{}^n P_2} = \frac{k}{n(n-1)} \quad (71)$$

$$NPDL(G) = \frac{j}{k} \quad (72)$$

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.

(**RQ3.1**) is answered by runtime simulation 1, which evaluates the runtime of the optimization model of DARS for different numbers of requirements (Figure 18(a)). We observed that increasing the number of requirements increased, as expected, the runtime of the optimization model of the DARS 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 the selection was completed. On the other hand, our results for Simulation 2 demonstrated (Figure 18(b)) that the runtime of the optimization model of the DARS method

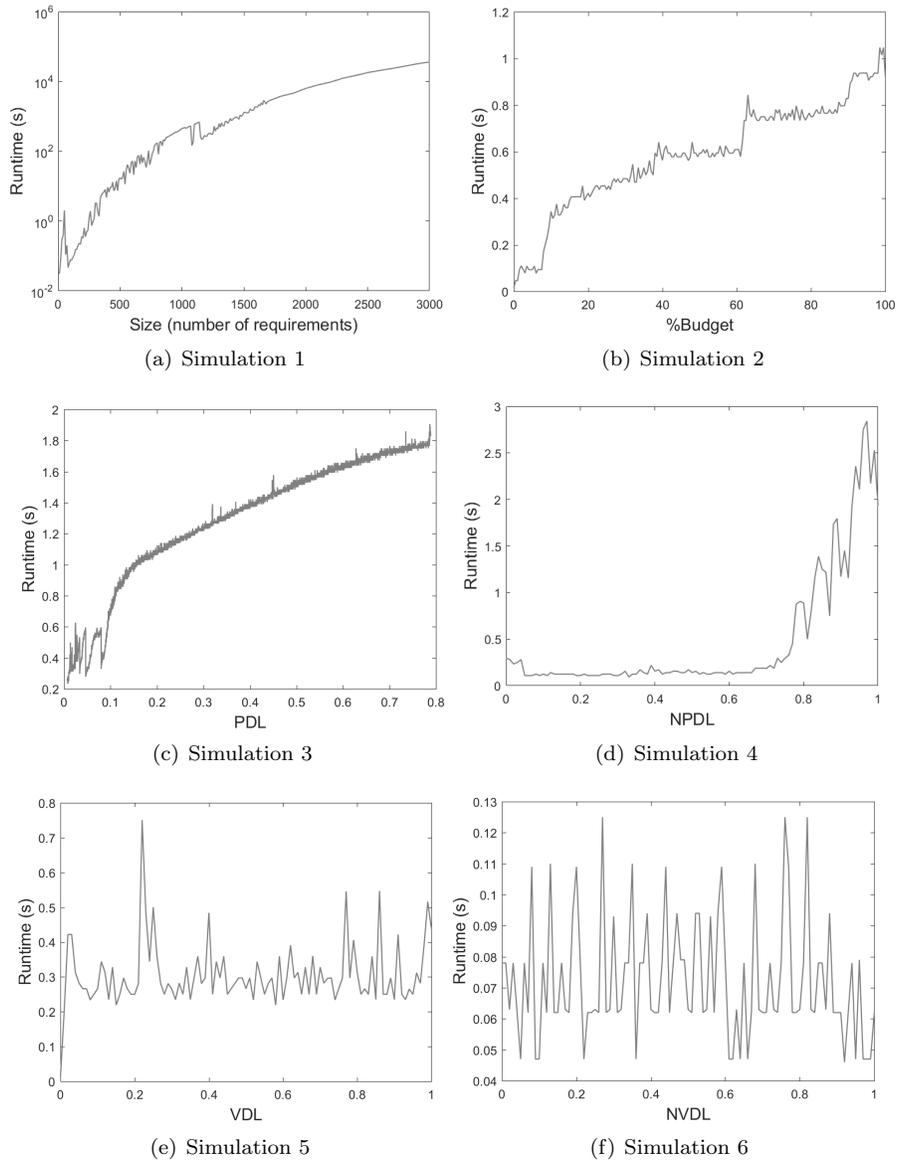


Figure 18: Runtime of DARS.

increased with a 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 DARS to find the optimal subset. This answers **(RQ3.2)**. To answer **(RQ3.3)**, we simulated requirements selection for various precedence dependency levels (PDLs). Our results (Figure 18(c)) demonstrated, that, in general, the runtime of the optimization model of DARS increased when PDL increased. The reason is increasing PDL limits the number of choices for the optimization model of the DARS method as the model needs to respect precedence dependencies; it takes longer for the selection task to complete. Increasing NPDL, on the other hand, had no significant impact on the runtime of the optimization model of DARS in most places. Nonetheless, for larger NPDLs ($NPDL \rightarrow 1$), runtime was increased. The reason is at such high NPDL, the optimization model of DARS 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.

Simulation 5 was carried out to answer **(RQ3.4)** by measuring the runtime of the selection models in the presence of various value dependency levels (VDLs). Our results demonstrate (Figure 18(e)) that increasing (decreasing) VDL has an inconsistent impact of negligible magnitude on the runtime of the optimization model of the DARS method. In a similar way, our simulations for various negative value dependency levels (NVDLs) showed (Figure 18(f)) that the impact of increasing (decreasing) NVDL on the runtime of the optimization model of the DARS method was unpredictable.

References

- van den Akker, M., Brinkkemper, S., Diepen, G., & Versendaal, J. (2005a). Determination of the next release of a software product: an approach using integer linear programming. In *CAiSE Short Paper Proceedings*.
- van den Akker, M., Brinkkemper, S., van Diepen, G., & Versendaal, J. (2005b). Flexible release planning using integer linear programming. *REFSQ'05*, .
- van den Akker, M., Brinkkemper, S., Diepen, G., & Versendaal, J. (2008). Software product release planning through optimization and what-if analysis. *Information and Software Technology*, 50, 101–111.
- Araújo, A. A., Paixao, M., Yeltsin, I., Dantas, A., & Souza, J. (2016). An architecture based on interactive optimization and machine learning applied to the next release problem. *Automated Software Engineering*, (pp. 1–49).
- Aydemir, F. B., Dalpiaz, F., Brinkkemper, S., Giorgini, P., & Mylopoulos, J. (2018). The next release problem revisited: A new avenue for goal models. In *2018 IEEE 26th International Requirements Engineering Conference (RE)* (pp. 5–16). IEEE.
- Bagnall, A. J., RaywardSmith, V. J., & Whittlely, I. M. (2001). The next release problem. *Information and Software Technology*, 43, 883–890.
- Baker, P., Harman, M., Steinhofel, K., & Skaliotis, A. (2006). Search based approaches to component selection and prioritization for the next release problem. In *Proceedings of the 22Nd IEEE International Conference on Software Maintenance* (pp. 176–185). IEEE.

- Boschetti, M. A., Golfarelli, M., Rizzi, S., & Turricchia, E. (2014). A lagrangian heuristic for sprint planning in agile software development. *Computers & Operations Research*, *43*, 116–128.
- Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.
- Brasil, M. M. A., Silva, T. G. N. d., Freitas, F. G. d., Souza, J. T. d., & Corts, M. I. (2012). 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, & J. Cordeiro (Eds.), *Enterprise Information Systems 102* (pp. 300–314). Springer Berlin Heidelberg.
- Carlshamre, P. (2002). Release planning in market-driven software product development: Provoking an understanding. *Requirements Engineering*, *7*, 139–151.
- Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., & Natt och Dag, J. (2001). An industrial survey of requirements interdependencies in software product release planning. In *Fifth IEEE International Symposium on Requirements Engineering, 2001. Proceedings* (pp. 84–91).
- Chen, W.-N., & Zhang, J. (2013). Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *IEEE Transactions on Software Engineering*, *39*, 1–17.
- Colares, F., Souza, J., Carmo, R., Padua, C., & Mateus, G. (2009a). A new approach to the software release planning. In *Software Engineering, 2009. SBES '09. XXIII Brazilian Symposium on* (pp. 207–215).
- Colares, F., Souza, J., Carmo, R., Pádua, C., & Mateus, G. R. (2009b). A new approach to the software release planning. In *Software Engineering, 2009. SBES'09. XXIII Brazilian Symposium on* (pp. 207–215). IEEE.
- Dahlstedt, Å. G., & Persson, A. (2005). Requirements interdependencies: state of the art and future challenges. In *Engineering and managing software requirements* (pp. 95–116). Springer.
- De Kleer, J., & Brown, J. S. (1984). A qualitative physics based on confluences. *Artificial intelligence*, *24*, 7–83.
- Eells, E. (1991). *Probabilistic causality* volume 1. Cambridge University Press.
- Finkelstein, A., Harman, M., Mansouri, S. A., Ren, J., & Zhang, Y. (2009). A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. *Requirements Engineering*, *14*, 231–245.
- Floyd, R. W. (1962). Algorithm 97: shortest path. *Communications of the ACM*, *5*, 345.
- Franch, X., & Ruhe, G. (2016). Software release planning. In *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 894–895). ACM.
- Freitas, F. G., Coutinho, D. P., & Souza, J. T. (2011). Software next release planning approach through exact optimization. *Int. J. Comput. Appl*, *22*, 1–8.
- Geng, L., & Hamilton, H. J. (2006). Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, *38*, 9.
- Gkioulekas, I., & Papageorgiou, L. G. (2019). Piecewise regression analysis through information criteria using mathematical programming. *Expert Systems with Applications*, *121*, 362–372.

- Greer, D., & Ruhe, G. (2004a). Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, *46*, 243 – 253.
- Greer, D., & Ruhe, G. (2004b). Software release planning: an evolutionary and iterative approach, . *46*, 243–253.
- Halpern, J. Y., & Hitchcock, C. (2015). Graded causation and defaults. *The British Journal for the Philosophy of Science*, *66*, 413–457.
- Harman, M., Krinke, J., MedinaBulo, I., PalomoLozano, F., Ren, J., & Yoo, S. (2014). Exact scalable sensitivity analysis for the next release problem. *ACM Trans. Softw. Eng. Methodol.*, *23*, 19:119:31.
- Henig, M. I. (1990). Risk criteria in a stochastic knapsack problem. *Operations research*, *38*, 820–825.
- Holland, S., Ester, M., & Kießling, W. (2003). Preference mining: A novel approach on mining user preferences for personalized applications. In *European Conference on Principles of Data Mining and Knowledge Discovery* (pp. 204–216). Springer.
- Hussain, W., Mougouei, D., & Whittle, J. (2018). Integrating social values into software design patterns. In *2018 IEEE/ACM International Workshop on Software Fairness (FairWare)* (pp. 8–14). IEEE. doi:10.1145/3194770.3194777.
- Janzing, D., Balduzzi, D., Grosse-Wentrup, M., Schölkopf, B. et al. (2013). Quantifying causal influences. *The Annals of Statistics*, *41*, 2324–2358.
- Jiang, H., Zhang, J., Xuan, J., Ren, Z., & Hu, Y. (2010). A hybrid aco algorithm for the next release problem. In *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on* (pp. 166–171). IEEE.
- Jung, H.-W. (1998). Optimizing value and cost in requirements analysis. *IEEE Software*, *15*, 74–78.
- K, P. (1996). *Process-Centered Requirements Engineering*. Taunton, Somerset, England New York: Research Studies Pre.
- Kalampakas, A., Spartalis, S., Iliadis, L., & Pimenidis, E. (2013). Fuzzy graphs: algebraic structure and syntactic recognition. *Artificial Intelligence Review*, (pp. 1–12).
- Karlsson, J., Olsson, S., & Ryan, K. (1997). Improved practical support for largescale requirements prioritising. *Requirements Engineering*, *2*, 51–60.
- Karlsson, J., & Ryan, K. (1997). A costvalue approach for prioritizing requirements. *IEEE Software*, *14*, 67–74.
- Karpoff, J. M. (1987). The relation between price changes and trading volume: A survey. *Journal of Financial and quantitative Analysis*, *22*, 109–126.
- Kroese, D. P., Chan, J. C. et al. (2014). *Statistical modeling and computation*. Springer.
- Kumari, A. C., Srinivas, K., & Gupta, M. (2012). Software requirements selection using quantum-inspired elitist multi-objective evolutionary algorithm. In *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on* (pp. 782–787). IEEE.
- Kusiak, A., & Wang, J. (1995). Dependency analysis in constraint negotiation. *Systems, Man and Cybernetics, IEEE Transactions on*, *25*, 1301–1313.
- Leung, C. W.-K., Chan, S. C.-F., Chung, F.-L., & Ngai, G. (2011). A probabilistic rating inference framework for mining user preferences from reviews. *World Wide Web*, *14*, 187–215.

- Li, C., Akker, M. v. d., Brinkkemper, S., & Diepen, G. (2010). An integrated approach for requirement selection and scheduling in software release planning. *Requirements Engineering*, 15, 375–396.
- Li, J., Le, T. D., Liu, L., Liu, J., Jin, Z., Sun, B., & Ma, S. (2016). From observational studies to causal rule mining. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 7, 14.
- Li, L., Harman, M., Letier, E., & Zhang, Y. (2014). Robust next release problem: handling uncertainty during optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (pp. 1247–1254). ACM.
- Li, L., Harman, M., Wu, F., & Zhang, Y. (2017). The value of exact analysis in requirements selection. *IEEE Transactions on Software Engineering*, 43, 580–596.
- Liu, X. F., & Yen, J. (1996). An analytic framework for specifying and analyzing imprecise requirements. In *Proceedings of the 18th international conference on Software engineering* (pp. 60–69). IEEE Computer Society.
- Luenberger, D. G., & Ye, Y. (2015). *Linear and nonlinear programming* volume 228. Springer.
- Macke, J. H., Berens, P., Ecker, A. S., Tolia, A. S., & Bethge, M. (2009). Generating spike trains with specified correlation coefficients. *Neural Computation*, 21, 397–423.
- Mathew, S., & Sunitha, M. (2013). Strongest strong cycles and theta fuzzy graphs. *IEEE Transactions on Fuzzy Systems*, 21, 1096–1104.
- de Melo França, V. J. A. T., Balancieri, R., Leal, G. C. L., & Rouiller, A. C. (2018). Mixed integer programming helping requirements allocation for the nrp in scrum teams. In *Proceedings of the 17th Brazilian Symposium on Software Quality SBQS* (pp. 279–286). New York, NY, USA: ACM. URL: <http://doi.acm.org/10.1145/3275245.3275272>. doi:10.1145/3275245.3275272.
- Mougouei, D. (2013). Goal-based requirement engineering for fault tolerant security-critical systems. *International Journal of Software Engineering and Its Applications*, 7, 1–14.
- Mougouei, D. (2016). Factoring requirement dependencies in software requirement selection using graphs and integer programming. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 884–887). IEEE. doi:10.1145/2970276.2975936.
- Mougouei, D. (2018). *A Mathematical Programming Approach to Considering Value Dependencies in Software Requirement Selection*. Ph.D. thesis Flinders University, School of Computer Science, Engineering and Mathematics.
- Mougouei, D., Moghtadaei, M., & Moradmand, S. (2012a). A goal-based modeling approach to develop security requirements of fault tolerant security-critical systems. In *2012 International Conference on Computer and Communication Engineering (ICCCCE)* (pp. 200–205). IEEE.
- Mougouei, D., & Nurhayati, W. (2013). A fuzzy-based technique for describing security requirements of intrusion tolerant systems. *International Journal of Software Engineering and its Applications*, 7, 99–112.
- Mougouei, D., Perera, H., Hussain, W., Shams, R., & Whittle, J. (2018). Operationalizing human values in software: A research roadmap. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering ESEC/FSE 2018* (pp. 780–784). New York, NY, USA: ACM.

- Mougouei, D., & Powers, D. M. (2017). Modeling and selection of interdependent software requirements using fuzzy graphs. *International Journal of Fuzzy Systems*, (pp. 1–17). doi:10.1007/s40815-017-0364-4.
- Mougouei, D., & Powers, D. M. (2019). Dependency-aware release planning for software projects using fuzzy graphs and integer programming. *Journal of Intelligent & Fuzzy Systems*, (pp. 3693–3707). doi:10.3233/JIFS-182810.
- Mougouei, D., & Powers, D. M. (2020). Dependency-aware software release planning through mining user preferences. *Soft Computing*, . doi:10.1007/s00500-018-3553-7.
- Mougouei, D., Powers, D. M., & Mougouei, E. (2019). A fuzzy framework for prioritization and partial selection of security requirements in software projects. *Journal of Intelligent & Fuzzy Systems*, (pp. 2671–2686). doi:10.3233/JIFS-182907.
- Mougouei, D., Powers, D. M. W., & Moeini, A. (2017a). Dependency-aware software release planning. In *Proceedings of the 39th International Conference on Software Engineering Companion ICSE-C 17* (p. 198200). IEEE Press. URL: <https://doi.org/10.1109/ICSE-C.2017.74>. doi:10.1109/ICSE-C.2017.74.
- Mougouei, D., Powers, D. M. W., & Moeini, A. (2017b). An integer linear programming model for binary knapsack problem with dependent item values. In W. Peng, D. Alahakoon, & X. Li (Eds.), *AI 2017: Advances in Artificial Intelligence: 30th Australasian Joint Conference, Melbourne, VIC, Australia, August 19–20, 2017, Proceedings* (pp. 144–154). Cham: Springer International Publishing. URL: https://doi.org/10.1007/978-3-319-63004-5_12. doi:10.1007/978-3-319-63004-5_12.
- Mougouei, D., Rahman, W., & Almasi, M. M. (2012b). Measuring security of web services in requirement engineering phase. *International Journal of Cyber-Security and Digital Forensics (IJCSDF)*, 1, 89–98.
- Mougouei, D., Rahman, W. N. W. A., & Almasi, M. M. (2012c). Evaluating fault tolerance in security requirements of web services. In *Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)* (pp. 111–116). IEEE.
- Mougouei, D., Shen, H., & Babar, M. A. (2015). Partial selection of agile software requirements. *International Journal of Software Engineering and Its Applications*, 9, 113–126.
- Ngo-The, A., & Ruhe, G. (2008). A systematic approach for solving the wicked problem of software release planning. *Soft Computing*, 12, 95–108.
- Ngo-The, A., & Ruhe, G. (2009). Optimized resource allocation for software release planning. *IEEE Transactions on Software Engineering*, 35, 109–123.
- Ngo The, A., & Saliu, M. O. (2005). Fuzzy structural dependency constraints in software release planning. In *The 14th IEEE International Conference on Fuzzy Systems, 2005. FUZZ'05*. (pp. 442–447). IEEE.
- Ngo-The, A., & Saliu, M. O. (2005). Measuring dependency constraint satisfaction in software release planning using dissimilarity of fuzzy graphs. In *Cognitive Informatics, 2005. (ICCI 2005). Fourth IEEE Conference on* (pp. 301–307). IEEE.
- Pearl, J. (2009). *Causality*. Cambridge university press.
- Perera, H., Hussain, W., Mougouei, D., Shams, R. A., Nurwidyantoro, A., & Whittle, J. (2019a). Towards integrating human values into software: Mapping principles and rights of gdpr to values. In *2019 IEEE 27th International Requirements Engineering Conference (RE)* (pp. 404–409). IEEE. doi:10.1109/RE.2019.00053.

- Perera, H., Nurwidyantoro, A., Hussain, W., Mougouei, D., Whittle, J., Shams, R. A., & Oliver, G. (2019b). A study on the prevalence of human values in software engineering publications, 2015-2018. *arXiv preprint arXiv:1907.07874*, .
- Pitangueira, A., Tonella, P., Susi, A., Maciel, R., & Barros, M. (2017). Minimizing the stakeholder dissatisfaction risk in requirement selection for next release planning. *Information and Software Technology*, .
- Pitangueira, A. M., Maciel, R. S. P., & Barros, M. (2015). Software requirements selection and prioritization using sbse approaches: A systematic review and mapping of the literature. *Journal of Systems and Software*, *103*, 267–280.
- Pitangueira, A. M., Tonella, P., Susi, A., Maciel, R. S., & Barros, M. (2016). Risk-aware multi-stakeholder next release planning using multi-objective optimization. In *International Working Conference on Requirements Engineering: Foundation for Software Quality* (pp. 3–18). Springer.
- Robinson, W. N., Pawlowski, S. D., & Volkov, V. (2003). Requirements interaction management. *ACM Comput. Surv.*, *35*, 132-190.
- Rosenfeld, A. (1975). Fuzzy graphs. *Fuzzy Sets and Their Applications*, *77*, 95.
- Ruhe, G., & Greer, D. (2003). Quantitative studies in software release planning under risk and resource constraints. In *Proceedings of the 2003 International Symposium on Empirical Software Engineering* (pp. 262–270).
- del Sagrado, J., ÁAguila, I. M., & Orellana, F. J. (2011). Requirements interaction in the next release problem. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation* (pp. 241–242). ACM.
- del Sagrado, J., del Aguila, I. M., & Orellana, F. J. (2010). Ant colony optimization for the next release problem: A comparative study. In *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on* (pp. 67–76). IEEE.
- Sagrado, J. d., guila, I. M. d., & Orellana, F. J. (2013). Multiobjective ant colony optimization for requirements selection. *Empirical Software Engineering*, (pp. 1–34).
- Saliu, M. O., & Ruhe, G. (2007). 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* (pp. 105–114). ACM.
- Saliu, O., & Ruhe, G. (2005a). Software release planning for evolving systems. *Innovations in Systems and Software Engineering*, *1*, 189–204.
- Saliu, O., & Ruhe, G. (2005b). Supporting software release planning decisions for evolving systems. In *29th Annual IEEE/NASA Software Engineering Workshop* (pp. 14–26). IEEE.
- Sayyad, A. S., Menzies, T., & Ammar, H. (2013). 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)* (pp. 492–501). IEEE.
- Tavana, M., Keramatpour, M., Santos-Arteaga, F. J., & Ghorbaniane, E. (2015). A fuzzy hybrid project portfolio selection method using data envelopment analysis, topsis and integer programming. *Expert Systems with Applications*, *42*, 8432–8444.
- Tonella, P., Susi, A., & Palma, F. (2010). Using interactive ga for requirements prioritization. In *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on* (pp. 57–66). IEEE.

- Tonella, P., Susi, A., & Palma, F. (2013). Interactive requirements prioritization using a genetic algorithm. *Information and software technology*, 55, 173–187.
- van Valkenhoef, G., Tervonen, T., de Brock, B., & Postmus, D. (2011). Quantitative release planning in extreme programming. *Information and software technology*, 53, 1227–1235.
- Veerapen, N., Ochoa, G., Harman, M., & Burke, E. K. (2015). An integer linear programming approach to the single and bi-objective next release problem. *Information and Software Technology*, 65, 1–13.
- Villarroel, L., Bavota, G., Russo, B., Oliveto, R., & Di Penta, M. (2016). Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering* (pp. 14–24). ACM.
- Wang, J., Li, J., Wang, Q., Zhang, H., & Wang, H. (2012). A simulation approach for impact analysis of requirement volatility considering dependency change. In B. Regnell, & D. Damian (Eds.), *Requirements Engineering: Foundation for Software Quality* 7195 (pp. 59–76). Springer Berlin Heidelberg.
- Wasserman, S., & Faust, K. (1994). *Social network analysis: Methods and applications* volume 8. Cambridge University Press.
- Wellman, M. P., & Derthick, M. (1990). *Formulation of tradeoffs in planning under uncertainty*. Pitman London.
- Xuan, J., Jiang, H., Ren, Z., & Luo, Z. (2012). Solving the large scale next release problem with a backbone-based multilevel algorithm. *IEEE Transactions on Software Engineering*, 38, 1195–1212.
- Zadeh, L. A. (1965). Fuzzy sets. *Inf. Comput.*, 8, 338–353.
- Zahedi, Z. M., Akbari, R., Shokouhifar, M., Safaei, F., & Jalali, A. (2016). Swarm intelligence based fuzzy routing protocol for clustered wireless sensor networks. *Expert Systems with Applications*, 55, 313–328.
- Zhang, H., Li, J., Zhu, L., Jeffery, R., Liu, Y., Wang, Q., & Li, M. (2014). Investigating dependencies in software requirements for change propagation analysis. *Information and Software Technology*, 56, 40–53.
- Zhang, W., Mei, H., & Zhao, H. (2005). A feature-oriented approach to modeling requirements dependencies. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on* (pp. 273–282). IEEE.
- Zhang, Y., & Harman, M. (2010). Search based optimization of requirements interaction management. In *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on* (pp. 47–56). IEEE.
- Zhang, Y., Harman, M., Finkelstein, A., & Mansouri, S. A. (2011). Comparing the performance of metaheuristics for the analysis of multi-stakeholder tradeoffs in requirements optimisation. *Information and Software Technology*, 53, 761–773.
- Zhang, Y., Harman, M., & Lim, S. L. (2013). Empirical evaluation of search based requirements interaction management. *Information and Software Technology*, 55, 126 – 152. Special section: Best papers from the 2nd International Symposium on Search Based Software Engineering 2010.
- Zhang, Y., Harman, M., & Mansouri, S. A. (2007). The multi-objective next release problem. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (p. 11291137). New York, NY, USA: ACM.

- Zhang, Y., Harman, M., Ochoa, G., Ruhe, G., & Brinkkemper, S. (2018a). An empirical study of meta- and hyper-heuristic search for multi-objective release planning. *ACM Trans. Softw. Eng. Methodol.*, 27, 3:1–3:32.
- Zhang, Y., Harman, M., Ochoa, G., Ruhe, G., & Brinkkemper, S. (2018b). An empirical study of meta- and hyper-heuristic search for multi-objective release planning. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27, 3.