2010

Representing and reasoning with qualitative preferences for compositional systems

Ganesh Ram Santhanam
Iowa State University

Representing and reasoning with qualitative preferences for compositional systems

by

Ganesh Ram Santhanam

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Vasant G. Honavar, Major Professor
Samik Basu
Carl Chang
Robyn Lutz
Giora Slutzki

Iowa State University

Ames, Iowa

2010

DEDICATION

To my master.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I am very grateful to Dr. Honavar for first of all suggesting to me the idea of pursuing a PhD, and for his valuable guidance throughout. I am thankful for the freedom he gave me to explore research directions of my choice. I am also very grateful to Dr. Basu for giving his valuable time and for enthusiastically evaluating my research ideas. I am grateful to the other members of my committee for giving their valuable time and inputs.

I am thankful to Bhavesh for his kind and friendly association, and the other AI lab members. I thank all the departmental staff for their timely help. I gratefully acknowledge the support I received from Dr. Honavar, Dr. Basu and Dr. McCalley through the NSF grants CNS0709217, CCF0702758, IIS0711356 and CNS0540293.

I am deeply indebted to Dr. P.V.Krishnan, who taught me that education should be aimed at developing perfect character. I am very grateful to Dr. Rangan and many other friends for their loving association during the program, which helped me make steady progress in my PhD and in all other aspects of life.

I am extremely grateful to my parents and family members for allowing me to pursue this program, and for their support and encouragement throughout.

ABSTRACT

Many applications call for techniques for representing and reasoning about preferences, i.e., relative desirability over a set of alternatives. Preferences over the alternatives are typically derived from preferences with respect to the various attributes of the alternatives (e.g., a student's preference for one course over another may be influenced by his preference for the topic, the time of the day when the course is offered, etc.). Such preferences are often qualitative and conditional. When the alternatives are expressed as tuples of valuations of the relevant attributes, preferences between alternatives can often be expressed in the form of (a) preferences over the values of each attribute, and (b) relative importance of certain attributes over others. An important problem in reasoning with multi-attribute qualitative preferences is dominance testing, i.e., to find if one alternative (assignment to all attributes) is preferred over another. This problem is hard (PSPACE-complete) in general for well known qualitative conditional preference languages such as TCP-nets.

We provide two practical approaches to dominance testing. First, we study a restricted unconditional preference language, and provide a dominance relation that can be computed in polynomial time by evaluating the satisfiability of an appropriately constructed logic formula. Second, we show how to reduce dominance testing for TCP-nets to reachability analysis in an *induced preference graph*. We provide an encoding of TCP-nets in the form of a Kripke structure for CTL. We show how to compute dominance using NuSMV, a model checker for CTL.

We address the problem of identifying a preferred outcome in a setting where the

outcomes or alternatives to be compared are composite in nature (i.e., collections of components that satisfy certain functional requirements). We define a dominance relation that allows us to compare collections of objects in terms of preferences over attributes of the objects that make up the collection, and show that the dominance relation is a strict partial order under certain conditions. We provide algorithms that use this dominance relation to identify only (sound), all (complete), or at least one (weakly complete) of the most preferred collections. We establish some key properties of the dominance relation and analyze the quality of solutions produced by the algorithms. We present results of simulation experiments aimed at comparing the algorithms, and report interesting conjectures and results that were derived from our analysis.

Finally, we show how the above formalism and algorithms can be used in preference-based service composition, substitution, and adaptation.

CHAPTER 1. Introduction

Many applications call for techniques for representing and reasoning about preferences over a set of alternatives or outcomes. Such preferences may be *qualitative* or *quantitative*. Qualitative preferences are expressed in the form of binary relations (preference structures) on the set of alternatives, whereas quantitative preferences are expressed in the form of real valued functions on the set of alternatives.

Much of the work on decision theory has focused on reasoning with quantitative preferences [Fishburn, 1970, Keeney and Raiffa, 1993]. Value theory provides tools for encoding quantitative preferences using a *value function*, which assigns a value to each alternative. If v is a value function on the set $\{A, B, C\}$ of alternatives, then A is said to be preferred to B if and only if $v(A) > v(B)$. *Utility functions* have further been developed as an extension of value functions to deal with settings where the outcomes are uncertain (e.g., by incorporating a probability distribution over the set of outcomes). However, in many settings quantitative preferences may not exist, or it may be more natural to express preferences in qualitative terms [Doyle and Thomason, 1999]. Hence, there is a growing interest on formalisms for representing and reasoning with qualitative preferences [Brafman and Domshlak, 2009] in AI.

An important problem in this context has to do with representing qualitative preferences over a set of alternatives, and reasoning with them to identify the most preferred ones. Typically, the alternatives are described by a set of attributes, and preferences over the alternatives are expressed with respect to their attributes. Representing and reasoning with preferences over multiple attributes is complicated by the fact that there

is an exponential increase in the number of possible outcomes over which preferences have to be specified. This brings the need for formalisms that compactly represent such preferences, and algorithms that reason with them.

Brafman's seminal work [Brafman et al., 2006] attempts to address this need by introducing *preference networks*, which are graphical formalisms for compactly encoding preferences over multiple attributes in the form of: (a) *intra-variable or intra-attribute* preferences specifying preferences over the domains of each of the attributes; (b) the *relative importance* among the attributes. They can capture conditional preferences, i.e., preferences that arise in settings where the preferences over the values of an attribute is dependent on the assignment to one or more other attributes. Preference networks use a graphical representation scheme to encode the above types of preferences, and employ the *ceteris paribus*[1] semantics to reason about the most preferred alternatives. Two of the well studied formalisms in this family are the conditional preference networks (CP-nets) that capture intra-attribute preferences, and tradeoff-enhanced conditional preference networks (TCP-nets) that capture intra-attribute preferences and relative importance over the attributes.

Against the above background, this thesis makes contributions in three related topics:

1. **Dominance testing**: Determining whether an outcome is preferred to another with respect to a set of preferences specified by a TCP-net is computationally hard. This thesis provides formal methods for performing dominance testing efficiently in many practical cases.

2. **Preference Reasoning for Compositional Systems**: In many AI applications such as planning and scheduling, the alternatives over which preferences are computed represent collections of objects rather than simple objects. This thesis develops formalisms to reason with preferences over collections of objects based

[1] A Latin term for 'all else being equal'

on the preferences over the attributes of the objects that make up the collections, and provides algorithms to compute the most preferred collections.

3. **Application to Web Services**: The service oriented computing paradigm offers a powerful approach for software development, where independently developed distributed software components called Web services are assembled together to build more complex applications or compositions. This thesis provides algorithms for automatically identifying, repairing and adapting compositions satisfying a given requirement that are also most preferred with respect to the preferences over the set of attributes.

1.1 Contributions

We describe the specific research challenges, objectives and the contributions made in this thesis in relation to the above topics in Sections 1.1.1, 1.1.2 and 1.1.3 respectively.

1.1.1 Dominance Testing

A problem of fundamental importance in reasoning with qualitative preferences over multiple attributes is *dominance testing*. Dominance testing is the problem of determining whether an outcome (an assignment to all the attributes) is preferred to another with respect to a given set of preferences. In languages such as CP-nets and TCP-nets, testing whether an outcome α is preferred to another outcome β with respect to a set of preferences is equivalent to finding a path in the graph of all possible outcomes from α to β [Brafman et al., 2006].

Polynomial algorithms exist for very special cases of CP-nets (such as when the dependencies of the attributes in a CP-net form a tree structure [Boutilier et al., 2004]).

However, the problem in the general case is hard (PSPACE-complete, [Goldsmith et al., 2008]). Hence, there is a need for practical algorithms for dominance testing.

Experience with other hard problems such as boolean satisfiability (SAT) suggests that many instances of the hard problem may be easily solvable. Specialized data structures and algorithms have thus been developed to obtain practically useful tools (i.e., SAT solvers).

The first main contribution of this thesis is to address the practical need for efficient dominance testing methods for preference languages such as CP-nets and TCP-nets. In particular, we (a) study a restricted preference language for which dominance testing is polynomial time solvable; and (b) employ the state-of-the-art tools in formal methods to make the search for a proof of dominance efficient. This allows us to leverage continuing advances in model checking for more efficient preference reasoning.

1.1.2 Preference Reasoning for Compositional Systems

In many AI applications such as planning and scheduling, the alternatives over which preferences are computed have a *composite* structure, i.e., an alternative represents a collection or a *composition* of objects rather than simple objects. In such settings, typically there are a set of user specified functional requirements that compositions are required to satisfy[2]. Among all the possible compositions that do satisfy the functional requirements, there is often a need to choose compositions that are most preferred with respect to a set of user preferences. In particular, we are interested in the setting where the user preferences are specified with respect to a set of *non-functional* attributes of the objects that make up the composition. We illustrate the above problem using the following example.

Consider the task of designing a program of study (POS) for a Masters student in the

[2]For example, in planning, a valid plan is a collection of actions that satisfies the goal; and in scheduling, a valid schedule is a collection of task-to-resource assignments that respects the precedence constraints.

Computer Science department. The POS consists of a collection of courses chosen from a given repository of available courses spanning across different areas of focus in computer science. Apart from the area of focus, each course also has an assigned instructor and a number of credit hours. A repository of available courses, their areas of focus, their instructors and the number of credit hours are specified in Table 1.1.

Course	Area	Instructor	Credits
CS501	Formal Methods (FM)	Tom	4
CS502	Artificial Intelligence (AI)	Gopal	3
CS503	Formal Methods (FM)	Harry	2
CS504	Artificial Intelligence (AI)	White	3
CS505	Databases (DB)	Bob	4
CS506	Networks (NW)	Bob	2
CS507	Computer Architecture (CA)	White	3
CS508	Software Engineering (SE)	Tom	2
CS509	Theory (TH)	Jane	3
CS510	Theory (TH)	Tom	3

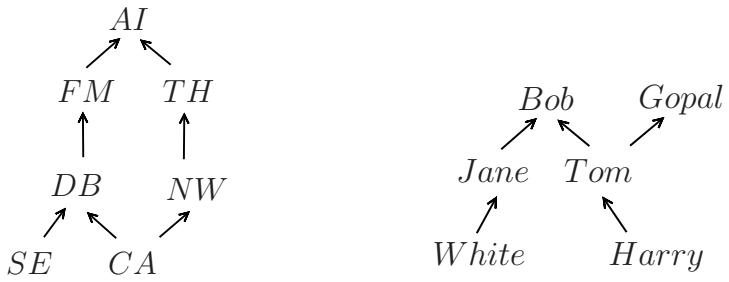
Table 1.1 List of courses the student can choose from

In this example, each POS can be viewed as a composition of courses. The requirements for an acceptable Masters POS (i.e., a feasible composition) are as follows.

- The POS should include at least 15 credits
- The POS should include the two core courses CS509 and CS510
- There should be courses covering at least two breadth areas of study (apart from the area of Theory (TH))

Given the repository of courses (Table 1.1), there may be one or more acceptable programs of study (i.e., feasible compositions). For example:

- $P_1 = CS501 \oplus CS502 \oplus CS503 \oplus CS504 \oplus CS509 \oplus CS510$
- $P_2 = CS501 \oplus CS502 \oplus CS505 \oplus CS506 \oplus CS509 \oplus CS510$



(a) $\succ_A$ (b) $\succ_I$

Figure 1.1 Intra-attribute preferences for Area ($\succ_A$) and Instructor ($\succ_I$).

- $P_3 = CS503 \oplus CS504 \oplus CS507 \oplus CS508 \oplus CS509 \oplus CS510$

Suppose that in addition to the above requirements, a student has some preferences over the course attributes such as the area of focus, the choice of instructors and difficulty level in terms of credit hours. Among several acceptable programs of study, the student may be interested in those programs of study that: (a) satisfy the minimum requirements (see above) for an acceptable POS, and (b) those that are most preferred with respect to his/her preferences specified above. The preferences of a student with respect to the course attributes Area (A) and Instructor (I) are illustrated in Figure 1.1. In addition let us say that the student prefers the POS that have lesser total number of credits (this specifies $\succ_C$). Further, let the relative importance among the attributes A, I and C be $I \rhd A \rhd C$[3], i.e., I is relatively more important than A, which is in turn relatively more important than C. Such preferences can be represented using TCP-nets.

The problems that we try to address in our research, given such a compositional system are:

- Given two programs of study, namely P_i and P_j, determine whether $P_i \succ_d P_j$ or vice versa with respect to the student's preferences;

- Given a repository of courses and an algorithm for computing a set of acceptable

[3]We will use these preferences as a running example.

programs of study, find the most preferred, acceptable programs of study with respect to the above dominance relation.

In this example, the functional requirements correspond to the three conditions[4], all of which must be satisfied for a collection of courses to be an acceptable POS. Area (A), instructor (I) and number of credits (C) constitute the non-functional attributes, and the user preferences over these attributes are given by $\{\succ_A, \succ_I, \succ_C\}$ and $I \rhd A \rhd C$. One can envision similar problems in several other applications, ranging from assembling hardware and software components in an embedded system (such as designing a pace-maker or anti-lock braking system) to putting together a complex piece of legislation (such as the one for reforming health care).

In general, we are interested in the problem of (a) reasoning about preferences over compositions of objects, given the preferences over a set of non-functional attributes describing the objects; and (b) identifying compositions that satisfy the functional requirements of the compositional system, and at the same time are optimal with respect to the stated preferences over the non-functional attributes.

Although there are existing preference formalisms [Boutilier et al., 2004, Brafman et al., 2006] for computing dominance over a simple set of alternatives (given a set of multi-attribute preferences), they are inadequate when it comes to computing dominance over a set of alternatives that are themselves *composite* in nature, as is the case in compositional systems. This is because when dealing with composite alternatives (i.e., compositions), the preference valuation of an alternative or composition is a function of the preference valuations of components that make up the composition. For instance, in the above example, whether one program of study is be preferred to another (with respect to the user specified preferences) or not depends on the subject area, instructors and credit hours of the individual courses that make up the respective programs

[4]A POS must have a minimum of 15 credits; must include the two core courses; and must include courses from at least two breadth areas apart from Theory

of study. Hence, there is a need for developing a preference formalism that provides methods for (a) reasoning about preferences over compositions given user preferences over attributes of the components that make up the compositions; and (b) identifying preferred compositions that satisfy the given functional requirement with respect to user preferences in a compositional system.

We aim to develop a formalism for representing and reasoning with intra-attribute and relative importance preferences over a set of attributes in compositional systems that provides decision procedures for computing dominance over compositions. The formalism should allow users flexibility to define how exactly the attributes of the components in composition influence the overall preference valuation of the composition itself in terms of each of the attributes, and how dominance between two compositions is computed.

We also aim to develop a suite of algorithms for compositional systems that use the above formalism to compare any two compositions; and produce a set of preferred compositions that satisfy a given functional requirement. We also aim to experimentally validate the performance of the developed algorithms, and to compare the algorithms with respect to parameters of interest.

We present a preference formalism for compositional systems that allows users to specify preferences in terms of intra-attribute and relative importance preferences over a set of attributes. We also allow the user to provide a custom *aggregation* function that computes the attributes of a composition in terms of the attributes of its components. We introduce a new dominance relation that compares compositions in terms of their attributes (computed using the user specified aggregation function) with respect to the stated preferences. We analyze some of the key properties of this dominance relation.

We develop a suite of algorithms for compositional systems that identify the set, or subset of the most preferred composition(s) with respect to the user preferences. For this purpose, we assume the existence of a functional composition that guides the search for

compositions that satisfy the given functional requirement. Based on the nature of the functional composition algorithm, we develop various algorithms to identify preferred compositions and compare them both theoretically and experimentally, in terms of the quality of solutions they produce (number of most preferred solutions), and relative performance of the algorithms in practice.

1.1.3 Application of Preference Reasoning for Compositional Systems: Web services

In this thesis we demonstrate the application of preference reasoning techniques to the development and maintenance of a class of software systems, namely service-oriented architectures. Service-oriented computing [Bichler and Lin, 2006, Papazoglou, 2003, Huhns and Singh, 2005] offers a powerful approach to assemble complex distributed applications from independently developed software components in many application domains such as e-Science, e-Business and e-Government. In a service oriented architecture, composite Web services are assembled from atomic Web services such that the overall composite service satisfies the functional and non-functional requirements of the user. Given a set of functional/non-functional requirements and a repository of atomic services, the user seeks a composite service assembled from the components in the repository that satisfies the requirements.

In a service oriented architecture, functional requirements refer to the functionality of the desired composite Web service (also called a goal service). Non-functional requirements refer to aspects such as security, reliability, performance, and cost of the goal service. For example, among the composite services that achieve the desired functionality, a user might prefer a more secure service over a less secure one; or one with a lower cost over one with a higher cost. As in other applications, such preferences may be *quantitative* or *qualitative*. In many settings, a user might need to trade off one non-functional attribute against another (e.g., performance against cost); In others, it

might be useful to assign relative importance to different non-functional attributes (e.g., security being more important than performance).

Barring a few notable exceptions [Zeng et al., 2003, Zeng et al., 2004, Yu and Lin, 2005, Berbner et al., 2006], much of the work on service composition has focused on algorithms for assembly of composite services from functional specifications. Some of the major approaches to service composition based on functional specifications include: AI planning [Pistore et al., 2005b, Traverso and Pistore, 2004, Sirin et al., 2004, Shaparau et al., 2006], labeled transition systems [Pathak et al., 2006b, Pathak et al., 2007, Pathak et al., 2008b], Petri nets [Hamadi and Benatallah, 2003], among others. (The interested reader is referred to [Dustdar and Schreiner, 2005, Pathak et al., 2008a, Pistore et al., 2005a] for surveys). Hence, there is an urgent need for principled methods that incorporate consideration of user-specified preferences with respect to the non-functional attributes, and the relative importance of the different non-functional attributes of the composite Web services.

Successful development and deployment of service oriented software applications relies on effective solution of three inter-related problems: (a) *service composition* [Pathak et al., 2007] - assembling a *composite Web service* (or *service composition*) from a set of component services in a repository that satisfy the given functional requirements; (b) *substitution* [Pathak et al., 2007] - identifying appropriate alternatives to replace failed or unavailable component services in a service composition; and (c) *adaptation* [Chafle et al., 2006, Pathak et al., 2006b] - altering existing composite Web services in response to changes in the functional/non-functional requirements, and/or the repository of available component services.

Solutions to all the above challenges are critical to meeting key goals of a successful service oriented architectures, namely that of supporting business agility and continuity.

Our objective is to develop a set of algorithms for service oriented architectures to *compose, substitute and adapt* composite Web services with respect to user specified

preferences over non-functional attributes of the Web services. We make use of the preference formalism developed as part of our research in pursuance of the objectives of Section 1.1.2.

In this research, we demonstrate the use of preference reasoning techniques and composition algorithms developed as part of Section 1.1.2 for the development and maintenance of composite Web services in service-oriented architectures. In addition, we develop a heuristic search algorithm for identifying the most preferred Web service compositions when one or more of the non-functional attributes of some of the component services in the repository are unknown. We also develop algorithms to address problems related to the lifecycle management of composite Web services in a service-oriented environment, namely, identifying preferred substitutions and adaptations of composite services after identifying and deploying a composite Web service. In particular, we develop different algorithms for dealing with substitution of components in a service composition: (a) single-component substitution; and (b) multi-component substitution, i.e., multiple components are replaced at once in a composite Web service. We also develop two algorithms for adaptation, both of which have the anytime property, i.e., they are guaranteed to produce a sequence of increasingly preferred adaptations to a composite service, given the revised user preferences and/or repository of components. Although these algorithms are tailored to suit the needs of service-oriented architectures, similar techniques can be applied in compositional systems other than Web services as well, such as AI planning, team formation, etc.

CHAPTER 2. Preliminaries

In this chapter, we will introduce some preliminary definitions and key concepts related to qualitative preferences and compositional systems that will be used in the rest of the thesis.

2.1 Properties of Binary Relations

We recall some basic properties and definitions concerning binary relations used in this thesis (see [Fishburn, 1985] for a comprehensive treatment of the same). Let $\succ$ be a binary relation on a set S, i.e., $\succ \subseteq S \times S$. We say that $\succ$ is an equivalence (eq), a (strict) partial order (po), an interval order (io), a weak order (wo) or a total order (to), as defined in Table 2.1.

A total order is also a weak order; a weak order is also an interval order; and an interval order is also a strict partial order.

2.2 Qualitative Preferences

The problem we are interested in this research has to do with representing qualitative preferences over multiple attributes and reasoning with them to find the most preferred among a set of alternatives. Qualitative preferences over a set S are typically represented in the form of a binary relation $\succ_P \subseteq S \times S$ such that for any two elements $u, v \in S$, $u \succ_P v$ if and only if u is preferred to v.

#	Property of relation	Definition	eq	po	io	wo	to
1.	reflexive	$\forall x \in S : x \succ x$	✓				
2.	irreflexive	$\forall x \in S : x \not\succ x$		✓	✓	✓	✓
3.	symmetric	$\forall x, y \in S : x \succ y \Rightarrow y \succ x$	✓				
4.	asymmetric	$\forall x, y \in S : x \succ y \Rightarrow y \not\succ x$		✓	✓	✓	✓
5.	transitive	$\forall x, y, z \in S : x \succ y \wedge y \succ z \Rightarrow x \succ z$	✓	✓	✓	✓	✓
6.	total or complete	$\forall x, y \in S : x \neq y \Rightarrow x \succ y \vee y \succ x$					✓
7.	negatively transitive	$\forall x, y, z \in S : x \succ y \Rightarrow x \succ z \vee z \succ y$				✓	✓
8.	ferrers	$\forall x, y, z, w \in S : (x \succ y \wedge z \succ w) \Rightarrow (x \succ w \vee z \succ y)$			✓	✓	✓

Table 2.1 Properties of binary relations

We focus only on *strict partial order* preference relations, i.e., relations that are both *irreflexive* and *transitive*, because transitivity is a natural property of any *rational* preference relation [Morgenstern and Von Neumann, 1944, French, 1986a, Mas-Colell et al., 1995], and irreflexivity ensures that the preferences are strict.

With respect to any strict partial order preference relation $\succ_P$, we say that two elements u and v are *indifferent*, denoted $u \sim_P v$, whenever $u \not\succ_P v$ and $v \not\succ_P u$. For preference relations $\succ_i$, $\succ'_i$, $\rhd$ and $\succ_d$, we denote the corresponding indifference relation by $\sim_i$, $\sim'_i$, $\sim_\rhd$ and $\sim_d$ respectively. We will drop the subscripts whenever they are understood from the context.

Proposition 1. *For any strict partial order preference relation $\succ_P$, the corresponding indifference relation $\sim_P$ is reflexive and symmetric.*

Proof. $\sim_P$ is reflexive because $u \not\succ_P u$ by the irreflexivity of $\succ_P$; it is symmetric because $u \sim_P v \Leftrightarrow u \not\succ_P v \wedge v \not\succ_P u \Leftrightarrow v \sim_P u$. □

It is important to note that indifference with respect to a strict partial order is not necessarily transitive. For instance, $\succ_X = \{(b, c)\}$ is a strict partial order on the set

$\{a, b, c\}$ with $b \sim_X a$, $a \sim_X c$ but $b \succ_X c$.

2.2.1 Qualitative Preference Languages

Let $V = \{X_i\}$ be a set of preference variables, each with a domain D_i. An outcome $\alpha \in \mathcal{O}$ is a complete assignment to all the preference variables, denoted by the tuple $\alpha := \langle \alpha(X_1), \alpha(X_2), \ldots, \alpha(X_m) \rangle$ such that $\alpha(X_i) \in D_i$ for each $X_i \in V$. The set of all possible outcomes is given by $\mathcal{O} = \prod_{X_i \in V} D_i$.

Brafman's seminal work [Brafman et al., 2006] attempts to address the problem of representing and reasoning with qualitative preferences by introducing *preference networks* that capture: (a) *intra-variable or intra-attribute* preferences specifying preferences over the domains of attributes; (b) the *relative importance* among the attributes. Therefore, we consider a preference language for specifying: (a) conditional intra-variable preferences $\succ_i$ that are strict partial orders (i.e., irreflexive and transitive relations) over D_i; and (b) conditional relative importance preferences $\rhd$ that are strict partial orders over V.

Definition 1 (Intra-variable Preference [Brafman et al., 2006])**.** *Intra-variable preference with respect to a attribute X_i (denoted $\succ_i$), is an irreflexive, transitive and complete binary preference relation on D_i. $\forall u, v \in D_i : u \succ_i v$ iff u is preferred to v with respect to X_i.*

The intra-variable preference relation can be conditional in some settings, and in that case, the preferences over the values of a variable X_i may be influenced by the values assigned to a set of *parent* variables $Pa(X_i)$. The parent variables are said to "influence" the preference over the values of X_i, and the the the variable X_i is said to be *dependent* on the variables in $Pa(X_i)$. Thus, there is a mapping from the set of all possible assignments of the parent variables $Pa(X_i)$ to a set of intra-variable preference relations $\{\succ_i\}$.

A set $X \subseteq \mathcal{X}$ of attributes is said to be *preferentially independent* of the set $Y = \mathcal{X} \backslash X$ of attributes if none of the variables in X are dependent on any of the variables in Y and vice versa. When the two sets contain exactly one variable each, then we simply say that the variables are preferentially independent of each other.

Definition 2 (Relative Importance [Brafman et al., 2006])**.** *Relative importance preference with respect to the non-functional attributes $\mathcal{X}$ (denoted $\rhd$), is an irreflexive, transitive binary preference relation on $\mathcal{X}$ such that $X_i \rhd X_j$ iff X_i is* relatively more important *than X_j.*

As with intra-variable preferences, relative importance may also be conditional. Conditional relative importance is specified as a preference relation on the set of variables conditioned on the values assigned to a set of *selector* variables.

2.2.2 Representing Qualitative Preferences: CP-nets and TCP-nets

Preference networks [Boutilier et al., 2004, Brafman et al., 2006] offer a compact, directed graph representation of intra-variable and relative importance qualitative preferences. The nodes of the graph represent the attributes ($\mathcal{X}$), with the node annotations representing the intra-variable preferences under various conditions specified in terms of the valuations of a set of other parent variables. There are various types of edges representing the conditional intra-variable dependency (between a variable and its parents) and the relative importance preferences among the attributes.

CP-nets [Boutilier et al., 2004] specify conditional intra-variable preferences $\succ_i$ over a set of variables V. Each node in the graph corresponds to a variable $X_i \in V$, and each *dependency* edge (X_i, X_j) in the graph captures the fact that the intra-variable preference $\succ_j$ with respect to variable X_j is dependent (or conditioned) on the valuation of X_i. For any variable X_j, the set of variables $\{X_i : (X_i, X_j)\text{is an edge}\}$ that influence $\succ_j$ are called the *parent* variables, denoted $Pa(X_j)$. Each node X_i in the graph is

associated with a *conditional preference table* (CPT) that maps all possible assignments to the parents $Pa(X_i)$ to a total order over D_i. An *acyclic* CP-net is one that does not contain any dependency cycles.

TCP-nets [Brafman et al., 2006] extend CP-nets by allowing additional edges (X_i, X_j) to be specified, describing the relative importance among variables $(X_i \rhd X_j)$. Each relative importance edge could be either unconditional (directed edge) or conditioned on a set of *selector* variables (analogous to parent variables in the case of intra-variable preferences). Each edge (X_i, X_j) describing conditional relative importance is undirected and is associated with a table (analogous to the CPT) mapping each assignment of the selector variables to either $X_i \rhd X_j$ or vice versa. Figure 2.1 illustrates a TCP-net.

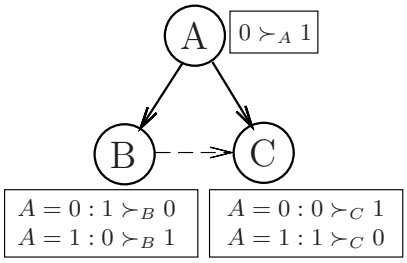


Figure 2.1 Example of a TCP-net with 3 binary variables, namely A, B and C. It encodes the fact that the preference over values of B and C are dependent on the value assigned to the variable A, and that it is relatively more important to have a preferred valuation for the variable B in comparison to having a preferred valuation for C. The intra-attribute preferences are given in the boxes next to the variables.

An extended preference language due to Wilson [Wilson, 2004b, Wilson, 2004a] allows arbitrary preference statements of the form $y : x \succ_i x'[\mathcal{Z}]$ where $X \in \mathcal{X}$, $x, x' \in D_X$, $y \in \mathcal{Y} \subseteq \mathcal{X} \backslash \{X\}$, $\mathcal{Z} \subseteq \mathcal{X} \backslash \mathcal{Y} \backslash \{X\}$.

2.2.3 Reasoning with Qualitative Preferences: *Ceteris Paribus* Semantics

Ceteris paribus is a Latin word that stands for "all else being equal". A formal semantics in terms of the *ceteris paribus* interpretation for preference languages involving conditional intra-variable and relative importance preferences (CP-nets and TCP-nets) was given by Brafman et al. in [Brafman et al., 2006].

A set $X \subseteq \mathcal{X}$ of attributes is *preferentially independent* of the set $Y = \mathcal{X} \setminus X$ of attributes iff for all $x_1, x_2 \in \prod_{X_i \in X} D_i$; $y_1, y_2 \in \prod_{X_i \in Y} D_i$, we have: $x_1 y_1$ is preferred to $x_2 y_1$ (denoted $x_1 y_1 \succ x_2 y_1$, where $\succ$ is the preference relation on complete assignments to all attributes) *iff* $x_1 y_2$ is preferred to $x_2 y_2$ (i.e., $x_1 y_2 \succ x_2 y_2$). We then say that x_1 is preferred to x_2 *ceteris paribus*, i.e., "all else being equal".

Given two preferentially independent attributes X_i and X_j, X_i is *relatively more important* than X_j, denoted by $X_i \rhd X_j$, if

$$\forall w \in \prod_{X_l \in W} D_l, \ \textit{where } W = \mathcal{X} - \{X_i, X_j\}$$
$$\forall x_1, x_2 \in D_i, \forall y_a, y_b \in D_j : x_1 \succ_i x_2 \Rightarrow x_1 y_a w \succ x_2 y_b w$$

Note that $x_1 y_a w \succ x_2 y_b w$ even if $y_b \succ_j y_a$, as any worsening of X_j is preferred to any worsening of X_i.

2.2.4 Dominance Testing

One of the important tasks in reasoning with qualitative intra-variable and relative importance preferences is *dominance* testing, i.e., given two complete assignments to variables or outcomes, determine whether one outcome is preferred to (dominates) the other. There are several ways in which one could define dominance between two outcomes. We list here two definitions based on the ceteris paribus semantics that require the existence of a sequence of outcomes called the *flipping sequence*. A *flip* from one outcome to the next in the sequence represents the change of one or more variables in

the outcome under certain conditions, while the outcomes are equal with respect to all other variables.

Definition 3 (Worsening flipping sequence: adapted from [Brafman et al., 2006])**.** *A sequence of outcomes* $\alpha = \gamma_1, \gamma_2, \cdots \gamma_{n-1}, \gamma_n = \beta$ *such that*

$$\alpha = \gamma_1 \succ^\circ \gamma_2 \succ^\circ \cdots \succ^\circ \gamma_{n-1} \succ^\circ \gamma_n = \beta$$

is a ***worsening flipping sequence*** *with respect to a set of preference statements if and only if, for* $1 \leq i < n$[1]*, either*

1. (*V-flip*) *outcome* γ_i *is different from the outcome* γ_{i+1} *in the value of exactly one variable* X_j*, and* $\gamma_i(X_j) \succ_j \gamma_{i+1}(X_j)$*, or*

2. (*I-flip*) *outcome* γ_i *is different from the outcome* γ_{i+1} *in the value of exactly* ***two*** *variables* X_j *and* X_k*,* $\gamma_i(X_j) \succ_j \gamma_{i+1}(X_j)$*, and* $X_j \rhd X_k$*.*

The V-flips are induced directly by the conditional intra-variable preferences $\succ_i$, and the I-flips are additional flips induced by the relative importance $\rhd$ over variables in conjunction with $\succ_i$. Note that the notion of an I-flip in this definition revises the one presented in [Brafman et al., 2006] in order to accurately reflect the semantics of $\succ^\circ$ [2]. Furthermore, this definition adapts the original definition such that flips are worsening rather than improving. Given a TCP-net N and a pair of outcomes α and β, Brafman et al. have shown that $\alpha \succ^\circ \beta$ with respect to N if and only if there is a worsening flipping sequence with respect to N from α to β.

An issue with dominance testing with respect to $\succ^\circ$ is that the interpretation of relative importance statements is *pairwise*, as illustrated by Wilson in [Wilson, 2004b, Wilson, 2004a]. In this case, I-flips allow *only two* variables to be flipped at a time. On

[1] *We assume the dominance relation is irreflexive and transitive in this thesis.*

[2] Specifically, Definition 3 relaxes the stronger requirement (see Definition 13 in [Brafman et al., 2006]) that "$\gamma_{i+1}(X_j) \succ_j \gamma_i(X_j)$ and $\gamma_i(X_k) \succ_k \gamma_{i+1}(X_k)$" to a weaker requirement that "$\gamma_{i+1}(X_j) \succ_j \gamma_i(X_j)$" – based on a personal communication exchanged by the authors with Ronen Brafman.

the other hand, Wilson's extended semantics [Wilson, 2004b, Wilson, 2004a] (denoted $\succ^w$) defines a worsening I-flip to allow multiple variables to be changed at a time in order to produce a worse outcome. This generalizes the search for *flipping* sequences to a search for *swapping*[3] sequences.

Definition 4 (Worsening flipping sequence with revised I-flip: adapted from [Wilson, 2004b, Wilson, 2004a]). *A sequence of outcomes* $\alpha = \gamma_1, \gamma_2, \cdots \gamma_{n-1}, \gamma_n = \beta$ *such that*

$$\alpha = \gamma_1 \succ^w \gamma_2 \succ^w \cdots \succ^w \gamma_{n-1} \succ^w \gamma_n = \beta$$

is a ***worsening flipping sequence*** *with respect to a set of preference statements if and only if, for* $1 \leq i < n$, *either*

1. (*V-flip*) *as in Definition 3*

2. (*I-flip*) *outcome* γ_i *is different from the outcome* γ_{i+1} *in the value of variables* X_j *and* $X_{k_1}, X_{k_2}, \cdots X_{k_n}$, $\gamma_i(X_j) \succ_j \gamma_{i+1}(X_j)$, *and* $X_j \rhd X_{k_1}, X_j \rhd X_{k_2}, \cdots, X_j \rhd X_{k_n}$.

Given a TCP-net N and a pair of outcomes α and β, according to Wilson's semantics we say that N entails $\alpha \succ^w \beta$ iff there is a worsening flipping sequence with respect to N from α to β.

Dominance testing has been shown to be PSPACE-complete [Goldsmith et al., 2008] for CP-nets and TCP-nets under all the above semantics which are based on the ceteris paribus interpretation of qualitative preferences.

2.3 Compositional Systems

A compositional system consists of a repository of pre-existing components from which we are interested in assembling compositions that satisfy a pre-specified functionality. Formally, a compositional system is a tuple $\langle R, \oplus, \models \rangle$ where:

[3]To simplify the terminology, we will henceforth use the term *flipping* sequence to refer to Wilson's *swapping* sequence as well.

- $R = \{W_1, W_2 \ldots W_r\}$ is a set of available components,
- $\oplus$ denotes a composition operator that functionally aggregates components and encodes all the functional details of the composition. $\oplus$ is a binary operation on components W_i, W_j in the repository that produces a composition $W_i \oplus W_j$.
- $\models$ is a satisfaction relation that evaluates to *true* when a composition satisfies some pre-specified functional properties.

Definition 5 (Compositions, Feasible Compositions and Extensions)**.** *Given a compositional system* $\langle R, \oplus, \models \rangle$*, and a functionality* φ*, a composition* $C = W_{i_1} \oplus W_{i_2} \oplus \ldots W_{i_n}$ *is an arbitrary collection of components* $W_{i_1}, W_{i_2}, \ldots, W_{i_n}$ *s.t.* $\forall j \in [1, n] : W_{i_j} \in R$*.*

i. C *is a* feasible composition *whenever* $C \models \varphi$*;*

ii. C *is a* partial feasible composition *whenever* $\exists W_{i_1} \ldots W_{i_n} \in R : C \oplus W_{i_1} \oplus \ldots \oplus W_{i_n}$ *is a feasible composition; and*

iii. $C \oplus W_i$ *is a* feasible extension *of a partial feasible composition* C *whenever* $C \oplus W_i$ *is a feasible or a partial feasible composition.*

2.3.1 Functional Composition

Given a compositional system $\langle R, \oplus, \models \rangle$ and a functionality φ, an algorithm that produces a set of feasible compositions (satisfying φ) is called a *functional composition algorithm*. The most general class of functional composition algorithms we consider can be treated as *black boxes*, simply returning a set of feasible compositions satisfying φ in a single step. Some other functional composition algorithms proceed by computing the set of feasible extensions of partial feasible compositions incrementally.

Definition 6 (Incremental Functional Composition Algorithm)**.** *A functional composition algorithm is said to be* incremental *if, given an initial partial feasible composition*

C and the desired functionality φ, the algorithm computes the set of feasible extensions to C.

An incremental functional composition algorithm can be used to compute the feasible compositions by recursively invoking the algorithm on the partial feasible compositions it produces starting with the empty composition ($\perp$), and culminating with a set of feasible compositions satisfying φ. In this sense, incremental functional composition algorithms are similar to their "black box" counterparts. However, (as we later show in Chapter 4.2.5) in contrast to their "black box" counterparts, incremental functional composition algorithms can be exploited in the search for the most preferred feasible compositions, by *interleaving* each step of the functional composition algorithm with the optimization of the valuations of non-functional attributes (with respect to the user preferences). This allows us to develop algorithms that can eliminate partial feasible compositions that will lead to less preferred feasible compositions from further consideration early in the search.

Different approaches to functional composition, (e.g., [Traverso and Pistore, 2004, Lago et al., 2002, Baier et al., 2008, Passerone et al., 2002]) differ in terms of (a) the languages used to represent the desired functionality φ and the compositions, and (b) the algorithms used to verify whether a composition C satisfies φ, i.e., $C \models \varphi$. We have intentionally abstracted the details of how functionality φ is represented (e.g., transition systems, logic formulas, plans, etc.) and how a composition is tested for satisfiability ($\models$) against φ, as the primary focus of our work is orthogonal to details of the specific methods used for functional composition.

2.3.2 Preferences over Non-functional Attributes

We now turn to the non-functional aspects of compositional systems. In addition to obtaining functionally feasible compositions, users are often concerned about the non-

functional aspects of the compositions, e.g., the reliability of a composite Web service. In such cases, users seek the *most preferred* compositions among those that are functionally feasible, with respect to a set of non-functional attributes describing the components. In order to compute the most preferred compositions, it is necessary for the user to specify his/her preferences over a set of non-functional attributes $\mathcal{X}$.

Notation. In general, for any relation $\succ_P$, we use the same to denote the transitive closure of the relation as well, and $\not\succ_P$ or $\neg \succ_P$ to denote its complement. The list of notations used in this thesis are given in Table 2.2.

Representing Multi-Attribute Preferences. Following the representation scheme introduced by Boutilier et al. [Boutilier et al., 2004] and Brafman et al. [Brafman et al., 2006], we model the user's preferences with respect to multiple attributes in two forms: (a) intra-attribute preferences with respect to each non-functional attribute in $\mathcal{X}$, and (b) relative importance over all attributes.

Definition 7 (Intra-attribute Preference)**.** *The intra-attribute preference relation, denoted by* $\succ_i$ *is a* strict partial order *(irreflexive and transitive) over the possible valuations of an attribute* $X_i \in \mathcal{X}$. $\forall u, v \in D_i : u \succ_i v$ *iff* u *is preferred to* v *with respect to* X_i.

Definition 8 (Relative Importance)**.** *The relative importance preference relation, denoted by* $\rhd$ *is a* strict partial order *(irreflexive and transitive) over the set of all attributes* $\mathcal{X}$. $\forall X_i, X_j \in \mathcal{X} : X_i \rhd X_j$ *iff* X_i *is* relatively more important *than* X_j.

[4]We will use the terms composition and collection; and component and object interchangeably.

Notation	Meaning
$R = \{W_1 \cdots W_r\}$	Set of components in the repository
$\oplus$	Operation that composes components from R
C_i	A composition or collection [4]of a set of components from R
$\mathcal{C}$	A set $\{C_i\}$ of compositions
$\mathcal{X} = \{X_1 \cdots X_m\}$	Set of non-functional attributes
$\mathcal{D} = \{D_1 \cdots D_m\}$	Set of possible valuations (domains) of attributes in $\mathcal{X}$ respectively
$u_i, v_i, a_i, b_i \cdots \in D_i$	Valuations of an attribute with domain D_i
$A_i, B_i \cdots \subseteq D_i$	Sets of valuations of an attribute with domain D_i
V_{W_i}	Overall valuation of the component W_i with respect to all attributes $\mathcal{X}$
V_{C_i}	Overall valuation of the composition C_i with respect to all attributes $\mathcal{X}$
$V_{W_i}(X_j)$	Valuation of the component W_i with respect to the attribute X_j
$V_{C_i}(X_j)$	Valuation of the composition C_i with respect to the attribute X_j
$\succ_i, \succ_X$	Intra-attribute preference over valuations of X_i or X respectively (user input)
$\rhd$	Relative importance among attributes (user input)
Φ_i	Aggregation function that computes the valuation of a composition with respect to X_i as a function of the valuation of its components
$\succ'_i$	Derived preference relation on the aggregated valuations with respect to X_i
$\succ_d$	Dominance relation that compares two compositions in terms of their aggregated valuations over all attributes
$\Psi_\succ(S)$	The non-dominated set of elements in S with respect to $\succ$
φ	User specified functionality to be satisfied by a feasible composition

Table 2.2 Notation

CHAPTER 3. Efficient Preference Reasoning Techniques

As we have seen in Chapter 1, CP-nets [Boutilier et al., 2004], TCP-nets [Brafman et al., 2006] and their extensions [Wilson, 2004b, Wilson, 2004a] capture qualitative intra-variable preferences and relative importance over a set of variables. Dominance testing for these languages has been shown to be PSPACE-complete [Goldsmith et al., 2008] based on the *ceteris paribus* ("all else being equal") interpretation of preferences.

3.1 Efficient Dominance Testing for Unconditional Preferences

In this section, we attempt to alleviate the complexity of dominance testing in TCP-nets by restricting the preference language to unconditional qualitative preferences. We consider *TUP-nets*, an unconditional fragment of TCP-nets. We introduce a dominance relation for TUP-nets and compare it with its unconditional counterparts for TCP-nets and their variants. We provide a polynomial time algorithm for dominance testing for TUP-nets. TUP-nets are *not* special cases of already known restrictions of CP-/TCP-nets for which polynomial time dominance testing algorithms exist [Boutilier et al., 2004].

3.1.1 A Language for Unconditional Preferences

Let $\mathcal{X} = \{X_i\}$ be a set of variables, each with a domain D_i. An outcome α is a complete assignment to all the variables, denoted by the tuple $\alpha := \langle \alpha(X_1), \alpha(X_2), \ldots, \alpha(X_m) \rangle$

such that $\alpha(X_i) \in D_i$ for each $X_i \in \mathcal{X}$. We consider a preference language $\mathcal{L}_{TUP}$ for specifying: (a) unconditional intra-variable preferences $\succ_i$ that are strict partial orders (i.e., irreflexive and transitive relations) over D_i for each $X_i \in \mathcal{X}$; and (b) unconditional relative importance preferences $\rhd$ that are strict partial orders over $\mathcal{X}$.

Let $\mathcal{L}_{CP}$, $\mathcal{L}_{TCP}$ and $\mathcal{L}_{Ext}$ denote the preference languages of CP-nets, TCP-nets (an extension of CP-nets) and Wilson's extension to TCP-nets respectively. We note that:

- $\mathcal{L}_{TUP}$ allows the expression of relative importance while $\mathcal{L}_{CP}$ does not; and $\mathcal{L}_{CP}$ allows the expression of conditional intra-variable preferences while $\mathcal{L}_{TUP}$ does not.
- $\mathcal{L}_{TUP}$ is less expressive than $\mathcal{L}_{TCP}$ because it does not allow the expression of conditional preferences.
- When restricted to unconditional preferences, $\mathcal{L}_{TCP} = \mathcal{L}_{TUP}$.
- $\mathcal{L}_{Ext}$ is more expressive than $\mathcal{L}_{TCP}$ [Wilson, 2004b, Wilson, 2004a], and hence, $\mathcal{L}_{TUP}$ as well.

3.1.2 Dominance Testing for $\mathcal{L}_{TUP}$

We now provide a polynomial time dominance testing approach for $\mathcal{L}_{TUP}$. We proceed by defining a relation $\succeq_i$ (for each variable $X_i \in \mathcal{X}$) that is derived from $\succ_i$.

Definition 9 ($\succeq_i$)**.** $\forall u, v \in D_i : u \succeq_i v \Leftrightarrow u = v \vee u \succ_i v$

Since $\succ_i$ is a strict partial order (irreflexive and transitive), it can be shown that $\succeq_i$ is a preorder (reflexive and transitive). We next define dominance of α over β with respect to $\{\succ_i\}$ and $\rhd$ using a first order logic formula.

Definition 10 (Dominance for Unconditional Preferences)**.** *Given input preferences* $\{\succ_i\}$ *and* $\rhd$*, and a pair of outcomes* α *and* β*, we say that* α ***dominates*** β *(denoted* $\alpha \succ^{\bullet} \beta$*) iff:*

$$\exists X_i \ : \alpha(X_i) \succ_i \beta(X_i)$$
$$\wedge \forall X_k \ : \ (X_k \rhd X_i \vee X_k \sim_\rhd X_i) \Rightarrow \ \alpha(X_k) \ \succeq_k \ \beta(X_k)$$

where $X_k \sim_\rhd X_i \Leftrightarrow X_k \not\rhd X_i \wedge X_i \not\rhd X_k$*, and* X_i *is called the* ***witness*** *of the relation.*

Intuitively, this definition of dominance of α over β (i.e., $\alpha \succ^\bullet \beta$) requires that α is *preferred* to β with respect to at least one variable, namely the witness. Further, it requires that for all variables that are relatively more important than or indifferent to the witness, α is *either equal to or is preferred to* β. In Example 12, $\alpha \succ^\bullet \beta$, with witness X_1.

3.1.2.1 Properties of Dominance

We now proceed to analyze some properties of $\succ^\bullet$. Specifically, we would like to ensure that $\succ^\bullet$ has two desirable properties of preference relations: irreflexivity and transitivity, which make it a strict partial order. First, it is easy to see that $\succ^\bullet$ is irreflexive, due to the irreflexivity of $\succ_i$ (since it is a partial order).

Proposition 2 (Irreflexivity of $\succ^\bullet$)**.** $\forall \alpha : \alpha \not\succ^\bullet \ \alpha$.

The above proposition ensures that the dominance relation $\succ^\bullet$ is strict over compositions. In other words, no composition is preferred over itself. Regarding transitivity, we observe that $\succ^\bullet$ is not transitive when $\succ_i$ and $\rhd$ are both arbitrary strict partial orders, as illustrated by the following example.

Example 1. *Let* $\mathcal{X} = \{X_1, X_2, X_3, X_4\}$*, and for each* $X_i \in \mathcal{X}$ *:* $D_i = \{a_i, b_i\}$ *with* $a_i \succ_i b_i$*. Suppose that* $X_1 \rhd X_3$ *and* $X_2 \rhd X_4$*. Let* $\alpha = \langle a_1, a_2, b_3, b_4 \rangle$*,* $\beta = \langle b_1, a_2, a_3, b_4 \rangle$ *and* $\gamma = \langle b_1, b_2, a_3, a_4 \rangle$*. Clearly, we have* $\alpha \succ^\bullet \beta$ *(with* X_1 *as witness),* $\beta \succ^\bullet \gamma$ *(with* X_2 *as witness), but there is no witness for* $\alpha \succ^\bullet \gamma$*, i.e.,* $\alpha \not\succ^\bullet \gamma$ *according to Definition 10.*

Because transitivity of preference is a necessary condition for rational choice [Morgenstern and Von Neumann, 1944, French, 1986a], we proceed to investigate the possibility

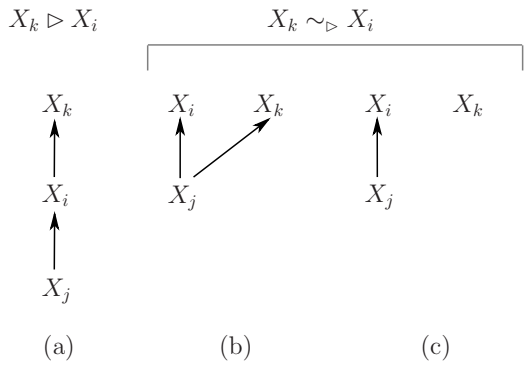


Figure 3.1 $X_i \rhd X_j \wedge (X_k \rhd X_i \vee X_k \sim_\rhd X_i)$

of obtaining such a dominance relation by restricting $\rhd$. In particular, we find that $\succ^\bullet$ is transitive when $\rhd$ is restricted to a special family of strict partial orders, namely *interval orders* as defined below. We prove that such a restriction is necessary and sufficient for the transitivity of $\succ^\bullet$.

Definition 11 (Interval Order)**.** *A binary relation $\boldsymbol{R} \subseteq \mathcal{X} \times \mathcal{X}$ is an interval order iff it is irreflexive and satisfies the* ferrers *axiom [Fishburn, 1985]: for all $X_i, X_j, X_k, X_l \in \mathcal{X}$, we have:*

$$(X_i \,\boldsymbol{R}\, X_j \wedge X_k \,\boldsymbol{R}\, X_l) \Rightarrow (X_i \,\boldsymbol{R}\, X_l \vee X_k \,\boldsymbol{R}\, X_j)$$

We now proceed to establish the transitivity of $\succ^\bullet$ when $\rhd$ is an interval order. We make use of two intermediate propositions 3 and 4 that are needed for the task.

In Proposition 3, we prove that if an attribute X_i is relatively more important than X_j, then X_i is not more important than a third attribute X_k implies that X_j is also not more important than X_k. This will help us prove the transitivity of the dominance relation. Figure 3.1 illustrates the cases that arise.

Proposition 3. $\forall X_i, X_j, X_k : X_i \rhd X_j \Rightarrow$
$\Big((X_k \rhd X_i \vee X_k \sim_\rhd X_i) \Rightarrow (X_k \rhd X_j \vee X_k \sim_\rhd X_j)\Big)$

The proof follows from the fact that $\rhd$ is a partial order.

Proof.

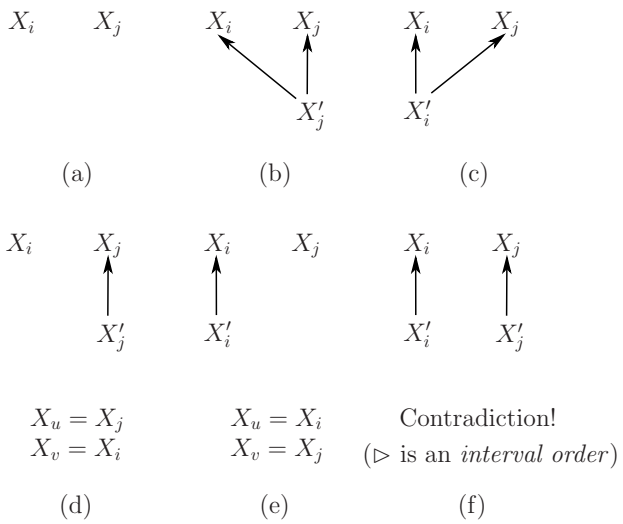


Figure 3.2 $X_i \sim_{\rhd} X_j$

1. $X_i \rhd X_j$ $(Hyp.)$

2. $X_k \rhd X_i \vee X_k \sim_{\rhd} X_i$ $(Hyp.)$ Show $X_k \rhd X_j \vee X_k \sim_{\rhd} X_j$

   (a) $X_k \rhd X_i \Rightarrow X_k \rhd X_j$ By transitivity of $\rhd$ and (1.); see Figure 3.1(a)

   (b) $X_k \sim_{\rhd} X_i \Rightarrow X_k \rhd X_j \vee X_k \sim_{\rhd} X_j$

      i. $X_k \sim_{\rhd} X_i$ $(Hyp.)$

      ii. $(X_k \rhd X_j) \vee (X_j \rhd X_k) \vee (X_k \sim_{\rhd} X_j)$ Always; see Figure 3.1(b,c)

      iii. $X_j \rhd X_k \Rightarrow X_i \rhd X_k$ (1.) Contradiction!

      iv. $X_k \rhd X_j \vee X_k \sim_{\rhd} X_j$ $(2.2.ii., iii.)$

3. $X_i \rhd X_j \Rightarrow \Big((X_k \rhd X_i \vee X_k \sim_{\rhd} X_i) \Rightarrow (X_k \rhd X_j \vee X_k \sim_{\rhd} X_j)\Big)$ $(1., 2.1, 2.2)$ □

Proposition 4 states that if attributes X_i, X_j are such that $X_i \sim_{\rhd} X_j$ then at least one of them, X_u is such that with respect to the other, X_v, there is no attribute X_k that is less important while at the same time $X_k \sim_{\rhd} X_u$. This result is needed to establish the transitivity of the dominance relation.

Proposition 4. *If $\rhd$ is an interval order, then*

$\forall X_i, X_j, u \neq v, X_i \sim_{\rhd} X_j$

$\Rightarrow \exists X_u, X_v \in \{X_i, X_j\}, \nexists X_k : (X_u \sim_{\triangleright} X_k \wedge X_v \triangleright X_k)$.

Proof. Let $X_i \sim_{\triangleright} X_j$, and X_i' and X_j' be variables that are less important than X_i and X_j respectively (if any). Figure 3.2 illustrates all the possible cases that arise. Figure 3.2(a, b, c, d, e) illustrates the cases when *at most* one of X_i' and X_j' exists, and in each case the claim holds trivially. For example, in the cases of Figure 3.2(a, b, c), both $X_u = X_i; X_v = X_j$ and $X_u = X_j; X_v = X_i$ satisfy the implication, and in the cases of Figure 3.2(d, e), the corresponding satisfactory assignments to X_u and X_v are shown in the figure. The final case (Figure 3.2(f)) corresponds to $\triangleright$ not being an interval order (see Definition 11). Hence, the proposition holds in all cases. □

The above proposition reflects the interval order property of the $\triangleright$ relation, and relates to Example 1 in which $\succ^\bullet$ was shown to be intransitive when $\triangleright$ is not an interval order. In fact, if relative importance was defined as a strict partial order instead, it is easy to see that the above proof does not hold. Given that $\alpha \succ^\bullet \beta$ with witness X_i and $\beta \succ^\bullet \gamma$ with witness X_j, the above proposition guarantees that one among X_i and X_j can be chosen as a potential witness for $\alpha \succ^\bullet \gamma$ so that the conditions demonstrated in Example 1 are avoided. Using the propositions 3 and 4, we are now in a position to prove the transitivity of $\succ^\bullet$ in Proposition 5.

Proposition 5 (Transitivity of $\succ^\bullet$)**.** $\forall \alpha, \beta, \gamma$,
$\alpha \succ^\bullet \beta \wedge \beta \succ^\bullet \gamma \Rightarrow \alpha \succ^\bullet \gamma$ *when* $\triangleright$ *is an interval order.*

The proof proceeds by considering all possible relationships between X_i, X_j, the respective attributes that are *witnesses* of the dominance of α over β and β over γ. Lines $5, 6, 7$ in the proof establish the dominance of α over γ in the cases $X_i \triangleright X_j$, $X_j \triangleright X_i$ and $X_i \sim_{\triangleright} X_j$ respectively. In the first two cases, the more important attribute among X_i and X_j is shown to be the witness for $\alpha \succ^\bullet \gamma$ with the help of Proposition 3; and in the last case we make use of Proposition 4 to show that at least one of X_i, X_j is a witness for $\alpha \succ^\bullet \gamma$.

Proof.

1. $\alpha \succ^{\bullet} \beta \quad (Hyp.)$
2. $\beta \succ^{\bullet} \gamma \quad (Hyp.)$
3. $\exists X_i \ : \ \alpha(X_i) \succ'_i \beta(X_i) \quad (1.)$
4. $\exists X_j \ : \ \beta(X_j) \succ'_j \gamma(X_j) \quad (2.)$

   Three cases arise: $X_i \rhd X_j (5.)$, $X_j \rhd X_i (6.)$ and $X_i \sim_{\rhd} X_j (7.)$.
5. $X_i \rhd X_j \Rightarrow \alpha \succ^{\bullet} \gamma$
   1. $X_i \rhd X_j \quad (Hyp.)$
   2. $\beta(X_i) \succeq'_i \gamma(X_i) \quad (2., 5.1.)$
   3. $\alpha(X_i) \succ'_i \gamma(X_i) \quad (3., 5.2.)$
   4. $\forall X_k : (X_k \rhd X_i \vee X_k \sim_{\rhd} X_i) \Rightarrow \alpha(X_k) \succeq'_k \gamma(X_k)$
      i. Let $X_k \rhd X_i \vee X_k \sim_{\rhd} X_i \quad (Hyp.)$
      ii. $\alpha(X_k) \succeq'_k \beta(X_k) \quad (1., 5.4.i.)$
      iii. $X_k \rhd X_j \vee X_k \sim_{\rhd} X_j \quad (5.4.i., Proposition\ 3)$
      iv. $\beta(X_k) \succeq'_k \gamma(X_k) \quad (2., 5.4.iii.)$
      v. $\alpha(X_k) \succeq'_k \gamma(X_k) \quad (5.4.ii., 5.4.iv.)$
   5. $X_i \rhd X_j \Rightarrow \alpha \succ^{\bullet} \gamma \quad (5.1., 5.3., 5.4.)$
6. $X_j \rhd X_i \Rightarrow \alpha \succ^{\bullet} \gamma$
   1. This is true by symmetry of X_i, X_j in the proof of (5.); in this case, it can easily be shown that $\alpha(X_j) \succ'_i \gamma(X_j)$ and $\forall X_k : (X_k \rhd X_j \vee X_k \sim_{\rhd} X_j) \Rightarrow \alpha(X_k) \succeq'_k \gamma(X_k)$.
7. $X_i \sim_{\rhd} X_j \Rightarrow \alpha \succ^{\bullet} \gamma$
   1. $X_i \sim_{\rhd} X_j \quad (Hyp.)$

2. $\exists X_u, X_v \in \{X_i, X_j\} : X_u \neq X_v \wedge \nexists X_k : (X_u \sim_\rhd X_k \wedge X_v \rhd X_k)$
   $(7.1., \textit{Proposition } 4)$
3. Without loss of generality, suppose that $X_u = X_i, X_v = X_j \quad (\textit{Hyp.})$.
4. $\beta(X_i) \succeq'_i \gamma(X_i) \quad (2., 7.1.)$
5. $\alpha(X_i) \succ'_i \gamma(X_i) \quad (3., 7.4.)$
6. $\forall X_k : X_k \rhd X_i \Rightarrow \alpha(X_k) \succeq'_k \gamma(X_k)$.
   i. $X_k \rhd X_i \quad (\textit{Hyp.})$
   ii. $\alpha(X_k) \succeq'_k \beta(X_k) \quad (1., 7.6.i.)$
   iii. $X_k \rhd X_j \vee X_k \sim_\rhd X_j \quad$ Because $X_j \rhd X_k$ Contradicts $(7.1., 7.6.i.)$!
   iv. $\beta(X_k) \succeq'_k \gamma(X_k) \quad (2., 7.6.iii.)$
   v. $\alpha(X_k) \succeq'_k \gamma(X_k) \quad (7.6.ii., 7.6.iv.)$
7. $\forall X_k : X_k \sim_\rhd X_i \Rightarrow \alpha(X_k) \succeq'_k \gamma(X_k)$
   i. $X_k \sim_\rhd X_i \quad (\textit{Hyp.})$
   ii. $\alpha(X_k) \succeq'_k \beta(X_k) \quad (1., 7.7.i.)$
   iii. $X_k \rhd X_j \vee X_k \sim_\rhd X_j \quad$ Because $X_j \rhd X_k$ Contradicts $(7.2., 7.3.)$!
   iv. $\beta(X_k) \succeq'_k \gamma(X_k) \quad (2., 7.7.iii.)$
   v. $\alpha(X_k) \succeq'_k \gamma(X_k) \quad (7.7.ii., 7.7.iv.)$
8. $\forall X_k : X_k \rhd X_i \vee X_k \sim_\rhd X_i \Rightarrow \alpha(X_k) \succeq'_k \gamma(X_k) \quad (7.6., 7.7.)$
9. $X_i \sim_\rhd X_j \Rightarrow \alpha \succ^\bullet \gamma \quad (7.5., 7.8.)$

8. $(X_i \rhd X_j \vee X_j \rhd X_i \vee X_i \sim_\rhd X_j) \Rightarrow \alpha \succ^\bullet \gamma \quad (5., 6., 7.)$
9. $\alpha \succ^\bullet \beta \wedge \beta \succ^\bullet \gamma \Rightarrow \alpha \succ^\bullet \gamma \quad (1., 2., 8.)$ □

From Propositions 2 and 5, we have the first main result of this chapter as follows.

Theorem 1. *$\succ^\bullet$ is a strict partial order when intra-attribute preferences $\succ_i$ are arbitrary strict partial orders and relative importance $\rhd$ is an interval order.* □

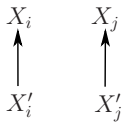


Figure 3.3 A $2 \oplus 2$ substructure, not an Interval Order

The above theorem applies to all partially ordered intra-variable preferences and a wide range of relative importance preferences including total orders, weak orders and semi orders [Fishburn, 1985] which are all interval orders. Having seen in Example 1 that the transitivity of $\succ^\bullet$ does not necessarily hold when $\rhd$ is an arbitrary partial order, a natural question that arises here is whether there is a condition *weaker* than the interval order restriction on $\rhd$ that still makes $\succ^\bullet$ transitive. The answer turns out to be negative, which we show next. We make use of a characterization of interval orders by Fishburn in [Fishburn, 1985], which states that $\rhd$ is an interval order if and only if $2 \oplus 2 \not\subseteq \rhd$, where $2 \oplus 2$ is a relational structure shown in Figure 3.3. In other words, $\rhd$ is an interval order if and only if it has *no restriction of itself* that is isomorphic to the partial order structure shown in Figure 3.3.

Theorem 2. *For arbitrary partially ordered intra-attribute preferences $\succ^\bullet$ is transitive only if relative importance $\rhd$ is an interval order.*

Proof. Assume that $\rhd$ is not an interval order. This is true if and only if $2 \oplus 2 \subseteq \rhd$. However, we showed in Example 1 that in such a case $\succ^\bullet$ is not transitive. Hence, $\succ^\bullet$ is transitive only if relative importance $\rhd$ is an interval order. □

For the special case of $\rhd = \emptyset$, i.e., when there are no relative importance preferences specified, $(X_k \sim_\rhd X_i)$ always holds for any pair of variables $X_i, X_k \in \mathcal{X}$. Hence, dominance testing reduces to:

$$\begin{aligned} \alpha \succ^\bullet \beta \Leftrightarrow \quad & \exists X_i \ : \alpha(X_i) \succ_i \beta(X_i) \wedge \\ & \forall X_k \ : \alpha(X_k) \ \succeq_k \ \beta(X_k) \end{aligned}$$

3.1.2.2 Complexity of Dominance Testing

We now analyze the complexity of evaluating $\alpha \succ^{\bullet} \beta$ in terms of the following parameters: (a) number of variables $m = |\mathcal{X}|$; (b) size of the domains of variables $n = max_{X_i \in \mathcal{X}} |D_i|$; (c) size of the intra-variable preference $k_{int} = max_{X_i \in \mathcal{X}} | \succ_i |$; and (d) size of the relative importance preference relation $k_{rel} = | \rhd |$.

The evaluation of $\succ^{\bullet}$ (Definition 10) involves the evaluation of two clauses for each variable $X_i \in \mathcal{X}$. The first clause checks if $\alpha(X_i) \succ_i \beta(X_i)$. The complexity of evaluating the first clause for each X_i is thus $O(k_{int})$. For each X_i, the second clause checks if the implication holds for each $X_k \in \mathcal{X} - \{X_i\}$. The left hand side of the implication computes $X_k \rhd X_i \vee X_k \sim_{\rhd} X_i$, or equivalently $X_i \not\rhd X_k$ and has complexity $O(k_{rel})$; and the right hand side computes $\alpha(X_k) \succeq_k \beta(X_k)$ and has complexity $O(k_{int})$ (similar to the complexity of the first clause). Hence, the overall complexity of evaluating $\alpha \succ^{\bullet} \beta$ is $O\big(m\big(k_{int} + m(k_{rel} + k_{int})\big)\big)$, or $O\big(m^2(k_{int} + k_{rel})\big)$. Since $k_{rel} = | \rhd |$ (and hence bounded by m^2); and $k_{int} = | \succ_i |$ (and hence bounded by n^2), the complexity can also be expressed as $O\big(m^2(m^4 + n^4)\big)$ in terms of the number of variables and domain size of variables.

3.1.3 Semantics: Relationship Between $\succ^{\circ}$, $\succ^{w}$ & $\succ^{\bullet}$

We investigate the relationship between the semantics $\succ^{\circ}$, $\succ^{\bullet}$, and $\succ^{w}$ for the language $\mathcal{L}_{TUP}$. We show that:

a) $\succ^{\bullet} \subseteq \succ^{w}$

b) $\succ^{\bullet} = \succ^{w}$ when $\rhd$ is an interval order

c) $(\succ^{\bullet})^{\star} = \succ^{w}$, where $(\succ^{\bullet})^{\star}$ is the transitive closure of $\succ^{\bullet}$

d) $\succ^{\bullet} \not\subseteq \succ^{\circ}$ and $\succ^{\circ} \not\subseteq \succ^{\bullet}$ in general; but $\succ^{\circ} \subseteq \succ^{\bullet}$ when $\rhd$ is an interval order

Theorem 3. $\succ^{\bullet} \subseteq \succ^{w}$.

Proof.

We will show that $\alpha \succ^{\bullet} \beta \Rightarrow \alpha \succ^{w} \beta$ for any pair of outcomes α, β.

Suppose that $\alpha \succ^{\bullet} \beta$ with witness X_i (see Definition 10). Define the sets $L = \{X_l \ : \ X_i \rhd X_l\}$, $M = \{X_l \ : \ (X_l \rhd X_i \vee X_l \sim_{\rhd} X_i) \wedge \alpha(X_l) \succ_l \beta(X_l) \wedge X_l \neq X_i\}$, and $M' = \{X_l \ : \ (X_l \rhd X_i \vee X_l \sim_{\rhd} X_i) \wedge \alpha(X_l) = \beta(X_l) \wedge X_l \neq X_i\}$. Clearly, the sets $\{X_i\}$, L, M, M' form a partition of $\mathcal{X}$. Let $X_{t1}, X_{t2}, \ldots X_{tn}$ be an enumeration of M.

We now construct a sequence of outcomes $\gamma_{t1}, \gamma_{t2}, \ldots, \gamma_{tn}$ corresponding to variables $X_{t1}, X_{t2}, \ldots X_{tn}$ as follows. $\gamma_{t1} = \langle \gamma_{t1}(X_1), \gamma_{t1}(X_2), \ldots \gamma_{t1}(X_m) \rangle$ such that $\gamma_{t1}(X_{t1}) = \alpha(X_{t1})$ and $\forall X_j \in \mathcal{X} - \{X_{t1}\} : \gamma_{t1}(X_j) = \beta(X_j)$. Similarly, construct the i^{th} outcome $\gamma_{ti} = \langle \gamma_{ti}(X_1), \gamma_{ti}(X_2), \ldots \gamma_{ti}(X_m) \rangle$ such that $\gamma_{ti}(X_{ti}) = \alpha(X_{ti})$; and $\forall X_j \in \mathcal{X} - \{X_{ti}\} : \gamma_{ti}(X_j) = \gamma_{ti-1}(X_j)$.

Now, we make use of Definition 4 to compare these outcomes with respect to $\succ^{w}$. $\gamma_{t1} \succ^{w} \beta$ because $\gamma_{t1}(X_{t1}) = \alpha(X_{t1}) \succ_{t1} \beta(X_{t1})$ with γ_{t1} and β being equal in all variables other than X_{t1} (V-flip). Also $\gamma_{ti+1} \succ^{w} \gamma_{ti}$ because $\gamma_{ti+1}(X_{ti}) = \alpha(X_{ti}) \succ_{ti} \gamma_{ti}(X_{ti}) = \beta(X_{ti})$, with γ_{ti+1} and γ_{ti} being equal in variables other than X_{ti}. For the last outcome in this sequence $\gamma_{t1}, \ldots, \gamma_{tn}$, we have $\alpha \succ^{w} \gamma_{tn}$ because $\alpha(X_i) \succ_i \gamma_{tn}(X_i) = \beta(X_i)$ and $\forall X_l \in M \cup M' : \alpha(X_l) = \gamma_{tn}(X_l)$, regardless of the assignments to variables $X_j \in L$ (they are less important than X_i) (I-flip). Hence, $\alpha \succ^{w} \gamma_{tn} \succ^{w} \ldots \succ^{w} \gamma_{t1} \succ^{w} \beta$. By the transitivity of $\succ^{w}$ [Wilson, 2004b, Wilson, 2004a], $\alpha \succ^{w} \beta$. □

We now investigate the other side of the inclusion. We recall Example 1 that is relevant in this context.

Example 1 (continued). *Recall that* $\alpha = \langle a_1, a_2, b_3, b_4 \rangle$, $\beta = \langle b_1, a_2, a_3, b_4 \rangle$ *and* $\gamma = \langle b_1, b_2, a_3, a_4 \rangle$ *with* $\alpha \succ^{\bullet} \beta$ *(with* X_1 *as witness),* $\beta \succ^{\bullet} \gamma$ *(with* X_2 *as witness), but* $\alpha \not\succ^{\bullet} \gamma$ *according to Definition 10. However, there exists a sequence of flips from* α *to* γ*, namely* α*,* β*,* γ *according to Definition 4. Hence,* $\alpha \succ^{w} \gamma$.

This example shows that $\succ^w \subseteq \succ^\bullet$ does not hold in general. However, observe that $\succ^\bullet$ holds for each consecutive pair of outcomes in the flipping sequence. Hence, if $\succ^\bullet$ is transitive, we can show that $\succ^w \subseteq \succ^\bullet$.

Theorem 4. *$\succ^w \subseteq \succ^\bullet$ when $\rhd$ is an interval order.*

Proof. Given a set of intra-variable preferences $\{\succ_i\}$ and relative importance $\rhd$, we show that $\alpha \succ^w \beta \;\Rightarrow\; \alpha \succ^\bullet \beta$ when $\rhd$ is an interval order.

Let $\alpha \succ^w \beta$. According to Definition 4, there exists a set of outcomes $\gamma_1, \gamma_2, \cdots, \gamma_{n-1}, \gamma_n$ such that $\alpha = \gamma_1 \succ^w \gamma_2 \succ^w \cdots \succ^w \gamma_{n-1} \succ^w \gamma_n = \beta$ such that for all $1 \leq i < n$ there is either a *V-flip* or an *I-flip* between γ_i and γ_{i+1}.

Case 1: (V-flip) γ_i and γ_{i+1} differ in the value of exactly one variable X_j and $\gamma_i(X_j) \succ_j \gamma_{i+1}(X_j)$. With X_j as the witness, the first clause in the definition of $\gamma_i \succ^\bullet \gamma_{i+1}$ is satisfied ($\gamma_i(X_j) \succ_j \gamma_{i+1}(X_j)$). Because $\gamma_i(X_k) = \gamma_{i+1}(X_k)$ for all $X_k \in \mathcal{X} - \{X_j\}$, we have $\forall X_k \;:\; (X_k \rhd X_j \vee X_k \sim_\rhd X_j) \;\Rightarrow\; \gamma_i(X_k) \succeq_k \gamma_{i+1}(X_k)$ by Definition 9. Therefore, we have $\gamma_i \succ^\bullet \gamma_{i+1}$ with X_j as the witness.

Case 2: (I-flip) γ_i and γ_{i+1} differ in the value of variables X_j and $X_{k_1}, X_{k_2}, \cdots X_{k_l}$, and $X_j \rhd X_{k_1}, X_j \rhd X_{k_2}, \cdots, X_j \rhd X_{k_l}$, such that $\gamma_i(X_j) \succ_j \gamma_{i+1}(X_j)$. With X_j as the witness, the first clause in the definition of $\gamma_i \succ^\bullet \gamma_{i+1}$ is satisfied ($\gamma_i(X_j) \succ_j \gamma_{i+1}(X_j)$).

By Definition 4, $\gamma_i(X_k) = \gamma_{i+1}(X_k)$ for all $X_k \in \mathcal{X} - \{X_j, X_{k_1}, X_{k_2}, \cdots, X_{k_l}\}$. In particular, $\gamma_i(X_k) = \gamma_{i+1}(X_k)$ for all X_k such that $X_k \rhd X_j \vee X_k \sim_\rhd X_j$, which means that $\forall X_k \;:\; (X_k \rhd X_j \vee X_k \sim_\rhd X_j) \;\Rightarrow\; \gamma_i(X_k) \succeq_k \gamma_{i+1}(X_k)$ by Definition 9. Therefore, we have $\gamma_i \succ^\bullet \gamma_{i+1}$ with X_j as the witness by Definition 10[1].

From Cases 1 and 2, $\gamma_i \succ^\bullet \gamma_{i+1}$ for every pair of consecutive outcomes γ_i and γ_{i+1}. Using the fact that $\succ^\bullet$ is transitive when $\rhd$ is an interval order (Theorem 1), we have $\alpha \succ^\bullet \beta$ (by Definition 10) when $\rhd$ is an interval order. Hence, $\succ^w \subseteq \succ^\bullet$ when $\rhd$ is an interval order. □

[1]Note that we do not care how γ_i and γ_{i+1} compare w.r.t. variables $\{X_{k_1}, \cdots, X_{k_l}\}$ that are less important than witness X_j.

The next observation follows from the fact that $\succ^{\bullet}$ holds for each pair of consecutive outcomes in a flipping sequence supporting $\alpha \succ^{w} \beta$.

Observation 1. $(\succ^{\bullet})^{\star} = \succ^{w}$, *where* $(\succ^{\bullet})^{\star}$ *is the transitive closure of* $\succ^{\bullet}$.

Note that this observation holds even when $\rhd$ is not an interval order. However, it does not yield a computationally efficient algorithm for dominance testing in general because computing $(\succ^{\bullet})^{\star}$ is in itself an expensive operation.

We now investigate the relationship between $\succ^{\circ}$ and $\succ^{\bullet}$. In Example 1, α, β, γ forms a flipping sequence from γ to α, resulting in $\alpha \succ^{\circ} \gamma$ (by Brafman et al.'s definition of a flipping sequence). However, $\alpha \not\succ^{\bullet} \gamma$. $\alpha \succ^{\circ} \beta$ implies that there exists a flipping sequence from α to β such that $\succ^{\bullet}$ holds for each pair of consecutive outcomes in the sequence. Hence, it follows that when $\succ^{\bullet}$ is transitive, $\succ^{\circ} \subseteq \succ^{\bullet}$. On the other hand, Example 12 shows that it is possible that $\alpha \succ^{\bullet} \beta$ but $\alpha \not\succ^{\circ} \beta$, and hence, the other side of the inclusion does not hold. This leads us to the following observation.

Observation 2. $\succ^{\bullet} \not\subseteq \succ^{\circ}$ *and* $\succ^{\circ} \not\subseteq \succ^{\bullet}$ *in general; but* $\succ^{\circ} \subseteq \succ^{\bullet}$ *when* $\rhd$ *is an interval order.*

3.1.4 Concluding Remarks

Dominance testing for conditional preference languages such as CP-nets, TCP-nets and their extensions have been shown to be computationally hard [Goldsmith et al., 2008]. Although polynomial time dominance testing algorithms exist for restricted classes of CP-/TCP-nets, there are no known polynomial time dominance testing algorithms for any preference language that allows expression of relative importance of variables. We study one such language, $\mathcal{L}_{TUP}$, an unconditional fragment of $\mathcal{L}_{TCP}$, the language of TCP-nets. Dominance testing in $\mathcal{L}_{TUP}$ amounts to evaluating the satisfiability of a logic formula that can be carried out in polynomial time.

Our results lead to two natural questions that would be interesting to explore: (1) whether dominance testing using a search for flipping sequences can be achieved in polynomial time in the case of unconditional preferences; and (2) whether the existing large body of work on efficient SAT solvers [Zhang and Malik, 2002] can be leveraged to perform efficient dominance testing for other more expressive preference languages.

3.2 Dominance Testing via Model Checking

Dominance testing, the problem of determining whether an outcome is preferred over another, is of fundamental importance in many applications. Hence, there is a need for algorithms and tools for dominance testing with CP-nets and TCP-nets, which are widely studied languages for representing and reasoning with preferences. One of the issues in reasoning with CP-nets and TCP-nets is that dominance testing is PSPACE-complete. Moreover, with the exception of special cases such as CP-nets with tree or polytree structured conditional dependencies [Boutilier et al., 2004, Brafman et al., 2006], dominance testing has been shown to be PSPACE-complete [Goldsmith et al., 2008]. Section 3.1 showed a way to alleviate this hardness by considering a language for representing only unconditional preferences. It also introduced a dominance relation that can be computed in polynomial time, established its properties and compared it with its counterparts in TCP-nets.

In this chapter, we present another approach for dealing with the complexity of dominance testing in TCP-nets – by making use of the state-of-the-art tools in model checking to compute dominance. Although dominance testing is hard, experience with other hard problems such as boolean satisfiability (SAT) suggests that it is often possible to realize acceptable performance in practice for such hard problems, using implementations that take advantage of specialized data structures and algorithms.

We reduce dominance testing in TCP-nets to reachability analysis in a graph of

outcomes. We provide an encoding of TCP-nets in the form of a Kripke structure for CTL. We show how to compute dominance using NuSMV, a model checker for CTL. We present results of experiments that demonstrate the feasibility of our approach to dominance testing.

Hence, this section explores a novel approach to dominance testing that leverages the state-of-the-art techniques in *model checking* [Clarke et al., 1986, Queille and Sifakis, 1982]. Our approach reduces dominance testing to reachability analysis in a graph of outcomes. Specifically, we formalize the ceteris paribus semantics of preferences in terms of a direct and succinct representation of preference semantics using *Kripke structures* [Clarke et al., 2000] that encode preferences over outcomes as reachability within a graph of outcomes. In this setting, we reduce dominance testing to the satisfiability of corresponding temporal formulas in the model. We provide a translation from TCP-nets to the Kripke model specification language of a widely used model checker NuSMV [Cimatti et al., 2002]. We demonstrate how a *proof* of dominance can be automatically generated whenever the dominance holds. This approach allows us to take advantage of the state-of-the-art model checkers that provide optimized computation of dominance using specialized data structures and algorithms. We present results of experiments that demonstrate the feasibility of this approach to dominance testing: Dominance queries over preference specifications involving 20 or more variables are answered within a few seconds. While the discussion here is restricted to TCP-nets, our approach to dominance testing via model checking can be used for any preference formalism whose semantics is given in terms of properties over a graph of outcomes.

3.2.1 Dominance Testing via Model Checking

We now proceed to describe our approach to dominance testing using model checking. Ceteris paribus semantics induces a graph, the *induced preference graph* [Boutilier et al., 2004, Brafman et al., 2006]; and an outcome α is said to dominate another outcome β

if there exists a path consisting of successively worsening outcomes in this graph from α to β. The key observation behind our approach is that dominance of α over β is given in terms of the reachability of the worse outcome (β) from the preferred outcome (α) in the *induced preference graph* [Boutilier et al., 2004, Brafman et al., 2006] that captures the preference semantics.

Definition 12. *Given a TCP-net N over a set of variables V, the* induced preference graph $\delta(N) = G(A, E)$ *is constructed as follows. The nodes A correspond to the set of all possible outcomes, i.e., complete assignments to all variables in V, and each directed edge $(\alpha, \beta) \in E$ corresponds to either a V-flip or an I-flip as dictated by the chosen semantics (Definitions 3, 4).*

An outcome α dominates β with respect to N if and only if the node corresponding to β in $\delta(N)$ is reachable from α. Note that $\delta(N)$ is guaranteed to be acyclic because it represents an irreflexive and transitive dominance relation.

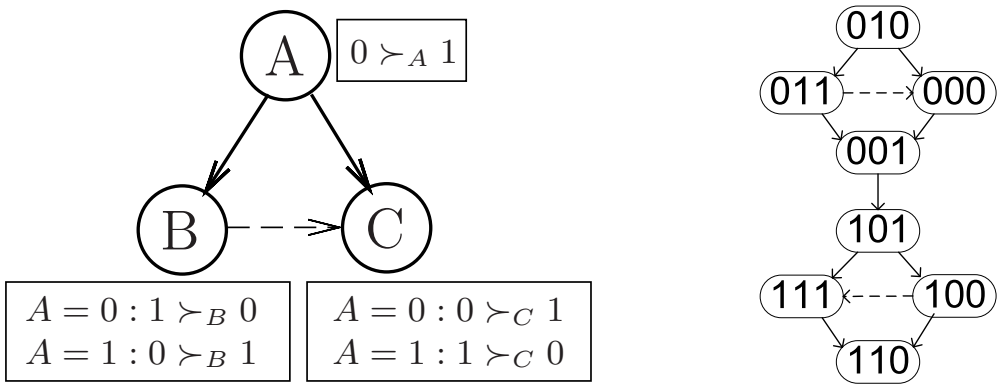


Figure 3.4 (a) TCP-net N; (b) Transitive Reduction of $\delta(N)$

Example. Consider a TCP-net N of three binary variables, namely $\{A, B, C\}$ as shown in Figure 3.4(a). $\succ_B$ and $\succ_C$ depend on the valuations A (solid directed edges), and the nodes are annotated with the respective CPTs. $B \rhd C$ is denoted by a dotted edge from B to C. Figure 3.4(b) shows the transitive reduction of the corresponding induced preference graph $\delta(N)$.

The above formulation of dominance testing in a TCP-net N in terms of verifying reachability properties in the corresponding induced preference graph $\delta(N)$ allows us to take advantage of the state-of-the-art approaches to model checking. This approach to dominance testing involves addressing two questions: (a) How to encode the induced preference graph $\delta(N)$ as an input graph to a model checker (we use NuSMV [Cimatti et al., 2002]), and (b) How to express a query regarding the dominance of an outcome (α) with respect to another (β) in the form of a test of reachability of β from α in the corresponding graph.

The preference variables of the TCP-net are mapped to the state variables of the model in a model checker. The V-flips and the I-flips (Definitions 3 and 4) are directly encoded as transitions in the *Kripke* structures for the language of the model checker. This ensures that the state space explored by the model checker corresponds to $\delta(N)$.

Dominance queries over the TCP-nets are then modeled as temporal logic properties (in CTL [Clarke et al., 2000]) over the state space of the model. This allows us to take advantage of all the specialized data structures (e.g., BDDs) and algorithms available in the model checking engine to efficiently verify the satisfiability of the corresponding temporal logic properties. In order to check whether an outcome α dominates the outcome β, we query the model checker with a CTL formula φ such that there is a model of φ if and only if α dominates β. If the dominance does hold, then the model checker can be used to obtain a proof of dominance, i.e., a worsening flipping sequence from α to β.

3.2.2 Kripke Structure Encoding of TCP-net Preferences

We now proceed to describe how TCP-net[2] preferences can be encoded in a *Kripke* structure [Clarke et al., 2000].

[2]To simplify the presentation, we will restrict our discussion to TCP-nets over binary variables, although our approach can be extended to variables with other domains as well.

Definition 13 (Kripke Structure)**.** *A* Kripke *structure is a tuple* $\langle S, S_0, T, L\rangle$ *where* S *is a set of states described by the valuations of a set of propositional variables* P*,* $S_0 \subseteq S$ *is a set of* initial *states,* $T \subseteq S \times S$ *is a transition relation inducing directed edges between states such that* $\forall s \in S : \exists s' \in S : (s, s') \in T$*, and* $L : S \rightarrow 2^P$ *is a labeling function such that* $\forall s \in S : L(s)$ *is the set of propositions that are* true *in* s*.*

Given a TCP-net N over a set $V = \{X_1, \ldots X_n\}$ of variables, a Kripke structure K_N corresponding to the induced preference graph $\delta(N)$ can be constructed as follows.

1. The states S are defined by the valuations of propositions $P = V \cup \{h_i | X_i \in V\}$, where each h_i is a binary variable indicating whether or not the value of X_i can change in a transition,

$$h_i = \begin{cases} 0 & \text{if value of } X_i \textbf{ must not} \text{ change in a} \\ & \text{transition in the Kripke structure } K_N \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

   The change variables h_i defined above will be used, as will be shown later, to construct the state space S of the Kripke structure K_N that encodes the induced preference graph $\delta(N)$. Note that each outcome α in $\delta(N)$ corresponds to a set $S^\alpha = \{s | s_{\downarrow V} = \alpha\}$ of states, where $s_{\downarrow V}$ denotes the *projection* of a state s described by P onto the set of variables $V \subseteq P$. By this construction, the various states in S^α differ precisely in the valuations of the change variables, and since there are $|V|$ change variables, $|S^\alpha| = 2^{|V|}$. The start state(s) S_0 of K_N are specified based on the dominance query (as described later); and the labeling function L is defined such that for any state $s \in S$, $L(s)$ corresponds to the set of variables that are 'true' in s.

2. The transition relation T is defined as follows, with $s(X_i)$ and $s'(X_i)$ denoting the valuation of the corresponding variable X_i in states s and s' respectively. For any

two states $s, s' \in S$, define $(s, s') \in T$ (denoted $s \rightarrow s'$) by the rules:

i. (V-flip)

$$s \rightarrow s' \Leftarrow \begin{cases} \exists X_i \in V : s(h_i) = 1 \ \wedge \ s(X_i) \succ_i s'(X_i) \\ \quad \wedge \forall X_j \in V \setminus \{X_i\} : s(h_j) = 0 \wedge \\ \qquad s(X_j) = s'(X_j) \end{cases}$$

ii. (I-flip)

$$s \rightarrow s' \Leftarrow \begin{cases} \exists X_i \in V : s(h_i) = 1 \ \wedge \ s(X_i) \succ_i s'(X_i) \\ \quad \wedge \exists W \subseteq V \setminus \{X_i\} : \forall X_j \in W : X_i \rhd X_j \\ \qquad \wedge \forall X_k \in V \setminus (W \cup \{X_i\}) : \\ \qquad\quad s(h_k) = 0 \ \wedge \ s(X_k) = s'(X_k) \end{cases}$$

iii. $s \rightarrow s' \Leftarrow \forall X_i \in V : s(X_i) = s'(X_i)$

In the above encoding of the Kripke structure, transition rules $2(i) - (iii)$ are exhaustive, thus satisfying the requirement that in a Kripke structure all states should have outgoing transitions. Transitions effected through the rules $2(i)$ and $2(ii)$ correspond to valid worsening V-flips and I-flips respectively according to Wilson's semantics[3] (Definition 4). Therefore, all possible edges E corresponding to V-flips and I-flips of the induced preference graph $\delta(N) = G(A, E)$ are captured by the above transition relation. $2(iii)$ allows only transitions from a state s to those states s' that agree with s on all variables in V. We now establish the main theorem which forms the basis for our approach to dominance testing via model checking.

Remark. The definition of a V- or an I- flip from α to β (Definitions 3 and 4) requires the equality of α and β with respect to some of the variables in V. Since NuSMV does not allow the specification of a transition by constraining the destination state variables,

[3]Brafman et al. semantics (Definition 3) can be similarly encoded with minor changes to the definition of the transition relation $2(ii)$ in the Kripke structure.

we use h_i to control the allowed changes to each variable $X_i \in V$. Observe that the variables h_i are allowed to take any value in the destination state of any transition (see rules $2(i) - (iii)$). This allows the model checker to explore all possible V-/I-flips from any given outcome.

Theorem 5. *Given a TCP-net N, and the corresponding Kripke structure $K_N = \langle S, S_0, T, L\rangle$ (constructed from $\delta(N) = G(A, E)$ as described above),*

1. $\forall \alpha, \beta : (\alpha, \beta) \in E \Rightarrow \exists s \rightarrow s' : s_{\downarrow V} = \alpha \wedge s'_{\downarrow V} = \beta$

2. $\forall s, s' \in S : s \rightarrow s' \wedge s_{\downarrow V} \neq s'_{\downarrow V} \Rightarrow \exists (\alpha, \beta) \in E$

Proof. For the first part, let $(\alpha, \beta) \in E$. Since $\delta(N)$ is a cycle-free graph over distinct outcomes, $\alpha \neq \beta$. Further, $(\alpha, \beta) \in E$ requires the existence of either a V-flip or I-flip from α to β by Definitions 4 and 12. By construction of the Kripke structure, there exist sets S^α and S^β of states such that $\forall s^\alpha \in S^\alpha : s^\alpha_{\downarrow V} = \alpha$ and $\forall s^\beta \in S^\beta : s^\beta_{\downarrow V} = \beta$ respectively. Therefore, $\forall s^\alpha \in S^\alpha, s^\beta \in S^\beta : (s^\alpha_{\downarrow V}, s^\beta_{\downarrow V}) \in E$. By the definition of the valuations of the change variables in Equation (1) and the transition rules $2(i)$ and $2(ii)$, it follows that $\exists s^\alpha \in S^\alpha, s^\beta \in S^\beta : s^\alpha \rightarrow s^\beta$.

For the second part, let $s, s' \in S : s \rightarrow s' \wedge s_{\downarrow V} \neq s'_{\downarrow V}$. Since $s \neq s'$, $2(iii)$ is not applicable, and hence, it must be the case that the conditions in the right hand side of $2(i)$ or $2(ii)$ is satisfied. This in turn implies that the transition $s \rightarrow s'$ is due to a V-flip or an I-flip, i.e., $(s_{\downarrow V}, s'_{\downarrow V}) \in E$. □

3.2.2.1 Encoding K_N in NuSMV

For the TCP-net N specified in Figure 3.4, the encoding of the corresponding Kripke structure $\delta(N)$ that is provided as input to the NuSMV model checker [Cimatti et al., 2002] is shown in Figure 3.5.

The VAR construct declares the binary preference variables (a, b, c) and the corresponding change variables (ha, hb, hc), and ASSIGN defines the transition rules for each variable in the form of the guards and corresponding next state valuations (as per rules $2(i) - (iii)$ in our construction). For example, the V-flips corresponding to variable b when $\mathtt{a} = 0$ is encoded as follows: "**if** $\mathtt{b} = 1 \& \mathtt{a} = 0 \& \mathtt{ha} = 0 \& \mathtt{hb} = 1 \& \mathtt{hc} = 0$ **then** $\mathtt{b} = 0$ in the next state". The I-flips induced by $\mathtt{b} \rhd \mathtt{c}$ are specified by guarded transitions allowing (1) b's valuation to change regardless of whether c's valuation changes or not, and (2) c's valuation to change in a transition whenever b's valuation changes.

3.2.2.2 Computing Dominance

Given a Kripke structure K_N that encodes the induced preference graph of a TCP-net N, determining whether α dominates β in N can be reduced to verifying appropriate temporal properties in CTL (see [Clarke et al., 2000]). Specifically, the CTL formula $\varphi_\alpha \rightarrow \mathtt{EF}\varphi_\beta$ is used to check whether α dominates β. In the formula, φ_α and φ_β are conjunctions of the assignments to the variables in V in α and β respectively. A state in the Kripke structure is said to satisfy the above formula if and only if when the state satisfies φ_α (i.e., valuations of variables of V in that state correspond to those in α), there exists a path or a sequence of transitions $s^\alpha = s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_n = s^\beta$ (s.t. $s^\alpha_{\downarrow V} = \alpha$ and $s^\beta_{\downarrow V} = \beta$) such that $n > 1$. In short, a state in the Kripke structure K_N corresponding to $\delta(N)$ satisfies the above CTL formula if and only if α dominates β with respect to N (Theorem 5). We will use the model checker NuSMV to verify the satisfiability of a CTL formula $\varphi_\alpha \rightarrow \mathtt{EF}\varphi_\beta$.

Example. For the TCP-net N in Figure 3.4, the dominance of $\alpha = \langle \mathtt{a} = 0, \mathtt{b} = 1, \mathtt{c} = 1 \rangle$ over $\beta = \langle \mathtt{a} = 1, \mathtt{b} = 0, \mathtt{c} = 0 \rangle$ corresponds to the satisfiability of the CTL formula $\varphi : (\mathtt{a} = 0 \& \mathtt{b} = 1 \& \mathtt{c} = 1 \rightarrow \mathtt{EF}(\mathtt{a} = 1 \& \mathtt{b} = 0 \& \mathtt{c} = 0))$. Note that NuSMV asserts that φ is verified only if *every* initial state satisfies φ. Therefore, we initialize X_i to $\alpha(X_i)$

to restrict the start states to S^{α} in the encoded Kripke structure. We also initialize all the change variables h_i to 0, so that transitions corresponding to all possible V-flips and I-flips from α are explored by the model checker. In NuSMV, the satisfiability of φ can be verified by the specification $\texttt{SPEC}\varphi$, and the verification returns 'true' in our example, thereby establishing that the outcome $\alpha = \langle \mathtt{a} = 0, \mathtt{b} = 1, \mathtt{c} = 1 \rangle$ dominates $\beta = \langle \mathtt{a} = 1, \mathtt{b} = 0, \mathtt{c} = 0 \rangle$.

3.2.2.3 Extracting a Proof of Dominance

We can use the NuSMV model checker to obtain a proof that an outcome α dominates another outcome β (i.e., a worsening flipping sequence from α to β) as follows. Suppose α dominates β. This implies that the CTL formula $\varphi : \varphi_{\alpha} \rightarrow \texttt{EF}\varphi_{\beta}$ holds. Hence, in this case if we provide the formula $\neg\varphi$ (i.e., $\neg(\varphi_{\alpha} \rightarrow \texttt{EF}\varphi_{\beta})$) as input to the model checker, the model checker will return 'false', and provide us with the sequence of states (as below) corresponding to the worsening flipping sequence from α to β.

In our example, since $\alpha = \langle \mathtt{a} = 0, \mathtt{b} = 1, \mathtt{c} = 1 \rangle$ dominates $\beta = \langle \mathtt{a} = 1, \mathtt{b} = 0, \mathtt{c} = 0 \rangle$, when input the formula

$$\texttt{SPEC} \ ! \ (\mathtt{a} = 0 \ \& \ \mathtt{b} = 1 \ \& \ \mathtt{c} = 1 \ \texttt{->} \ \texttt{EF} \ (\mathtt{a} = 1 \ \& \ \mathtt{b} = 0 \ \& \ \mathtt{c} = 0))$$

NuSMV returns the sequence: $(0, 1, 1) \rightarrow (0, 0, 0) \rightarrow (1, 0, 0)$ as shown below.

```
-> State: 1.1 <-          -> State: 1.2 <-          ha = 1
  a = 0                     hb = 1                  hb = 0
  b = 1                     hc = 1                  hc = 0
  c = 1
  ha = 0                  -> State: 1.3 <-        -> State: 1.4 <-
  hb = 0                    b = 0                   a = 1
  hc = 0                    c = 0                   ha = 0
```

In the above, the transition from state 1.1 to 1.2 is effected by transition rule 2(iii); that from state 1.2 to 1.3 by rule 2(ii); and that from state 1.3 to 1.4 by rule 2(i).

3.2.3 Summary and Discussion

We have described, to the best of our knowledge, the first practical solution to the problem of determining whether an outcome dominates another with respect to a set of qualitative preferences. Our approach relies on a reduction of the dominance testing problem to reachability analysis in a graph of outcomes. We have provided an encoding of TCP-nets in the form of a Kripke structure for CTL.

We have shown how to: (a) directly and succinctly encode preference semantics as a Kripke structure; (b) compute dominance by verifying CTL temporal properties against this Kripke structure; and (c) generate a proof of dominance. We have shown how to compute dominance using NuSMV, a model checker for CTL. The results of our experiments demonstrate the feasibility of this approach to dominance testing. This approach to dominance testing via model checking allows us to take advantage of continuing advances in model-checking.

Although our treatment focused on acyclic CP-nets and TCP-nets, our approach can be applied to any preference language for which the semantics is given in terms of the satisfiability of graph properties (including GCP-nets, cyclic CP-nets and the language due to Wilson). Our approach can also be used for reasoning tasks other than dominance testing such as finding whether a given outcome is the *least* (*or most*) *preferred* among all the outcomes.

In our experiments we have reported running times for dominance testing that are intended merely to demonstrate the feasibility of our approach. The running times are averages taken over multiple dominance queries over sets of randomly generated preference networks. The running time could depend on many factors such as the structure of the preference network, the number of preference variables and the CPT size, in addition to the dominance query itself. Hence, it remains to be studied how the running times and memory usage of our solution approach are affected by such factors.

Although we have used the NuSMV model checker in our implementation, any model checker that accepts a Kripke structure as input can be used to realize our approach to dominance testing. Hence, it should be possible to take advantage of specialized techniques that have recently been developed to improve the performance of model checkers [Kahlon et al., 2009, Cook and Sharygina, 2005, Ciardo et al., 2001, Wang, 2000, Biere et al., 1999].

```
MODULE main
VAR a:{0,1}; b:{0,1}; c:{0,1};
     ha:{0,1};  hb:{0,1};  hc:{0,1};
ASSIGN --init(a):=1; init(b):=1; init(c):=1;
  init( ha):=0; init( hb):=0; init( hc):=0;
  next(a) :=
     case    -- conditional preferences:
        a=0 &  hc=0 &  hb=0 &  ha=1 : 1;
        1 : a;
     esac;
  next(b) :=
     case    -- conditional preferences:
        b=1 & a=0 &  hc=0 &  hb=1 &  ha=0 : 0;
        b=0 & a=1 &  hc=0 &  hb=1 &  ha=0 : 1;
             -- relative importance: b imp. than c
        b=1 & a=0 &  hb=1 &  ha=0 : 0;
        b=1 & a=0 &  hb=0 &  ha=0 : 1;
        b=0 & a=1 &  hb=1 &  ha=0 : 1;
        b=0 & a=1 &  hb=0 &  ha=0 : 0;
        1 : b;
     esac;
  next(c) :=
     case    -- conditional preferences:
        c=0 & a=0 &  hc=1 &  hb=0 &  ha=0 : 1;
        c=1 & a=1 &  hc=1 &  hb=0 &  ha=0 : 0;
             -- relative importance: c less imp. than b
        ((b=1 & a=0) | (b=0 & a=1))
                          &  hb=1 &  ha=0 &  hc=1 : !c;
        ((b=1 & a=0) | (b=0 & a=1))
                          &  hb=1 &  ha=0 &  hc=0 : c;
        1 : c;
     esac;
```

Figure 3.5 Listing of Kripke encoding in NuSMV

CHAPTER 4. Preference Reasoning for Compositional Systems: Theory & Algorithms

In this chapter, we turn our attention to the problem of reasoning with multi-attribute preferences for compositional systems. We develop a formalism for representing intra-attribute and relative importance preferences over multiple attributes of the components, and reasoning with them to compare compositions using a dominance preference relation. Further, we develop a set of algorithms to identify the most preferred compositions, given a functional specification (functional requirement), user preferences over the attributes of the components (non-functional requirements), a repository of available components, and a functional composition algorithm.

4.1 Preference Formalism

Given a compositional system with a repository of components described by attributes $\mathcal{X}$ and preferences $(\{\succ_i\}, \rhd)$ over them, we are interested in reasoning about preferences over different compositions. Note that based on preferences $\{\succ_i\}$ and $\rhd$, one can make use of existing formalisms such as TCP-nets [Brafman et al., 2006] to select the most preferred components. However, the problem of comparing compositions (as opposed to comparing components) with respect to the attribute preferences is complicated by the fact that the valuation of a composition is a function of the valuations of its components. Our approach to developing the preference formalism is as follows.

First, given a composition and the valuations of its components with respect to the

attributes, we obtain the *aggregated valuation* of the composition with respect to each attribute as a function of the valuations of its components. Next, we define preference relations to compare the aggregated valuations of two compositions with respect to each attribute. Finally, we build a *dominance* preference relation $\succ_d$ that *qualitatively* compares any two compositions with respect to their aggregated valuations across all attributes.

4.1.1 Aggregating Attribute Valuations across Components

In order to reason about preferences over compositions, it is necessary to obtain the valuation of a composition with respect to each attribute X_i *in terms of* its components, using some *aggregation* function Φ_i. There are several ways to aggregate the preference valuations attribute-wise across components in a composition. The aggregation function Φ_i computes the valuation of a composition with respect to an attribute X_i as a function of the valuations of its components.

Remark. In the compositional systems considered here, we assume that the valuation of a composition with respect to its attributes is a function of only the valuations of its components. In other words, if $C = W_1 \oplus W_2 \oplus \ldots \oplus W_n$, then V_C is a function of only $\{V_{W_1}, V_{W_2}, \ldots, V_{W_n}\}$. However, in the most general setting, the aggregation functions Φ_i need to take into account, in addition to the valuations of the components themselves, the structural or functional details of a composition encoded by $\oplus$ (e.g., the reliability of a Web service composition depends on whether the service components are composed in a series or parallel structure; the power rating of an embedded system depends on the details of the circuitry in addition to the power ratings of the components themselves).

Definition 14 (Aggregation Function)**.** *The aggregation function on the set of subsets of possible valuations (D_i) of attribute X_i is*

$$\Phi_i : \mathscr{P}(D_i) \longrightarrow \mathscr{F}(X_i)$$

where $\mathscr{F}(X_i)$ denotes the range of the aggregation function.

Aggregation with respect to an attribute X_i amounts to devising an appropriate aggregation function Φ_i that computes the valuation of a composition in terms of the valuations of its components for X_i. The range $\mathscr{F}(X_i)$ of Φ_i depends on the choice of aggregation function. Some examples of aggregation functions are given below.

1. *Summation.* This is applicable in cases where an attribute is real-valued and represents some kind of cost. For example, the cost of a shopping cart is the sum of the costs of the individual items it includes. In our running example, the total number of credits in a POS consisting of a set of courses is the sum of the credits of all the courses it includes. That is, if S is the set of credit hours (valuations of the courses with respect to the attribute C) of courses in a POS, then

$$\Phi_C(S) := \{\Sigma_{s \in S} s\}$$

2. *Minimum/Maximum.* Here, the valuation of a composition with respect to an attribute is the worst, i.e., the minimum among the valuations of its components. This type of aggregation is a natural one to consider while composing embedded systems or Web services. For example, when putting together several components in an embedded system, the system is only as secure (or safe) as its least secure (or safe) component.

$$\Phi_i(S) := \{min_{s \in S} s\}$$

Analogously, one could chose as the valuation of the composition the maximum (best) among the valuations of its components. Such an aggregation function may be useful in applications such as parallel job scheduling, where the maximum response time is used to measure the quality of a schedule.

3. *Best/Worst Frontier.* In some settings, it is possible that the intra-attribute preference over the values of an attribute is a partial order (not necessarily a ranking or a total order). Hence, it may not be possible to compute the valuation of a composition as the best or worst among the valuations of its components because a unique maximum or minimum may not exist. For example, it may be useful to compute the valuation of a composition as the *minimal set* of valuations among the valuations of its components, which we call the *worst frontier*. The *worst frontier* represents the worst possible valuations of an attribute X_i with respect to $\succ_i$, i.e., the minimal set[1] among the set of valuations of the components in a composition.

Definition 15 (Aggregation using Worst Frontier)**.** *Given a set S of valuations of an attribute X_i, the worst frontier aggregation function is defined by*

$$\forall S \subseteq D_i : \Phi_i(S) := \{v : v \in S \wedge \nexists u \in S : v \succ_i u\}$$

In our running example (see Section 1.1.2), the user would like to avoid courses not in his interest area and professors whom he is not comfortable with. That is, a program of study is considered only as good as the least interesting areas of study it covers, and the set of professors he is least comfortable with. Hence, worst frontier aggregation function is chosen for the breadth area and instructor attributes.

Example 2. *The "worst possible" valuations of attributes A and I for the programs of study (compositions) P_1, P_2 and P_3 with respect to $\succ_A$ and $\succ_I$ are $(\{FM, TH\}, \{White, Harry\})$, $(\{DB, NW\}, \{Jane, Tom\})$ and $(\{CA, SE\}, \{Harry, White\})$ respectively. These sets correspond to the "worst frontiers" of the respective attributes. The different areas of focus covered in the POS P_2 are $\{FM, AI, DB, NW, TH\}$, and the worst frontier*

[1]Note that if $\succ_i$ is a total order, then worst frontier represents the minimum or lowest element in the set with respect to the total order.

of this set is $\Phi_A(\{FM, AI, DB, NW, TH\}) = \{DB, NW\}$ *because* $AI \succ_A DB, FM \succ_A DB, TH \succ_A NW$. *Similarly the set of instructors in* P_2 *are* $\{Tom, Gopal, Bob, Jane\}$, *and hence we have* $\Phi_I(\{Tom, Gopal, Bob, Jane\}) = \{Jane, Tom\}$ *because* $Bob \succ_I Jane$ *and* $Gopal \succ_I Tom$. *For attribute* C, *the aggregation function computes the sum of credits of the constituent courses in a POS, so for* P_2, $\Phi_C(\{4, 3, 4, 2, 3, 3\}) = 4+3+4+2+3+3 = 19$. ◇

We note that other choices of the aggregation function can be accommodated in our framework (such as average or a combination of best and worst frontier sets), and that the above is only a representative list of choices.

Proposition 6 (Indifference of Frontier Elements)**.** *Consider an attribute* X_i, *whose valuations are aggregated using the best or worst frontier aggregation function. For any* $A \in \mathscr{F}(X_i)$, $\forall u, v \in A$, $u \sim_i v$.

Proof. Follows from Definition 15 (or the analogous definition of a best frontier) and a well known result due to Fishburn in [Fishburn, 1985]. □

Definition 16 (Valuation of a Composition w.r.t X_i using worst frontier aggregation)**.** *The valuation of a component* W *with respect to an attribute* X_i *is denoted as* $V_W(X_i) \in D_i$. *The valuation of a composition of two components* W_1 *and* W_2 *with respect to an attribute* X_i, *each with valuation* $V_{W_1}(X_i)$ *and* $V_{W_2}(X_i)$ *respectively, is given by*

$$V_{W_1 \oplus W_2}(X_i) := \Phi_i(V_{W_1}(X_i) \cup V_{W_2}(X_i))$$

Example 3. *Consider* $P_2 = CS501 \oplus CS502 \oplus CS505 \oplus CS506 \oplus CS509 \oplus CS510$ *in our running example (see Section 1.1.2).*

$$\begin{aligned} V_{P_2}(A) &= \Phi_A(V_{CS501}(A) \cup V_{CS502}(A) \cup V_{CS505}(A) \cup V_{CS506}(A) \cup V_{CS509}(A) \cup V_{CS510}(A)) \\ &= \Phi_A(\{Tom\} \cup \{Gopal\} \cup \{Bob\} \cup \{Bob\} \cup \{Jane\} \cup \{Tom\}) \\ &= \Phi_A(\{Tom, Gopal, Bob, Jane\}) \\ &= \{Tom, Jane\} \end{aligned}$$

◇

It must be noted that $V_{W_1 \oplus W_2}(X_i) = V_{W_2 \oplus W_1}(X_i)$, because the valuations of compositions are *subsets* of the union of individual component valuations.

4.1.2 Comparing Aggregated Valuations

Having obtained an aggregated valuation with respect to each attribute, we next proceed to discuss how to compare aggregated valuations attribute-wise. We denote the preference relation used to compare the aggregated valuations for an attribute X_i by $\succ_i'$. In the simple case when an aggregation function Φ_i with respect to an attribute X_i returns a value in D_i ($\mathscr{F}(X_i) = D_i$), the intra-attribute preference $\succ_i$ can be (re)used to compare aggregated valuations, i.e., $\succ_i' = \succ_i$. Other choices of $\succ_i'$ can be considered as long as $\succ_i'$ is a partial order. In order to obtain a strict preference relation, we require irreflexivity, and to obtain a rational preference relation, we require transitivity[2].

For worst frontier-based aggregation (Definition 15), we present a preference relation that uses the following idea: Given two compositions with different aggregated valuations (worst frontiers) A, B with respect to an attribute X_i, we say that A is preferred to B if for *every* valuation of X_i in B, there is *some* valuation in A that is strictly preferred.

Definition 17 (Preference over Worst Frontiers)**.** *Let $A, B \in \mathscr{F}(X_i)$ be two worst frontiers with respect to attribute X_i. We say that valuation A is* preferred *to B with respect to X_i, denoted by $A \succ_i' B$, when all elements in B are dominated by some element in A.*

$$\forall A, B \in \mathscr{F}(X_i) : A \succ_i' B \Leftrightarrow \forall b \in B, \exists a \in A : a \succ_i b$$

Example 4. *In our running example (see Section 1.1.2), we have $\{FM, TH\} \succ_A' \{DB, NW\}$ because $FM \succ_A DB$ and $TH \succ_A NW$.* ◇

[2]Any preference relation, including the one that compares only the uncommon elements of two sets can be used, provided it is irreflexive and transitive.

Given a preference relation over a set of elements, there are several ways of obtaining a preference relation over subsets of elements from the set (see [Barbera et al., 2004] for a survey on preferences over sets). As mentioned earlier, any such preference relation can be used in our setting, provided it is irreflexive and transitive.

Proposition 7 (Irreflexivity of $\succ_i'$)**.** $A \in \mathscr{F}(X_i) \Rightarrow A \not\succ_i' A$.

Proof. $\forall a, b \in A, a \sim_i b$ (follows from *Proposition* 6) □

Proposition 8 (Transitivity of $\succ_i'$)**.** *If* $A, B, C \in \mathscr{F}(X_i)$, *then* $A \succ_i' B \wedge B \succ_i' C \Rightarrow A \succ_i' C$.

Proof. Immediate from Definition 17. □

Definition 18. *Let* $A, B \in \mathscr{F}(X_i)$. *We say that valuation* A *is* at least as preferred *to* B *with respect to* X_i, *denoted* $\succeq_i'$ *iff*

$$A \succeq_i' B \Leftrightarrow A = B \vee A \succ_i' B$$

Proposition 9. $\succeq_i'$ *is reflexive, transitive.*

Proof. Follows from the facts that $=$ is reflexive and transitive, and $\succ_i'$ is irreflexive and transitive. □

Definition 19 (Complete Valuation)**.** *The complete valuation or outcome or assignment of a composition* C *is defined as a tuple* $V_C := \langle A_1, \ldots A_m \rangle$, *where* $A_i = V_C(X_i) \in \mathscr{F}(X_i)$. *The set of all possible valuations or outcomes is denoted as* $\prod_{i=1}^{m} \mathscr{F}(X_i)$.

Example 5. *In case of our example in Section 1.1.2:*

$$\begin{aligned}
V_{P_1} &= \langle \Phi_A(\{FM, AI, TH\}), \Phi_I(\{Tom, Gopal, Harry, White, Jane\}), \\
&\qquad \Phi_C(\{4, 3, 2, 3, 3, 3\})\rangle \\
&= \langle \{FM, TH\}, \{White, Harry\}, \{18\}\rangle \\
V_{P_2} &= \langle \Phi_A(\{FM, AI, DB, NW, TH\}), \Phi_I(\{Tom, Gopal, Bob, Jane\}), \\
&\qquad \Phi_C(\{4, 3, 4, 2, 3, 3\})\rangle \\
&= \langle \{DB, NW\}, \{Tom, Jane\}, \{19\}\rangle \\
V_{P_3} &= \langle \Phi_A(\{FM, AI, CA, SE, TH\}), \Phi_I(\{Harry, White, Tom, Jane\}), \\
&\qquad \Phi_C(\{2, 3, 3, 2, 3, 3\})\rangle \\
&= \langle \{CA, SE\}, \{Harry, White\}, \{16\}\rangle
\end{aligned}$$

◇

4.1.3 Dominance: Preference over Compositions

In the previous sections, we have discussed how to evaluate and compare a composition with respect to the attributes as a function of its components. In order to identify preferred compositions, we need to compare compositions with respect to their aggregated valuations over all attributes, based on the originally specified intra-attribute and relative importance preferences. We present a specific dominance relation for performing such a comparison.

Definition 20 (Dominance)**.** *Dominance* $\succ_d$ *is a binary relation defined as follows: for all* $\mathcal{U}^3, \mathcal{V} \in \prod_{i=1}^{m} \mathscr{F}(X_i)$

$$\begin{aligned}
\mathcal{U} \succ_d \mathcal{V} \Leftrightarrow \quad & \exists X_i \ : \ \mathcal{U}(X_i) \succ_i' \mathcal{V}(X_i) \ \wedge \\
& \forall X_k \ : \ (X_k \rhd X_i \vee X_k \sim_{\rhd} X_i) \ \Rightarrow \ \mathcal{U}(X_k) \succeq_k' \mathcal{V}(X_k)
\end{aligned}$$

[3] *To avoid excessively cluttering the notation, for a given composition* C*, we will slightly abuse notation by using* C *interchangeably with* V_C*.*

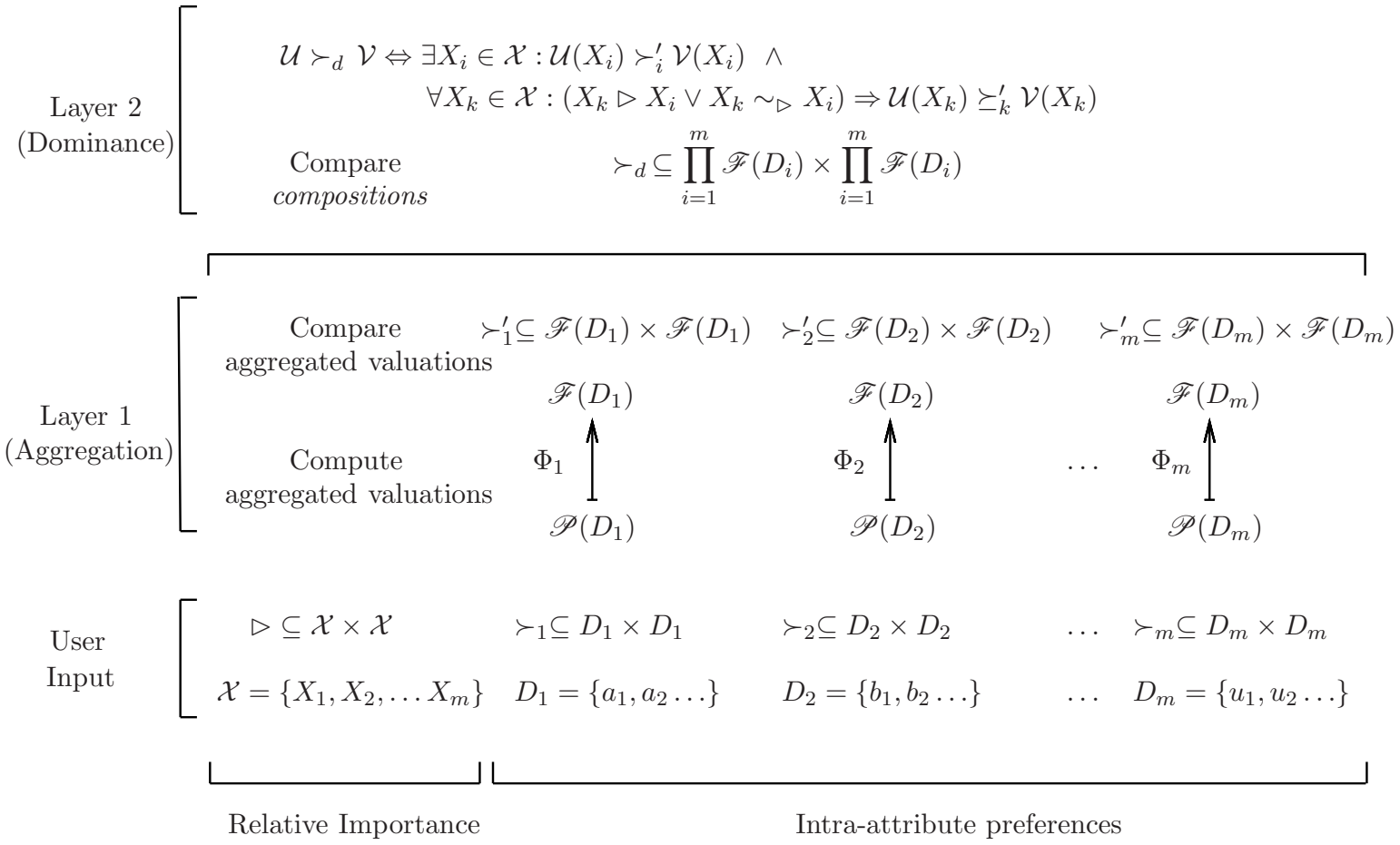


Figure 4.1 Dominance: Preference over compositions

In Definition 20, we call the attribute X_i as the *witness* of the relation. The dominance relation $\succ_d$ is derived from and respects both the intra-attribute preferences ($\succ_i$) as well as the relative importance preferences ($\rhd$) asserted by the user. Figure 4.1 graphically illustrates how dominance is derived from user-specified preferences. First, to start with we have user specified preferences, namely intra-attribute ($\succ_i$) and relative importance ($\rhd$) preferences. Next, from $\succ_i$ preferences, the valuations of compositions with respect to attributes are computed using the aggregation function (Φ_i). Then the intra-attribute preference relation to compare the aggregated valuations ($\succ'_i$) is derived from $\succ_i$. Finally, the global dominance ($\succ_d$) is defined in terms of $\succ'_i$ and $\rhd$.

The definition of dominance states that a composition $\mathcal{U}$ dominates $\mathcal{V}$ iff we can find a witness attribute X_i such that with respect to the intra-attribute preference $\succ_i$, the valuation of $\mathcal{U}$ dominates $\mathcal{V}$ in terms of $\succ'_i$, and for all attributes X_k which the user considers more important than ($\rhd$) or indifferent to ($\sim_\rhd$) X_i, the valuation of X_k in $\mathcal{U}$ is at least as preferred ($\succeq'_i$) as the valuation of X_k in $\mathcal{V}$.

Example 6. *In our running example (see Section 1.1.2), we have* $V_{P_2} \succ_d V_{P_1}$ *with* I *as witness and* $V_{P_1} \succ_d V_{P_3}$ *with* A *as witness. If* $I \rhd A$, $I \rhd C$ *but* $A \sim_\rhd C$ *then* $V_{P_2} \succ_d V_{P_1}$ *and* $V_{P_2} \succ_d V_{P_3}$ *with* I *as witness, but* $V_{P_1} \not\succ_d V_{P_3}$ *and* $V_{P_3} \not\succ_d V_{P_1}$. *This is because* P_1 *is preferred to* P_3 *with respect to* A *(*$\{FM, TH\} \succ'_A \{CA, SE\}$*); but* P_3 *is preferred to* P_1 *with respect to* C *(*$\{16\} \succ'_C \{18\}$*), and neither* A *nor* C *is relatively more important than the other.* ◇

4.1.3.1 Properties of $\succ_d$

We now proceed to analyze some properties of $\succ_d$. First, we show that a partial feasible composition is not dominated with respect to $\succ_d$ by any of its extensions. This property will be useful in establishing the soundness of algorithms that compute the most preferred compositions (see Section 4.2.2). Next, we observe that $\succ_d$ is irreflexive

(follows from the irreflexivity of $\succ_i$), and proceed to identify the conditions under which $\succ_d$ is transitive. Transitivity of $\succ_d$ is a necessary condition for it to be a *rational* preference relation [Morgenstern and Von Neumann, 1944, French, 1986a, Mas-Colell et al., 1995].

Proposition 10. *For any partial feasible composition B, there is no feasible extension $B \oplus W$ that dominates it, i.e., $V_{B\oplus W} \not\succ_d V_B$.*

Proof. The proof proceeds by showing that with respect to each attribute X_i, $V_{B\oplus W}(X_i) \not\succ'_i V_B(X_i)$, thereby ruling out the existence of a witness for $V_{B\oplus W} \succ_d V_B$. Suppose that by contradiction, $B \oplus W$ is a feasible extension of B such that $V_{B\oplus W} \succ_d V_B$. By Definition 20, $V_{B\oplus W} \succ_d V_B$ requires the existence of a witness attribute $X_i \in \mathcal{X}$ such that $V_{B\oplus W}(X_i) \succ'_i V_B(X_i)$, i.e.,

$$\forall b \in V_B(X_i)\ \exists a \in V_{B\oplus W}(X_i) : a \succ_i b \tag{4.1}$$

By Definition 16, we have $V_{B\oplus W}(X_i) = \Phi_i(V_B(X_i) \cup V_W(X_i))$. However, by Definition 15 $a \in \Phi_i(V_B(X_i) \cup V_W(X_i)) \Rightarrow \nexists b \in V_B(X_i) \cup V_W(X_i) : a \succ_i b$, which contradicts Equation (4.1). This rules out the existence of a witness for $V_{B\oplus W} \succ_d V_B$. Hence, $V_{B\oplus W} \not\succ_d V_B$. □

We next proceed to show that $\succ_d$ is not necessarily transitive when intra-attribute and relative importance preference relations are both arbitrary strict partial orders.

Proposition 11. *When intra-attribute preferences $\succ_i$ as well as relative importance among attributes $\rhd$ are arbitrary partial orders, $\mathcal{U} \succ_d \mathcal{V} \wedge \mathcal{V} \succ_d \mathcal{Z} \nRightarrow \mathcal{U} \succ_d \mathcal{Z}$*

Proof. We show a counter example of a compositional system with partially ordered $\{\succ_i\}, \rhd$ and compositions $\mathcal{U}, \mathcal{V}, \mathcal{Z}$ such that $\mathcal{U} \succ_d \mathcal{V}$, $\mathcal{V} \succ_d \mathcal{Z}$ but $\mathcal{U} \not\succ_d \mathcal{Z}$.

Consider a system with a set of attributes $\mathcal{X} = \{X_1, X_2, X_3, X_4\}$, each with domains $D_1 = \{a_1, b_1\}, \ldots D_4 = \{a_4, b_4\}$. Let the relative importance relation $\rhd$ on $\mathcal{X}$ and

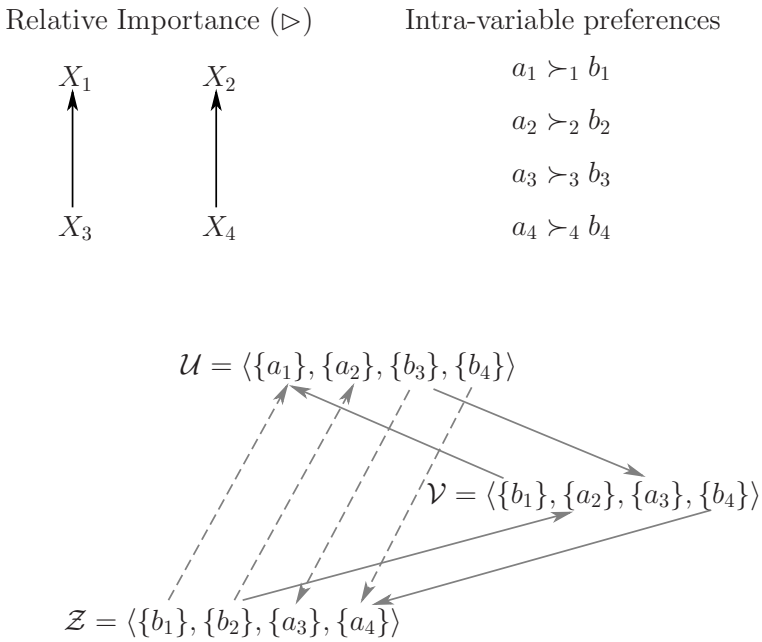


Figure 4.2 Counter example

Comp. (C)	$V_C(X_1)$	$V_C(X_2)$	$V_C(X_3)$	$V_C(X_4)$
$\mathcal{U}$	a_1	a_2	b_3	b_4
$\mathcal{V}$	b_1	a_2	a_3	b_4
$\mathcal{Z}$	b_1	b_2	a_3	a_4

Table 4.1 Valuations of $\mathcal{U}, \mathcal{V}, \mathcal{Z}$

the intra-attribute preferences $\succ_1 \ldots \succ_4$ be given by $\triangleright = \{(X_1, X_3), (X_2, X_4)\}$ and $\succ_i = \{(a_i, b_i)\}, i = 1, 2, 3, 4$ respectively (Figure 4.2). The valuations of $\mathcal{U}, \mathcal{V}, \mathcal{Z}$ with respect to the attributes $\mathcal{X}$ are given in Table 4.1.

Clearly $\mathcal{U} \succ_d \mathcal{V}$ with X_1 as the witness, and $\mathcal{V} \succ_d \mathcal{Z}$ with X_2 as the witness. In addition, note that:

$$\mathcal{Z}(X_3) \succ'_3 \mathcal{U}(X_3) \tag{4.2}$$

$$\mathcal{Z}(X_4) \succ'_4 \mathcal{U}(X_4) \tag{4.3}$$

However, we observe that $\mathcal{U} \not\succ_d \mathcal{Z}$:

a. X_1 is not a witness due to $X_4 \sim_{\rhd} X_1$ and Equation (4.3).

b. X_2 is not a witness due to $X_3 \sim_{\rhd} X_2$ and Equation (4.2).

c. X_3 is not a witness due to Equation (4.2).

d. X_4 is not a witness due to Equation (4.3). □

The above proposition shows that the dominance relation $\succ_d$ is not transitive when $\succ_i$ and $\rhd$ are arbitrary partial orders, when considering worst-frontier based aggregation. Because transitivity of preference is a necessary condition for rational choice [Morgenstern and Von Neumann, 1944, French, 1986a, Mas-Colell et al., 1995], we proceed to investigate the possibility of obtaining such a dominance relation by restricting $\rhd$. We later prove that such a restriction is necessary and sufficient for the transitivity of $\succ_d$.

Definition 21 (Relative Importance as an Interval Order)**.** *Relative importance is a binary relation $\rhd$ on $\mathcal{X}$ such that $X_i \rhd X_j$ iff X_i is* relatively more important *than X_j that is irreflexive and satisfies the following axiom:*

$$\forall X_i, X_j, X_k, X_l \in \mathcal{X} : (X_i \rhd X_j \wedge X_k \rhd X_l) \Rightarrow (X_i \rhd X_l \vee X_k \rhd X_j) \tag{4.4}$$

Proposition 12 (Transitivity of $\rhd$ [Fishburn, 1985])**.** *$\rhd$ is transitive.* □

Remarks.

1. Definition 21 imposes an additional restriction on the structure of the relative importance relation $\rhd$, over a strict partial order. A strict partial order is just irreflexive and transitive; however, the relative importance relation in Definition 21 should in addition satisfy Equation (4.4), thereby yielding an *interval order* [Fishburn, 1985].

2. The indifference relation with respect to $\rhd$, namely $\sim_{\rhd}$ is *not* transitive. For example, if there are three attributes $\mathcal{X} = \{X_1, X_2, X_3\}$, and $\rhd = \{(X_1, X_2)\}$. $\rhd$ satisfies the condition for an *interval order*, and we have $X_1 \sim_{\rhd} X_3$ and $X_3 \sim_{\rhd} X_2$, but $X_1 \not\sim_{\rhd} X_2$ because $X_1 \rhd X_2$.

Recall that from Section 3.1, Propositions 2-5 establish the properties of the dominance relation $\succ_d$. In particular, we showed that $\succ_d$ is irreflexive (Proposition 2) and transitive (Proposition 5), making $\succ_d$ a *strict partial order* (Theorem 1).

4.1.4 Choosing the Most Preferred Solutions

Given a set $\mathcal{C} = \{C_i\}$ of compositions and a preference relation $\succ$ (e.g., $\succ_d$) that allows us to compare any pair of compositions, the problem is to find the most preferred composition(s). When the preference relations are totally ordered (e.g., a ranking) over a set of alternative solutions, rationality of choice suggests ordering the alternatives with respect to the complete preference and choosing the "*best*" alternative, i.e., the one that ranks the highest. However, when the preference relation is a strict partial order, e.g., in the case of $\succ_d$, not every pair of solutions (compositions) may be comparable. Therefore, a solution that is *the most preferred* with respect to the preference relation may not exist. Hence, we use the notion of the *non-dominated* set of solutions defined as follows.

Definition 22 (Non-dominated Set)**.** *The non-dominated set of elements (alternatives or solutions or compositions) of a set $\mathcal{C}$ with respect to a (partially ordered) preference relation $\succ$ (e.g., $\succ_d$), denoted $\Psi_{\succ}(\mathcal{C})$, is a subset of $\mathcal{C}$ such that none of the elements in S are preferred to any element in $\Psi_{\succ}(\mathcal{C})$.*

$$\Psi_{\succ}(\mathcal{C}) = \{C_i \in \mathcal{C} | \nexists C_j \in \mathcal{C} : C_j \succ C_i\}$$

Note that as per this definition, $\Psi_{\succ}(\mathcal{C})$ is the *maximal* set of elements in $\mathcal{C}$ with respect to the relation $\succ$. It is also easy to observe that $\mathcal{C} \neq \emptyset \Leftrightarrow \Psi_{\succ}(\mathcal{C}) \neq \emptyset$.

4.2 Algorithms for Computing the Most Preferred Compositions

We now turn to the problem of identifying from a set of feasible compositions (that satisfy a pre-specified functionality (φ)), the most preferred subset, i.e., the non-dominated set.

4.2.1 Computing the Maximal/Minimal Subset with respect to a Partial Order

The straightforward way of computing the maximal (non-dominated) elements in a set S of n elements with respect to any preference relation $\succ$ is the following algorithm: For each element $s_i \in S$, check if $\exists s_j \in S : s_j \succ s_i$, and if not, s_i is in the non-dominated set. This simple "compare all pairs and delete dominated" approach involves computing dominance with respect to $\succ$ $O(n^2)$ times.

Recently Daskalakis et al. provided an algorithm [Daskalakis et al., 2009] that performs at most $O(wn)$ pairwise comparisons to compute the maximal elements of a set S with respect to a partial order $\succ$, where $n = |S|$ and w is the *width* of the partial order $\succ$ on S (the size of the maximal set of pairwise incomparable elements in S with respect to $\succ$). The algorithm presented in [Daskalakis et al., 2007] finds the minimal elements; the corresponding algorithm for finding the maximal elements is as follows.
Let $T_0 = \emptyset$. Let the elements of the set S be $x_1, x_2, \cdots x_n$. At step $t(\geq 1)$:

- Compare x_t to all elements in T_{t-1}.
- If there exists some $a \in T_{t-1}$ such that $a \succ x_t$, do nothing.
- Otherwise, remove from T_{t-1} all elements a such that $x_t \succ a$ and put x_t into T_t.

Algorithm 1 ComposeAndFilter($\succ, f, \varphi$)

1. Find the set $\mathcal{C}$ of feasible compositions w.r.t. φ using f
2. **return** $\Psi_{\succ}(\mathcal{C})$

On termination, the set T_n contains all the maximal elements in S, i.e., non-dominated subset of S with respect to $\succ$. We make use of the above algorithm to compute the non-dominated (maximal) subsets (namely, $\Psi_{\succ}(\cdot)$), and the original version of the algorithm given in [Daskalakis et al., 2007] to compute the worst-frontiers (minimal subsets).

4.2.2 Algorithms for Finding the Most Preferred Feasible Compositions

We proceed to develop algorithms for finding the most preferred feasible compositions, given a compositional system $\langle R, \oplus, \models \rangle$ consisting of a repository R of pre-existing components, a user specified functionality φ, user preferences $\{\succ_i\}$ and $\rhd$ and a functional composition algorithm. Two desirable properties of an algorithm that computes the non-dominated set of most preferred feasible compositions are soundness and completeness as defined below.

Definition 23 (Soundness and Completeness)**.** *An algorithm A that, given a set $\mathcal{C}$ of feasible compositions, computes a set of feasible compositions $S_A \subseteq \Psi_{\succ_d}(\mathcal{C})$ is said to be sound with respect to $\mathcal{C}$. Such an algorithm is complete with respect to $\mathcal{C}$ if $S_A \supseteq \Psi_{\succ_d}(\mathcal{C})$.*

Given a compositional system $\langle R, \oplus, \models \rangle$ consisting of a repository R of pre-existing components, and a user specified functionality φ, the most straightforward approach to finding the most preferred feasible compositions involves: (a) computing the set $\mathcal{C}$ of functionally feasible compositions using a functional composition algorithm f, and (b) choosing the non-dominated set according to preferences over non-functional attributes.

Algorithm 1 follows this simple approach to produce the set $\Psi_{\succ_d}(\mathcal{C})$ of all non-dominated feasible compositions, when invoked with the preference relation $\succ_d$, the functional composition algorithm f and the desired functionality φ. $\Psi_{\succ_d}(\mathcal{C})$ can be

computed using the procedure described in Section 4.2.1. Algorithm 1 is both sound and complete with respect to $\mathcal{C}$.

4.2.3 A *Sound and Weakly Complete* Algorithm

Note that in the worst case, Algorithm 1 evaluates the dominance relation $\succ_d$ between all possible pairs of feasible compositions $\mathcal{C}$. However, this can be avoided if we settle for a non-empty subset of $\Psi_{\succ_d}(\mathcal{C})$. Note that every solution in such a subset is guaranteed to be *"optimal"* with respect to user preferences $\succ_d$. We introduce the notion of *weak completeness* to describe an algorithm that computes a set of feasible compositions, at least one of which is non-dominated with respect to $\succ_d$.

Definition 24 (Weak Completeness)**.** *An algorithm A that, given a set $\mathcal{C}$ of feasible compositions, computes a set S_A of feasible compositions is said to be weakly complete with respect to $\mathcal{C}$ if $\Psi_{\succ_d}(\mathcal{C}) \neq \emptyset \Rightarrow S_A \cap \Psi_{\succ_d}(\mathcal{C}) \neq \emptyset$.*

We now proceed to describe a sound and weakly complete algorithm, i.e., one that computes a non-empty subset of $\Psi_{\succ_d}(\mathcal{C})$. The algorithm is based on the following observation: Solutions that are non-dominated with respect to each of the relatively most-important attributes are guaranteed to include some solutions that are non-dominated overall with respect to $\succ_d$ as well. Hence, the solutions that are most preferred with respect to each such attribute can be used to compute a non-empty subset of $\Psi_{\succ_d}(\mathcal{C})$. We proceed by considering solutions that are most preferred with respect to an attribute X_i.

Definition 25 (Non-dominated solutions w.r.t. attributes)**.** *The set $\Psi_{\succ'_i}(\mathcal{C})$ of solutions that are* non-dominated with respect to an attribute X_i *is defined as*

$$\Psi_{\succ'_i}(\mathcal{C}) = \{\mathcal{U} \mid \mathcal{U} \in \mathcal{C} \wedge \nexists \mathcal{V} \in \mathcal{C} : \mathcal{V}(X_i) \succ'_i \mathcal{U}(X_i)\}.$$

Let $I \subseteq \mathcal{X}$ be the set of most important attributes with respect to $\rhd$, i.e., $I = \Psi_{\rhd}(\mathcal{X}) = \{X_i | \nexists X_j \in \mathcal{X} : X_j \rhd X_i\}$. Clearly, $I \neq \emptyset$ because there always exists a non-empty maximal set of elements in the partial order $\rhd$. The following proposition states that for every $X_i \in I$, at least one of the solutions in $\Psi_{\succ'_i}(\mathcal{C})$ is also contained in $\Psi_{\succ_d}(\mathcal{C})$.

Proposition 13. $\forall X_i \in I : \Psi_{\succ_d}(\mathcal{C}) \neq \emptyset \Rightarrow \Psi_{\succ'_i}(\mathcal{C}) \cap \Psi_{\succ_d}(\mathcal{C}) \neq \emptyset$.

Proof. Let $X_i \in I$ and $\mathcal{U} \in \Psi_{\succ'_i}(\mathcal{C})$. There are two possibilities: $\mathcal{U} \in \Psi_{\succ_d}(\mathcal{C})$ and $\mathcal{U} \notin \Psi_{\succ_d}(\mathcal{C})$. If $\mathcal{U} \in \Psi_{\succ_d}(\mathcal{C})$, then there is nothing left to prove.

Suppose that $\mathcal{U} \notin \Psi_{\succ_d}(\mathcal{C})$. Then we show that $\exists \mathcal{V} \neq \mathcal{U}$ such that $\mathcal{V} \in \Psi_{\succ'_i}(\mathcal{C}) \cap \Psi_{\succ_d}(\mathcal{C})$.

$$\mathcal{U} \in \Psi_{\succ'_i}(\mathcal{C}) \wedge \mathcal{U} \notin \Psi_{\succ_d}(\mathcal{C}) \Rightarrow \exists \mathcal{V} \in \Psi_{\succ_d}(\mathcal{C}) : \mathcal{V} \succ_d \mathcal{U}.$$

By Definitions 20 and 25, it follows that $\nexists \mathcal{V} \in \Psi_{\succ_d}(\mathcal{C}) : \mathcal{V}(X_i) \succ'_i \mathcal{U}(X_i)$. Hence, X_i cannot be a witness for $\mathcal{V} \succ_d \mathcal{U}$. Now there are two cases to consider.

Case 1: $\mathcal{U}(X_i) \succ'_i \mathcal{V}(X_i)$.

Let attribute $X_j \neq X_i$ be a witness for $\mathcal{V} \succ_d \mathcal{U}$. Since $X_i \in I$, $(X_i \rhd X_j) \vee (X_i \sim_{\rhd} X_j)$. It therefore follows that $\mathcal{V}(X_i) \succeq'_i \mathcal{U}(X_i)$, which contradicts our assumption that $\mathcal{U}(X_i) \succ'_i \mathcal{V}(X_i)$. Hence, $\mathcal{U}(X_i) \not\succ'_i \mathcal{V}(X_i)$.

Case 2: $\mathcal{U}(X_i) \sim'_i \mathcal{V}(X_i)$.

Let attribute $X_j \neq X_i$ be a witness for $\mathcal{V} \succ_d \mathcal{U}$. Since $X_i \in I$, $(X_i \rhd X_j) \vee (X_i \sim_{\rhd} X_j)$. From Definition 20, $\mathcal{V} \succ_d \mathcal{U}$ only if $\mathcal{V}(X_i) \succeq'_i \mathcal{U}(X_i)$. Because of our assumption that $\mathcal{U}(X_i) \sim'_i \mathcal{V}(X_i)$, it must be the case that $\mathcal{V}(X_i) = \mathcal{U}(X_i)$, i.e., $\mathcal{V} \in \Psi_{\succ'_i}(\mathcal{C})$. Thus, we have:

$$\mathcal{U} \in \Psi_{\succ'_i}(\mathcal{C}) \setminus \Psi_{\succ_d}(\mathcal{C}) \Rightarrow \exists \mathcal{V} \in \Psi_{\succ'_i}(\mathcal{C}) \cap \Psi_{\succ_d}(\mathcal{C}) : \mathcal{V} \succ_d \mathcal{U} \tag{4.5}$$

This completes the proof. □

Algorithm 2 constructs a subset of $\Psi_{\succ_d}(\mathcal{C})$, using the sets $\{\Psi_{\succ'_i}(\mathcal{C}) \mid X_i \in I\}$. First, the algorithm computes the set I of most important attributes in $\mathcal{X}$ with respect to $\rhd$ (Line 1). The algorithm iteratively computes $\Psi_{\succ'_i}(\mathcal{C})$ for each $X_i \in I$ (Lines 2, 3), identifies the subset of solutions that are non-dominated with respect to $\succ_d$ in each case, and combines them to obtain $\theta \subseteq \Psi_{\succ_d}(\mathcal{C})$.

Algorithm 2 WeaklyCompleteCompose($\{\succ_i\}, \rhd, f, \varphi$)

1. $I \leftarrow \Psi_{\rhd}(\mathcal{X}) = \{X_i \mid \nexists X_j : X_j \rhd X_i\}$
2. **for all** $X_i \in I$ **do**
3. $\quad \Psi_{\succ'_i}(\mathcal{C}) \leftarrow$ ComposeAndFilter($\succ'_i, f, \varphi$)
4. $\quad \theta \leftarrow \theta \cup \Psi_{\succ_d}(\Psi_{\succ'_i}(\mathcal{C}))$
5. **end for**
6. **return** θ

Theorem 6 (Soundness and Weak Completeness of Algorithm 2)**.** *Given a set of attributes $\mathcal{X}$, preference relations $\rhd$ and $\succ'_i$, Algorithm 2 generates a set θ of feasible compositions such that $\theta \subseteq \Psi_{\succ_d}(\mathcal{C})$ and $\Psi_{\succ_d}(\mathcal{C}) \neq \emptyset \Rightarrow \theta \neq \emptyset$.*

Proof.

Soundness: The proof proceeds by contradiction. Suppose that the algorithm returns a solution $\mathcal{U} \in \theta$ such that $\mathcal{U} \notin \Psi_{\succ_d}(\mathcal{C})$. Because $\mathcal{U} \in \theta$, it is necessary (by Line 4) that $\exists X_i \in I : \mathcal{U} \in \Psi_{\succ'_i}(\mathcal{C}) \setminus \Psi_{\succ_d}(\mathcal{C})$. Then, from Equation (4.5) in the proof of Proposition 13, $\exists \mathcal{V} \in \Psi_{\succ'_i}(\mathcal{C}) \cap \Psi_{\succ_d}(\mathcal{C}) : \mathcal{V} \succ_d \mathcal{U}$, which means that $\mathcal{U} \notin \Psi_{\succ_d}(\Psi_{\succ'_i}(\mathcal{C}))$. However, this contradicts Line 4 of the algorithm. Hence, $\theta \subseteq \Psi_{\succ_d}(\mathcal{C})$, i.e., Algorithm 2 is sound.

Weak Completeness: Because $I \neq \emptyset$, Line 4 is executed by the algorithm at least once for some $X_i \in I$. By Definition 22, we have $\mathcal{C} \neq \emptyset \Rightarrow \Psi_{\succ'_i}(\mathcal{C}) \neq \emptyset \Rightarrow \Psi_{\succ_d}(\Psi_{\succ'_i}(\mathcal{C})) \neq \emptyset \Rightarrow \theta \neq \emptyset$. Hence, Algorithm 2 is weakly complete by Definition 24.

□

In general, Algorithm 2 is not guaranteed to yield a complete set of solutions, i.e., $\theta \neq \Psi_{\succ_d}(\mathcal{C})$. The following example illustrates such a case.

Example 7. *Consider a compositional system with two attributes* $\mathcal{X} = \{X_1, X_2\}$*, with domains* $\{a_1, a_2, a_3\}$ *and* $\{b_1, b_2, b_3\}$ *respectively. Let their intra-attribute preferences be total orders:* $a_1 \succ_1 a_2 \succ_1 a_3$ *and* $b_1 \succ_2 b_2 \succ_2 b_3$ *respectively, and let both attributes by equally important* ($\rhd = \emptyset$)*. Suppose the user-specified goal* φ *is satisfied by three feasible compositions* C_1, C_2, C_3 *with valuations* $V_{C_1} = \langle\{a_1\}, \{b_3\}\rangle$*,* $V_{C_2} = \langle\{a_3\}, \{b_1\}\rangle$ *and* $V_{C_3} = \langle\{a_2\}, \{b_2\}\rangle$ *respectively. Given the above preferences,* $\Psi_{\succ'_1}(\mathcal{C}) = \{C_1\}$ *and* $\Psi_{\succ'_2}(\mathcal{C}) = \{C_2\}$*. Thus,* $\theta = \{C_1, C_2\}$ *However,* $\Psi_{\succ_d}(\mathcal{C}) = \{C_1, C_2, C_3\} \neq \theta$*.* ◇

The above example shows that the most preferred valuation for one attribute (e.g., X_1) can result in poor valuations for one or more other attributes (e.g., X_2). Algorithm 2 may thus leave out solutions like C_3 that are not most preferred with respect to any one $\succ'_i$, but nevertheless may correspond to a good compromise when we consider multiple most important attributes. It is a natural question ask what are the minimal conditions under which Algorithm 2 is complete. A related question is whether Algorithm 2 can be guaranteed to produce a certain minimum number of non-dominated solutions ($|\theta|$) under some specific set of conditions. Note that in general, the cardinality of θ depends not only on the user preferences $\succ_i, \rhd$, but also on the user specified functionality φ which together with the repository R determines the set $\mathcal{C}$ of feasible compositions. However, in the special case when $\rhd$ specifies a single attribute X_t that is relatively more important than all other attributes, we can show that Algorithm 2 is complete.

Proposition 14. *If* $I = \{X_t\} \wedge \forall X_k \neq X_t \in \mathcal{X} : X_t \rhd X_k$*, then* $\Psi_{\succ_d}(\mathcal{C}) \subseteq \theta$*, i.e., Algorithm 2 is complete.*

Proof. The proof proceeds by contradiction. Let $I = \{X_t\}$ and $\forall X_k \neq X_t \in \mathcal{X} : X_t \rhd X_k$, and suppose that $\exists \mathcal{V} \in \Psi_{\succ_d}(\mathcal{C}) \setminus \Psi_{\succ'_t}(\mathcal{C})$. Since $\mathcal{V} \notin \Psi_{\succ'_t}(\mathcal{C})$, by Definition 22 it must be the case that $\exists \mathcal{U} \in \Psi_{\succ'_t}(\mathcal{C}) : \mathcal{U}(X_t) \succ'_t \mathcal{V}(X_t)$. However, then $\mathcal{U} \succ_d \mathcal{V}$ by Definition 20 thus contradicting our assumption that $\mathcal{V} \in \Psi_{\succ_d}(\mathcal{C})$. □

It remains to be seen what are all the necessary and sufficient conditions for ensuring the completeness of Algorithm 2, and we plan to address this problem in future.

4.2.4 Optimizing with Respect to One of the Most Important Attributes

As we will see in Section 4.2.6, Algorithm 2 has a high worst case complexity, especially if the set I of most important attributes is large. This is due to the fact that for each most important attribute $X_i \in I$, the algorithm computes the non-dominated set over the feasible compositions with respect to $\succ'_i$ first, and then with respect to $\succ_d$, i.e., $\theta \cup \Psi_{\succ_d}(\Psi_{\succ'_i}(\mathcal{C}))$ (Line 4). The computation of the non-dominated set with respect to $\succ_d$, although expensive, is crucial to ensuring the soundness of Algorithm 2.

While soundness is a desirable property, there may be settings requiring faster computation of feasible compositions, where it may be acceptable to obtain a set S of feasible compositions that contains *at least one* (whenever there exists one) of the most preferred feasible compositions (one that is non-dominated by any other feasible composition with respect to $\succ_d$). In such a case, it might be useful to have an algorithm with lower complexity that finds a set of feasible compositions of which *at least one* is most preferred (i.e., weakly complete), as opposed to one with a higher complexity that finds a set of feasible compositions *all* of which are most preferred (i.e., sound).

Algorithm 3 AttWeaklyCompleteCompose($\{\succ_i\}, \rhd, f, \varphi$)

1. $I \leftarrow \Psi_{\rhd}(\mathcal{X}) = \{X_i \mid \nexists X_j : X_j \rhd X_i\}$
2. **for some** $X_i \in I$
3. $\theta \leftarrow \Psi_{\succ'_i}(\mathcal{C}) = \text{ComposeAndFilter}(\succ'_i, f, \varphi)$
4. **return** θ

We consider one such modification of Algorithm 2, namely Algorithm 3, that arbitrarily picks one of the most important attributes $X_i \in I$ (as opposed to the entire set I as in Algorithm 2) and finds the set of all feasible compositions that are non-dominated with respect to $\succ'_i$, i.e., $\theta = \Psi_{\succ'_i}(\mathcal{C})$ for Algorithm 3.

The weak completeness of Algorithm 3 follows directly from Proposition 13. In the following example, however, we show that some of the feasible compositions produced by Algorithm 3 may be dominated by some other feasible composition with respect to $\succ_d$, i.e., Algorithm 3 is not sound.

Example 8. *Consider a compositional system with two attributes $\mathcal{X} = \{X_1, X_2\}$, with domains $\{a_1, a_2\}$ and $\{b_1, b_2\}$ respectively. Let their intra-attribute preferences be: $a_1 \succ_1 a_2$ and $b_1 \succ_2 b_2$ respectively, and let both attributes by equally important ($\rhd = \emptyset$; $I = \{X_1, X_2\}$). Suppose the user-specified goal φ is satisfied by three feasible compositions C_1, C_2, C_3 with valuations $V_{C_1} = \langle\{a_1\}, \{b_1\}\rangle$, $V_{C_2} = \langle\{a_2\}, \{b_1\}\rangle$ and $V_{C_3} = \langle\{a_1\}, \{b_2\}\rangle$ respectively. Given the above preferences, if we choose to maximize the preference with respect to attribute $X_1 \in I$, then $\theta = \Psi_{\succ'_1}(\mathcal{C}) = \{C_1, C_3\}$. If we chose $X_2 \in I$ instead, we get $\theta = \Psi_{\succ'_2}(\mathcal{C}) = \{C_1, C_2\}$. However, in any case $\Psi_{\succ_d}(\mathcal{C}) = \{C_1\} \neq \theta$.* $\diamond$

We note that Algorithm 3 and Algorithm 2 are identical when $|I| = 1$, i.e., when there is a unique most important attribute in $\mathcal{X}$ with respect to $\rhd$. Hence, Algorithm 3 is sound (Theorem 6) and complete (Proposition 14) in this case.

4.2.5 Interleaving Functional Composition with Preferential Optimization

Algorithms 1, 2 and 3 identify the most preferred feasible compositions using the two step approach: (a) find the feasible compositions $\mathcal{C}$; and (b) compute a subset of $\mathcal{C}$ that is preferred with respect to the user preferences. We now develop an algorithm that eliminates some of the intermediate partial feasible compositions from consideration based on the user preferences. This is particularly useful in settings (such as when $|\mathcal{C}|$ is large relative to $|\Psi_{\succ_d}(\mathcal{C})|$), where it might be more efficient to compute only a subset of $\mathcal{C}$ that are likely (based on $\succ_i$ and $\rhd$) to be in $\Psi_{\succ_d}(\mathcal{C})$.

Algorithm 4 requires that the functional composition algorithm f is incremental (see Definition 6), i.e., that it produces a set $f(C)$ of functionally feasible extensions given

Algorithm 4 $InterleaveCompose(\mathcal{L}, \succ, f, \varphi)$

1. **if** $\mathcal{L} = \emptyset$ **then**
2. **return** $\emptyset$
3. **end if**
4. $\theta = \Psi_{\succ}(\mathcal{L})$
5. $\theta' = \emptyset$
6. **for all** $C \in \theta$ **do**
7. **if** $C \not\models \varphi$ **then**
8. $\theta' = \theta' \cup f(C)$
9. **else**
10. $\theta' = \theta' \cup \{C\}$
11. **end if**
12. **end for**
13. **if** $\theta' = \theta$ **then**
14. **return** θ
15. **else**
16. $InterleaveCompose((\mathcal{L} \setminus \theta) \cup \theta', \succ, f, \varphi)$
17. **end if**

any existing partial feasible composition C. At each step, Algorithm 4 chooses a subset of the feasible extensions produced by applying f on all the non-dominated partial feasible compositions, based on the user preferences. Algorithm 4 computes the non-dominated set of feasible compositions by *interleaving* the execution of the incremental functional composition algorithm f with the ordering of partial solutions with respect to preferences over non-functional attributes.

Algorithm 4 is initially invoked using the parameters $\mathcal{L} = \{\bot\}$ [4], $\succ_d$, the functional composition algorithm f and φ. The algorithm maintains at each step a list $\mathcal{L}$ of partial feasible compositions under consideration. If $\mathcal{L}$ is empty at any step, i.e., there are no more partial feasible compositions to be explored, then the algorithm terminates with no solution (Lines $1-3$); otherwise it selects from $\mathcal{L}$, the subset θ that is non-dominated with respect to some preference relation $\succ$ (Line 4). If all the partial feasible

[4] It is not necessary to invoke the algorithm with $\mathcal{L} = (\bot)$ initially. There may be functional composition algorithms that begin with an non-empty composition C and proceed to obtain a feasible composition by iteratively altering C. For instance, one could think of randomized or evolutionary algorithms that begin with a random, non-empty composition which is somehow repeatedly "improved" during the course of composition.

compositions in θ are also feasible compositions, then the algorithm outputs θ and terminates (Lines $13-14$). Otherwise, it replaces the partial feasible compositions in θ that are not feasible compositions, with their one-step extensions (Lines $7-8$).

Proposition 15 (Termination of Algorithm 4)**.** *Given a finite repository of components, Algorithm 4 terminates in a finite number of steps.*

Proof. Given a finite repository R of components, and an algorithm f that computes feasible extensions of partial feasible compositions[5], and due to the fact that Algorithm 4 does not re-visit any partial feasible composition, the number of recursive calls is finite. □

We next investigate the soundness, weak-completeness and completeness properties of Algorithm 4. Proposition 16 states that the algorithm is in general not sound with respect to $\mathcal{C}$, i.e., it is not guaranteed to produce feasible compositions that are non-dominated with respect to $\succ_d$. However, this does not discount the usefulness of the algorithm, as we will show that it is sound under some other assumptions (see Theorem 7).

Proposition 16 (Unsoundness of Algorithm 4)**.** *Given a functional composition algorithm f and user preferences $\succ_i'$ and $\rhd$ over a set of attributes $\mathcal{X}$, Algorithm 4 is not guaranteed to generate a set of feasible compositions θ such that $\theta \subseteq \Psi_{\succ_d}(\mathcal{C})$.*

Proof. We provide an example wherein Algorithm 4 returns a feasible composition that is dominated by some other feasible composition. Consider a compositional system with a single attribute $\mathcal{X} = \{X_1\}$, with a domain of $\{a_1, a_2, a_3, a_4\}$. Let the intra-attribute preference of the user over those values be the partial order: $a_4 \succ_1 a_1$ and $a_2 \succ_1 a_3$ (Figure 4.3). Let $R = \{W_1, W_2, W_3, W_4\}$ be the repository of components in the compositional system such that $V_{W_i}(X_1) = \{a_i\}$.

[5] An f that terminates with a set of feasible extensions is guaranteed by the decidability of φ.

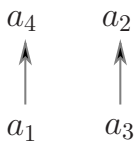


Figure 4.3 Intra-attribute preference $\succ_1$ for attribute X_1

Suppose that there are three feasible compositions in $\mathcal{C}$ satisfying the user specified functionality φ, namely $C_1 = W_1, C_2 = W_2, C_3 = W_3 \oplus W_4$. Their respective valuations are: $V_{C_1} = \langle\{a_1\}\rangle$, $V_{C_2} = \langle\{a_2\}\rangle$ and $V_{C_3} = \langle\{a_3, a_4\}\rangle$. Clearly, $\Psi_{\succ_d}(\mathcal{C}) = \{C_2, C_3\}$, because $V_{C_3} \succ_d V_{C_1}$ (due to the fact that $\{a_3, a_4\} \succ'_1 \{a_1\}$).

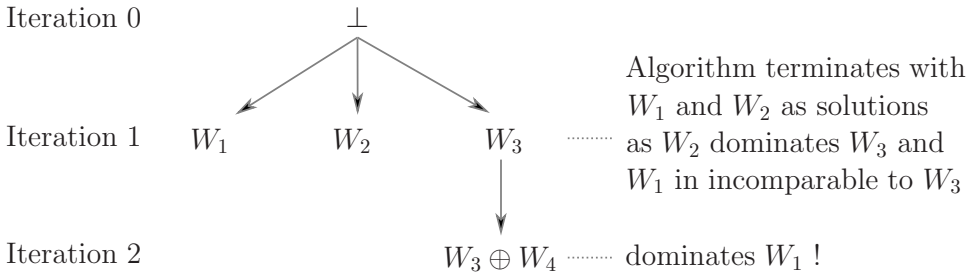


Figure 4.4 Execution of Algorithm 4

Now suppose that there exists a functional composition algorithm f that produces the following sequence of partial feasible compositions (Figure 4.4): $\{\perp\}$, $\{W_1, W_2, W_3\}$, $\{W_1, W_2, W_3 \oplus W_4\}$. According to Line 13 of Algorithm 4, the algorithm will terminate after the first invocation of f, i.e., when the set $\{W_1, W_2, W_3\}$ of partial feasible compositions is produced by f. This is because after the first iteration, $\theta = \{W_1, W_2\}$, with $V_{W_2} \succ_d V_{W_3}$, and both W_1 and W_2 are feasible compositions. This results in $\theta = \{C_1, C_2\} \not\subseteq \Psi_{\succ_d}(\mathcal{C})$. □

This result implies that in general, not all feasible compositions returned by Algorithm 4 (θ) are in $\Psi_{\succ_d}(\mathcal{C})$. The example shown in Figure 4.5 illustrates this problem. At the time of termination, there may exist some partial feasible composition B in the list

$\mathcal{L}$ that *is dominated* by some feasible composition E in θ; however, it may be possible to extend B to a feasible composition $B \oplus W$ that *dominates* one of the compositions F in θ (as illustrated by the counter example in the proof). In other words, $V_E \succ_d V_B$, $V_F \sim_d V_E$, $V_F \sim_d V_B$ and $V_{B \oplus W} \succ_d V_F$.

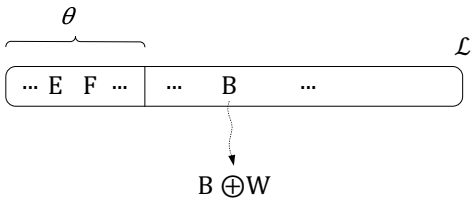


Figure 4.5 The case when Algorithm 4 is not sound

Although Algorithm 4 is not sound in general, we show that it is sound when the $\succ_d$ relation is an interval order (as opposed to an arbitrary partial order).

Theorem 7 (Soundness of Algorithm 4)**.** *If $\succ_d$ is an interval order, then given a functional composition algorithm f and user preferences $\{\succ'_i\}, \rhd$ over a set of attributes $\mathcal{X}$, Algorithm 4 generates a set θ of feasible compositions such that $\theta \subseteq \Psi_{\succ_d}(\mathcal{C})$.*

Proof. Suppose that by contradiction, $F \in \theta$ and there is a feasible composition $D \notin \theta$ such that $V_D \succ_d V_F$. If D is present in the list $\mathcal{L}$ upon termination of the algorithm, then D should have been in θ, because the algorithm terminates only when *all* compositions in $\Psi_{\succ_d}(\mathcal{L})$ are feasible. This implies that the algorithm did not terminate with an $\mathcal{L}$ containing D.

The algorithm keeps track of all partial feasible compositions that can be extended from $\perp$ in $\mathcal{L}$, without discarding any of them before termination. Therefore, the existence of any such feasible composition D that is not in $\mathcal{L}$ at the time of termination must imply the existence of some partial feasible composition B in the list (at the time of termination) that can be extended to produce the feasible composition D, i.e., $B \oplus W_1 \oplus W_2 \oplus \ldots \oplus W_n = D$ such that $B \not\models \varphi$ and $D \models \varphi$.

$B \not\models \varphi \Rightarrow B \notin \theta$ at the time of termination, and therefore $\exists E \in \theta : V_E \succ_d V_B$. Because $\succ_d$ is transitive (by Proposition 5), since $V_D \not\succ_d V_B$ (by Proposition 10), it follows that $V_D \not\succ_d V_E$ (otherwise, $V_D \succ_d V_E \wedge V_E \succ_d V_B \Rightarrow V_D \succ_d V_B$, a contradiction). Hence, D must dominate some composition other than E, say $F \in \theta$ at the time of termination, i.e., $V_D \succ_d V_F$. Because $E, F \in \theta$, it follows that $V_F \sim_d V_E$, which in turn implies that $V_E \not\succ_d V_D$. Therefore, $\exists F \in \theta : V_D \succ_d V_F$, $V_F \sim_d V_E$ and $V_E \sim_d V_D$ (see Figure 4.6).

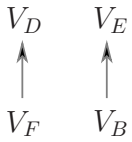


Figure 4.6 Dominance relationships that violate the interval order restriction on $\succ_d$

From $V_E \succ_d V_B$, $V_D \succ_d V_F$, $V_F \sim_d V_E$ and $V_D \not\succ_d V_B$, it follows that $V_D \sim_d V_B$ (because $V_B \succ_d V_D$ would otherwise imply $V_E \succ_d V_F$, a contradiction). Finally, it must be the case that: $V_B \not\succ_d V_F$, since otherwise it would contradict $V_F \sim_d V_E$; and $V_F \not\succ_d V_B$, since otherwise it would contradict $V_D \sim_d V_B$. Therefore, $V_B \sim_d V_F$. Thus, the only possible dominance relationships among the compositions B, D, E, F are as follows (see Figure 4.6):

- $V_E \succ_d V_B$
- $V_D \succ_d V_F$

However, this scenario is ruled out by the fact that $\succ_d$ is an interval order. Hence $\forall F \in \theta, \forall D \in \mathcal{C} \setminus \theta : V_D \not\succ_d V_F$, i.e., $\theta \subseteq \Psi_{\succ_d}(\mathcal{C})$. □

Because Theorem 7 requires $\succ_d$ to be an interval order, an important question arises: What are the conditions under which $\succ_d$ an interval order? We investigate the answer

to this question later in Section 5.3, where we prove that if $\{\succ_i\}$ and $\rhd$ are total orders, then $\succ_d$ is a weak order (i.e., also an interval order). Based on the experimental results we obtain, we further conjecture that if $\{\succ_i\}$ are totally ordered and $\rhd$ is partially ordered, then $\succ_d$ is a weak order.

Theorem 8 (Weak Completeness of Algorithm 4)**.** *If $\succ_d$ is an interval order, then given a functional composition algorithm f and user preferences $\{\succ'_i\}, \rhd$ over a set of attributes $\mathcal{X}$, Algorithm 4 produces a set θ of feasible compositions such that $\Psi_{\succ_d}(\mathcal{C}) \neq \emptyset \Rightarrow \theta \cap \Psi_{\succ_d}(\mathcal{C}) \neq \emptyset$.*

Proof. From Theorem 7, we have $\theta \subseteq \Psi_{\succ_d}(\mathcal{C})$ when $\succ_d$ is an interval order. It suffices to show that $\Psi_{\succ_d}(\mathcal{C}) \neq \emptyset \Rightarrow \theta \neq \emptyset$. The algorithm terminates with the non-dominated set of compositions in the current list $\mathcal{L}$, i.e., the maximal elements of $\mathcal{L}$ with respect to $\succ_d$. The set of maximal elements of any partial order on the set of elements in $\mathcal{L}$ is not empty whenever $\mathcal{L}$ is not empty, and the set of elements in $\mathcal{L}$ is in turn not empty whenever $\mathcal{C}$ is not empty. Therefore, $\Psi_{\succ_d}(\mathcal{C}) \neq \emptyset \Rightarrow \mathcal{C} \neq \emptyset \Rightarrow \mathcal{L} \neq \emptyset \Rightarrow \theta \neq \emptyset$ as required.

□

Theorem 9 (Completeness of Algorithm 4)**.** *If $\succ_d$ is a weak order, then given a functional composition algorithm f and user preferences $\{\succ'_i\}, \rhd$ over a set of attributes $\mathcal{X}$, Algorithm 4 generates a set θ of feasible compositions such that $\Psi_{\succ_d}(\mathcal{C}) \subseteq \theta$.*

Proof. It suffices to show that there is no feasible composition $D \in \Psi_{\succ_d}(\mathcal{C}) \setminus \theta$.

Suppose by contradiction that $D \in \Psi_{\succ_d}(\mathcal{C})$, and $D \notin \theta$. This means that D was not present in the list $\mathcal{L}$ upon the termination of the algorithm (because otherwise $D \in \theta$ as per Lines $4, 6, 13$ in Algorithm 4). Hence, D must be a feasible extension of some partial feasible composition B that is present in $\mathcal{L}$ at the time of termination such that $B \oplus W_1 \oplus W_2 \oplus \ldots \oplus W_k = D$.

From Proposition 10, we have $V_D \not\succ_d V_B$. Because $\succ_d$ is a weak order, (a) $\forall E \in \theta : V_E \succ_d V_B$; and (b) $V_D \not\succ_d V_B \wedge V_E \succ_d V_B \Rightarrow V_E \succ_d V_D$. However, this contradicts our assumption that $D \in \Psi_{\succ_d}(\mathcal{C})$. □

A summary of the conditions (properties of the dominance relation $\succ_d$) under which the algorithms are sound, complete and weak complete of are given in Table 4.2.

Algorithm	Sound	Weakly Complete	Complete
$A1$	po	po	po
$A2$	po	po	$\|I\| = 1$
$A3$	$\|I\| = 1$	po	$\|I\| = 1$
$A4$	io	io	wo

Table 4.2 Properties of $\succ_d$ for which the algorithms are sound, weakly complete and complete. *po* stands for partial order; *io* stands for interval order; *wo* stands for weak order; and $|I| = 1$ is when there is a unique most important attribute.

4.2.6 Complexity

In this section, we study the complexity of dominance testing (evaluating $\succ_d$, see Chapter 4, Section 4.1.3) as well as the complexity of the algorithms for computing the non-dominated set of feasible compositions. We express the worst case time complexity of dominance testing in terms of the size of user specified intra-attribute, relative importance preference relations and the attribute domains (see Table 4.3).

Computing the maximal(non-dominated)/minimal subset. Let $\succ$ be a partial order on the set S, with a width of w (size of the maximal set of elements which are pairwise incomparable) and $n = |S|$. The algorithm due to Daskalakis et al. discussed in Section 4.2.1 finds the maximal or minimal subset of S with respect to $\succ$ within $O(wn)$ pairwise comparisons. Note that the maximum width of any partial order is $w = n$, when $\succ = \emptyset$. Hence, in the worst case $O(n^2)$ comparisons are needed.

Relation / Set	Symbol	Cardinality	Remarks
Attributes	$\mathcal{X}$	m	Number of attributes
Domain of Attributes	D_i	n	Number of possible valuations of X_i
Intra-attribute preferences	$\succ_i$	w_{int}	Width of the partial order $\succ_i$
Intra-attribute preferences	$\succ_i$	k_{int}	Size of the relation $\succ_i$
Relative Importance	$\rhd$	w_{rel}	Width of the partial order $\rhd$
Relative Importance	$\rhd$	k_{rel}	Size of the relation $\rhd$
Most Important Attributes	I	m_I	Number of most important attributes
Repository	R	r	Number of components in R
Feasible Compositions	$\mathcal{C}$	c	Number of feasible compositions
Dominance Relation	$\succ_d$	w_{dom}	Width of the dominance relation

Table 4.3 Cardinalities of sets and relations

4.2.6.1 Complexity of Dominance Testing

Computing Worst Frontiers (Φ_i). Let $S \subseteq D_i$. Recall from Definition 15 that the worst frontier of a set S with respect to an attribute X_i is $\Phi_i(S) := \{v : v \in S, \nexists u \in S \ s.t. \ v \succ_i u\}$, i.e., the minimal set of elements in S with respect to the preference relation $\succ_i$. Using the algorithm due to Daskalakis et al. to find the minimal set with respect to a partial order (see above), the complexity of computing $\Phi_i(S)$ is $O(nw_{int})$.

Comparing Worst Frontiers ($\succ'_i$). Let $A, B \in \mathscr{F}(X_i)$. As per Definition 17, $A \succ'_i B \Leftrightarrow \forall b \in B, \exists a \in A : a \succ_i b$. In the worst case, computing $A \succ'_i B$ would involve checking whether $a \succ_i b$ for each pair a, b, which would cost $O(k_{int})$. Hence, the complexity of comparing the worst frontiers A and B is $O(n^2 k_{int})$.

Dominance Testing ($\succ_d$). Recall from Definition 20 the definition of dominance:

$$\begin{aligned} \mathcal{U} \succ_d \mathcal{V} \Leftrightarrow \quad & \exists X_i \ : \ \mathcal{U}(X_i) \succ'_i \mathcal{V}(X_i) \ \wedge \\ & \forall X_k, (X_k \rhd X_i \vee X_k \sim_{\rhd} X_i) \ \Rightarrow \ \mathcal{U}(X_k) \succeq'_k \mathcal{V}(X_k) \end{aligned}$$

The complexity of dominance testing is the complexity of finding a witness attribute in $\mathcal{X}$ for $\mathcal{U} \succ_d \mathcal{V}$. For each attribute X_i, the complexity of computing the first clause in the conjunction of the definition of $\mathcal{U} \succ_d \mathcal{V}$ is $O(n^2 k_{int})$; and that of computing the second clause is $O\big(m(n^2 k_{int} + k_{rel})\big)$, where $O(k_{rel})$ and $O(n^2 k_{int})$ are the complexities of evaluating the left and right hand sides of the implication (respectively) for each $X_k \in \mathcal{X}$. Hence, the complexity of dominance testing is $O\big(m\big(n^2 k_{int} + m(n^2 k_{int} + k_{rel})\big)\big)$, or simply $O\big(m^2(n^2 k_{int} + k_{rel})\big)$. We will use the shorthand d to denote $m^2(n^2 k_{int} + k_{rel})$.

4.2.6.2 Complexity of Algorithms

Each of the algorithms for computing the non-dominated feasible compositions make use of a functional composition algorithm f to find feasible compositions. Hence, the complexity analysis of the algorithms needs to incorporate of the complexity of the functional composition algorithm as well.

Recall that Algorithms 1, 2 and 3 begin by computing the set of all feasible compositions in a *single shot* using a functional composition algorithm as a *black box*, and then proceed to find the most preferred among them. Algorithm 4 instead makes use of a functional composition algorithm that produces the set of feasible compositions by iteratively extending partial feasible compositions. Specifically, it *interleaves* the execution of the functional composition algorithm with the ordering of partial solutions with respect to preferences over non-functional attributes.

We denote by $O(f_e)$ and $O(f_g)$ respectively, the complexity of computing the set of feasible extensions of a partial feasible composition with respect to φ and the complexity of computing the set of all feasible compositions with respect to φ.

4.2.6.3 Complexity of Algorithm 1

The overall complexity for finding the set of all non-dominated feasible compositions is $O(f_g + cw_{dom} d)$, where $O(d)$ is the complexity of evaluating $\succ_d$ for any pair of com-

positions. The first term f_g accounts for Line 1 of the algorithm which computes the set of all feasible compositions, and the term $cw_{dom}d$ corresponds to the computation of $\Psi_{\succ_d}(\mathcal{C})$ as per the algorithm given in Section 4.2.1.

4.2.6.4 Complexity of Algorithm 2

The complexity of identifying the most important attributes I with respect to $\rhd$ (Line 1) is $O(mw_{rel}k_{rel})$. For *each* most important attribute $X_i \in I$, Algorithm 2 (a) invokes Algorithm 1 using the derived intra-attribute preference $\succ'_i$ to compute $\Psi_{\succ'_i}(\mathcal{C})$; (b) identifies the subset of $\Psi_{\succ'_i}(\mathcal{C})$ that is non-dominated with respect to $\succ_d$; and (c) adds them to the set of solutions. Hence, the complexity of Algorithm 2 is $O\big(mw_{rel}k_{rel} + m_I(f_g + cw_{dom}n^2k_{int}) + m_I|\Psi_{\succ'_i}(\mathcal{C})|^2d\big)$.

Since the feasible compositions with respect to any given φ are fixed, by computing the feasible compositions only once (during the first invocation of Algorithm 1 and storing them), the complexity of Algorithm 2 can be further reduced to $O(f_g + mw_{rel}k_{rel} + m_I cw_{dom}n^2k_{int} + m_I|\Psi_{\succ'_i}(\mathcal{C})|^2d)$.

4.2.6.5 Complexity of Algorithm 3

The complexity of identifying the most important attributes I with respect to $\rhd$ (Line 1) is $O(mw_{rel}k_{rel})$. In contrast to Algorithm 2, Algorithm 3 invokes Algorithm 1 using the derived intra-attribute preference $\succ'_i$ to compute $\Psi_{\succ'_i}(\mathcal{C})$ for *exactly one* of the most important attributes, $X_i \in I$. Hence, the complexity of Algorithm 3 is $O\big(f_g + mw_{rel}k_{rel} + cw_{dom}n^2k_{int}\big)$.

4.2.6.6 Complexity of Algorithm 4

We consider the worst case wherein the space of partial feasible compositions explored by Algorithm 4 is a tree rooted at $\perp$; let b be its maximum branching factor

(corresponding to the maximum number of extensions produced by the functional composition algorithm), and h its height (corresponding to the maximum number of components used in a composition that satisfies φ). In the worst case, in each iteration of Algorithm 4, every element of $\mathcal{L}$, the list of current partial feasible compositions, ends up in the non-dominated set θ.

Each level in the tree corresponds to one iteration of Algorithm 4, and at the lth iteration, in the worst case there are b^l nodes in $\mathcal{L}$. Hence, the complexity of the lth iteration is $O\big((b^l)^2 d + b^l f_e\big)$, where the first term corresponds to computing the non-dominated set among the current set of partial feasible compositions, and the second term corresponds to computing the feasible extensions of each partial feasible composition. Hence, the overall complexity of Algorithm 4 is $O\big(\sum_{l=0}^{h} (b^{2l} d + b^l f_e)\big) \approx O(b^{2h} d + b^h f_e)$.

We further conducted experiments on the algorithms using simulated problem instances to study how the algorithms perform in practice, which we describe next.

4.3 Related Work

Techniques for representing and reasoning with user preferences over a set of alternatives have been studied extensively in the areas of decision theory, microeconomics, psychology, operations research, etc. The seminal work by vonNeumann and Morgenstern [Morgenstern and Von Neumann, 1944] models user preferences using *utility functions* that map the set of possible alternatives to numeric values. More recently, models for representing and reasoning with quantitative preferences over multiple attributes have been developed [Fishburn, 1970, Keeney and Raiffa, 1993, Bacchus and Grove, 1995, Boutilier et al., 2001]. Such models have been used to address problems such as identifying the most preferred tuples resulting from database queries [Agrawal and Wimmers, 2000, Hristidis and Papakonstantinou, 2004, Börzsönyi et al., 2001], assembling preferred composite Web services [Zeng et al., 2003, Zeng et al., 2004, Yu and Lin,

2005, Berbner et al., 2006], and in other composition problems.

However, in many applications it is more natural for users to express preferences in qualitative terms [Doyle and McGeachie, 2003, Doyle and Thomason, 1999, Dubois et al., 2002] and hence, there is a growing interest in AI on formalisms for representing and reasoning with qualitative preferences [Brafman and Domshlak, 2009]. We now proceed to place our work in the context of some of the recent work on representing and reasoning with qualitative preferences.

4.3.1 TCP-nets

Notable among *qualitative* frameworks for preferences are *preference networks* [Boutilier et al., 2004, Brafman et al., 2006] that deal with qualitative and conditional preferences. A class of preference networks, namely Tradeoff-enhanced Conditional Preference networks (TCP-nets) [Brafman et al., 2006] are closely related to our work, and we now proceed to discuss where our framework departs from and adds to the existing TCP-net framework.

TCP-nets provide a very elegant and compact graphical model to represent *qualitative* intra-attribute and relative importance preferences over a set of attributes. In addition, TCP-nets can also model conditional preferences using dependencies among attributes. While TCP-nets allow us to represent and reason about preferences in general over simple objects (each of which is described by a set of attributes), the focus of our work is to reason about such preferences over *compositions* of simple objects (i.e., a collection of objects satisfying certain functional properties). For example, in the domain of Web services, the problem of identifying the most preferred Web services from a repository of available ones based on their non-functional attributes, namely *Web service selection* can be solved using the TCP-net formalism. On the other hand, in addition to Web service selection, our formalism can also address the more complicated problem of identifying the most preferred composite Web services that collectively satisfy a certain functional

requirement, namely *Web service composition*.

Our formalism is based on the intra-attribute and relative importance preferences over a set of attributes describing the objects. As a result, the graphical representation scheme of TCP-nets can still be used to compactly encode the intra-attribute and relative importance preferences of the users within our formalism [6].

We have extended reasoning about preferences over single objects to enable reasoning about preferences over collections of objects. We have: (a) provided an aggregation function for computing the valuation of a composition as a function of the valuations of its components; (b) defined a dominance relation for comparing the valuations of compositions and established some of its properties; and (c) developed algorithms for identifying a subset or the set of most preferred composition(s) with respect to this dominance relation.

Our formalism departs from TCP-nets in the interpretation of the intra-attribute and relative importance preferences over objects: the dominance relation in a TCP-net is defined as *any* partial order relation that is *consistent* with the given preferences over attributes of the objects, based on the *ceteris-paribus* semantics. We introduce a dominance relation (see Definition 20) that allows us to reason about preferences over *collections* of objects in terms of *sets* of valuations of the attributes of objects that make up the collection. For instance, our worst frontier aggregation function returns the set of worst possible attribute valuations among all the components.

When our dominance relation is applied in the simpler setting where each collection consists of a *single object*, the aggregation function for each attribute reduces to the identity function, and the preference relation $\succ_i'$ over *sets* of valuations of each attribute X_i reduces to the intra-attribute preference $\succ_i$. We have recently shown [Santhanam et al., 2010b, Santhanam et al., 2009a] that in general, when TCP-nets are restricted

[6] In our setting, we do not consider conditional preferences that correspond to edges denoting conditional dependencies in the TCP-nets.

to unconditional preferences, our dominance relation (when each collection consists of a single object) and the dominance relation used in TCP-nets are incomparable; When relative importance is restricted to be an interval order, our dominance relation is more general than the dominance relation used in TCP-nets with only unconditional preferences. In the latter case, our dominance relation is computable in polynomial time, whereas there are no known polynomial time algorithms for computing TCP-net dominance [Santhanam et al., 2010b, Santhanam et al., 2009a].

4.3.2 Preferences over Collections of Objects

Several authors have considered ways to extend user preferences to obtain a ranking of collections of objects (see [Barbera et al., 2004] for a survey). In all these works, preferences are specified over individual objects in a set as opposed to preferences over valuations of the attributes of the objects. The preferences over objects are in turn used to reason about preferences over collections of those objects. This scenario can be simulated by our framework, by introducing a single attribute whose valuations correspond to objects in the domain.

desJardins et al. [desJardins and Wagstaff, 2005] have considered the problem of finding subsets that are optimal with respect to user specified *quantitative* preferences over a set of attributes in terms of the desired *depth*, *feature weight* and *diversity* for each attribute. In contrast, our framework focuses on *qualitative* preferences. In our setting, depth preferences that map attribute valuations to their relative desirability can be mapped to qualitative intra-attribute preferences and *feature weights* can be mapped to relative importance. Diversity preferences over attributes refer to the *spread* (e.g., variance, range, etc.) of component valuations with respect to the corresponding attributes. It would be interesting to explore whether a suitable dominance relation can be defined so as to simultaneously capture in our framework the user preferences with respect to the depth, diversity and feature weights.

More recently, Binshtok et al. [Binshtok et al., 2009] have presented a language for specification of preferences over sets of objects. This framework, in addition to intra-attribute and relative importance preferences over attributes, allows users to express preferences over the *number* ($|\varphi|$) of elements in a set that satisfy a desired property φ. The preference language in this case allows statements such as "$S_i : |\varphi|\ REL\ n$" (number of elements in the preferred set with property φ should be $REL\ n$), "$S_j : |\varphi|\ REL\ |\psi|$" (number of elements in the preferred set with property φ should be REL number of elements in the preferred set with property ψ), etc., where REL is one of the arithmetic operators $>, <, =, \geq, \leq$ and n is an integer. In addition, there can be relative importance between the various preference statements such as "S_i is more important than S_j" as well as *external* cardinality constraints such as a bound on the number of elements in the preferred set.

Our formalism can accommodate such preference statements, by representing each preference statement S_i as a new binary valued attribute in the compositional system. For example, preference statements $S_i : |\varphi|\ \geq\ n$ and $S_j : |\varphi|\ \leq\ |\psi|$ can be represented in our formalism by creating new binary attributes X_i and X_j with intra-attribute preferences $1\ \succ_i\ 0$ and $1\ \succ_j\ 0$ respectively. The relative importance statements such as "S_i is more important than S_j" can then be directly mapped to $X_i \rhd X_j$. Any external cardinality constraints on the size of the preferred set can be encoded in our setting by *functional* requirements, so as to restrict the feasible solutions to only those that satisfy the cardinality constraints.

Consider the example discussed in [Binshtok et al., 2009], with preferences over senate members described by attributes: Party affiliation (Republican, Democrat), Views (liberal, conservative, ultra conservative), and Experience (experienced, inexperienced). The attributes and their domains are listed in Table 4.4. The set preferences are given by:

Property	Denoted by	New Attribute	Attribute Domain
Party Affiliation	P	X_P	$\{Re, De\}$
Views	V	X_V	$\{Li, Co, Ul\}$
Experience	E	X_E	$\{Ex, In\}$

Table 4.4 Properties/Attributes describing the senators

- $S_1 : \langle |P = Re \vee V = Co| \geq 2 \rangle$
- $S_2 : \langle |E = Ex| \geq 2 \rangle$
- $S_3 : \langle |V = Li| \geq 1 \rangle$

Note that the senate members (i.e., the individual objects) are described by three attributes X_P, X_V, X_E representing the party affiliation, views and experience respectively. The valuation function for these attributes is defined in the obvious manner, e.g., if a senator W_j is a republican, then $V_{W_j}(X_P) = Re$. We introduce three additional boolean attributes X_1, X_2, X_3 corresponding to the preference statements S_1, S_2, S_3 respectively. The valuation function for each new attribute of a senator W_i can then be defined as follows.

- $V_{W_i}(X_1) = \begin{cases} 1 \ , \text{ if } W_i \models S_1 \text{ i.e., } V_{W_i}(X_P) = Re \text{ or } V_{W_i}(X_V) = Co \\ 0 \ , \text{ otherwise} \end{cases}$
- $V_{W_i}(X_2) = \begin{cases} 1 \ , \text{ if } W_i \models S_2 \text{ i.e., } V_{W_i}(X_E) = Ex \\ 0 \ , \text{ otherwise} \end{cases}$
- $V_{W_i}(X_3) = \begin{cases} 1 \ , \text{ if } W_i \models S_3 \text{ i.e., } V_{W_i}(X_V) = Li \\ 0 \ , \text{ otherwise} \end{cases}$

The valuation of the collection of senators $W_1 \oplus W_2 \oplus \ldots \oplus W_n$ for $i \in \{1, 2, 3\}$ is:

$$V_{W_1 \oplus W_2 \oplus \ldots \oplus W_n}(X_i) = \Phi_i(V_{W_1}, V_{W_2}, \ldots V_{W_n}) = V_{W_1}(X_i) + V_{W_2}(X_i) + \cdots + V_{W_n}(X_i)$$

Note that the aggregation function Φ_i defined above differs from the worst-frontier based aggregation function adopted in Definition 15. The preference relation for comparing groups of senators with respect to each new attribute X_i can then be defined based on the preference statement S_i. For example, in the case of X_1 we define $\succ_1'$ such that any value ≥ 2 is preferred to any value < 2, etc. Having defined the above aggregation function and comparison relation for each new attribute, any dominance relation can be adopted to compare compositions (arbitrary subsets) with respect to all attributes (including the dominance relation used by Binshtok et al. 2009).

In contrast to the framework of Binshtok et al., our formalism focuses on collections of objects that satisfy some desired criteria, rather than arbitrary subsets. We provide algorithms for finding the most preferred compositions that satisfy the desired criteria.

4.3.3 Database Preference Queries

Several authors [Brafman, 2004, Börzsönyi et al., 2001, Chomicki, 2002, Chomicki, 2003, Kiessling and Kostler, 2002, Kiessling, 2002] have explored techniques for incorporating user specified preferences over the result sets of relational database queries. For instance, Chomicki's framework [Chomicki, 2002, Chomicki, 2003] allows user preferences over each of the attributes of a relation to be expressed as first order logic formulas. Suppose S_q is the set of tuples that match a query q. For each attribute X_i, from S_q, a subset S_{q_i} of tuples that have the most preferred value(s) for X_i is identified. The result set for the query q is then given by $\cap_i S_{q_i}$. A similar framework for expressing and combining user preferences is presented in [Kiessling and Kostler, 2002, Kiessling, 2002]. Brafman and Domshlak have pointed out [Brafman, 2004] some of the semantic difficulties associated with above approaches, and considered an alternative approach to identifying the preferred result set based on the CP-net [Boutilier et al., 2004] dominance

relation. Because of the high computational complexity of dominance testing for CP-nets [Boutilier et al., 2004], they proposed an efficient alternative based on an ordering operator that orders the tuples in the result set in a way that is consistent with the user preferences. Our formalism can be used in the database setting, similar in spirit to that of Brafman and Domshlak, by considering each tuple in S_q as a collection with a single object. The differences in the semantics of the CP-net dominance and our dominance relation is discussed in Section 4.3.1.

A host of algorithms have also been proposed for computing the non-dominated result set in response to preference queries, especially for the efficient evaluation of *skyline* queries. A skyline query yields the non-dominated result set from a database, where dominance is evaluated based on the notion of *pareto dominance*. Pareto dominance requires that the dominating element is preferred or equal to the dominated element with respect to all attributes (with respect to their respective intra-attriubute preference), and is strictly preferred to the dominated element with respect to at least one attribute. Another aspect of pareto dominance is that it does not consider the relative importance preferences of the user over the attributes, i.e., it considers all attributes to be equally important.

Most of the proposed algorithms for computing the skylines (e.g., Blocked-nested-loop (BNL) [Börzsönyi et al., 2001], Sort-filter-skyline (SFS) [Chomicki, 2003]; see [Jain, 2009, Preisinger, 2009] for a survey) are applicable only when intra-attribute preferences are totally or weakly ordered. Some other algorithms have been recently developed to handle partially ordered attribute domains (such as BNL+, BBS+, SDC, SDC+ [Chan et al., 2005], TSS [Sacharidis et al., 2009] and FAST-SKY [Jung et al., 2010]). These algorithms rely on creating and maintaining indexes over the attributes in the database, and on data structures specifically designed to identify the skyline with respect to pareto dominance. These algorithms may be considered if a particular problem instance involves such a large set of components are already stored in a database and indexed. However,

it is not obvious that they generalize to an arbitrary notion of dominance such as the one presented here. On the other hand, our algorithms for finding the non-dominated set are applicable to any notion of dominance, provided the user preferences are such that the dominance relation is a partial order.

4.4 Summary

Many applications, e.g., planning, Web service composition, embedded system design, etc., rely on methods for identifying collections (compositions) of objects (components) satisfying some functional specification. Among the compositions that satisfy the functional specification (feasible compositions), it is often necessary to identify one or more compositions that are most preferred with respect to user preferences over non-functional attributes. Of particular interest are settings where user preferences over attributes are expressed in qualitative rather than quantitative terms [Doyle and Thomason, 1999].

In this chapter, we have proposed a framework for representing and reasoning with qualitative preferences over compositions in terms of the qualitative preferences over attributes of their components; and developed a suite of algorithms to compute the most preferred feasible compositions, given an algorithm that computes the functionally feasible compositions. Specifically, we have:

a) Defined a generic *aggregation function* to compute the valuation of a composition as a function of the valuations of its components. We have also presented a strict partial order preference relation for comparing two compositions with respect to their aggregated valuations of each attribute;

b) Introduced a *dominance* relation for comparing compositions based on user specified preferences and established some of its key properties. In particular, we have

shown that this dominance relation is a strict partial order when intra-attribute preferences are strict partial orders and relative importance preferences are interval orders.

c) Developed four algorithms for identifying the most preferred composition(s) with respect to the user preferences. The first three algorithms first compute the set of all feasible compositions (solutions) using a functional composition algorithm as a *black box*, and then proceed to find the most preferred among them (1) based on the dominance relation (*ComposeAndFilter*); and (2) based on the preferred valuations with respect to the most important attribute(s) (*WeaklyCompleteCompose* and *AttWeaklyCompleteCompose*). The fourth algorithm interleaves the execution of a functional composition algorithm that produces the set of solutions by iteratively extending partial solutions and the ordering of partial solutions with respect to user preferences (*InterleaveCompose*).

d) Established some key properties of the above algorithms. *ComposeAndFilter* is guaranteed to return the set of all non-dominated solutions; *WeaklyComplete-Compose* is guaranteed to return a non-empty subset of non-dominated solutions; *AttWeaklyCompleteCompose* is guaranteed to return at least one of the non-dominated solutions; and *InterleaveCompose* is guaranteed to return (i) a non-empty subset of non-dominated solutions when the dominance relation is an interval order; and (ii) the entire set of non-dominated solutions when the dominance relation is a weak order.

The proposed techniques for reasoning with preferences over non-functional attributes are independent of the language used to express the desired functionality φ of the composition, and the method used to check whether a composition C satisfies the desired functionality, i.e., $C \models \varphi$. Our formalism and algorithms may be applicable to a broad range of domains including Web service composition (see [Dustdar and Schreiner, 2005, Pathak

et al., 2008a] for surveys), planning (see [Hendler et al., 1990, Baier and McIlraith, 2008]), team formation (see [Lappas et al., 2009, Donsbach et al., 2009]) and indeed any setting that calls for choosing the most preferred solutions from a set of candidate solutions, where each solution is made up of multiple components.

4.5 Discussion

Our preference formalism is generic and flexible in terms of the choice of aggregation functions and the overall dominance. In the following, we discuss some of the alternate choices that one could make in applying our formalism for specific applications.

Aggregation Functions. In our previous work [Santhanam et al., 2008], we had proposed the use of TCP-net representation with ceteris paribus semantics [Brafman et al., 2006] for reasoning with preferences in addressing the problem of Web service composition. We had assumed that the intra-attribute preferences are total orders; however, this assumption does not hold in many practical settings involving qualitative preferences over non-functional attributes. In this chapter, we have relaxed this requirement, allowing intra-attribute preferences that are strict partial orders.

In this chapter we demonstrated the use of the summation (e.g., number of credits in a POS) and worst frontier (e.g., areas of study and instructors) aggregation functions. In some scenarios, it might be necessary to consider other ways of aggregating the valuations of the components, for example, using the *best frontier* denoting the best possible valuations of the components (i.e., the maximal valuations for each attribute X_i with respect to $\succ_i$). Any aggregation function can be used in our formalism, provided that the preference relation over the aggregated valuations is a strict partial order. Otherwise, the choice of aggregation function and the preference relation to compare aggregated valuations may impact the properties of the resulting dominance relation, and as a result, may also affect the soundness and completeness properties of some of

the proposed algorithms.

The aggregation functions demonstrated in this chapter are independent of *how* the components interact or are assembled, i.e., the *structure* of a composition. However, in general, it may be necessary for the aggregation function to take into account the structure and/or other interactions between the valuations of components in a composition. For example, in evaluating the reliability of a composition, one needs to consider the precise structure of the composition. The reliability of a composition C_i is the product of the reliabilities of the components ($\prod_{i=1}^{n} V_{W_i}(Reliability)$) when the components are arranged in a series configuration [Rausand and Høyland, 2003]. On the other hand, when the same set of components $\{W_i\}$ are arranged in a parallel configuration, the reliability of C_i is computed as $(1 - \prod_{i=1}^{n}(1 - V_{W_i}(Reliability)))$. In general, it might be necessary to introduce aggregation functions that take into consideration a variety of factors including the structure, the function, as well as the non-functional attributes of the composition.

Comparing Sets of Aggregated Valuations. In this chapter, we presented a preference relation ($\succ'_i$) to compare sets of valuations computed using the worst frontier aggregation function (Definition 17). This preference relation requires that given two sets of valuations, every element in the dominated set is preferred to at least one of the elements in the dominating set of valuations. Other choices of $\succ'_i$ can be used as well, but care should be taken because the properties of the chosen preference relation may affect the properties of the dominance ($\succ_d$) relation and the properties of the algorithms. However, as long as $\succ'_i$ is a strict partial order (irreflexive and transitive), the dominance relation continues to remain a strict partial order (subject to $\rhd$ being an interval order), and hence the properties of the algorithms hold. This provides the user with a wide range of preference relations for comparing sets of valuations to choose from (see [Barbera et al., 2004] for a survey of preferences over sets).

Dominance and its Properties. The dominance relation ($\succ_d$) adopted in this chapter is a strict partial order when the intra-attribute preferences are arbitrary strict partial orders and the relative importance is an interval order. It would be interesting to explore alternative notions of dominance that preserve the rationality of choice, by requiring a different set of properties (e.g., those that satisfy *negative-transitivity* instead of transitivity). It would also be of interest to examine the relationships between $\succ_d$ and alternative dominance relations. Some results comparing $\succ_d$ with the dominance relations proposed in [Brafman et al., 2006, Wilson, 2004b, Wilson, 2004a] can be found in [Santhanam et al., 2010b, Santhanam et al., 2009a].

CHAPTER 5. Preference Reasoning for Compositional Systems: Experiments & Results

In this chapter, we present results of experiments we performed to compare the algorithms developed in Chapter 4 in terms of the following criteria.

a) **Quality of solutions produced by the algorithms.** The quality is measured in terms of the percent of the most preferred solutions that an algorithm produces, and the percent of the overall set of solutions produced by an algorithm that are most preferred.

b) **Performance and efficiency of the algorithms.** The performance of an algorithm is measured in terms of response time (time taken to return the set of solutions), and the efficiency is measured in terms of the number of times an algorithm invokes the functional composition algorithm.

5.1 Experimental Setup

5.1.1 Modeling the Search Space of Compositions using Recursive Trees

The *uniform recursive tree* [Smythe and Mahmoud, 1995] serves as a good choice to model the search space of partial compositions and their feasible extensions. A tree with n vertices labeled by $1, 2, \ldots n$ is a *recursive tree* if the node labeled 1 is distinguished as the root, and $\forall k : 2 \leq k \leq n$, the labels of the nodes in the unique path from the root to the node labeled with k form an increasing sequence. A *uniform recursive tree*

of n nodes (denoted $URTree(n)$) is one that is chosen with equal probability from the space of all such trees.

A simple growth rule can be used to generate a uniform random recursive tree of n nodes, given such a tree of $n-1$ nodes: Given $URTree(n-1)$, choose uniformly at random a node k in $URTree(n-1)$, and add a node labeled n with k as parent to obtain $URTree(n)$. The properties of this class of uniform random recursive trees are well studied in the literature of random data structures (see [Smythe and Mahmoud, 1995] for a survey).

The rationale behind choosing the uniform recursive tree data structure to model the search space of our problem is that the growth rule that generates the recursive tree is similar in intuition to the process of searching for a feasible composition. Recall that the search space of partial compositions is generated by the recursive application of the functional composition algorithm f. The nodes in the recursive tree correspond to components in the repository of the composition problem. The tree is built starting with the root node – the search for feasible compositions correspondingly begins with $\perp$. The recursive tree is further grown by attaching new nodes to any of the existing nodes – this corresponds to extending feasible partial compositions by adding (composing) new components to any of the existing feasible partial compositions. Finally, the leaves of a recursive tree at depth d from the root correspond to a (possibly feasible) composition of d components from the repository in the composition problem.

We now show the precise correspondence between a recursive tree data structure and a search space of partial compositions.

- Each node k in the tree corresponds to a composition $C(k)$. We denote the parent of a node k as $Pa(k)$.
- The root node 1 corresponds to the empty composition, i.e., $C(1) = \perp$,
- Each node k at level 1 corresponds to a composition of $\perp$ with a component W in

the repository, i.e., $C(k) = \perp \oplus W = W, W \in R$,

- Each node k at level i corresponds to a composition of $C(Pa(k))$ with a component W in the repository, i.e., $C(k) = C(Pa(k)) \oplus W, W \in R$,
- A leaf node l corresponding to a composition $C(l)$ is called a *feasible node* if $C(l) \models \varphi$.

For the purpose of experimentally evaluating our algorithms for finding the most preferred compositions and to compare them, we generate random recursive trees with varying number of nodes (or $|R|$, the number of components in the repository). In the generated random recursive tree, a certain fraction ($feas$) of leaves are picked uniformly randomly and are labeled to be feasible compositions. For each node in the generated and labeled random recursive tree, the valuation of attributes $\mathcal{X} = \{X_i\}$ (corresponding to the partial composition it represents) is randomly generated based on the respective domains ($\{D_i\}$).

User Preferences

We generate user preferences by generating random partial/total orders for each $\succ_i$ and random interval/total order for $\rhd$ for varying number of attributes $m = |\mathcal{X}|$ and domain size of attributes $n = |D_i|$.

A summary of the simulation parameters is given in Table 5.1.

5.1.2 Implementation of Algorithms

Computing Dominance

In order to check if one valuation dominates another with respect to the user preferences $\{\succ_i\}$ and $\rhd$, we iterate through all attributes $\mathcal{X}$ and check if there exists a witness for the dominance to hold (see Definition 20).

Parameter	Meaning	Range
$feas$	Fraction of leaves in the search tree that are feasible compositions	$\{0.25, 0.5, 0.75, 1.0\}$
$\|D_i\|$	Domain size of preference attributes	$\{2, 4, 6, 8, 10\}$
$\|\mathcal{X}\|$	Number of preference attributes	$\{2, 4, 6, \ldots 20\}$
$\|R\|$	Number of components in the repository (nodes in the search tree)	$\{10, 20, \ldots 200\}$
T_f	Overhead (in milliseconds) per invocation of the step-by-step functional composition algorithm f	$\{1, 10, 100, 1000\}$
$\succ_i$	Intra-attribute preference over the values of X_i	$\{po, to\}$
$\rhd$	Relative importance preference over $\mathcal{X}$	$\{io, to\}$

Table 5.1 Simulation parameters and their ranges

Computing the most preferred solutions

We implemented algorithms $A1$, $A2$, $A3$ and $A4$ in Java. Preliminary experiments with $A2$ showed that the algorithm did not scale up for large problem instances. In particular, when the number of attributes are large and dominance testing is computationally intensive, $A2$ timed out due to the computation of the non-dominated set multiple times for each of the most important attributes. Hence we did not proceed to run experiments on the samples with $A2$. However, we were able to run experiments with algorithm $A3$ that arbitrarily picks *one* of the most important attributes and finds the most preferred solutions with respect to the intra-attribute preferences of that attribute.

In algorithms $A1$ and $A3$ we first compute all solutions using the functional composition algorithm (simulated by f), whereas in $A4$, we interleave calls to f with choosing preferred compositions (partial solutions) at each step. At each step, $A4$ chooses a subset of the feasible extensions of the current compositions for further exploration. Table 5.2 gives a brief description of the implemented algorithms.

Table 5.3 shows the attributes that are recorded during the execution of each of the algorithms $A1, A3$ and $A4$ for each composition problem.

Alg.	Name of Algorithm	Remarks
$A1$	$ComposeAndFilter$	First identifies functionally feasible compositions; then finds the non-dominated set of feasible compositions with respect to $\succ_d$
$A2$	$WeaklyCompleteCompose$	First identifies functionally feasible compositions; then finds the non-dominated set of feasible compositions with respect to $\succ_i$ for the most important attributes $\{X_i\}$
$A3$	$AttWeaklyCompleteCompose$	First identifies functionally feasible compositions; then picks an arbitrary most important attribute X_i and finds the non-dominated set of feasible compositions with respect to $\succ'_i$
$A4$	$InterleaveCompose$	Identifies the non-dominated set of feasible extensions with respect to $\succ_d$ at each step; and recursively identifies their feasible extensions until all the non-dominated feasible extensions are feasible compositions

Table 5.2 Implemented Algorithms

5.2 Results

We compare the algorithms $A1, A3, A4$ with respect to:

1. Quality of solutions produced by the algorithms, in terms of SP/PF and SP/S
2. Performance and efficiency in terms of running time and number of calls to the functional composition algorithm f

5.2.0.1 Quality of Solutions

We compare the quality of solutions produced by the algorithms in terms of the following measures.

Attribute	Meaning	Remarks
F	Set of solutions (feasible compositions) in a sample problem instance	$F = \mathcal{C}_\varphi$
PF	Set of most preferred solutions in a sample problem instance with respect to the user preferences and the dominance relation	$PF = \Psi_{\succ_d}(\mathcal{C}_\varphi) \subseteq F$
S	Set of solutions produced by the composition algorithm	
SP	Set of solutions produced by the composition algorithm that are also most preferred solutions with respect to the user preferences and the dominance relation	$SP = PF \cap S \subseteq S$
T	Running time of the composition algorithm (ms)	
$fcount$	Number of times the algorithm invokes the step-by-step functional composition algorithm f	

Table 5.3 Attributes observed during the execution of each algorithm

- SP/PF[1]: Proportion of most preferred solutions produced by the algorithm (fraction of all optimal solutions produced by the algorithm). If the algorithm is complete, then $SP/PF = 1$.

- SP/S: Proportion of solutions produced by the algorithm that are most preferred (fraction of solutions produced solutions, that are optimal). If the algorithm is sound, then $SP/S = 1$.

The algorithm $A1$ exhaustively searches the entire space of compositions to identify all the feasible compositions F, and then finds the most preferred among them with respect to the user preferences $\rhd$ and $\{\succ_i\}$. Because it computes the set $\Psi_{\succ_d}(F)$, we observed that for $A1$, $SP = PF = S$, i.e., it is both sound (finds only the most preferred solutions) and complete (finds all the most preferred solutions).

[1]For the sake of readability, we use the notation used to denote the set to denote its cardinality as well, e.g., SP is used to denote both the set and its cardinality ($|SP|$).

We next compare the algorithms $A3$ and $A4$ with respect to SP/PF and SP/S for various types of ordering restrictions on the user preferences $\{\succ_i\}$ and $\rhd$. Table 5.4 reports results for the following combinations: (i) $\rhd$ is an interval order, $\{\succ_i\}$ are partial orders; (ii) $\rhd$ is an interval order, $\{\succ_i\}$ are total orders; (iii) $\rhd$ is a total order, $\{\succ_i\}$ are partial orders; and (iv) $\rhd$ is a total order, $\{\succ_i\}$ are total orders.

Figures 5.1 and 5.2 show a comparison of the algorithms $A3$ and $A4$ with respect to the proportion of most preferred solutions produced by the algorithms (SP/PF) and proportion of solutions produced by the algorithms that are most preferred (SP/S), when the intra-attribute preferences are partial orders and relative importance is an interval order ((a) - (b)), or a total order ((c) - (d)) respectively.

$\rhd$	$\succ_i$	$A3$	$A4$
io	*po*	77.50	83.95
io	*to*	71.00	100.00
to	*po*	100.00	85.88
to	*to*	100.00	100.00

Table 5.4 Comparison of SP/PF for algorithms $A3$ and $A4$ with respect to various ordering restrictions on $\{\succ_i\}, \rhd$. The percent of problem instances for which $SP/PF = 1$ is shown in each row with respect to the corresponding ordering restrictions on the preference relations $\rhd$ and $\{\succ_i\}$. Plots from simulation experiments are shown in Figures 5.1 and 5.2.

Comparison of SP/PF

- In general, most of the most preferred solutions were found by both the algorithms (see Table 5.4).

- We observe that when relative importance ($\rhd$) is a total order and $\{\succ_i\}$ are arbitrary partial orders, 100% of the most preferred solutions are produced by $A3$. This suggests that there may be some relationship between $\succ_i'$ and $\succ_d$ under

this condition. The precise implications of this observation is explored later in Section 5.3.

$\rhd$	$\succ_i$	$A3$	$A4$
io	*po*	41.78	98.45
io	*to*	30.78	100.00
to	*po*	33.90	96.98
to	*to*	27.30	100.00

Table 5.5 Comparison of SP/S for algorithms $A3$ and $A4$ with respect to various ordering restrictions on $\{\succ_i\}, \rhd$. The percent of problem instances for which $SP/S = 1$ is shown in each row with respect to the corresponding ordering restrictions on the preference relations $\rhd$ and $\{\succ_i\}$. Plots from simulation experiments are shown in Figures 5.1 and 5.2.

Comparison of SP/S

- In general, most of the solutions that were found by the interleaved algorithm $A4$ were the most preferred solutions (see Table 5.5). On the other hand, algorithm $A3$ produced many solutions that were not the most preferred.

- The second (and fourth) row(s) of Tables 5.5 and 5.4 suggests that when intra-attribute preferences ($\{\succ_i\}$) are total orders and $\rhd$ is an arbitrary interval order, the interleaved algorithm $A4$ is sound and complete, i.e., it produces exactly the non-dominated set of solutions with respect to $\succ_d$. In the light of Theorems 7 and 9, this observation suggests a possible relationship between the properties of $\succ_d$ and $\{\succ_i\}, \rhd$ which we will explore in further detail in Section 5.3.

5.2.0.2 Performance and Efficiency

We compare the performance and efficiency of $A3, A4$ in terms of the number of times the functional composition algorithm f is invoked, and running time (in milliseconds)

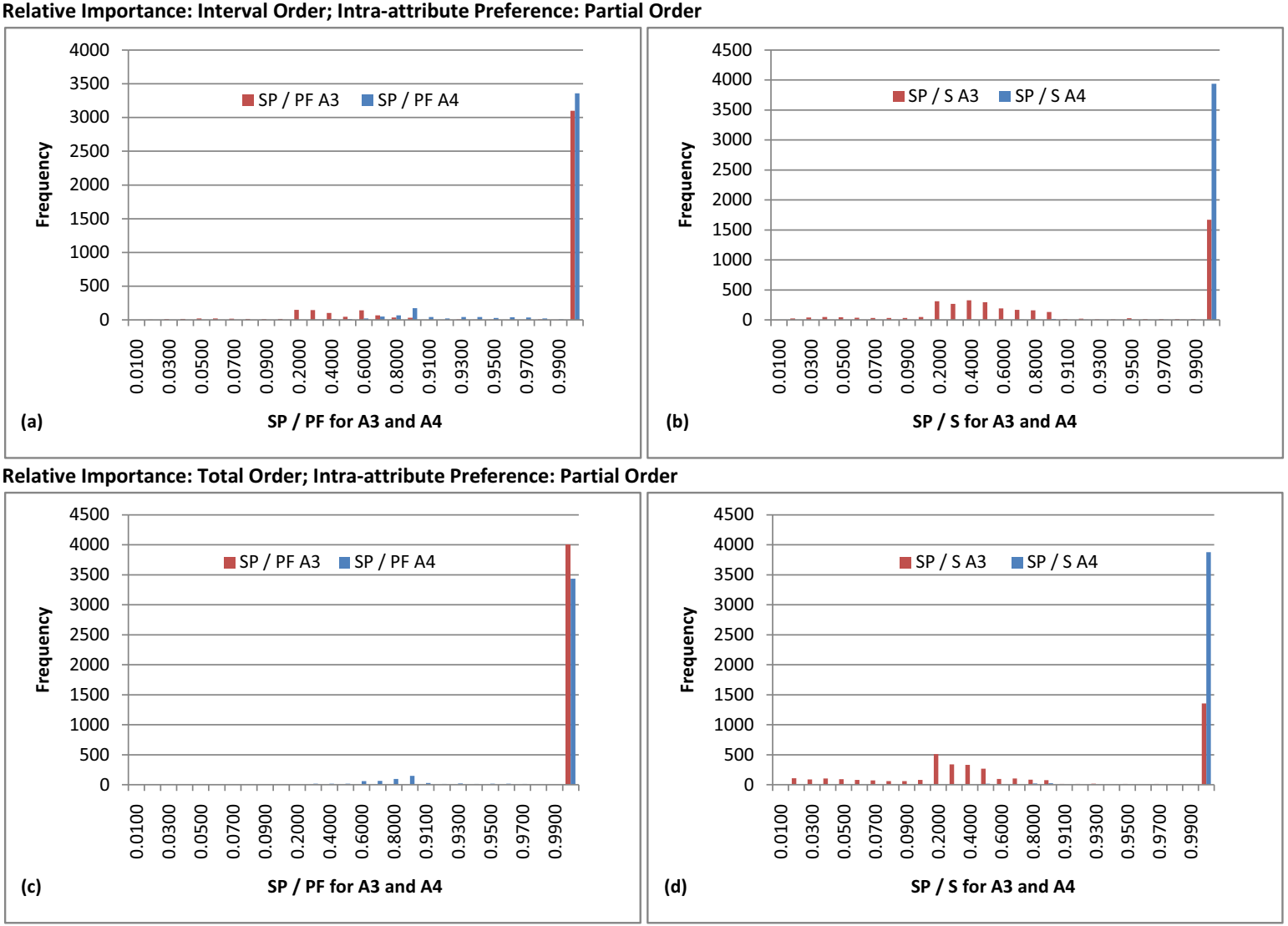


Figure 5.1 A comparison of the algorithms $A3$ and $A4$ with respect to SP/PF and SP/S.

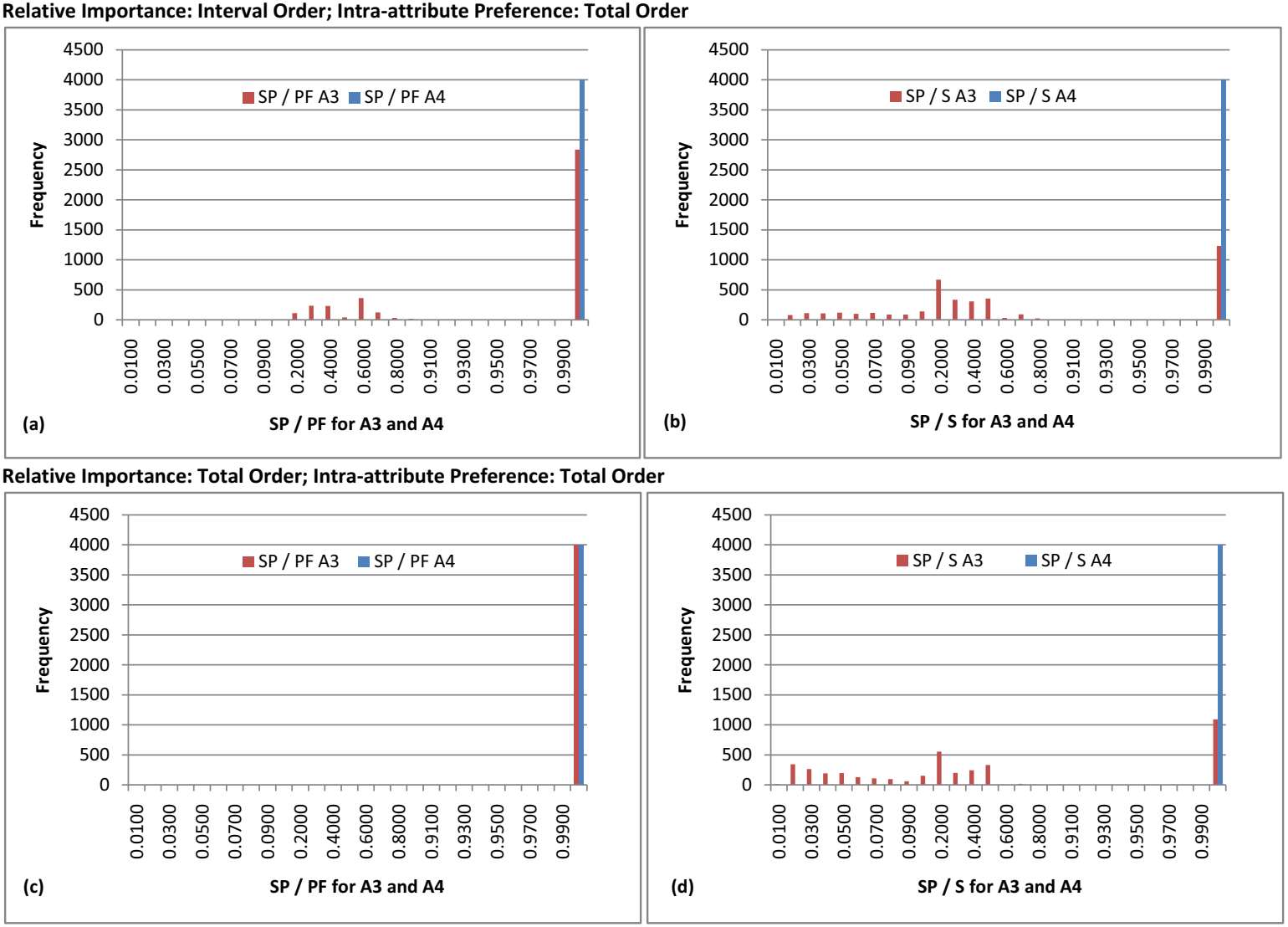


Figure 5.2 A comparison of the algorithms $A3$ and $A4$ with respect to SP/PF and SP/S.

for the algorithms to compute their solutions.

Number of calls to functional composition f

The plots in Figure 5.3 show the results of our experiments comparing the algorithms $A1$, $A3$ and $A4$ with respect to the number of times they invoke the step-by-step functional composition algorithm during their execution under various ordering restrictions on the intra-attribute preferences ($\succ_i$) and relative importance ($\rhd$). The four distinct "bands" seen in the plots correspond to various fractions of leaves in the search tree of the problem instance that are feasible compositions: $feas = 0.25, 0.5, 0.75, 1.0$.

The results yield the following observations.

- In general, our experiments show that the interleaved algorithm $A4$ makes lesser number of calls to f compared to $A3$ (see Figure 5.3). This is because $A4$ explores only the most preferred subset of the available feasible extensions at each step in the search. On the other hand, $A3$ exhaustively explores all feasible extensions at each step.

- When the intra-attribute preferences $\{\succ_i\}$ are total orders, the difference in the number of calls to f made by $A3$ and $A4$ is more pronounced. This can be explained by the fact that in this case the dominance relation is larger, due to which the number of incomparable pairs of compositions is smaller. Therefore, at each interleaving step the non-dominated set computed for extension is smaller.

- In the case of both the algorithms $A3, A4$, the number of calls to f decreases as the fraction of feasible compositions ($feas$) increases.

Running time

We observed that the running times of both algorithms $A3, A4$ are dependent on two key factors:

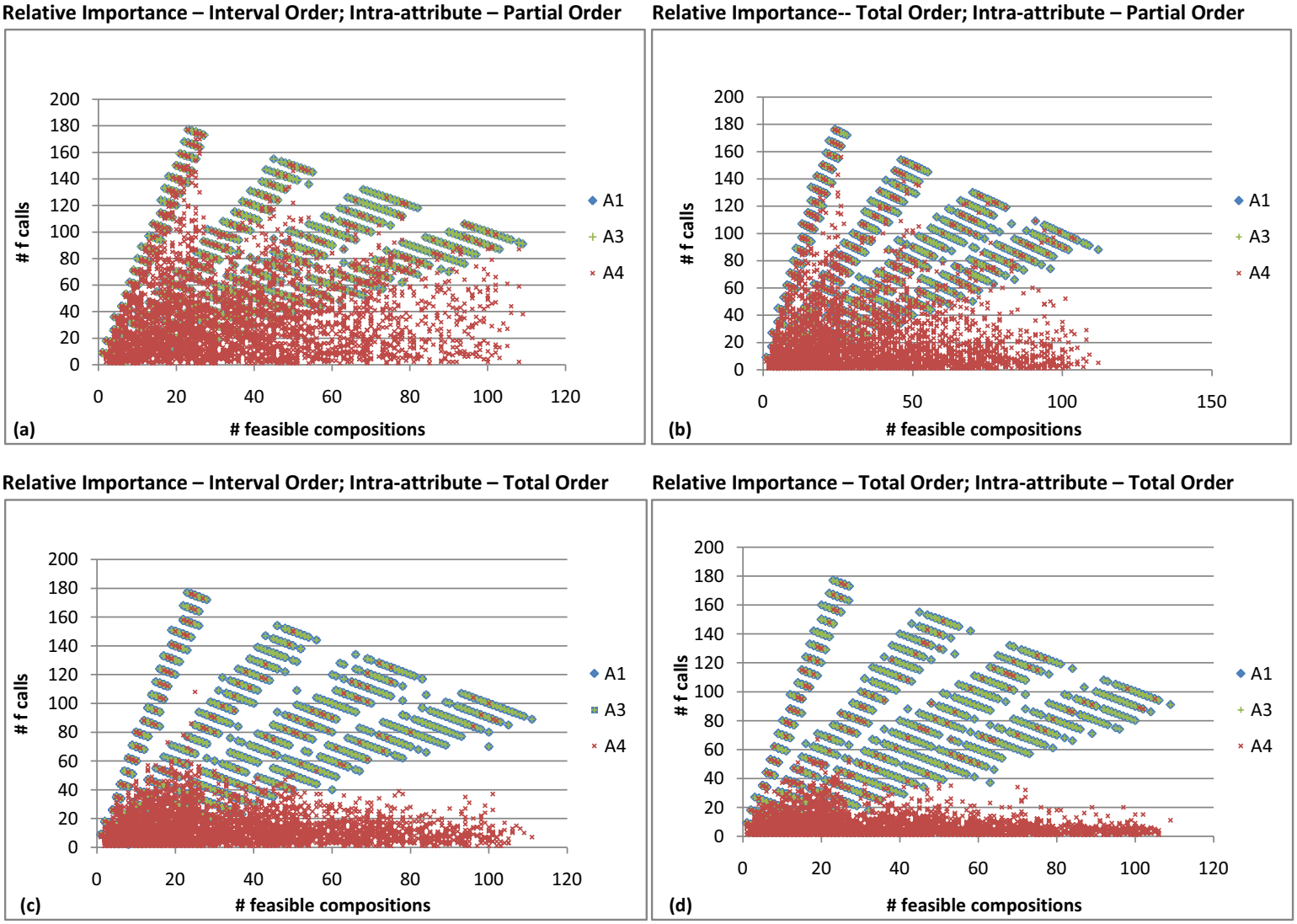


Figure 5.3 Number of invocations of functional composition algorithm for $A1$, $A3$ and $A4$ when $feas = 0.25, 0.5, 0.75, 1.0$.

- $fdelay$, the time taken per execution of the functional composition at each step
- Complexity of dominance testing which is in turn a function of $|D_i|$, $|\mathcal{X}|$ and the properties of $\{\succ_i\}, \rhd$ (see Section 4.2.6.1).

In order to understand the effect of $fdelay$ on the running times of the algorithms, we ran experiments with $fdelay = 10ms$ and $fdelay = 1000ms$. Figures 5.4 and 5.5 show a comparison of the algorithms $A1$, $A3$ and $A4$ with respect to their running times as a function of the number of feasible compositions, when the each invocation step in the step-by-step functional composition algorithm has a overhead of 10 milliseconds and 1000 milliseconds respectively. The plots (a) - (d) correspond to results of running the algorithms on simulated problem instances with various ordering restrictions on the intra-attribute preferences ($\succ_i$) and relative importance ($\rhd$). The four distinct "bands" seen in the plots correspond to various fractions of leaves in the search tree of the problem instance that are feasible compositions: $feas = 0.25, 0.5, 0.75, 1.0$.

The results yield the following observations.

- In general, the interleaved algorithm $A4$ is faster when the intra-attribute preferences ($\succ_i$) are total orders, as compared to the case when they are partial orders.
- The algorithm $A3$ almost always outperforms the blind search algorithm $A1$ in terms of running time, because $A3$ computes the non-dominated set in the last step with respect to the intra-attribute preference over the valuations of one attribute $\succ_i'$ (in place of the dominance relation $\succ_d$ used by $A1$).
- The interleaved algorithm $A4$ is more sensitive to the complexity of dominance than $A1$ and $A3$, because at each step $A4$ computes the non-dominated subset of extensions to explore. On the other hand, $A1$ and $A3$ involve computation of dominance only in the last step. $A3$ is faster than $A1$, more than $A4$, because it computes the non-dominated set with respect to the intra-attribute preference

over the valuations of one attribute $\succ_i'$ (in place of the dominance relation $\succ_d$ used by $A1$ and $A4$).

- Algorithms $A1$ and $A3$ are more sensitive to $fdelay$ than the interleaved algorithm $A4$, because at each step $A1$ and $A3$ explore all feasible extensions. On the other hand, $A4$ only explores the preferred subset of the feasible extensions at each step.

- The overall running time of $A1$, $A3$ and $A4$ depend on the relative trade-offs among $|D_i|$, $|\mathcal{X}|$, the properties of $\{\succ_i\}, \rhd$ (those that influence the complexity of dominance testing) on the one hand and $fdelay$ on the other.

5.3 Analysis of Experimental Results

Apart from demonstrating the feasibility of our algorithms in various settings, the experimental results also point us towards possible relationships between the properties of the algorithms and dominance relation under various restrictions on the intra-attribute and relative importance preferences. In particular, a closer observation of the quality of solutions produced by the algorithms (see Tables 5.4 and 5.5) under various ordering restrictions on the user preferences lead us to the following conjectures/results.

- When relative importance ($\rhd$) is a total order and $\{\succ_i\}$ are arbitrary partial orders, 100% of the most preferred solutions are produced by $A3$ (Table 5.4). This suggests that $\succ_i' \subseteq \succ_d$, because in such a case $A3$ would be complete (in the sense that it produces all the most preferred solutions, or those that are non-dominated with respect to $\succ_d$). In the following, we prove Propositions 17 and 18 based on this insight.

- Table 5.5 suggests that when intra-attribute preferences ($\{\succ_i\}$) are total orders and $\rhd$ is an arbitrary interval order, the interleaved algorithm $A4$ is sound and

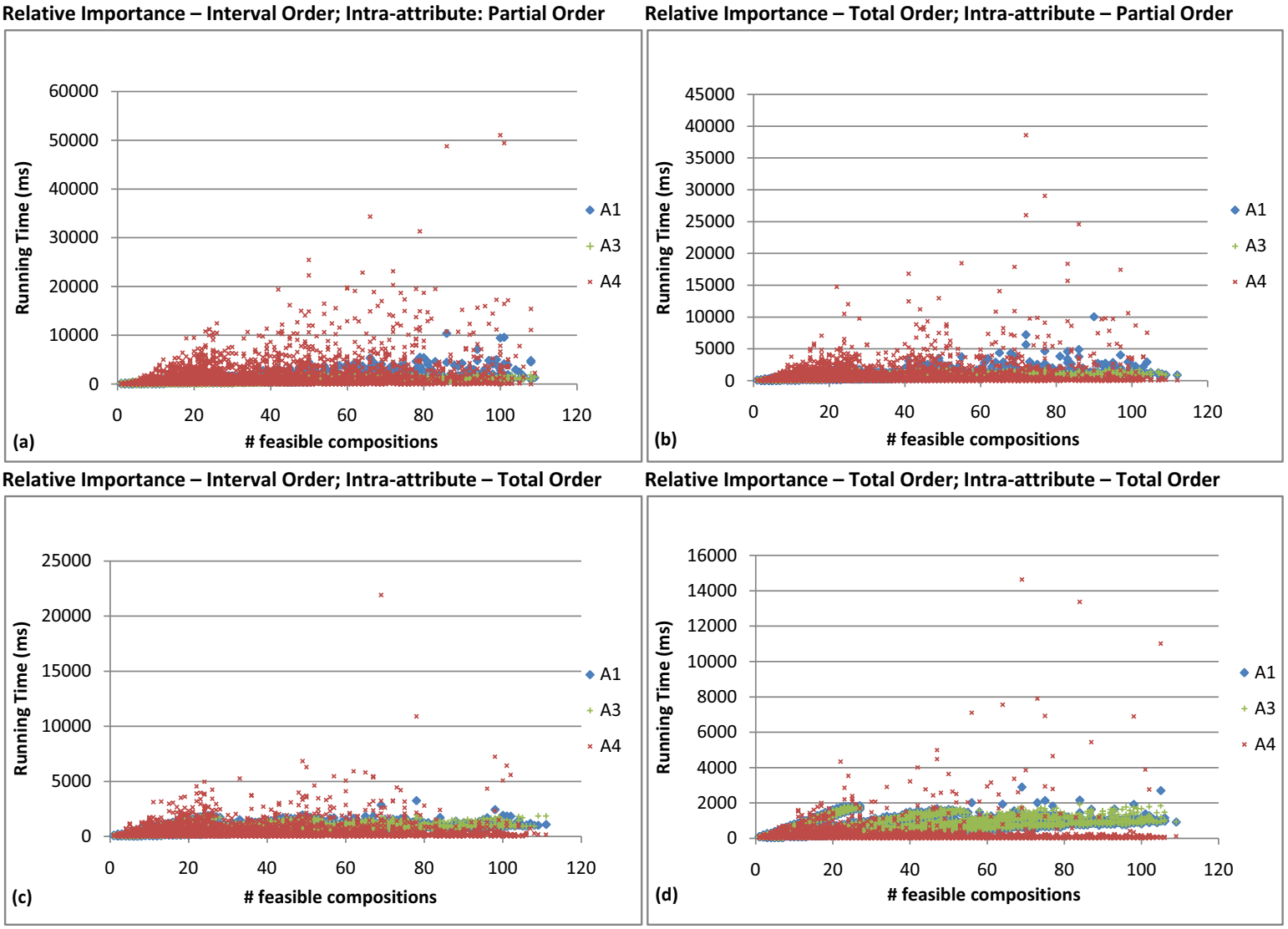


Figure 5.4 Running times of algorithms $A1$, $A3$ and $A4$ with 10 milliseconds per functional composition step for $feas = 0.25, 0.5, 0.75, 1.0$.

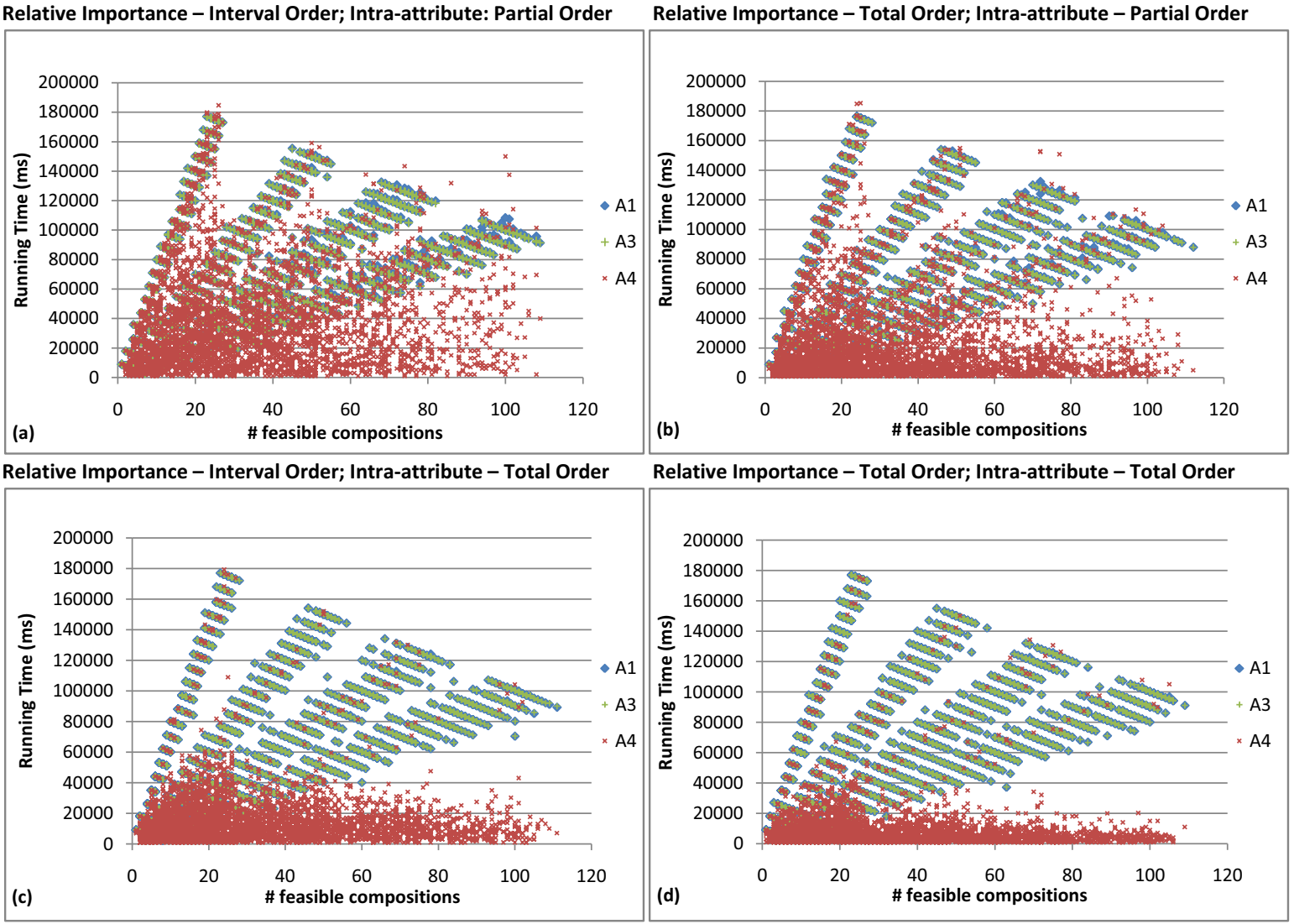


Figure 5.5 Running times of algorithms $A1$, $A3$ and $A4$ with 1000 milliseconds per functional composition step for $feas = 0.25, 0.5, 0.75, 1.0$.

complete, i.e., it produces exactly the non-dominated set of solutions with respect to $\succ_d$. Secondly, Theorems 7 and 9 give sufficient conditions for the soundness and weakness of $A4$ in terms of the properties of $\succ_d$: $A4$ is complete if $\succ_d$ is interval ordered, and $A4$ is sound if $\succ_d$ is a weak order. These two observations suggest a direct relationship between the properties of $\succ_d$ and $\{\succ_i\}$, $\rhd$. This leads us to a conjecture (Conjecture 1), part of which we prove (Theorem 10) in the following.

Proposition 17. *If $\rhd$ is a total order and X_i is the most important attribute in $\mathcal{X}$ with respect to $\rhd$, then $\succ_i' \subseteq \succ_d$.*

Proof. Let X_i be the (unique) most important attribute in $\mathcal{X}$. Suppose that $\mathcal{U}(X_i) \succ_i' \mathcal{V}(X_i)$, thereby making X_i a potential witness for $\mathcal{U} \succ_d \mathcal{V}$. Since X_i is the most important attribute, $\forall X_k \in \mathcal{X} : X_i \rhd X_k$, the second clause in the definition of $\mathcal{U} \succ_d \mathcal{V}$ trivially holds. Hence, X_i is a witness for $\mathcal{U} \succ_d \mathcal{V}$ (see Definition 20). □

Note that the proof of the above proposition only made use of the fact that $\forall X_k \in \mathcal{X} : X_i \rhd X_k$, which is a weaker condition than $\rhd$ being a total order. Hence, we have the following more general result.

Proposition 18. *If $\rhd$ is such that there is a unique most important attribute X_i, i.e., $\exists X_i \in \mathcal{X} : \forall X_k \in \mathcal{X} \setminus \{X_i\} : X_i \rhd X_k$, then $\succ_i' \subseteq \succ_d$.*

Because $\succ_i' \subseteq \succ_d \Rightarrow \Psi_{\succ_d}(S) \subseteq \Psi_{\succ_i'}(S)$ for any set S, the above proposition also gives a condition under which $A3$ is complete. This is not surprising since under this condition $A3$ is the same as $A2$ (see Section 4.2.4), and hence the completeness of $A3$ follows from Proposition 14.

Conjecture 1. *If $\{\succ_i\}$ are total orders and $\rhd$ is an arbitrary interval order, then $\succ_d$ is an weak order.*

We now prove a restricted version of the above conjecture, when both $\rhd$ as well as $\{\succ_i\}$ are total orders.

Theorem 10. *If $\rhd$ as well as $\{\succ_i\}$ are total orders, then $\succ_d$ is a weak order.*

Proof. $\succ_d$ is a weak order if and only if it is a strict partial order and negatively transitive. We have already shown that $\succ_d$ is a strict partial order in Theorem 1, and hence we are only left with proving that $\succ_d$ is negatively transitive, i.e., $\mathcal{U} \not\succ_d \mathcal{V} \wedge \mathcal{V} \not\succ_d \mathcal{Z} \Rightarrow \mathcal{U} \not\succ_d \mathcal{Z}$.

First, we note that since $\succ_i$ is a total order, $\succ'_i$ is also a total order (see Definition 17).

$\mathcal{U} \not\succ_d \mathcal{V} \Rightarrow (\forall X_i : \mathcal{U}(X_i) \succ'_i \mathcal{V}(X_i) \Rightarrow \exists X_k : (X_k \rhd X_i \wedge \mathcal{U}(X_k) \not\succeq'_k \mathcal{V}(X_k)))$ ($X_k \sim_\rhd X_i$ is not possible because $\rhd$ is a total order). (1)

Let X_i and X_j be the most important attributes such that $\mathcal{U}(X_i) \succ'_i \mathcal{V}(X_i)$ and $\mathcal{V}(X_j) \succ'_j \mathcal{Z}(X_j)$ respectively. (2)

Let X_p and X_q be the most important attributes s.t. $X_p \rhd X_i \wedge \mathcal{U}(X_p) \not\succeq'_p \mathcal{V}(X_p)$ and $X_q \rhd X_j \wedge \mathcal{V}(X_q) \not\succeq'_q \mathcal{Z}(X_q)$ respectively (such X_p and X_q must exist by (1)). (3)

Case 1 Both X_i and X_j as defined in (2) exist (cases when such X_i and/or X_j don't exist will be dealt with separately).

Three sub-cases arise: $X_p \rhd X_q$, $X_q \rhd X_p$ and $X_p = X_q$.

Case 1a: Suppose that $X_p \rhd X_q$ (see Figure 5.6). (4)

- From (3) we know that $X_p \rhd X_i \wedge \mathcal{U}(X_p) \not\succeq'_p \mathcal{V}(X_p)$, i.e., $\mathcal{V}(X_p) \succ'_p \mathcal{U}(X_p)$. (5)
- From (3) and (4) we know that $\mathcal{V}(X_p) \succeq'_p \mathcal{Z}(X_p)$, because X_q is the most important attribute that is also more important than X_j and $\mathcal{V}(X_q) \not\succeq'_q \mathcal{Z}(X_q)$, and X_p is more important than X_q (and hence X_j as well). (6)
- But because X_j is the most important attribute with $\mathcal{V}(X_j) \succ'_j \mathcal{Z}(X_j)$, and $X_p \rhd X_j$ (since $X_q \rhd X_j$ and $X_p \rhd X_q$), we have $\mathcal{V}(X_p) \not\succ'_p \mathcal{Z}(X_p)$ (as X_j is the most

important attribute with $\mathcal{V}(X_j) \succ'_j \mathcal{Z}(X_j)$, using (2)). Along with (6), this means that $\mathcal{V}(X_p) = \mathcal{Z}(X_p)$. (7)

- From (5) and (7), $\mathcal{Z}(X_p) \succ'_p \mathcal{U}(X_p)$. (8)
- Also, $\forall X_k : X_k \rhd X_p \Rightarrow \mathcal{U}(X_k) = \mathcal{V}(X_k) \wedge \mathcal{V}(X_k) = \mathcal{Z}(X_k)$ (because X_k is more important than X_i, X_j and X_p, X_q). (9)
- From (8) and (9), $\mathcal{Z} \succ_d \mathcal{U}$ with X_p as witness. Hence, $\mathcal{U} \not\succ_d \mathcal{Z}$.

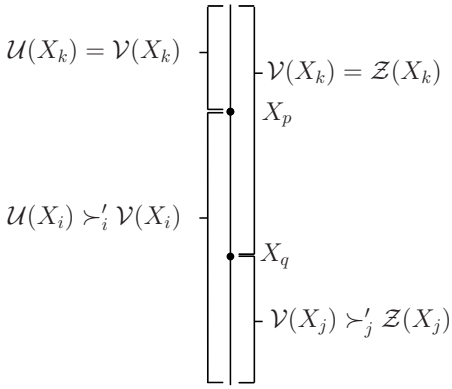


Figure 5.6 The case when $X_p \rhd X_q$

Case 1*b*: Suppose that $X_q \rhd X_p$. The claim holds by symmetry.

Case 1*c*: Suppose that $X_p = X_q$.

- From (3) we know that $X_p \rhd X_i \wedge \mathcal{U}(X_p) \not\succeq'_p \mathcal{V}(X_p)$, i.e., $\mathcal{V}(X_p) \succ'_p \mathcal{U}(X_p)$.
- Similarly, $\mathcal{Z}(X_p) \succ'_p \mathcal{V}(X_p)$.
- Hence, $\mathcal{Z}(X_p) \succ'_p \mathcal{U}(X_p)$. Moreover, $\forall X_k : X_k \rhd X_p \Rightarrow \mathcal{U}(X_k) = \mathcal{V}(X_k) \wedge \mathcal{V}(X_k) = \mathcal{Z}(X_k)$ (because X_k is more important than X_i, X_j and X_p, X_q).
- Therefore, $\mathcal{Z} \succ_d \mathcal{U}$ with X_p as witness. Hence, $\mathcal{U} \not\succ_d \mathcal{Z}$.

Case 2 : If X_i (say) does not exist, then $\forall X_i : \mathcal{U}(X_i) \not\succ'_i \mathcal{V}(X_i)$. Let X_p be the most important attribute s.t. $\mathcal{V}(X_p) \succ'_p \mathcal{U}(X_p)$ (if X_p does not exist, then trivially $\mathcal{U} \not\succ_d \mathcal{Z}$ because $\mathcal{U} = \mathcal{V}$). (10)

Case 2*a*: Suppose $X_p \rhd X_q$. Then $\forall X_k : X_k \rhd X_p \Rightarrow \mathcal{V}(X_k) = \mathcal{Z}(X_k)$ (because $X_k \rhd X_q$ as well). Moreover, $X_p \rhd X_q \Rightarrow \mathcal{V}(X_p) = \mathcal{Z}(X_p)$. Hence, $\mathcal{Z} \succ_d \mathcal{U}$ with X_p as witness and therefore $\mathcal{U} \not\succ_d \mathcal{Z}$.

Case 2*b*: Suppose $X_q \rhd X_p$. Then $\forall X_k : X_k \rhd X_q \Rightarrow \mathcal{U}(X_k) = \mathcal{V}(X_k)$ (because $X_k \rhd X_p$ as well). Moreover, $X_q \rhd X_p \Rightarrow \mathcal{U}(X_q) = \mathcal{V}(X_q)$. Hence, $\mathcal{Z} \succ_d \mathcal{U}$ with X_q as the witness and therefore $\mathcal{U} \not\succ_d \mathcal{Z}$.

Case 2*c*: Suppose $X_p = X_q$. Then $\forall X_k : X_k \rhd X_p \Rightarrow \mathcal{V}(X_k) = \mathcal{Z}(X_k)$ (because $X_k \rhd X_q$ as well) and similarly $\forall X_k : X_k \rhd X_q \Rightarrow \mathcal{U}(X_k) = \mathcal{V}(X_k)$ (because $X_k \rhd X_p$ as well). Moreover, since $\mathcal{V}(X_q) \not\succeq'_q \mathcal{Z}(X_q)$ (by (3)), $\mathcal{V}(X_p) \succ'_p \mathcal{U}(X_p)$ (using (10)) we have $\mathcal{Z}(X_p) \succ'_p \mathcal{U}(X_p)$. Hence, $\mathcal{Z} \succ_d \mathcal{U}$ with X_p as the witness and therefore $\mathcal{U} \not\succ_d \mathcal{Z}$.

Case 3 : If X_j (say) does not exist, the proof is symmetric to Case 2.

Case 4 : Suppose that both X_i and X_j do not exist. Then, for any variable X_i, $\mathcal{V}(X_i) \succeq'_i \mathcal{U}(X_i)$ and $\mathcal{Z}(X_i) \succeq'_i \mathcal{V}(X_i)$, i.e., $\forall X_i : \mathcal{Z}(X_i) \succeq'_i \mathcal{U}(X_i)$. Hence, there is no witness for $\mathcal{U} \succ_d \mathcal{Z}$, or $\mathcal{U} \not\succ_d \mathcal{Z}$.

Cases 1 - 4 are exhaustive, and in each case $\mathcal{U} \not\succ_d \mathcal{Z}$. This completes the proof. □

Remark.

As stated, Conjecture 1 and Theorem 10 apply whenever $\{\succ_i\}$ are totally ordered, and when using our method of comparing two aggregated valuations ($\succ'_i$) (see Definition 17). More generally, we note that they hold whenever $\{\succ'_i\}$ are total orders, regardless of the chosen method of comparing two aggregated valuations, and regardless of the properties of the input intra-attribute preferences $\{\succ_i\}$. For example, suppose that $\{\succ_i\}$ are ranked

weak orders (i.e., not total orders). As such, Conjecture 1 and Theorem 10 do not apply. However, for each attribute X_i if we define $\Phi_i(S)$ to be the rank number corresponding to the worst frontier of S, and $\succ_i'$ as the natural total order over the ranks in the weak order, then the consequences of Conjecture 1 and Theorem 10 hold.

We summarize the theoretical results relating the properties of the dominance relation and the properties of the preference relations $\rhd$ and $\{\succ_i'\}$ in Table 5.6.

$\rhd$	$\succ_i'$	$\succ_d$	Remarks
io	*po*	*po*	Theorem 1
io	*to*	*wo*	Conjecture 1
to	*to*	*wo*	Theorem 10

Table 5.6 Summary of results and conjectures relating to the properties of $\succ_d$ with respect to the properties of $\rhd$ and $\{\succ_i'\}$.

5.4 Summary and Discussion

In the following, we summarize the contents of this chapter.

a) Performed simulation experiments to compare the algorithms with respect to (i) the ratio of most preferred solutions produced to the actual set of most preferred solutions, and the ratio of the most preferred solutions produced to the entire set of solutions produced by the algorithm; (ii) their running times as a function of the search space and the overhead in each call to the functional composition algorithm; and (iii) the number of calls each algorithm makes to the functional composition algorithm during the course of its execution. The results showed the feasibility of our algorithms for composition problems involving hundreds of components.

b) Analyzed the results of experiments to obtain additional theoretical properties of the dominance relation as a function of the properties of the underlying intra-

attribute preference relations and relative importance preference relation. In particular, we obtained non-trivial results as a consequence of our analysis of experimental results, which were not known apriori, including conditions under which the dominance relation is a weak order. In particular:

i) when the relative importance preference of the user is such that there is a unique most important attribute (X_i, that is more important than all others), then the dominance relation includes the intra-attribute preferences aggregated valuations ($\{\succ_i'\}$) and the algorithm $A3$ is complete (it produces all the solutions that are non-dominated with respect to $\succ_d$);

ii) when the intra-attribute preferences and the relative importance are totally ordered, the dominance relation is a weak order. Further, we also conjectured that (ii) holds even when the relative importance is an interval order.

These conjectures/results are significant because they give the properties of the dominance relation directly as a function of the input user preferences. In turn, they also throw light on the soundness, weak-completeness and/or completeness properties of the algorithms.

The current implementation of dominance testing with respect to $\succ_d$ is based on iteratively searching all the attributes to find a witness. It would be interesting to compare this with other methods for dominance testing such as the one proposed in [Santhanam et al., 2010a] that uses efficient model checking techniques. We would also like to use other multi-attribute preference formalisms that include conditional preferences in our framework for compositional systems and compare the performance of the resulting implementation with the current implementation.

CHAPTER 6. Preference Reasoning for Compositional Systems: Applications to Web Services

6.1 Service-oriented Systems as Compositional Systems

Service-oriented computing [Bichler and Lin, 2006, Papazoglou, 2003, Huhns and Singh, 2005] offers a powerful approach to assemble complex distributed applications from independently developed software components in many application domains such as e-Science, e-Business and e-Government. Consequently, there is a growing body of work on specification, discovery, selection, and composition of services.

Successful development and deployment of service oriented software applications relies on effective solution of three inter-related problems:

- (a) *service composition* [Pathak et al., 2007] - assembling a *composite service* (or *composition*) from a set of components in a repository that satisfy the given requirements;
- (b) *substitution* [Pathak et al., 2007] - identifying appropriate alternatives to replace failed or unavailable component services in a composition; and
- (c) *adaptation* [Chafle et al., 2006, Pathak et al., 2006b] - altering existing composite services in response to changes in the functional/non-functional requirements, and/or the repository of available components.

Given a set of functional/non-functional requirements and a repository, the user seeks

a composite service assembled from the components in the repository that satisfies the requirements.

In this chapter, we demonstrate the use of the qualitative preference representation and reasoning techniques that we developed in our earlier chapters to address the problems of Web service composition in Section 6.2, substitution in Section 6.2 and adaptation in Section 6.2.

6.2 Web Service Composition

In many service-oriented applications, trade-offs involving non-functional attributes e.g., availability, performance play an important role in selecting component services in assembling a feasible composition, i.e., a composite service that achieves the desired functionality. We present `TCP-Compose*`, an algorithm for service composition that identifies, from a set of candidate solutions that achieve the desired functionality, a set of composite services that are *non-dominated* by any other candidate with respect to the user-specified qualitative preferences over non-functional attributes. We use TCP-net, a graphical modeling paradigm for representing and reasoning with qualitative preferences and importance. We propose a heuristic for estimating the preference ordering over the different choices at each stage in the composition to improve the efficiency of `TCP-Compose*`. We establish the conditions under which `TCP-Compose*` is guaranteed to generate a set of composite services that (a) achieve the desired functionality and (b) constitute a *non-dominated* set of solutions with respect to the user-specified preferences and tradeoffs over the non-functional attributes.

6.2.1 Background

The focus of this section is on service composition, i.e., the problem of assembling a composite service (goal service) from component services based on user preferences over

non-functional attributes.

Functional requirements specify the desired goal service functionality. Barring a few notable exceptions [Zeng et al., 2003, Zeng et al., 2004, Yu and Lin, 2005, Berbner et al., 2006], much of the work on service composition has focused on algorithms for assembly of composite services from functional specifications. Some of the major approaches to service composition based on functional specifications include: AI planning [Pistore et al., 2005b, Traverso and Pistore, 2004, Sirin et al., 2004, Shaparau et al., 2006], labeled transition systems [Pathak et al., 2006b, Pathak et al., 2007, Pathak et al., 2008b], Petri nets [Hamadi and Benatallah, 2003], among others. (The interested reader is referred to [Dustdar and Schreiner, 2005, Pathak et al., 2008a, Pistore et al., 2005a] for surveys).

Non-functional requirements refer to aspects such as security, reliability, performance, and cost of the goal service. For example, among the composite services that achieve the desired functionality, a user might prefer a more secure service over a less secure one; or one with a lower cost over one with a higher cost. Such preferences may be *quantitative* or *qualitative*. In many settings, a user might need to trade off one non-functional attribute against another (e.g., performance against cost); In others, it might be useful to assign relative importance to different non-functional attributes (e.g., security being more important than performance). Hence, there is an urgent need for principled methods that incorporate consideration of user-specified preferences with respect to the non-functional attributes, and the relative importance of the different non functional attributes. Of particular interest are algorithms that ensure that a set of solutions generated constitute a *non-dominated set*. We say that a set N of composite services is a *non-dominated set* if there is no composite service that is not in N that is strictly preferred over one or more of the composite services in N with respect to a set of user-specified preferences over non-functional attributes (and their relative importance).

Against this background, we present a procedure, `TCP-Compose`* for generating, given (i) a set of functional specifications; (ii) a set of preferences with respect to non-functional

attributes and their relative importance; (iii) a repository of candidate services with specified input-output behaviors and non-functional attributes; and (iv) *any* sound functional composition algorithm for assembling, from a repository of component services: a set of composite services that (a) achieve the desired functionality and (b) are non-dominated with respect to the user-specified preferences over non-functional attributes by any other composite service in the solution set of the algorithm used for functional specification based service composition.

`TCP-Compose*` makes use of Tradeoff-enhanced Conditional Preference Network (TCP-net) [Brafman et al., 2006], a variant of Conditional Preference Network [Boutilier et al., 2004], a framework for representing and reasoning with qualitative preferences. *CP*-net provides a compact representation of user-specified preferences with respect to non-functional attributes, by taking advantage of the independence or conditional independence of user preferences with respect to an attribute from preferences with respect to other attributes. *TCP*-net extends the CP-net framework by allowing the specification of the *relative importance* of different attributes (e.g., security is more important than cost).

`TCP-Compose*` uses a heuristic estimate of the preference ordering of alternative partial solutions to a service composition problem that corresponds to different choices of each component service, to improve the efficiency of search for a set of non-dominated solutions. We establish the conditions under which the proposed algorithm is guaranteed to find a set of non-dominated compositions with respect to user-specified qualitative preferences over possible values of each non-functional attribute and the relative importance of different non-functional attributes.

The rest of the section is organized as follows: Section 6.2.2 introduces the problem of service composition from user-specified functional and non-functional specifications; Section 6.2.4 describes the algorithm `TCP-Compose*` and establishes the conditions under which `TCP-Compose*` is guaranteed to find the set of non-dominated compositions

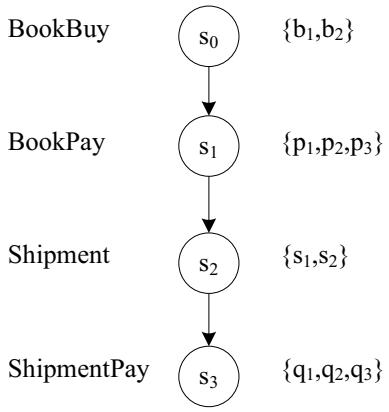


Figure 6.1 Goal Service

with respect to user-specified qualitative preferences over possible values of each non-functional attribute and the relative importance of different non-functional attributes; Section 6.2.5 concludes with a summary, discussion of related work, and an outline of some directions for further research.

6.2.2 Problem Specification

We introduce the problem of service composition from user-specified functional and non-functional requirements using a simple example. Suppose a user is interested in assembling a goal service G shown in Figure 6.1 from a repository of services $R = \{b_1, b_2, p_1, p_2, p_3, s_1, s_2, q_1, q_2, q_3\}$–where b_i's are book buying services, s_i's are shipment services and p_i's and q_i's are payment services that can work with b_i's and s_i's respectively. Suppose $(p_3, q_2), (b_2, s_2), (b_2, q_2), (b_2, q_3)$ are functionally incompatible and hence cannot be used together in any valid composition. The goal service should allow the user to buy book(s) from an online store, pay the store through a credit card service, arrange for shipping the book through a shipment service and pay for the shipping. In Figure 6.1, each of the steps in the goal service is annotated with the set of services from the repository that provide the respective functionality.

What we have so far is an informal specification of a service composition task based on

Preference Variable	Domain of Preference Variable
Reliability(R)	$\{L_R, H_R\}$
Security (S)	$\{L_S, M_S, H_S\}$
Availability(A)	$\{L_A, H_A\}$

Table 6.1 Domain Definition

user-specified functional requirements. We now turn to specification of user preferences with respect to three non-functional attributes: reliability, security, and availability of the goal service denoted by R, S, and A respectively. Suppose the available services can have Low (L_R) or High (H_R) reliability; Low (L_S), Medium (M_S) or High (H_S) security; and Low (L_A) or High (H_A) availability as shown in Table 1. Assume that the following non-functional attributes are known of each of the component services: $b_1 : L_R, b_2 : H_R, p_1 : L_S, p_2 : M_S, p_3 : H_S, s_1 : L_A, s_2 : H_R, q_1 : L_S, q_2 : M_S, q_3 : H_S$.

Now suppose that the user's preference with respect to security level is *not* independent of the reliability of the service. Suppose further that when the reliability is low, the user prefers high security; and when reliability is high the user is willing to settle for lower security (say, because of the prohibitive cost of achieving both high security and reliability); Suppose further that the user prefers high availability to low availability irrespective of the reliability and security of the service. Such information can be represented concisely using a CP-net with three nodes denoting the three attributes R, S, and A. The single headed arrows (e.g., from R to S) denote dependence among user preferences with respect to the attributes under consideration. The qualitative preferences of the user with respect to each attribute (conditioned on the preferences over attributes that such preference is dependent on) are specified by the conditional preference table (CPT) that annotate each node (Figure 6.2(a)). Suppose further that the user attaches greater importance to availability relative to security. Such assertions of relative importance of one attribute over another are represented using double headed

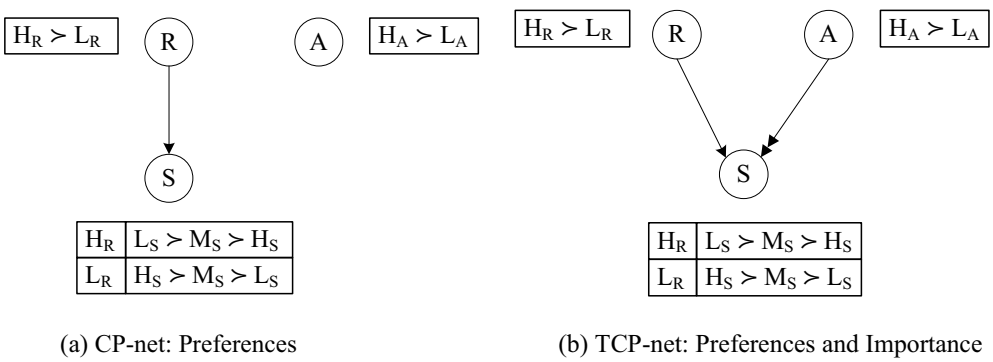


Figure 6.2 Example CP-net and TCP-net

arrows in TCP-net shown in Figure 6.2(b). The information regarding preferences with respect to R, S, and A in Figure 6.2(b) are the same as those shown in Figure 6.2(a).

For more background on CP-nets and TCP-nets, please see Chapter 1.

Given the preferences with respect to the non-functional attributes and their relative importance, our task is to identify from the solution space, i.e., the set of composite services that satisfy the user-specified functional requirements, a subset that forms a *non-dominated* set with respect to a set of user-specified preferences over non-functional attributes (and tradeoffs among them) that are captured by a TCP-net. It should be noted that a unique optimal composition exists only when the corresponding TCP-net induces a total ordering over the set of candidate feasible composite services, that is, the set of composite services that satisfy the user-specified functionality. `TCP-Compose*` does not assume the existence of a total order induced by the TCP-net over user-specified preferences and relative importance among attributes. Instead, we return a set of feasible composite services that constitute a non-dominated set with respect to the TCP-net that reflects the users preferences and tradeoffs with respect to the non-functional attributes.

Definition 26 (Completion)**.** *[Brafman et al., 2006] The* completion *of a partial assignment z is defined as a complete assignment or an outcome consistent with z, denoted $Comp(z)$. By consistency, we mean that if a preference attribute X_i has a valuation v_i*

in z then the valuation of X_i is also v_i in $Comp(z)$.

Definition 27 (Most Preferred Completion)**.** *[Brafman et al., 2006] The* most preferred completion *of a partial assignment z, denoted $PrefComp(z, \mathcal{N})$ is defined as a* completion *of z that is preferentially optimal with respect to the TCP-Net $\mathcal{N}$, i.e. $\nexists\, o \in O :\ o \succ PrefComp(z, \mathcal{N})$ such that o is a completion of z and consistent with z.*

Remarks.

1. We restrict our discussion to the class of *conditionally acyclic* TCP-nets that have been shown to be satisfiable with respect to a preference relation [Brafman et al., 2006].

2. Given a *conditionally acyclic* TCP-net, it is possible to order the set of all outcomes O [Brafman et al., 2006]. In other words, there exists a total order (that can be obtained using a topological sort) of the set of outcomes O that is *consistent with* the given TCP-net. However, several orderings of O can be consistent with a given conditionally acyclic TCP-net. For example, in a total preorder, there could be an outcome o such that $\nexists o' \succ o$ with respect to $\mathcal{N}$, but one cannot define o as the unique most preferred outcome. In our example, considering tuples of valuations of the non-functional attributes of a service, if the user did not give the information that R is relatively more important than A, then we would not be able to assert a preference among compositions with outcomes $o_1 = (H_R, L_S, H_A)$ and $o_2 = (L_R, H_S, L_A)$ (where subscripts denote the corresponding non-functional attributes reliability (R), security (S) and availability (A)). In this case, the user may like the composition system to return both the compositions if both o_1 and o_2 are non-dominated, i.e., $\nexists o' \succ o_1$ and $\nexists o' \succ o_2$. The algorithm we present, `TCP-Compose`* guarantees that in the absence of a *unique total order* over the

outcomes, the outcome corresponding to each composition in the solution set is non-dominated by the outcome corresponding to any other feasible composition.

3. We also note that there is another variant of the TCP-net, known as UCP-nets [Brafman et al., 2006] that capture quantitative preferences and relative importance information using utility functions. However, since we are not dealing with quantitative preferences, we stick to the basic qualitative TCP-nets.

6.2.3 Utilizing TCP-nets in Web service composition

We now proceed to describe how TCP-nets can be used to model qualitative preferences during Web service composition. For this we will use *dominance queries* [Brafman et al., 2006] of the form $o \overset{?}{\succ} o'$ with respect to $\mathcal{N}$ (in other words whether o is preferred to or dominates o'). The problem of Web service composition is to assemble a composite service that achieves a desired functionality from a set of component services. More precisely, we have:

Definition 28 (Web service composition problem)**.** *Given a target or goal service G and a repository of available services $R = \{W_1, W_2 \ldots W_n\}$, Web service composition amounts to creating a set of composite services $C = \{C_1, C_2 \ldots C_m\}$ such that $\forall i \leq m, C_i = W_{i_1} \oplus W_{i_2} \ldots \oplus W_{i_k}$ and $\forall l \leq i_k, W_l \in R$ such that C_i is functionally equivalent* [1] *to the G, denoted by $C_i \equiv G$. In the above, $\oplus$ is the* composition *operator for composing two services.*

Note that $\oplus$ is a *generic* composition operator and $W_{i_1}, W_{i_2} \ldots W_{i_k}$ is an arbitrary ordering of the components in C_i such that W_{i_j} is selected before $W_{i_{j+1}}$ in constructing C_i. We now proceed to describe an approach for using the TCP-net representation of

[1] *Functional equivalence can be defined in many ways depending on the particular formalism used to describe the services. For example, if labeled transition systems are used for describing the services, checking the functional equivalence of a composite service to a goal service reduces to checking the* bisimulation equivalence *of the corresponding labeled transition systems [Pathak et al., 2006b, Pathak et al., 2007]*

user-specified *non-functional* requirements to guide service composition using any of the standard methods that can generate compositions that satisfy user-specified *functional* requirements.

6.2.4 TCP-Compose*

We present an algorithm, TCP-Compose*, that uses a preference guided heuristic to come up with the most preferred compositions among the candidates.

6.2.4.1 Search Space of TCP-Compose*

We cast the problem of assembling from a set of available component services, a composite service with the desired functionality as a state space search problem. The empty composition $\perp$ is the start state; the set of *feasible extensions* using one of the available components from any given state define the successors of that state; and the set of feasible candidate compositions correspond to the goal states. The cost function at any state is given by the *preference valuation* of the partial composition corresponding to that state.

Definition 29 (Feasible Extension)**.** *A feasible extension to a partial composition P is defined as a partial composition $P' = P \oplus W_i, W_i \in R$ such that the partial composition P' is functionally equivalent to a part of the goal service.*

Let $\mathcal{N}$ be a TCP-net with a set of preference attributes $V = \{X_1, X_2, \ldots X_p\}$ with finite domains $D(X_1), D(X_2), \ldots D(X_k)$ respectively where each preference attribute corresponds to a non-functional attribute of a composition. We assume that such a TCP-net specification is given by the user as input to the algorithm TCP-Compose*.

Each of the leaf nodes is a goal node and corresponds to valid or feasible candidate composite services that are functionally equivalent to the goal service. Note that the the nodes of the tree may have varying but finite branching factors. Figure 6.3 illustrates the

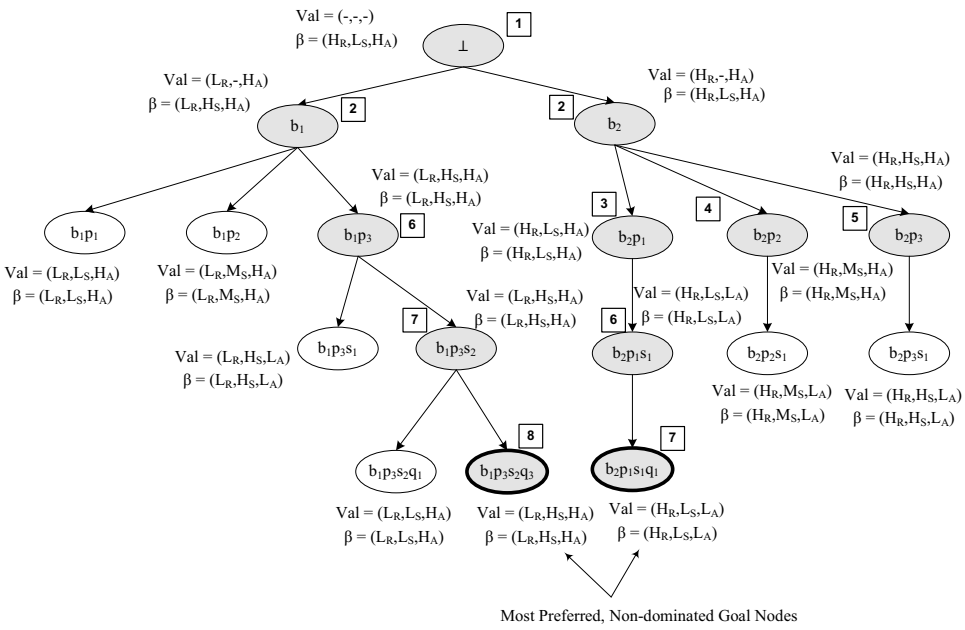


Figure 6.3 Search Space for `TCP-Compose`* when the TCP-net does not induce a total order over the set of valuations

search space for our goal service given in Figure 6.1 with respect to the TCP-net given in Figure 6.2. The shaded nodes correspond to partial compositions that were actually expanded further. The numbers in the boxes next the nodes show the order in which the corresponding nodes are expanded. The nodes that are not shaded are generated but not further pursued by the algorithm: For example, although `TCP-Compose`* explores partial composition $b_1 \oplus p_1$, its feasible extension $b_1 \oplus p_1 \oplus s_1$ is not explored. The annotation Val denotes the *preference valuation* and β denotes the *most preferred completion* of the partial composition corresponding to each node. They are formally defined below.

Definition 30 (Preference Valuation)**.** Preference valuation *is a function* $F : W \times X \rightarrow \bigcup(D(X_i) \cup \{-\})$, *where* $W = \{W_1, W_2 \ldots W_n\}$, $X = \bigcup X_i$. *The value* $\{-\}$ *denotes that the valuation of the corresponding attribute is unknown. We denote the valuation of an attribute* X_i *in a Web service* W *as* $F(W)(X_i) = v_i, v_i \in D(X_i) \cup \{-\}$. *We define the valuation of an attribute* X_i *in a* composition *of two services* W_i, W_j *as* $F(W_i \oplus W_j)(X_i) = F(W_i)(X_i) \odot F(W_j)(X_i)$, *where*

$$F(W_i)(X_p) \quad \odot \quad F(W_j)(X_p) \qquad = \qquad \begin{cases} F(W_j)(X_p), & F(W_i)(X_p) \succ F(W_j)(X_p) \\ F(W_i)(X_p), & otherwise. \end{cases}$$

The preference valuation of a partial composition $P = W_1 \oplus W_2 \oplus \ldots W_l$ *with respect to an attribute* X_p *is defined inductively as* $F(P)(X_p) = F(W_1)(X_p) \odot F(W_2)(X_p) \ldots \odot F(W_l)(X_p)$. *We also denote the preference valuation (over all attributes) of a partial composition* P *as the tuple* $Val(P) = (F(P)(X_i), F(P)(X_2) \ldots F(P)(X_k))$.

Thus, the preference valuation of a partial composition with respect to a non-functional attribute corresponds to the least preferred valuation of that attribute among the participating component services in the composition. For example, in Figure 6.3 the valuation of the partial composition $b_2 \oplus p_1 \oplus s_1$ with respect to the attribute availability (A) is low (L_A) because the component s_1 has low availability although the component b_2 has high availability.

Definition 31 (Most Preferred Completion)**.** *The* most preferred completion *of a preference valuation of a partial composition* P *is defined as a complete assignment to all attributes* $X_1, X_2, \ldots X_k$, $\beta(P) = PrefComp(Val(F(P)), \mathcal{N})$ *where the function* $PrefComp$ *is as defined in Def.27.*

Intuitively, it is easy to see why $\beta(P_i)$ is a heuristic estimate of the most preferred composition that can be realized by extending the given P_i. In our search for compositions, $\beta(P_i)$ denotes the estimate of most preferred feasible candidate composite service that is equivalent to the goal service that is realizable from the current partial composition P_i.

Definition 32 (Heuristic Function h)**.** *We define the heuristic function* h *as* $h: 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$, *where* $\mathcal{P}$ *is a set of partial compositions and* $S := h(\varphi)$, $S \subseteq \varphi \subseteq \mathcal{P}$ *is such that* $\forall P_0 \in S$ *and* $\forall P_i \in \varphi$, $\beta(P_i) \not\succ \beta(P_0)$. *We also define* $h(\phi) = \perp$ *and* $\perp \oplus W_i \equiv W_i$.

Algorithm 5 `TCP-Compose`* ($\mathcal{N}$, φ, G, R)

1. Compute heuristic $\rho \leftarrow h(\varphi)$
2. **for all** $P \in \rho$ **do**
3. **if** ($P \equiv G$ and $\nexists\, Q \in \theta : \beta(Q) \succ \beta(P)$) **then**
4. $\theta \leftarrow \theta \cup \{P\}$
5. $\varphi \leftarrow \varphi - \{P\}$
6. **for all** $Q \in \theta$ **do**
7. **if** $\beta(P) \succ \beta(Q)$ **then**
8. $\theta \leftarrow \theta - \{Q\}$
9. **end if**
10. **end for**
11. **else**
12. Find the next set of feasible partial compositions expanding P
$\psi \leftarrow \{P_i \mid P_i = P \oplus W_i, \quad W_i \in R \quad and \quad P \oplus W_i \equiv G\}$
13. $\varphi \leftarrow \varphi \cup \psi - \{P\}$
14. **end if**
15. **end for**
16. **if** ($\varphi = \phi$) **then**
17. **return** θ
18. **end if**
19. `TCP-Compose`* ($\mathcal{N}$, φ, G, R)

The above definition of the heuristic function h makes it clear that, for any set of partial compositions φ, $h(\varphi)$ is the set of partial compositions whose valuations are non-dominated by the valuation of any other partial composition in φ.

Algorithm 1 shows the listing for `TCP-Compose`*. The main idea behind `TCP-Compose`* is to use a best first search strategy to find a set of non-dominated feasible candidate compositions equivalent to the goal service. To guide the search, the algorithm applies the heuristic function h to the set of partial compositions under consideration. The algorithm is initially invoked with the parameters $(\mathcal{N}, \phi, G, R)$. The set of partial compositions under consideration for expansion are maintained in φ, and the algorithm uses $h(\varphi)$ to select the set of non-dominated compositions ρ to expand (line 1). Next, for each of the compositions in the non-dominated set ρ, if there is a candidate composition P which is functionally equivalent to the goal service, then the algorithm adds P to the solution set θ, provided none of the existing solutions already in θ strictly dominate it

(lines 2 - 5). If P now strictly dominates any of the existing solutions in θ then those candidate solutions are removed from θ (lines 6 - 10). For partial compositions that are not candidate compositions in ρ, the algorithm proceeds to compute the set of feasible extensions and adds them to φ (lines 11 and 14). The algorithm terminates with the set of candidate solutions θ if there are no more compositions to explore (lines 16 - 18), and finally, the process is repeated (line 19) until there are no more compositions left to explore.

We now proceed to describe how the algorithm explores the search space when the TCP-net does not induce a total order on the set of non-functional attribute valuations. In the search space illustrated in Figure 6.3 in the first run, when expanding the root node there are two possible partial compositions namely b_1 and b_2 respectively. Note that the valuations of partial compositions b_1, b_2 are incomparable with respect to the TCP-net $\mathcal{N}$, and in the second run, both the partial compositions in φ are expanded. In runs 3, 4 and 5 the compositions $b_2 \oplus p_1, b_2 \oplus p_2, b_2 \oplus p_3$ are expanded, as their valuations strictly dominated others in their respective iterations. However, in the sixth run, the algorithm again finds that the non-dominated compositions $b_1 \oplus p_2$ and $b_2 \oplus p_1 \oplus s_1$ are incomparable, and hence expands both. Notice that when expanding $b_2 \oplus p_1 \oplus s_1$, the feasible extensions also have the component s_1 (we assumed that b_2, s_2 are functionally incompatible and cannot be composed together) which has lower reliability and availability, but there is a still a possibility of a service with b_1 ending up with a candidate composition with high availability. In the seventh run, the algorithm identifies $b_2 \oplus p_1 \oplus s_1 \oplus q_1$ as a solution, and finally in the eighth run the algorithm terminates with both the candidate compositions $b_1 \oplus p_3 \oplus s_2 \oplus q_3$ and $b_2 \oplus p_1 \oplus s_1 \oplus q_1$ as the non-dominated candidate compositions. This illustrates how the less preferred compositions like $b_1 \oplus p_1$ are actually not explored by the algorithm, consequently pruning of the search space.

6.2.4.2 Properties of TCP-Compose*

We show that the algorithm TCP-Compose* is guaranteed to return the set of composite services each of which is functionally equivalent to the user-specified goal service that constitute a non-dominated set with respect to a set of user preferences over the non-functional attributes.

Lemma 1. *For any partial composition P, $\beta(P) \succeq \beta^\star(P)$, where $\beta^\star$ gives the valuation of the* actual *most preferred feasible composition starting with P.*

Proof. Suppose by contradiction, there exists a partial composition P and a $\beta^\star$ such that $\beta^\star(P) \succ \beta(P)$. This implies that there is a feasible candidate composition C starting from the partial composition P such that $Val(C) \succ \beta(P)$, or there is a sequence of feasible extensions from P to C such that $Val(C) \succ PrefComp(Val(P), \mathcal{N})$ with respect to $\mathcal{N}$, by the definition of $\beta(P)$. This clearly contradicts the definition of $PrefComp(Val(P), \mathcal{N})$. □ □

Theorem 11. *TCP-Compose* is guaranteed to return the set of feasible composite services that constitute a non-dominated set with respect to a given TCP-net.*

Sketch, by contradiction. Suppose TCP-Compose* does not terminate with the set of non-dominated candidate compositions. There are three cases to consider.

1. TCP-Compose* terminates with a set of compositions such that one of the solutions returned by TCP-Compose* is a not feasible composition. This contradicts the Step 3 of the algorithm where the terminating condition is clearly only satisfied for feasible compositions.

2. TCP-Compose* fails to terminate. This is not possible because although the algorithm is recursive, the search tree is finite, and in each iteration, previously examined partial compositions are not revisited.

3. Starting with a partial composition P, TCP-Compose* terminates with a set of candidate compositions such that one of the solutions corresponds to a feasible candidate composition C' with a sub-optimal preference valuation $Val(C')$, i.e. $\beta^\star(P) \succ Val(C')$, where $\beta^\star$ gives the *actual* most preferred feasible composition starting with P.

   By Lemma 1, at each step, $\beta(P) \succeq \beta^\star(P) \succ Val(C') \Rightarrow \beta(P) \succ Val(C')$. So in the last step just before termination, by the definition of the heuristic function h and Line 1 of TCP-Compose*, the algorithm would have chosen to expand the composition P rather than the partial composition that just preceded C'. Hence, the algorithm could not have terminated with any composition C' such that $\beta^\star(P) \succ Val(C')$, and hence it would return only non-dominated candidate compositions.

This proves that TCP-Compose* is guaranteed to return the set of feasible composite services that constitute a non-dominated set with respect to a given TCP-net. □ □

6.2.5 Summary and Discussion

Most of the work on service composition has focused on algorithms for assembling, from a set of available component services, a composite service that achieves the user-specified functionality. However, in many applications, preferences over non-functional attributes e.g., availability, performance as well as tradeoffs among them can influence the choice of the component services in assembling a composite service that achieves the desired functionality. Hence, there is a growing interest in techniques that incorporate such non-functional considerations into service composition. For example, Zeng et al. [Zeng et al., 2003, Zeng et al., 2004] have explored linear programming methods for optimizing the choice of services based on non-functional attributes based on user-

specified weights and *utility functions*. Yu et al. [Yu and Lin, 2005] have explored a formulation of service composition as a combinatorial optimization (multi-choice multi-dimensional 0-1 Knapsack problem) and as a graph search problem wherein the non-functional constraints are encoded by the edges in the graph. Berbner et al. [Berbner et al., 2006] have proposed a heuristic approach, using simulated annealing and integer programming, for selecting services based on non-functional attributes. Each of these approaches assume a *quantitative* measure of preference over alternative valuations of non-functional attributes. This is tantamount to assuming the existence of a *cardinal* utility function [Keeney and Raiffa, 1993]. Eliciting such a utility function, over multiple not necessarily independent attributes, from users presents a significant challenge in practice. Hence, methods that can utilize *qualitative* information regarding preferences over non-functional attributes are of significant interest.

Against this background, we have presented `TCP-Compose`*, a procedure for generating, given (i) a set of functional specifications; (ii) a set of preferences with respect to non-functional attributes and their relative importance; (iii) a repository of candidate services with specified input-output behaviors and non-functional attributes; and (iv) *any* sound algorithm for assembling, from a repository of component services: a set of composite services that (a) achieve the desired functionality and (b) are non-dominated with respect to the user-specified preferences over non-functional attributes by any other composite service in the solution set of the algorithm used for functional specification based service composition. `TCP-Compose`* uses TCP-net, a graphical modeling paradigm for representing and reasoning with qualitative preferences and importance of alternative partial solutions to a service composition problem that corresponds to different choices of each component service. An important feature of `TCP-Compose`* is that it offers a generic approach to augment *any* of a broad range of available algorithms for assembling feasible composite services that achieve the user-specified functionality with the ability to consider qualitative preferences and tradeoffs over non-functional attributes of the

desired goal service.

Schropfer et al. [Schropfer et al., 2007] have recently proposed a TCP-net based formulation of qualitative user preferences over non-functional attributes for ranking and selecting *individual* services. In contrast, the focus of `TCP-Compose*` is on the assembly of a set of *composite* services that constitute a non-dominated set with respect to a set of user-specified preferences and tradeoffs over non-functional attributes.

Shaparau et al. [Shaparau et al., 2006] have proposed an algorithm for contingent planning with goal preferences which can also be used for service composition. The planning algorithm requires the user to specify *explicit* preferences over goals. This is tantamount to explicitly specifying an ordering over all feasible composite services. Hence, this approach is likely to be impractical in settings where the number of component services in the repository is large. In contrast, the approach used in `TCP-Compose*` requires the user to specify only the preferences and trade-offs over the non-functional attributes that in turn *induce* a preference over the feasible composite services.

Work in progress is aimed at the implementation and experimental evaluation of `TCP-Compose*` on a range of benchmark problems of varying complexity. Some interesting directions for further research include: investigation of approaches for handling of *global* non-functional constraints (e.g., no composite service which has security level below a specified threshold is acceptable); customized versions of `TCP-Compose*` that take advantage of specific representations and algorithms used in the search for feasible solutions.

6.3 Web Service Substitution

As we have seen in Section 6.2, in service-oriented applications users often prefer some composite services over the others based on their preferences over non-functional attributes such as performance, security and cost. In such applications involving com-

posite Web services, one or more component services may become unavailable. This presents us with the problem of identifying other components that can take their place, while maintaining the overall functionality of the composite service. Given a choice of candidate substitutions that offer the desired functionality, it is often necessary to select the most preferred substitution based on non-functional attributes of the service, e.g., security, reliability, etc.

We propose an approach to this problem using *preference networks* for representing and reasoning about preferences over non-functional properties. We present algorithms for solving several variants of this problem: a) when the choice of the preferred substitution is independent of the other constituents of the composite service; b) when the choice of the preferred substitution depends on the other constituents of the composite service; and c) when multiple constituents of a composite service need to be replaced simultaneously. The proposed solutions to the service substitution problem based on preferences over non-functional properties are independent of the specific formalism used to represent functional requirements of a composite service as well as the specific algorithm used to assemble the composite service.

After a composite service assembled from a repository of component services has been deployed, one or more constituents of the composite service may become unavailable. Hence there arises a need to replace such components with other components from the repository while maintaining the overall functionality of the composite service. Among the candidate substitutions that offer the desired functionality, the user might prefer some substitutions over others based on non-functional attributes of the service, e.g., security, reliability, etc.

Service substitution based on the functional properties of components has been addressed by many authors [Benatallah et al., 2006, Bordeaux et al., 2005, Liu et al., 2005, Martens et al., 2006, Pathak et al., 2007]. This section is aimed at addressing the service substitution problem taking into account the user preferences over non-functional

properties. We associate with each non-functional property, a corresponding domain and assume that the non-functional properties as well as their respective domains are specified by some agreed upon standards. Service substitution in such a setting requires the user to be able to express preferences over non-functional properties of services. For example, a user might prefer a more secure service to a less secure one; or one with a lower cost over one with a higher cost. Furthermore, some attributes may be more important than others, in which case, it is useful to assign *relative importance* to different non-functional attributes (e.g., security being more important than performance).

Such preferences may be *qualitative* or *quantitative*. Qualitative preferences are asserted based on *relative goodness* of two alternatives, whereas quantitative preferences force the user to quantify *by how much* he/she prefers one alternative to another, for example in the form of *utility functions* [Fishburn, 1970]. While quantitative preferences over multiple attributes can be difficult to elicit from users, qualitative preferences are often easier to elicit [French, 1986b, Schneider and Shanteau, 2003].

Service substitution considering user preferences over non-functional attributes is complicated by the fact that the value of a particular attribute of the composite service is a function of all its constituent services. For instance, the reliability of a composite service is only as high as the reliability of its *least* reliable constituent. Further, there can be interactions among the user preferences with respect to different non-functional attributes. For example, due to prohibitive cost, the user may prefer higher security when there is low reliability and lower security when reliability is high.

Against this background, this section addresses the problem of service substitution based on user specified qualitative preferences over non-functional attributes. The contributions of the section are as follows:

1. We introduce an approach to service substitution based on user preferences over non-functional properties of services. Our approach utilizes *preference networks*

[Brafman et al., 2006] for representing and reasoning about preferences over non-functional properties in this setting.

2. We consider and solve two variants of the service substitution problem, namely, *context-insensitive* and *context-sensitive* substitution. The former assumes that the preferred substitution can be identified independent of the *context*, i.e., the other constituents in the composition, whereas the latter takes into account the context of the substitution.

3. We consider and solve the service substitution problem in a more general setting, wherein multiple constituents of a composition need to be replaced.

The proposed solutions to the service substitution problem based on preferences over non-functional properties are independent of the specific formalism used to represent functional requirements of a composite service as well as the specific algorithm used to assemble the composition.

Organization. Section 6.3.1 describes our solutions for identifying the preferred substitutions for both context-insensitive and context-sensitive substitution problems using the TCP-net preference model. Section 6.3.3 describes our algorithm for substitution of multiple services in a composition. Section 6.3.5 concludes with a summary, discussion of related work and an outline of future directions for research.

Example 9. *Suppose that a user specifies preferences over three non-functional attributes: reliability (R), security (S) and availability (A). The domains of the attributes are* $\{L_R, H_R\}$, $\{L_S, M_S, H_S\}$ *and* $\{L_A, H_A\}$ *respectively, where* L_i *represents* low *"level" of the attribute* i, M_i *represents* medium *and* H_i *represents* high. *The user specifies that reliability is more important than availability, and availability is more important than security. I.e., the user prefers high valuations of reliability and availability. Finally, the user states that his/her preference with respect to the security is* not *independent of the*

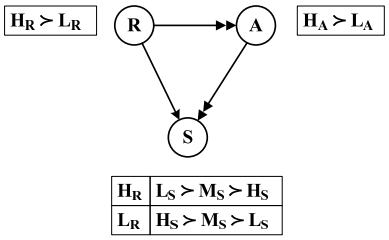


Figure 6.4 TCP-net: Representing Preferences and Importance

reliability. At lower levels of reliability, the user prefers higher security, but the user tends to prefer lower security when the reliability is high (say, due to prohibitive costs of having higher levels for both attributes). The TCP-net in Figure. 6.4 represents the above user preferences. It shows that S is preferentially dependent on R (i.e. $Pa(S) = \{R\}$). Double-headed arrow captures the fact that R is relatively more important than A which, in turn, is relatively more important than S. Finally, the conditional preference tables annotate each node presenting the total order of the domain of each attribute (w.r.t. the parents of the node, if any).

Remark 1. *We limit our scope of discussion to a class of TCP-nets called* conditionally acyclic *TCP-nets, as only this particular class of TCP-nets have been proved to be satisfiable with a preference relation [Brafman et al., 2006]. We also assume that the preferences over variable domains are total orders, although TCP-nets in general allow specification of partial orders as well. We note that there is another variant of the TCP-net, known as UCP-nets [Brafman et al., 2006] that captures quantitative preferences and relative importance information using utility functions. However, since we are not dealing with quantitative preferences, we stick to the basic qualitative TCP-nets.*

Non-dominated Set of Outcomes. Given a *conditionally acyclic* TCP-net, there exists a total order (that can be obtained using a topological sort) of the set of outcomes $\mathcal{O}$ that is *consistent with* the given TCP-net [Brafman et al., 2006]. However, several orderings of $\mathcal{O}$ can be consistent with a given conditionally acyclic TCP-net (corre-

sponding to distinct topological sorting of the variables in the preference network). In a total preorder, there could be an outcome o such that $\nexists o' : o' \succ o$ with respect to the TCP-net, but one cannot define o as the unique most preferred outcome.

Consider the example in Figure 6.4. If the user did not provide the information that R is relatively more important than A, then we will not be able to assert the preference between two valuations of attributes: (H_R, H_S, L_A) and (L_R, H_S, H_A). In other words, the user is indifferent with respect to the above outcomes and we say that the outcomes form a *non-dominated* set, one where any element in the set is *not preferred* over any other element in the set. To handle such situations, it is necessary that the reasoning about preferences considers non-dominated outcomes instead of the unique most preferred one.

6.3.1 Preference Reasoning for Web Service Substitution

We now proceed to describe the problem of Web service substitution, and how to use TCP-nets to compute the preferred substitutions from a set of functionally feasible alternatives. For this purpose, we will use *dominance queries* [Brafman et al., 2006] of the form $o \overset{?}{\succ} o'$ with respect to TCP-net (in other words whether o is preferred to or dominates o').

Definition 33 (Web Service Composition [Santhanam et al., 2008])**.** *A Web service composition* $C = W_1 \oplus W_2 \ldots \oplus W_k$ *such that* $\forall 1 \leq l \leq k, W_l \in R$ *is an assembly of component services from a repository of available services* $R = \{W_1, W_2 \ldots W_n\}$ *such that* C *is functionally equivalent* [2] *to a target or goal service* G*, denoted by* $C \equiv G$*. In the above,* $\oplus$ *is the* composition *operator for composing two services.*

The problem of Web service substitution refers to identifying a component service from the repository of services that can suitably replace a particular component in an

[2] *Functional equivalence can be defined in many ways including bisimulation of labeled transition systems [Pathak et al., 2006b, Pathak et al., 2007]*

existing composite service. The identified component service must achieve the desired functionality in the context of the composite service as a whole. Formally, Web service substitution is defined as follows.

Definition 34 (Web Service Substitution)**.** *Given an existing composite service* $C = W_1 \oplus W_2 \ldots \oplus W_{i-1} \oplus W_i \oplus W_{i+1} \ldots \oplus W_k$ *that achieves the functionality of the target or goal service* G *such that* $C \equiv G$*, Web service substitution amounts to identifying a replacement,* W_s*, of a component* W_i *in the composite service, such that* $C' = W_1 \oplus W_2 \ldots \oplus W_{i-1} \oplus W_s \oplus W_{i+1} \ldots \oplus W_k$ *and* $C' \equiv C$*.*

We will also use the notation $C \ominus W$ to denote the partial composition obtained by removing the service W from the composite service C i.e., $C \ominus W_i = W_1 \oplus W_2 \ldots \oplus W_{i-1} \oplus W_{i+1} \ldots \oplus W_k$. The above definition, however, does not take into consideration the preferences over non-functional attributes of the composition and the components.

Given the user-preferences and trade-offs over non-functional attributes in the form of a TCP-net, we can define the *preference valuation* [Santhanam et al., 2008] of a service as follows.

Definition 35 (Preference Valuation)**.** *Preference valuation is a function* $F : \mathcal{W} \times \mathcal{X} \rightarrow \bigcup(D(X_i))$ $\mathcal{W} = \{W_1, W_2 \ldots W_k\}$, $\mathcal{X} = \bigcup\{X_i\}$*. We denote the valuation of an attribute* X_i *in a Web service* W *as* $F(W)(X_i) = v_i$ *where* $v_i \in D(X_i)$*. We define the valuation of an attribute* X_i *in a* composition *of two services* $W_i \oplus W_j$ *as* $F(W_i \oplus W_j)(X_i) = F(W_i)(X_i) \odot F(W_j)(X_i)$*, where*

$$F(W_i)(X_p) \odot F(W_j)(X_p) = \begin{cases} F(W_j)(X_p) & \text{if } F(W_i)(X_p) \succ F(W_j)(X_p) \\ F(W_i)(X_p) & \text{otherwise} \end{cases}$$

The ***preference valuation*** *of a composition* $P = W_1 \oplus W_2 \oplus \ldots W_l$ *with respect to attribute* X_p *is defined (inductively) as* $F(P)(X_p) = F(W_1)(X_p) \odot F(W_2)(X_p) \ldots \odot$

$F(W_l)(X_p)$. *We also denote the* ***complete preference valuation*** *over all attributes* $\mathcal{X}$ *of a composition* P *as the tuple* $V_P = \langle F(P)(X_i), F(P)(X_2) \ldots F(P)(X_k) \rangle$.

The function F defines how the valuations of P's components are aggregated. Our definition of F computes the the least preferred valuation of that attribute among the participating component services in the composition. For example, in the case of reliability, the valuation of a composition can be defined as the valuation of its *least* reliable component. We note that other attributes such as cost may require other ways of aggregating the attribute valuations of the components in a composition, which can be handled by having appropriate definitions of F and $\odot$.

Definition 36 (Sole Dependence)**.** *Given a composition* C *and its component* W*, we say that* C *is* solely dependent *on* W *with respect to an attribute* X_i *iff* $V_{C \ominus W}(X_i) \succ V_C(X_i)$.

In other words, improving the valuation $V_W(X_i)$ improves C's valuation of X_i as well, i.e., all other components in C have a strictly better valuation for X_i than W.

6.3.2 Computing Preferred Substitutions

Now we address the problem of finding the preferred substitutions from a set of functionally feasible substitutions using a TCP-net model of preferences over non-functional attributes. Given a composite service C and a component W to be replaced, we assume that there exists a mechanism[3] that generates the set of functionally feasible substitutions, $\mathcal{W}$, from the repository of available services. Our goal is to compute the set $\mathcal{W}' \subseteq \mathcal{W}$ of preferentially non-dominated substitutions with respect to the given TCP-net.

We further distinguish between two variants of the substitution problem, namely, *context-insensitive* and *context-sensitive* substitution. The first approach assumes that

[3]We refer the reader to [Pathak et al., 2007, Liu et al., 2005, Martens et al., 2006, Benatallah et al., 2006, Bordeaux et al., 2005] for more details on functional aspects of service substitution

	Services	Reliability	Security	Availability
Composite	C	L_R	L_S	H_A
To-replace	W	L_R	L_S	H_A
Substitutes	W_1	L_R	L_S	L_A
	W_2	H_R	H_S	L_A
	W_3	L_R	M_S	H_A
	W_4	L_R	H_S	H_A

Table 6.2 Preference Valuations

the preferred substitution can be obtained independent of the *context*, i.e., the non-functional properties of the other components in the composition, while in the second approach, the context of the substitution is taken into account.

Example 10. *Consider a composite service C that and a component W in C that needs to be substituted. Suppose that there is a set of* functionally feasible substitutions $\mathcal{W} = \{W_1, W_2, W_3, W_4\}$ *such that each component $W_i \in \mathcal{W}$ can substitute W in C satisfying all the functional requirements of the substitution. The non-functional attributes of interest to the user are reliability, availability and security, and the user-preferences over these attributes are represented using the TCP-Net in Figure 6.4. The valuations of the non-functional attributes for C, W and the substitutes are presented in Table 6.2. The objective is to identify the preferred substitution(s) for W in the composition C.*

6.3.2.1 Context-Insensitive Substitution

This approach simply computes the preferentially non-dominated substitutions for the component to be replaced, from the set of functionally feasible substitutions, without considering how the non-functional properties of the other components in the composition may affect the valuation of the overall composition.

In Example 10, W_2 is the most preferred substitution in $\mathcal{W}$ as $V_{W_2} \succ V_{W_4} \succ V_{W_3} \succ V_{W_1}$ (see Figure 6.4 and Table 6.2). If C is *solely dependent* on W with respect to reliability and security, then the choice of W_2 is the most preferred one. As the existing

composite service C has the valuation $V_C = (L_R, L_S, H_A)$, following Definition 35, the new composition with W_2 as the substitute will have a valuation $V_C{}' = (H_R, H_S, L_A) \succ V_C$.

However, consider the scenario when with respect to the attributes reliability and security, C is *not solely dependent* on W, e.g., there is some other service in C in addition to W that has low reliability and low security. In that case, W_2 as the preferred substitution would be a bad choice, as the overall valuation of C goes down: $V_C{}' = (L_R, L_S, L_A)$, i.e., $V_C \succ V_C{}'$. This is because the computed substitutions do not take into account the *context* of the composite service as a whole. A better substitution in terms of preference can be obtained, if we take into account some context information, i.e., the preference valuation of the composite service C in the absence of W, $V_{C \ominus W}$. Thus, the context-insensitive approach works well only when all the non-functional attribute valuations of the composition are solely dependent on the component being replaced.

6.3.2.2 Context-Sensitive Substitution

In this approach, when identifying a replacement, we take into account how the valuation of the identified substitution will affect the overall valuation of the composition. As a result, here we consider the valuation of the composition in the absence of the component to be replaced in order to identify substitutions, i.e., $V_{C \ominus W}$, in contrast to the context-insensitive approach where we instead considered V_C. The following algorithm describes this approach.

1. Compute $V_{W_j}{}' = V_{(C \ominus W) \oplus W_j}, \forall W_j \in \mathcal{W}$
2. Compute the preferentially non-dominated set of valuations $\varphi = \{V_{W_j}{}' \mid \nexists W_k{}' : V_{W_k}{}' \succ V_{W_j}{}'\}$ w.r.t. TCP-net
3. Return the set of substitutions corresponding to each valuation in φ i.e., $\mathcal{W}' = \{W_j \mid V_{W_j} \in \varphi\}$

To see the subtle distinction between the context-insensitive and sensitive approaches, suppose that in Example 10, C is solely dependent on W with respect to security; and not with respect to reliability and availability. Let the valuation of $V_{C \ominus W}$ yield (L_R, H_S, H_A) – assuming all other components in C have high security level. The above algorithm would return the solution as W_4 because $V_{(C \ominus W) \oplus W_4} = (L_R, H_S, H_A)$ clearly dominates $V_{(C \ominus W) \oplus W_3} = (L_R, M_S, H_A)$, as well as $V_{(C \ominus W) \oplus W_1} = (L_R, L_S, L_A)$ and $V_{(C \ominus W) \oplus W_2} = (L_R, H_S, L_A)$. Thus, this approach yields the best solution $\mathcal{W}' = \{W_4\}$ taking into account the context information.

6.3.3 Multiple Component Substitution

The solutions we have seen so far are aimed at finding preferred substitutions for one component at a time. We now address the problem of finding substitutes for more than one component at a time.

Consider a composite service C that needs to replace a *set* of n of its components: $W = \{W_1, W_2, \ldots W_n\}$. We again assume that the user-preferences over the non-functional attributes are modeled using a TCP-net, and that there exists a mechanism to find out the set of functionally feasible substitutions for any given component in the composite. Suppose that each W_i has a set of functionally feasible substitutions: $R_{W_i} = \{W_{i_1}, W_{i_2}, \ldots W_{i_k}\}$. The problem is to find one substitution W_{i_j} from R_{W_i} for each component W_i to be replaced. There are many feasible sets of substitutions functionally possible, given by the space $\mathcal{P} = R_{W_1} \times R_{W_2} \cdots \times R_{W_n}$. We denote the composition obtained by making the set of substitutions $S = \{W_1^s, W_2^s, \ldots W_n^s\}$ (where W_i^s denotes a substitute service in the set R_{W_i}) as $(C \ominus R) \oplus S$. We are interested in finding a set of substitutions $S \in \mathcal{P}$ that maximizes the preference of the resulting composite service, i.e., $\nexists S' \in \mathcal{P} : V_{(C \ominus R) \oplus S'} \succ V_{(C \ominus R) \oplus S}$.

One way to search for an optimal solution is by brute force: exploring the entire space of sets of substitutions $\mathcal{P}$, and finding the set of non-dominating set of substitu-

$V_{W_{11}} = \langle x_1, y_3 \rangle$	$V_{W_{21}} = \langle x_0, y_2 \rangle$	$V_{W_{31}} = \langle x_3, y_1 \rangle$
$V_{W_{12}} = \langle x_2, y_2 \rangle$	$V_{W_{22}} = \langle x_2, y_1 \rangle$	$V_{W_{32}} = \langle x_2, y_1 \rangle$
$V_{W_{13}} = \langle x_1, y_1 \rangle$	$V_{W_{23}} = \langle x_0, y_1 \rangle$	$V_{W_{33}} = \langle x_1, y_1 \rangle$

Table 6.3 Valuations of replacements

tions from that set. However, given that the number of components to be replaced is n and considering that each component has k functionally feasible substitutes, it is computationally expensive (number of possible sets of substitutions is exponential: $O(k^n)$) to explore the entire space.

A more efficient but naive approach for finding multiple component substitutions according to the user preferences would be to execute the one-component substitution algorithm multiple times, once for each component to be substituted. However, there is no guarantee that this approach would give the best possible set of substitutions, as illustrated by the following example.

Example 11. *Consider a composition C that requires three services to be replaced, $W = \{W_1, W_2, W_3\}$. Let there be functionally feasible substitutions, $R_{W_1} = \{W_{11}, W_{12}, W_{13}\}$, $R_{W_2} = \{W_{21}, W_{22}, W_{23}\}, R_{W_3} = \{W_{31}, W_{32}, W_{33}\}$ respectively. Table 6.3 shows their valuations over two non-functional attributes X and Y with domains $\{x_0, x_1, x_2, x_3\}$ and $\{y_0, y_1, y_2, y_3\}$ respectively. The preferences over the variables are given in Figure 6.5.*

Let $V_{C \ominus W} = \langle x_2, y_3 \rangle$; then the naive substitution approach will execute the single-component substitution for W_1, W_2 and W_3 independently, yielding replacements W_{11}, W_{21} and W_{31} respectively according to the algorithm presented in Section 6.3.2.2. The resulting preference value $V_{C \ominus W \oplus \{W_{11}, W_{21}, W_{31}\}} = \langle x_0, y_1 \rangle$. However, note that there exists another solution, namely replacements W_{12}, W_{22} and W_{31} for W_1, W_2 and W_3 respectively, which yields a better solution with valuation $V_{C \ominus W \oplus \{W_{12}, W_{22}, W_{31}\}} = \langle x_2, y_1 \rangle$.

The reason for the sub-optimal solution obtained by the naive approach in the above example is that it does not consider the effect of the choice of replacement for one component on the choice of replacement for others.

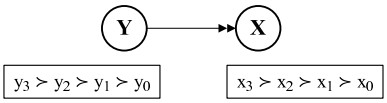


Figure 6.5 Preferences: Multiple Component Substitution

The solution obtained by the naive approach can be improved by having a search procedure that chooses the optimal replacement for a component W_i, *contingent* on the previous replacement choices already made. However, note that even in such an approach, the *order* in which we choose replacements for components plays an important role in determining the solutions. For example, if we choose the replacement for W_1 first, followed by a replacement for W_2 (given the choice for W_1), followed by a choice for W_3 (given the choices for W_1 and W_2), we obtain the sub-optimal solution W_{11}, W_{21} and W_{31} with valuation $\langle x_0, y_1 \rangle$. Instead, if we choose the replacement for W_3 first, followed by W_2 and W_1, then the resulting solution is optimal: the choice for W_3 is W_{31}; the choice for W_2 (*given* the choice for W_3) is W_{22}, and the choice for W_1 (*given* the choices for W_3 and W_2) is W_{12}. The resulting valuation of the substituted composite service is $\langle x_2, y_1 \rangle$ and it is optimal. Thus, the order in which the replacements for the components are chosen impacts the optimality.

In the absence of any information regarding the order in which components have to be replaced, the *only* way to guarantee an optimal solution is to explore all possible orders. In effect, this problem generalizes the known NP-hard traveling salesman problem [Cormen et al., 2001] that involves finding the optimal ordering of points in a plane such that the overall real-valued cost is minimized. The difference in our case is that we deal with qualitative valuations instead of real-valued costs.

We propose an approach to finding the optimal solution by exploiting that fact that the optimal solution corresponds to some *preferred order* in which the services are considered to be replaced. In the above example, such orders are (W_3, W_2, W_1) and (W_3, W_1, W_2). We present an algorithm that organizes the possible orderings in the form

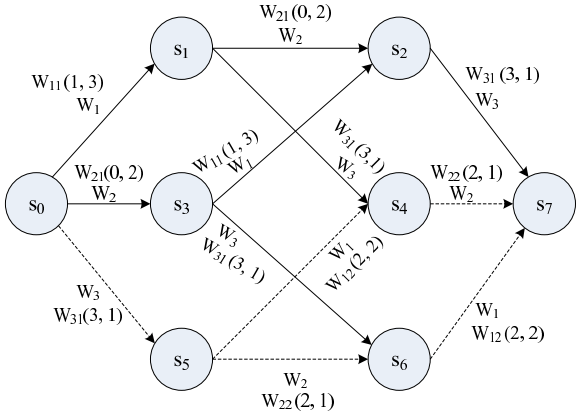


Figure 6.6 Multiple Component Substitution

of a lattice and obtains the preferred order as the shortest path between two nodes in the lattice.

6.3.4 Finding Preferred Order

We construct a lattice with the bottom of the lattice as the partial composition $C \ominus W$, and the top of the lattice as the fully substituted or *repaired* composition, namely $C \ominus W \oplus S$. Each node in level l (i.e., nodes that are l steps away from the bottom) of the lattice is the partial composition $C \ominus W$ composed with l different substitutes. There are a total of $n+1$ levels in the lattice and at each level l of the lattice, there are $\binom{n}{l}$. In particular, the level 0 of the lattice is the bottom and level $n+1$ is the top. The lattice for Example 11 is illustrated in Figure 6.6, where s_0 corresponds to $C \ominus W$ and s_7 corresponds to $C \ominus W \oplus S$, where S is the optimal substitution. We note that each path in the lattice specifies one order of selecting substitutes in W, and we denote the valuation of the fully substituted composition obtained through a path p in the lattice as V_p.

We assign the cost of each node in the lattice as follows. The cost of the bottom of the lattice (s_0 in Figure 6.6) is $V_{C \ominus W}$. The cost of a path of length l from s_0 to any

other node is the given by $V_{C\ominus W\oplus\{\widehat{W}_1,\ldots,\widehat{W}_l\}}$, where $\widehat{W}_i$ is locally-preferred substitute for

$$W_i \in \{W_1, W_2, \ldots, W_n\} - \bigcup_{j<i}\{W_j \mid \widehat{W}_j \text{ substitutes } W_j\}$$

For example, in Figure 6.6, there are two paths from s_0 to s_4. The cost of substitution along the path s_0, s_1, s_4 is $V_{C\ominus W\oplus\{W_{11},W_{31}\}} = \langle x_1, y_1\rangle$; in this path substitution of W_1 is selected before the substitution for W_3. While the cost of substitution along the path s_0, s_5, s_4 is computed from the selection of substitution for W_3 followed by that for W_1. Cost of any node other than the bottom is the most preferred cost among the paths from the bottom of the lattice to itself. For example, cost of node s_4 is $\langle x_2, y_1\rangle$ due to the cost of the path s_0, s_5, s_4. Finally, the most preferred substitution is given by the path from the bottom to the top of the lattice which corresponds to the cost of the top. Note that, this path corresponds to the preferred-order.

Algorithm 6, inspired by Dijkstra's shortest path algorithm [Cormen et al., 2001], computes the cost of the top of the lattice. Whereas Dijkstra's shortest path algorithm works for quantitative costs, or cases when the valuations of the paths are totally ordered, Algorithm 6 works for partial orders as well. This is needed because a node can be associated with multiple costs as the cost of the paths from the bottom to that node may be incomparable. Furthermore, unlike Dijkstra's algorithm which works on real value comparison operator, $>$, Algorithm 6 uses dominance relation $\succ$ between non-functional attribute valuations.

Lines 1–4 initializes the cost of a node V_n and its parent $\rho(n)$ to $\triangledown$ and `undef` respectively. The symbol $\triangledown$ denotes the worst valuations of the non-functional attributes in our setting (e.g., $\langle L_R, L_S, L_A\rangle$). At Line 5, the bottom of the lattice is assigned the cost of $V_{C\ominus W}$, i.e., the value of the non-functional attributes of the composition C without the services in W. Line 7 obtains the set of nodes whose associated cost is not dominated by that of any other node. Initially, this set will be singleton containing only the bottom element (unless $V_{C\ominus W} = \triangledown$, in which case any substitution for the

Algorithm 6 PreferredSubstitutions($\succ_d$, $G(N, E)$)

1: **for all** $n \in N$ **do**
2: $V_n \leftarrow \nabla$
3: $\rho(n) \leftarrow$ `undef`
4: **end for**
5: $V_{\perp \in N} \leftarrow V_{C \ominus W}$
6: **while** $|N| > 0$ **do**
7: $N_1 \leftarrow \{n \in N : \nexists m \in N : V_m \succ V_n\}$
8: $N \leftarrow N \setminus N_1$
9: **for all** $n \in N_1$ **do**
10: **for all** $m \in N$ such that $(n, m) \in E$ **do**
11: **if** $V_{n \oplus m} \succ_d V_m$ **then**
12: $V_m \leftarrow V_{n \oplus m}$
13: $\rho(m) \leftarrow n$
14: **else if** $V_m \sim V_{n \oplus m}$ **then**
15: Create a node m' with same edges as m
16: $V'_m \leftarrow V_{n \oplus m}$
17: $N \leftarrow N \cup \{m'\}$
18: **end if**
19: **end for**
20: **end for**
21: **end while**

services in W is a preferred one). Line 10–13 identifies the one-step neighbors of the current node and updates their cost appropriately if the costs are comparable (Line 11). If there are two paths to m with incomparable costs (specifically with non-dominating costs – Line 14), then the node is split into two (Lines 15–17). In general, if there are p paths leading to a node in the lattice, and p' paths form the non-dominating set with respect to their valuations, then the node is split into p' nodes (replicating all the edges of the original node in the split nodes), one for each of the paths with non-dominated valuations. However, such splitting, which will increase the computational complexity, can be effectively avoided by soliciting information from the user to break the tie due to non-dominance whenever the condition at Line 14 is satisfied. In that case, our algorithm will be able to assign a unique cost to each node, thereby avoiding node splitting.

The algorithm terminates when the minimum cost of all nodes are assigned; At this

point, the valuations at the top element in the lattice correspond to the most preferred non-dominated substitutions.

6.3.4.1 Complexity

Let there be n components to be replaced, and k feasible substitutes for each of them. By our construction, for each edge of the lattice, we make one substitution, i.e., we choose the best substitute from k candidates. Assuming that we obtain unique cost for each node (i.e., costs of all paths leading to a node are comparable), the number of times we make such a selection is equal to the total number of edges in the lattice, which is $n.2^{n-1}$. Hence, we consider $k.n.2^{n-1}$ sets of substitutions in all. It can be shown that $\forall k > 2, \forall n > 4 : k^n > k.n.2^{n-1}$, i.e., our approach will be more efficient than brute force whenever $k > 2$ and $n > 4$.

Remark 2. *If the cost of a node is not unique, as would be the case if there are at least two paths to that node with non dominating costs, then an alternative to splitting the node is to solicit information from the user to break the tie between these costs. This leads to unique cost at each node. In the event, the user fails to provide such information, the algorithm can proceed by splitting the nodes. However, such splitting will lead to increase in computational cost of our method. Let m nodes be split at each level of our lattice representation. Then the complexity for computing the globally-preferred substitutions is*[4]

$$C(n, m) = k.n.2^{n-1} + k.m \sum_{i=2}^{n-1} (i! - 1)(n - i) \tag{6.1}$$

Using this, we can design a method that switches from our algorithm to brute-force method when m splits are made and any further splitting will make k^n less than $C(n, m)$.

[4] *The derivation of Equation 6.1 is not presented due to space constraint.*

6.3.5 Summary and Discussion

Web service substitution approaches [Pathak et al., 2007, Liu et al., 2005, Martens et al., 2006, Benatallah et al., 2006, Bordeaux et al., 2005] have been developed in the past that identify functionally feasible replacements from a given repository. Regarding non-functional properties, existing techniques have focused on analyzing non-functional properties as hard-constraints (e.g., reliability after substitution *must* be 70%) or modeled them as a minimization/maximization problem (e.g., reliability after substitution must be maximized) [Ardagna and Pernici, 2007, Claro et al., , Sohrabi et al., 2006]. These techniques are either based on constraint satisfiability or require the user to precisely quantify their preferences over non-functional attributes (or both), or they do not address the problem of substitution in detail. To the best of our knowledge, there has been little work on identifying *preferred substitutions* in accordance with user-specified *qualitative* intra-variable and relative importance preferences over non-functional attributes.

We have introduced two different variants of the substitution problem given preferences over non-functional attributes: context-insensitive and context-sensitive. Our treatment of these two variants parallels the work of Pathak et al. [Pathak et al., 2007] on service substitution based on functional attributes. However, it departs from the work of Pathak et al. in an important respect: it offers a solution to the service substitution problem taking into consideration the preferences over non-functional attributes. We have also addressed the problem of substitution of multiple services in a composition. We showed that, in the context-sensitive setting, the order in which services are substituted influences the quality of the solution obtained (relative to user-specified preferences over non-functional attributes). We refer to the substitution order(s) corresponding to the most preferred solution as the preferred order(s). In the absence of any additional information regarding the preferred order, solving the multiple service sub-

stitution problem in the context-sensitive setting appears to be NP-hard. We show how to represent all possible substitution orders compactly by arranging the services to be replaced within a lattice structure. Finally, we reduce the problem of obtaining the preferred order(s) to identifying the shortest path(s) between the bottom and top elements of the lattice. We identify the conditions under which our method is computationally more efficient than a brute force search. Our approach to service substitution based on preferences over non-functional properties is independent of the specific formalism used to represent functional requirements of a composite service as well as the specific algorithm used to assemble the composite service.

Work in progress includes experimental studies of the performance and scalability of our algorithms. We next plan to implement our approach in an existing system like MoSCoE that does functional substitution. In future, we would like to extend our framework for service substitution to preference models that allow specification of partial orders over variable domains. We also plan to extend our framework to allow hard constraints over non-functional attribute values in addition to qualitative preferences, so that constrained preferential optimization supported by TCP-nets [Brafman et al., 2006] can be leveraged to obtain preferred substitutions.

6.4 Web Service Adaptation

Overview. After a composite service that satisfies the functional requirements is deployed, the requirements of the user may change. Apart from changes in the functional requirements, service-oriented architectures often have to deal with changes in the user preferences over the non-functional attributes and/or repository of available components. We formulate the problem of adaptation in the face of such changes as iterative substitution of appropriate components in a composite service. We provide two *anytime* algorithms that produce a sequence of increasingly preferred adaptations

with time: a fast algorithm that searches for preferred adaptations by improving the valuation of the relatively more important attributes, and another that is computationally more intensive but guaranteed to produce at least one preferred adaptation, if one exists.

We consider the problem of service adaptation when there is a change in one or more of the following of a composite service: (a) user preferences over the non-functional attributes (e.g., due to a recent budget cut, cost is now relatively more important than the performance of the composition); (b) repository of available components (e.g., new components have been added to the repository that could potentially improve the non-functional attributes of an existing composition with respect to user preferences). While techniques for service adaptation with respect to changes in functional requirements [Chafle et al., 2006] have been proposed, the problem of adaptation with respect to changes in the preferences over non-functional attributes and/or to the repository has received little attention.

Non-functional requirements over attributes such as performance and cost are usually expressed in the form of user preferences over values for each attribute. Such preferences could be *quantitative* or *qualitative*. In general, quantitative preferences are hard to elicit from users whereas qualitative preferences are easier to elicit [Doyle and Thomason, 1999] because users often lack enough information to quantify their preferences. We consider the problem of identifying preferred adaptations when the users specify two types of qualitative preferences: (a) preferences over the various values of each non-functional attribute (e.g., the higher the performance, the better); (b) relative importance over different non-functional attributes (e.g., cost is more important than performance).

Example 12. *Let $C = W_1 \oplus W_2$ be a composite service that satisfies a functionality φ, with W_1 and W_2 being two atomic services such that W_1 has low performance (p_L) and high availability (a_H), and W_2 has high performance (p_H) and low availability (a_L). The*

user prefers services with high performance and availability in general ($p_H \succ p_L; a_H \succ a_L$). The attributes of C are then $\langle p_L, a_L \rangle$ considering the fact that a composite service is only as performant (or available) as its least performant (or available) component.

Suppose that the user changes his preference so that additionally availability is relatively more important than performance ($A \rhd' P$). Further, assume that the repository is updated with new atomic services W_1' with attributes $\langle p_M, a_H \rangle$ and W_2' with attributes $\langle p_H, a_M \rangle$ (subscript M stands for 'medium') such that two new compositions $C_1 = W_1' \oplus W_2$ and $C_2 = W_1 \oplus W_2'$ satisfy φ. Then the attributes of C_1 and C_2 are $\langle p_M, a_L \rangle$ and $\langle p_L, a_M \rangle$ respectively. Here, C_1 and C_2 are both preferred adaptations of C with respect to the updated repository and preferences because C_1 has better performance, and C_2 has better availability respectively compared to C. Furthermore, C_2 is the most preferred adaptation because the relatively more important attribute, namely availability, has a better valuation in C_2 compared to C_1.

We use the TCP-net [Brafman et al., 2006] preference formalism for eliciting user preferences over the non-functional attributes in the form of qualitative, totally ordered *intra-attribute* preferences (over the valuations of each attribute), and partially ordered *relative importance* (over the attributes). We however note that our solution approaches can be applied equally well with any other suitable formalism for representing and reasoning with such preferences. Our approach is based on the fact that any adaptation of a composition can be obtained by replacing one or more components in the existing composition. Our main contributions are:

1. We reduce the problem of identifying increasingly preferred adaptations to the problem of iteratively substituting one or more components in the given composition with respect to the user preferences. We identify components that have to be necessarily replaced in a composition in order to improve the valuation of the composition with respect to an attribute.

2. We develop two sound adaptation algorithms with the *anytime* property: they produce a sequence of increasingly preferred (with respect to the new preferences and repository) compositions with time, each composition in the sequence being an adaptation of its predecessors. Both algorithms search the space of possible adaptations by substituting some of the components in the given composition. While the first algorithm iteratively attempts to improve the valuation of the composition with respect to the relative importance order of attributes and is computationally faster, the second algorithm is more computationally intensive but is guaranteed to produce at least one preferred adaptation, if one exists.

3. We provide a comparative analysis of our algorithm that is guaranteed to produce a preferred adaptation if one exists against the blind search algorithm in terms of the number of subsets of components that they consider to substitute.

6.4.1 Service Adaptation via Service Substitution

We focus on the problem of service adaptation in the face of changes in (a) the user preferences ($\{\succ_i\}$ and $\rhd$), and/or (b) in the repository (R) of available components. Note that the deletion of components in the repository can be handled using service substitution algorithms [Santhanam et al., 2009b]. Hence, we consider we consider only changes to the repository in the form of addition of new components. We refer to the updated preferences and repository by $\{\succ'_i\}$, $\rhd'$, $\succ'$ and R' respectively. In the event of such changes, it is possible that an already deployed composition C satisfying a functionality φ is not the most preferred composition satisfying φ. In such a case, it is often desirable to explore possible changes to C that could result in a composition C' such that $C' \models \varphi$, and C' is preferred to C as per the new preferences ($V_{C'} \succ' V_C$).

Consider a setting where a user attributes higher importance to the availability of a composite service and deploys a composition with high availability at a higher cost.

Due to the changes in the financial policy or recently imposed budget constraints in an organization, the user may change his preferences over the non-functional attributes of a composite service such that cost of the composition is relatively more important than its availability. In this case, the user may want to *adapt* the existing composition so that the adaptation has lesser cost but is also less available.

In another setting, for the same composition C, let the user's preference remain the same, and suppose that new components were added to the repository (i.e., $R \subseteq R'$) such that a new component $W_k \in R' \setminus R$ can replace an existing component W in C, resulting in a composition $C' = (C \ominus W) \oplus W_k \models \varphi$ with $V_{C'} \succ V_C$. Here, it is desirable to adapt the existing composition C to C' in return for a better preference valuation.

Indeed, there may well be settings in which both the user preferences over the non-functional attributes, as well as the repository change at the same time. In all the above scenarios, it is desirable to adapt the existing composite service C with respect to such changes, so as to obtain a more preferred composition that may not have been possible in the absence of such changes. We now formally define the problem of service adaptation.

Definition 37 (Service Adaptation)**.** *Given an existing composite service* $C = W_1 \oplus W_2 \ldots \oplus W_n$ *that achieves the desired functionality* φ *(*$C \models \varphi$*), an updated set of preferences* $\succ'$ *(specified by* $\{\succ'_i\}$ *and* $\rhd'$ *of the new TCP-net), and an updated repository* R'*,* service adaptation *amounts to identifying one or more compositions* $\{C' \mid C' \models \varphi\}$ *such that* $V_{C'} \succ' V_C$*. Each* C' *is said to be a preferred adaptation of* C*.*

Example 13. *Consider the problem of identifying the preferred adaptation of* $C = W_1 \oplus W_2$*, with non-functional attributes* P *(performance) and* A *(availability) with domains* $D_P = \{p_L, p_M, p_H\}$ *and* $D_A = \{a_L, a_M, a_H\}$ *(subscripts* L, M, H *stand for low, medium, high respectively). Suppose that the updated repository is* $R' = \{W_1, W'_1, W''_1, W_2, W'_2, W''_2\}$ *with the valuations of the components services as given in Table 6.4, and updated preferences are* $p_H \succ'_P p_M \succ'_P p_L$*,* $a_H \succ'_P a_M \succ'_C a_L$ *and* $A \rhd' P$ *(availability more important*

than performance). The feasible compositions $\{C_1 \ldots C_9\}$ *and their respective valuations are shown in Table 6.5. In this case, the ordering among the valuations of the adaptations is:* $V_{C_5} \succ \{V_{C_8}, V_{C_9}\} \succ \{V_{C_4}, V_{C_6}, V_{C_7}\} \succ \{V_{C_2}, V_{C_3}\} \succ V_{C_1}$. *Hence,* $C_2 \ldots C_9$ *are all preferred adaptations to* C, C_5 *being the most preferred adaptation with low performance and high availability, in accordance with the user's updated relative importance preference* ($A \rhd' P$).

Service	Valuation
W_1	$\langle p_M, a_L \rangle$
W_1'	$\langle p_L, a_H \rangle$
W_1''	$\langle p_H, a_M \rangle$
W_2	$\langle p_L, a_M \rangle$
W_2'	$\langle p_H, a_H \rangle$
W_2''	$\langle p_M, a_M \rangle$

Table 6.4 Components

Composition	Valuation
$C_1 = W_1 \oplus W_2$	$\langle p_L, a_L \rangle$
$C_2 = W_1 \oplus W_2'$	$\langle p_M, a_L \rangle$
$C_3 = W_1 \oplus W_2''$	$\langle p_M, a_L \rangle$
$C_4 = W_1' \oplus W_2$	$\langle p_L, a_M \rangle$
$C_5 = W_1' \oplus W_2'$	$\langle p_L, a_H \rangle$
$C_6 = W_1' \oplus W_2''$	$\langle p_L, a_M \rangle$
$C_7 = W_1'' \oplus W_2$	$\langle p_L, a_M \rangle$
$C_8 = W_1'' \oplus W_2'$	$\langle p_H, a_M \rangle$
$C_9 = W_1'' \oplus W_2''$	$\langle p_M, a_M \rangle$

Table 6.5 Compositions

6.4.2 Computing Preferred Adaptations

In general, given any composition C satisfying φ and the associated new preferences $\succ'$ and repository R', we could aim to achieve one of two goals, namely (a) finding the set of *most preferred* adaptation(s) of C among all possible adaptations with respect

to $\succ'$ and R'; or (b) finding a set of adaptations that are *more preferred* than C with respect to $\succ'$ and R'.

Recall from Definition 37 that a composition C' is an adaptation of C if and only if $C' \models \varphi$. Because C' differs from C precisely in terms of the set $S = W_C \setminus \mathcal{C}'$ of components, C' can be obtained by substituting the set S of components in C by any sound and complete functional substitution algorithm. Therefore, in order to obtain the *most* preferred adaptations, we need to address the following question: What is the set of components in C that need to be substituted to obtain the most preferred adaptations?

In the absence of any additional information regarding which subsets of W_C have to be substituted in C, the most straightforward method for obtaining the *most* preferred adaptation(s) is to compute compositions resulting from substitution of *all* possible subsets of W_C in C, and then selecting those with the most preferred valuations. However, in the worst case, such a procedure for finding the most preferred adaptation(s) to C requires us to solve $2^{|W_C|}$ instances of the substitution problem, which is computationally expensive. Although discouraging, this does not rule out the possibility of finding adaptations to C that are *more* preferred (not necessarily *most* preferred) compared to C with reasonable efficiency.

One way to make the search for adaptations more efficient is to explore only some of the subsets of W_C that (when substituted in C) are likely to yield preferred adaptations. We note that we can obtain a preferred adaptation C' of C by substituting a set S of components in C such that at least one of the attribute valuations of C' is preferred to that of C (by the TCP-net preference semantics [Brafman et al., 2006]). In order to identify such a set S to be substituted in C, we use the notion of *sole dependence*.

Definition 38 (Sole Dependence and Sole Cause[Santhanam et al., 2009b])**.** *A composition C is said to be* solely dependent *with respect to an attribute X_i on the set $\delta(C, X_i) = \{W_j' \mid V_{W_j'}(X_i) = V_C(X_i) \wedge W_j' \in^{\oplus} C\}$ of components. Alternatively, the set*

$\delta(C, X_i)$ *of components is said to be the* sole cause *of the valuation of* C *with respect to* X_i.

Intuitively, if a component is in $\delta(C, X_i)$, then the valuation of C with respect to X_i can never be improved without replacing it with another component having a better valuation with respect to X_i. Moreover, note that if $\delta(C, X_i) = \{W_1' \ldots W_k'\}$ then $V_{C \ominus W_1' \ominus \ldots \ominus W_k'}(X_i) \succ_i V_C(X_i)$. Hence, improving the valuation $V_{W_j'}(X_i)$ for each $W_j' \in \delta(C, X_i)$ improves C's valuation of X_i as well (provided there is at least one other component $W \in^{\oplus} C$ such that $V_W(X_i) \succ V_C(X_i)$). In terms of Definition 35, $\delta(C, X_i)$ represents the set of components in C that have the worst possible valuation with respect to X_i. Example 14 illustrates this concept.

Example 14. *Consider the composition* $C_1 = W_1 \oplus W_2$ *with valuation* $V_{C_1} = \langle p_L, a_L \rangle$ *in Example 13.* C_1 *is solely dependent on* W_1 *with respect to* A *and on* W_2 *with respect to* P*, i.e.,* $\delta(C, P) = \{W_2\}$ *and* $\delta(C, A) = \{W_1\}$*. This means that the poor availability of* C_1 *is due to the presence of component* W_1*, and also that if* W_1 *can be replaced in* C_1 *with a component having higher availability, then* C_1 *would have higher availability.*

We next describe an algorithm that uses the above notion of sole dependence to compute a sequence of adaptations of C, namely $C^0, C^1, \ldots C^n$ such that $C = C^0$ and $V_{C^{i+1}} \succ' V_{C^i}$ for all $0 \leq i < n$. The key idea underlying the algorithm is that it successively improves the preference valuation of the given composition C through $C^0, C^1, \ldots C^n$ by iteratively formulating and solving specific instances of the substitution problem. Rather than considering all possible subsets of W_C to substitute, this algorithm chooses a specific subset of W_C at each step, such that adaptations with respect to the relatively more important attributes are considered ahead of others, i.e., if $X_i \rhd X_j$ then the algorithm explores adaptations that are likely to have a better valuation of X_i prior to exploring adaptations that are likely to have a better valuation of X_j.

Algorithm 7 `AttributeAdapt`$(C, \succ', X_i, M)$

1: $S = \{W_j \mid V_{W_j}(X_i) = V_C(X_i) \wedge W_j \in^{\oplus} C\}$
2: Substitute S in C: $\psi = \{C' \mid C' \in$ `PrefSub`$(C, S)\}$
3: Pick only the adaptations that are preferred to C:
$\theta = \{C' \mid C' \in \psi \wedge V_{C'} \succ' V_C\}$
4: **if** $(\theta = \emptyset)$:
5: **if** $(i \neq m)$: `AttributeAdapt`$(C, \succ', X_{i+1}, M)$
6: **else**: **return**
7: **else**:
8: Select and Output $C' \in \theta$
9: `AttributeAdapt`$(C', \succ', X_i, M)$

6.4.3 A Sound Adaptation Algorithm

In order to identify adaptations of any C^i that are likely to have a better valuation for an attribute X_j, `AttributeAdapt` formulates an instance of the service substitution problem that substitutes the set $\delta(C^i, X_j)$ of components in C^i. The rationale behind choosing $\delta(C^i, X_j)$ as the set of components to substitute is that these components are the *sole cause* of the poor valuation of C^i with respect to X_j, i.e., if all the components in $\delta(C^i, X_j)$ are substituted to produce an adaptation C^{i+1} with components having better valuations for X_j, then $V_{C^{i+1}}(X_j) \succ' V_{C^i}(X_j)$ (provided there are other components in C^i besides those in $\delta(C^i, X_j)$). Furthermore, the algorithm produces adaptations by seeking to substitute components $\delta(C^i, X_j)$ in C^i iteratively until there exist no more preferred adaptations, and then repeats the same with respect to all attributes in $\mathcal{X}$, in the order of their relative importance.

Let $M = (X_1, X_2, \ldots X_m)$ be an ordering of $\mathcal{X}$ such that $X_{j+1} \not\succ X_j$ for all $0 < j < m$. For each attribute X_j in the order M, we consider adapting C^i by substituting the components $\delta(C^i, X_j)$ in C^i (Line 1) using a sound and complete multiple substitution algorithm (such as the one given in [Santhanam et al., 2009b]; `PrefSub` in Line 2). Note that although the substitution algorithm `PrefSub` may return many compositions (ψ), each of which is an adaptation of C^i, not all of them may be more preferred compared to

C. Hence, among the set (ψ) of all compositions returned by the substitution algorithm, we select only those that are strictly preferred to C^i (θ in Line 3). `AttributeAdapt` then chooses one of them arbitrarily, say $C^{i+1} \in \theta$, and outputs it to the user (Line 8) as a (more) preferred adaptation of C (than C^i). Similarly, the algorithm attempts to produce a sequence $C^{i+2}, C^{i+3}, \ldots$ of compositions by repeating (Line 9) the same procedure (i.e., substituting $\delta(C^{i+k}, X_j)$ in C^{i+k} for all $k > 1$) such that $V_{C^{i+k}} \succ' V_{C^{i+k-1}}$ for all $k > 1$. This iteration is continued until the substitution of $\delta(C^i, X_j)$ in C^i ceases to improve the preference valuation of C^i, i.e., $\theta = \emptyset$ (Line 4) or $\forall C' \in \psi : V_{C'} \not\succ' V_{C^i}$. The algorithm then proceeds to the next attribute X_{j+1} in the ordering M and repeats the same procedure (Line 5). The algorithm terminates when all the attributes in the order M have been considered (Line 6).

We next study some properties of `AttributeAdapt`. In particular, we would like to investigate if any adaptation returned to the user by the algorithm would be preferred (with respect to the user's updated preferences) to the original composition (soundness), and whether the user would be provided with all the preferred adaptations by the algorithm (completeness).

6.4.4 Properties of `AttributeAdapt`

Proposition 19 (Termination)**.** *Given a composition C, $\succ'$, R' and a substitution algorithm* ***`PrefSub`*** *that terminates in a finite number of steps,* ***`AttributeAdapt`*** *terminates in a finite number of steps.*

Proof. The only steps in which `AttributeAdapt` recurses are Lines 5 and 9. The recursion at Line 9 occurs whenever there exists a preferred adaptation C' to C ($V_{C'} \succ' V_C$) that has been obtained in the current iteration by substituting $\delta(C, X_i)$ in C. This recursion then invokes the algorithm with C', all other parameters being the same. Because there are only a finite number of possible compositions (the size of the repository

is finite), and since $\succ'$ is an irreflexive preference relation, the recursion at Line 9 must terminate in finite steps. The recursion at Line 5 iterates the algorithm for each $X_i \in \mathcal{X}$ in the order determined by M. Since $\mathcal{X}$ (number of non-functional attributes) is finite, this recursion is invoked a finite number of times.

Note that although there are two recursive calls, (a) they are mutually exclusive; and (b) the recursion of Line 9 for a single X_i completes *entirely within* each recursion of Line 5 corresponding to that X_i. Hence, the algorithm terminates in a finite number of steps. □

Proposition 20 (Soundness)**.** *Given a composition C, $\succ'$, R' and a sound substitution algorithm* `PrefSub`*,* `AttributeAdapt` *is sound, i.e., if a composition $C' \neq C$ is output by the algorithm, then $V_{C'} \succ' V_C$ and $C' \models \varphi$.*

Proof. From Lines 3 and 8, any composition C' ($\in \theta$) output by `AttributeAdapt` (in Line 8) is such that $V_{C'} \succ' V_C$. Moreover, $C' \in \theta \Rightarrow C' \in \psi$ as per Lines $2-3$, which in turn implies that $C' \models \varphi$ under the assumption that `PrefSub` is a sound substitution algorithm. Hence, `AttributeAdapt` is sound. □

Proposition 21. `AttributeAdapt` *is not complete.*

Proof. Example 15 (see below) provides a counter example to the completeness of `AttributeAdapt`, where there exists a preferred adaptation $C' \neq C$ to C, but `AttributeAdapt` fails to find it. □

Example 15. *Consider an adaptation problem where $\mathcal{X} = \{X_1, X_2\}$, $C = W_1 \oplus \ldots \oplus W_5$ (i.e., $W_C = \{W_1 \ldots W_5\}$), $\delta(C, X_1) = \{W_1, W_2\}$ and $\delta(C, X_2) = \{W_3, W_4\}$. Suppose that there exist $W_1', W_2' \ldots W_5' \in R'$ such that each W_i' can functionally substitute W_i in C, however there are some compatibility issues among the components such that the only functionally feasible adaptation to C is $C' = W_1' \oplus W_2' \oplus W_3' \oplus W_4 \oplus W_5'$ and $V_{C'} \succ' V_C$. However,* `AttributeAdapt` *considers only the following sets to substitute*

Algorithm 8 `ExhaustiveAdapt`$(C, \succ', M)$

```
1: for each S ∈ P(W_C) do
2:  for each X_i ∈ X do
3:   if (S ⊇ δ_i):
4:     Substitute S in C: ψ = {C' | C' ∈ PrefSub(C, S)}
5:     Pick only adaptations that are preferred to C:
       θ = {C' | C' ∈ ψ ∧ V_C' ≻' V_C}
6:     if (θ ≠ ∅):
7:       Select and Output C' ∈ θ
8:       ExhaustiveAdapt(C', ≻', M)
9:       return
```

in C *:* $\{W_1, W_2\}$ *and* $\{W_3, W_4\}$*. Since both have no feasible adaptations (due to their incompatibility),* ***`AttributeAdapt`*** *fails to find* C'*, and terminates with* C*.*

Although `AttributeAdapt` is computationally fast, it is not exhaustive and hence it may terminate with no preferred adaptation even when one exists (as evident from Example 15). We next present algorithm `ExhaustiveAdapt` that, in contrast to the previous algorithm, searches the space of adaptations more exhaustively and is guaranteed to find a preferred adaptation if there exists one.

6.4.5 A Sound and Weakly Complete Algorithm

We call an algorithm that is guaranteed to identify a preferred adaptation if there exists one, as *weakly complete* (see Definition 24).

`ExhaustiveAdapt` considers adaptations resulting from the substitution of *all* possible subsets of W_C that include at least one of the $\delta(C, X_i)$'s. Note that the number of such subsets of W_C considered is still less than $2^{|W_C|}$. `ExhaustiveAdapt` eliminates from consideration other subsets of W_C to substitute in C, as substituting subsets of W_C that do not include any of the $\delta(C, X_i)$'s cannot yield a preferred adaptation.

Proposition 22. *Given a composition* C*, preference* $\succ'$ *and repository* R'*, the substitution of a set* $S \subseteq W_C$ *of components in* C *can possibly result in a preferred adaptation*

C' only if S includes at least one of the sets $\delta(C, X_i)$ such that $X_i \in \mathcal{X}$, i.e., if C' was obtained by substituting S in C, then $V_{C'} \succ' V_C \Rightarrow \exists X_i \in \mathcal{X} : S \supseteq \delta(C, X_i)$.

Proof. Suppose that by contradiction, there exists a set $S \subseteq W_C$ such that $\forall X_i \in \mathcal{X} : S \not\supseteq \delta(C, X_i)$ and the substitution of S in C produces an adaptation C' with $V_{C'} \succ' V_C$. Clearly, for each attribute $X_i \in \mathcal{X}$, because not all components in $\delta(C, X_i)$ were substituted in C, at least one of the components in $\delta(C, X_i)$ is present in C', i.e., $\forall X_i \in \mathcal{X} : \exists W \in \delta(C, X_i) : W \in^{\oplus} C \wedge W \in^{\oplus} C'$. Because $\delta(C, X_i)$ represents the worst valuation among the valuations of all components in C with respect to X_i (by Definition 35), it follows that $\forall X_i \in \mathcal{X} : \exists W \in^{\oplus} C' : V_W(X_i) = V_C(X_i)$. However, the existence of such a component W for each X_i in C' implies that the valuation of C' with respect to X_i can never be better than that of W with respect to X_i, i.e., $\forall X_i \in \mathcal{X} : V_W(X_i) = V_C(X_i) \wedge W \in^{\oplus} C' \Rightarrow V_{C'}(X_i) \not\succ'_i V_W(X_i) = V_C(X_i)$ (again, by Definition 35). Further, $(\forall X_i \in \mathcal{X} : V_{C'}(X_i) \not\succ'_i V_C(X_i)) \Rightarrow V_{C'} \not\succ' V_C$ due to semantics of $\succ'$ [Brafman et al., 2006]. This contradicts our assumption that $V_{C'} \succ' V_C$. □

From the above proposition, it follows that the set S of components to be substituted in C must include at least one $\delta(C, X_i)$ to produce a preferred adaptation. Based on this, `ExhaustiveAdapt` enumerates and explores all possible subsets of W_C except those that do not include any of the sets $\delta(C, X_i)$. The algorithm chooses (Lines $1-3$) all the sets $S : \exists X_i \in \mathcal{X} : \delta(C, X_i) \subseteq S \subseteq W_C$, and attempts to substitute each of them in C (Line 4) to obtain a set θ of preferred adaptations (Line 5). For each such S, the algorithm chooses one (if one exists), say C', outputs it to the user (Line 7) and recursively calls itself in an attempt to further improve the new adaptation C' (Line 8). The algorithm terminates when there are no more preferred adaptations to C'. The algorithm ignores all the other subsets S of W_C such that $\forall X_i \in \mathcal{X} : S \not\supseteq \delta(C, X_i)$.

6.4.6 Properties of `ExhaustiveAdapt`

Proposition 23 (Termination)**.** *Given a composition C, $\succ'$, R' and a substitution algorithm* ***`PrefSub`*** *that terminates in a finite number of steps,* ***`ExhaustiveAdapt`*** *terminates in a finite number of steps.*

Proof. The number of iterations over $\mathcal{P}(W_C)$ for each $X_i \in \mathcal{X}$ is finite. The only recursive call occurs in Line 8, and each time the recursion is on a different adaptation C'. Since there are only a finite number of such C' (due to the finite number of components in the repository), and $\succ'$ is a strict partial order on the set of all such C's, this recursion must terminate in a finite number of calls. □

Proposition 24 (Soundness)**.** *Given a composition C, $\succ'$, R' and a sound substitution algorithm* ***`PrefSub`****,* ***`ExhaustiveAdapt`*** *is sound, i.e., if a composition $C' \neq C$ is output by the algorithm, then $V_{C'} \succ' V_C$ and $C' \models \varphi$.*

Proof. Any composition C' output by `ExhaustiveAdapt` (in Line 7) is such that $V_{C'} \succ' V_C$ (Line 5), $C' \in \psi$ (Line 4). This implies that $C' \models \varphi$ (by the soundness the substitution algorithm `PrefSub`). Hence, `ExhaustiveAdapt` is sound. □

Proposition 25 (Weak Completeness)**.** ***`ExhaustiveAdapt`*** *is weakly complete.*

Proof. Suppose there exists a preferred adaptation C' to C (i.e., such that $V_{C'} \succ' V_C$). By the soundness and completeness of `PrefSub`, it follows that C' can be obtained from C by the substitution of some subset S of W_C in C. Moreover, Proposition 22 states that such a set S must satisfy the condition $\exists X_i \in \mathcal{X} : S \supseteq \delta(C, X_i)$. `ExhaustiveAdapt` explores all such subsets S, and therefore must find such a C' should one exist. Hence, `ExhaustiveAdapt` is weakly complete (Definition 24). □

In Example 15, while `AttributeAdapt` fails to find any preferred adaptation to $C = W_1 \oplus \ldots W_5$, `ExhaustiveAdapt` does find the preferred adaptation $C' = W_1' \oplus W_2' \oplus W_3' \oplus$

$W_4 \oplus W_5'$ due to the above property, by considering the set $S = \{W_1, W_2, W_3, W_5\}$ for substitution in C. Note that as a direct consequence of Proposition 25, it also follows that if `ExhaustiveAdapt` does not return any preferred adaptation to C, then in fact there is no preferred adaptation.

6.4.7 Efficiency of `ExhaustiveAdapt`

A blind search algorithm to compute preferred adaptations would involve considering all possible subsets of W_C to be substituted in C. In contrast, `ExhaustiveAdapt` omits from consideration exactly the subsets S of W_C that do not contain any of the $\delta(C, X_i)$'s entirely, i.e., it does not consider $S \subseteq W_C : \forall X_i \in \mathcal{X} : \delta(C, X_i) \not\subseteq S$. In Example 15, for instance, the sets S omitted by `ExhaustiveAdapt` are $\{W_1, W_3\}$, $\{W_1, W_4\}$, $\{W_2, W_3\}$, $\{W_2, W_4\}$, $\{W_1, W_3, W_5\}$, $\{W_1, W_4, W_5\}$, $\{W_2, W_3, W_5\}$ and $\{W_2, W_4, W_5\}$ (note that adding even one element to any of these sets makes them eligible for consideration by `ExhaustiveAdapt`). The worst case for `ExhaustiveAdapt` arises when each $\delta(C, X_i)$ constitutes a unique component $W \in W_C$ (i.e., $|\mathcal{X}| = |W_C|$ and $\forall X_i \in \mathcal{X} : \exists W_j \in W_C : \delta(C, X_i) = \{W_j\}$). In this case, each of the $2^{|W_C|}$ sets are explored by `ExhaustiveAdapt`.

The advantage of using `ExhaustiveAdapt` over the blind search can be measured by counting the number of subsets of W_C that are explored by the blind search but omitted by `ExhaustiveAdapt`. For simplicity of discussion, let us consider the difference between the number of subsets explored by `ExhaustiveAdapt` and the brute force approach when all the $\delta(C, X_i)$'s are disjoint. The number of such subsets S of W_C such that $\forall X_i : S \not\supseteq \delta_i$, i.e., those subsets that do not contain any of the δ_i's is obtained as follows.

Let $S \subseteq W_C$ be a set omitted from consideration by `ExhaustiveAdapt`. S can then be partitioned into elements belonging to $W_C - \cup_i \delta_i$, those belonging to δ_1, those belonging to δ_2 and so on because the sets $W_C - \cup_i \delta_i$, δ_1, δ_2 ... δ_m are pairwise disjoint (see Figure 6.7). That is, the sets S that are omitted from consideration by `ExhaustiveAdapt` are precisely those that can be constructed from various combinations of its partitions

$S \cap (W_C - \cup_i \delta_i)$, $S \cap \delta_1$, $S \cap \delta_2$ ... $S \cap \delta_m$ such that:

a) $S \neq \emptyset$

b) S can contain any subset of $W_C - \cup_i \delta_i$

c) For each δ_i, S can contain any subset of δ_i, except δ_i

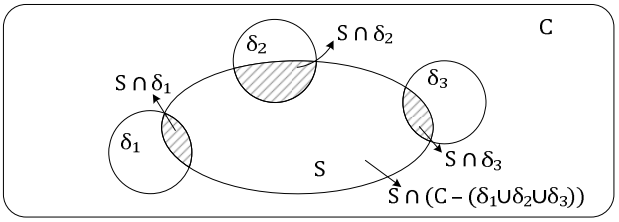


Figure 6.7 Partitions of $S \subseteq C$ excluding all δ_is for $m = 3$.

The number of ways of constructing the partition $S \cap (W_C - \cup_i \delta_i)$ is $2^{(|W_C| - \sum_i |\delta_i|)}$ and that for the partition $S \cap \delta_i$ is $2^{|\delta_i|} - 1$ (for each $X_i \in \mathcal{X}$). The total number of subsets S that are omitted by `ExhaustiveAdapt` is therefore $2^{(|W_C| - \sum_i |\delta_i|)} \times \prod_i (2^{|\delta_i|} - 1) - 1$ (the last term excludes the possibility where $S = \emptyset$). Hence, the number of sets explored by `ExhaustiveAdapt` is $2^{|W_C|} - 2^{(|W_C| - \sum_i |\delta_i|)} \times \prod_i (2^{|\delta_i|} - 1) - 1$. This is a tight bound for the case when all the δ_is are pairwise disjoint, and it is an upper bound when the δ_is overlap. Figure 6.8 shows a 3-dimensional plot of this function. The plot shows that `ExhaustiveAdapt` explores considerably lesser number of subsets of W_C to be substituted in C compared to blind search, to produce at least one preferred adaptation.

6.4.8 Summary and Discussion

Among functionally feasible compositions in service oriented applications, users often prefer some compositions over the others based on their non-functional attributes such as performance, security and cost. Service oriented architectures often admit changes in the user preferences over non-functional attributes of the deployed services and/or updates

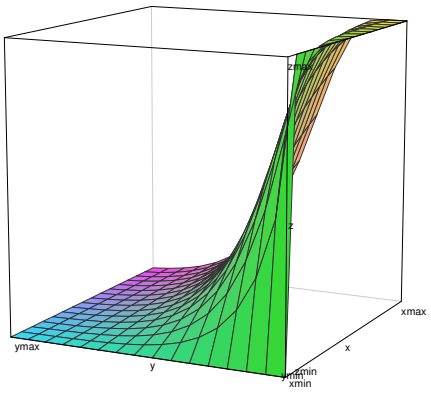


Figure 6.8 Number of subsets $S \subseteq W_C$ explored by `ExhaustiveAdapt` (z-axis) as a function of m (number of δ_is; the x-axis) and d (size of each δ_i; the y-axis). The plot is shown for $c = 10$ (number of components in W_C). We observed similar trends for plots for $c = 20, 30 \ldots 100$.

to the repository of available components. In the face of such changes, it is desirable to obtain and re-deploy a composition with the same functionality that is more preferred with respect to the changes.

We addressed the problem of automatically adapting a given composition when (a) there are changes to the user preferences over non-functional attributes and/or (b) new components are added to the repository of available components. We did not consider adaptation in response to changes involving deletion of components from the repository, as identifying preferred adaptations in such a case reduces to identifying preferred substitutions [Santhanam et al., 2009b]. We provided two algorithms that produce a sequence of increasingly preferred adaptations with time. Our algorithms take advantage of existing work on finding preferred substitutions to components in a composition, and can work with any substitution algorithm or preference formalism. Our algorithms do not consider changes in the functional requirements of a composition. However, they can be used in conjunction with any functional service adaption algorithm.

Our algorithms are sound and satisfy the anytime property: they produce a sequence of adaptations over time, such that each new adaptation in the sequence is strictly

preferred to its predecessors. The anytime property is especially useful in dynamic service oriented environments, as newer, more preferred adaptations generated by the algorithm can be re-deployed as and when they are computed by the algorithms without affecting the business continuity of the service. While the first algorithm we presented explores ways to iteratively improve the valuation of the composition with respect to the relative importance order of attributes and is computationally faster, the second algorithm is computationally more intensive but is guaranteed to produce at least one preferred adaptation, if one exists. A key element of our algorithms is the preference based comparison of alternatives (valuations of compositions), which can be handled efficiently for a large class of input preferences as demonstrated in [Santhanam et al., 2010b, Santhanam et al., 2010a].

In future, we plan to implement the algorithms in an existing service composition system such as MoSCoE [Pathak et al., 2006a] and investigate the performance of our algorithms in practice. We also plan to identify the precise computational complexity class for the problem of finding preferred adaptation in the face of change in preferences and the repository.

6.5 Discussion

Although throughout this chapter we have used TCP-nets to represent preferences, the formal methods and algorithms developed for composition, substitution and adaptation of Web services developed here can be used along with any other preference formalism. In particular, we note that the unconditional preference language as well as the efficient dominance testing techniques developed in Chapter 3 and Chapter 4 (for compositional systems) can be used along with the algorithms developed in this chapter to identify the preferred Web service compositions, substitutions and adaptations.

In addition, we also note that the algorithms for identifying preferred substitutions

and adaptations that we developed in this chapter for composite Web services can also be applied to other compositional systems. For example, in AI planning, a pre-compiled preferred plan can be altered (using the adaptation algorithm) when the agent's preferences change suddenly.

CHAPTER 7. Conclusion

Representing and reasoning with preferences is a problem of significant importance and interest in AI. Reasoning with qualitative preferences over a set of alternatives that are described by multiple attributes is known to be hard in general. In this thesis, we have studied three related problems that arise in the context of reasoning with qualitative preferences.

The first problem we address in this thesis is dominance testing, i.e., determining whether an outcome is preferred to another with respect to a set of preferences. This problem is is computationally hard for TCP-nets and other well known preference languages. This thesis provides formal methods for performing dominance testing efficiently in many practical cases. Secondly, we take up preference reasoning for compositional systems, where alternatives over which preferences are computed represent collections of objects rather than simple objects. We develop formalisms to reason with preferences over collections of objects based on the preferences over the attributes describing the objects, and provide algorithms to compute the most preferred collections. This thesis also addresses some problems of preference reasoning that arise in a service oriented software environment, where distributed Web services are assembled to build more complex compositions. Given preferences over the attributes of the Web services and some minimum requirements to be satisfied by any compositions, this thesis provides algorithms for identifying preferred service compositions.

Section 7.1 summarizes the primary contributions of this thesis, and Section 7.2 concludes the chapter with future work.

7.1 Contributions

The primary contributions of this thesis are as follows.

1. **Efficient Dominance Testing:** We provided techniques for efficient dominance testing with TCP-nets. In particular, we explored two ways of achieving efficient dominance testing:

   a) We first presented a preference language that allows expression of *unconditional* intra-variable and relative importance preferences and present a polynomial time dominance testing approach for this language.

   b) We explored a novel approach to dominance testing for TCP-nets in general that leverages the state-of-the-art techniques in *model checking* [Clarke et al., 1986, Queille and Sifakis, 1982, Cimatti et al., 2002]. To the best of our knowledge, this is the first practical solution to this problem. Our approach relies on a reduction of the dominance testing problem to reachability analysis in a graph of outcomes.

2. **Representing and Reasoning with Preferences for Compositional Systems:** We developed a preference formalism for compositional systems that allows users to specify preferences in terms of intra-attribute and relative importance preferences over a set of attributes, and includes mechanisms for:

   a) Computing the valuation of a composition: With respect to each attribute, we defined a generic *aggregation function* to compute the valuation of a composition as a function of the valuations of its components. We also presented a strict partial order preference relation for comparing two compositions with respect to their aggregated valuations of each attribute.

b) Comparing the valuations of compositions: We introduced a *dominance* relation that compares compositions (in terms of their aggregated valuations) with respect to the stated preferences, and established some of its key properties. In particular, we showed that this relation is a strict partial order whenever the intra-attribute preferences are strict partial orders and relative importance preference is an interval order.

3. **Algorithms for Identifying Preferred Compositions:** We developed a suite of algorithms for compositional systems that identify the set, or subset of the most preferred composition(s) with respect to the user preferences. In particular, we showed that under certain conditions, the algorithms are guaranteed to return only (sound), all (complete), or at least one (weakly complete) of the most preferred compositions. The algorithms we developed fall into two classes:

   a) those that first compute the set of all feasible compositions using a functional composition algorithm as a black box, and then proceed to find the most preferred among them using the preference relations developed in (1); and

   b) an algorithm that *interleaves* at each step the execution of a functional composition algorithm and the ordering of partial solutions with respect to user preferences. It requires the functional composition algorithm to be able to construct a composition satisfying the functional requirement incrementally, i.e., by iteratively extending partial compositions with additional components.

   We analyzed some key properties of the algorithms that yield specific conditions on the structure of preferences, under which the algorithms produce only/at least one/all of the most preferred solutions.

4. **Experimental Comparison of Composition Algorithms:** We presented results of experiments that compare performance of the above algorithms for com-

puting the most preferred compositions on a set of simulated composition problem instances. The results demonstrate the feasibility of our approach in practice, and compare our algorithms with respect to the quality of (number of *good* or most preferred) solutions produced by the algorithms and their performance (running time) and efficiency (the number of times they invoke the functional composition algorithm). Based on an analysis of the experimental results, we also identified key theoretical properties of the dominance relation directly as a function of the user preferences, which were unknown apriori.

5. **Application to Web Services:** Finally, we applied the above techniques to the domain of Web services and provided a set of algorithms for the composition, substitution and adaptation of Web services with respect to the user's preferences over the non-functional attributes and a given repository of component services. The algorithms are tailored to suit the needs of service-oriented architectures, but similar techniques can be applied in compositional systems other than Web services as well, such as AI planning, team formation, etc.

Although our approach to efficient dominance testing focuses on acyclic CP-nets and TCP-nets, our techniques for efficient dominance testing have a broader applicability. They can be applied to any preference language as long as the semantics of the preference language is given in terms of the satisfiability of graph properties (including GCP-nets, cyclic CP-nets and the language due to Wilson).

The preference formalism we developed for assembling compositional systems is generic in the sense that one can use any aggregation function that appropriately represents the valuation of the composition as a function of the valuations of its constituents. In particular, we showed examples of aggregation functions that compute the summation (numeric), the minimum/maximum valuation (totally ordered), or the set of *worst* valuations (partially ordered) of the constituents of a composition. Our formalism also

provides flexibility in choosing the preference relation that compares sets of valuations of two compositions, so that *any* strict partial order preference relation can be used.

All our algorithms for assembling compositions are completely independent of various aspects of the preference formalism, namely, the choice of aggregation functions, the preference relation used to compare aggregated valuations over a single attribute, and the dominance relation used to compare compositions over all attributes, except that the preference relations are strict partial orders. The theoretical and experimental results provide precise conditions under which the algorithms produce only/at least one/all of the most preferred solutions. This enables the user to choose an algorithm of his/her choice for particular problem instance, depending on the quality of solutions that is needed. In addition, our analysis also allows the user to tradeoff the quality of solutions produced against performance and efficiency or vice-versa.

Application of the above techniques for the development and maintenance of service-oriented software systems poses some additional challenges. For example, the algorithm for identifying the most preferred Web service composition may need to deal with unknown valuations of some of attributes of some of the components in the repository. We showed how the developed preference formalism can be used to address problems related to the lifecycle maintenance of composite Web services in a service-oriented environment, namely, identifying preferred substitutions and adaptations of composite services after identifying and deploying a composite Web service.

7.2 Future Work

Our work opens up the following directions for future research.

1. **Efficient Dominance Testing:** Preference networks are popular as graphical preference representation tool, but their applicability in practice is restricted by the fact that dominance testing for TCP-nets is PSPACE-complete. In this thesis,

we considered the unconditional fragment of the conditional preference language encoded by the TCP-net family, and defined a dominance relation for this fragment that can be computed in polynomial time. In another approach, we employed the state-of-the-art model checker NuSMV for computing dominance for TCP-net preferences. There are at least two possible directions for future work.

a) It would be useful to identify other fragments of the TCP-net language and corresponding dominance relations that can be efficiently computed, such as those that may include conditional preferences but restrict the structure of the conditional dependencies or relative importance preferences.

b) We would like to implement our solution for efficient dominance testing using NuSMV model checker, and perform experiments to determine the performance of our approach in practice. It would be interesting to see how big of a preference network (in terms of number of variables, dependencies, etc.) can be handled by such an implementation in practice.

c) Model checkers other than NuSMV such as SPIN can be employed for dominance testing. It would be interesting to compare the performance between the implementations that use different model checkers.

2. **Preference Formalism for Compositional Systems:** We have already seen that our preference formalism is generic and flexible in terms of the choice of aggregation functions to compute the valuation of a composition in terms of the valuations of its components, as well as in the overall dominance relation used to compare different compositions. It would be interesting to see if aggregation functions that may need to be employed for other applications will still work with the rest of the formalism as is. The same applies to preference relations for comparing aggregated preference valuations as well.

3. **Dominance:** It would be interesting to explore alternative notions of dominance that preserve the rationality of choice, by requiring a different set of properties (e.g., those that satisfy *negative-transitivity* instead of transitivity). The impact of such a dominance relation on the preference formalism and algorithms, and the semantic relationship between those and the dominance relation proposed in this thesis will be of interest.

4. **Experiments:** In evaluating the performance of our composition algorithms for compositional systems, we made use of the growth process of a uniform recursive tree data structure to simulate the searching of a functionally feasible composition. Based on the nature of specific target applications, it would be useful to consider other possible methods to generate the search space for composition.

5. **Application to Web Services:** We would like to implement our composition, substitution and adaptation algorithms for composite Web services in a Web service composition framework such as MoSCoE, in conjunction with various types of functional composition algorithms.

6. **Other Applications:** We would like to extend the application of our preference formalism and algorithms to other application domains such as preference-based planning, team formation, etc.

Bibliography

[Agrawal and Wimmers, 2000] Agrawal, R. and Wimmers, E. L. (2000). A framework for expressing and combining preferences. *SIGMOD Rec.*, 29(2):297–306.

[Ardagna and Pernici, 2007] Ardagna, D. and Pernici, B. (2007). Adaptive service composition in flexible processes. *IEEE Trans. Softw. Eng.*, 33(6):369–384.

[Bacchus and Grove, 1995] Bacchus, F. and Grove, A. J. (1995). Graphical models for preference and utility. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-1995)*, pages 3–10.

[Baier et al., 2008] Baier, J. A., Fritz, C., Bienvenu, M., and McIlraith, S. (2008). Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI), Nectar Track*, pages 1509–1512, Chicago, Illinois, USA.

[Baier and McIlraith, 2008] Baier, J. A. and McIlraith, S. A. (2008). Planning with preferences. *AI Magazine*, 29(4):25–36.

[Barbera et al., 2004] Barbera, S., Bossert, W., and Pattanaik, P. K. (2004). Ranking sets of objects. In *Handbook of Utility Theory. Volume II Extensions*, chapter 17, pages 893–977. Kluwer Academic Publishers.

[Benatallah et al., 2006] Benatallah, B., Casati, F., and Toumani, F. (2006). Representing, analysing and managing web service protocols. *Data Knowl. Eng.*, 58(3):327–357.

[Berbner et al., 2006] Berbner, R., Spahn, M., Repp, N., Heckmann, O., and Steinmetz, R. (2006). Heuristics for qos-aware web service composition. In *Proceedings of the IEEE International Conference on Web Services*, pages 72–82.

[Bichler and Lin, 2006] Bichler, M. and Lin, K. J. (2006). Service-oriented computing. *Computer*, 39(3):99–101.

[Biere et al., 1999] Biere, A., Cimatti, A., Clarke, E. M., and Zhu, Y. (1999). Symbolic model checking without bdds. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS 1579*, pages 193–207. Springer.

[Binshtok et al., 2009] Binshtok, M., Brafman, R. I., Domshlak, C., and Shimony, S. E. (2009). Generic preferences over subsets of structured objects. *Journal of Artificial Intelligence Research*, 34:133–164.

[Bordeaux et al., 2005] Bordeaux, L., Salaun, G., Berardi, D., and Mecella, M. (2005). When are two web services compatible? *Lecture Notes in Computer Science*, 3324:15–28.

[Börzsönyi et al., 2001] Börzsönyi, S., Kossmann, D., and Stocker, K. (2001). The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, Washington, DC, USA. IEEE Computer Society.

[Boutilier et al., 2001] Boutilier, C., Bacchus, F., and Brafman, R. I. (2001). Ucp-networks: A directed graphical representation of conditional utilities. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI-2001)*, pages 56–64.

[Boutilier et al., 2004] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., and Poole, D. (2004). Cp-nets: A tool for representing and reasoning with conditional

ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191.

[Brafman and Domshlak, 2009] Brafman, R. and Domshlak, C. (2009). Preference handling - an introductory tutorial. *AI magazine*, 30(1).

[Brafman, 2004] Brafman, R. I. (2004). Database preference queries revisited extended abstract.

[Brafman et al., 2006] Brafman, R. I., Domshlak, C., and Shimony, S. E. (2006). On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research*, 25:389424.

[Chafle et al., 2006] Chafle, G., Dasgupta, K., Kumar, A., Mittal, S., and Srivastava, B. (2006). Adaptation in web service composition and execution. In *Int. Conf. on Web Services*, pages 549–557.

[Chan et al., 2005] Chan, C.-Y., Eng, P.-K., and Tan, K.-L. (2005). Stratified computation of skylines with partially-ordered domains. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 203–214, New York, NY, USA. ACM.

[Chomicki, 2002] Chomicki, J. (2002). Querying with intrinsic preferences. In *EDBT*, pages 34–51.

[Chomicki, 2003] Chomicki, J. (2003). Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466.

[Ciardo et al., 2001] Ciardo, G., Lüttgen, G., and Siminiceanu, R. (2001). Saturation: an efficient iteration strategy for symbolic state space generation. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 328–342. Springer-Verlag.

[Cimatti et al., 2002] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. Intl. Conf. on Computer-Aided Verification*, Copenhagen, Denmark. Springer.

[Clarke et al., 2000] Clarke, E., Grumberg, O., and Peled, D. (2000). *Model Checking*. MIT Press.

[Clarke et al., 1986] Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263.

[Claro et al.,] Claro, D. B., Albers, P., and Hao, J. K. Selecting web services for optimal composition. In *Proc. of ICWS 2005*.

[Cook and Sharygina, 2005] Cook, B. and Sharygina, N. (2005). Symbolic model checking for asynchronous boolean programs. In *SPIN*, pages 75–90.

[Cormen et al., 2001] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). *Introduction to Algorithms*. The MIT Press.

[Daskalakis et al., 2007] Daskalakis, C., Karp, R. M., Mossel, E., Riesenfeld, S., and Verbin, E. (2007). Sorting and selection in posets. *CoRR*, abs/0707.1532.

[Daskalakis et al., 2009] Daskalakis, C., Karp, R. M., Mossel, E., Riesenfeld, S., and Verbin, E. (2009). Sorting and selection in posets. In *SODA*, pages 392–401.

[desJardins and Wagstaff, 2005] desJardins, M. and Wagstaff, K. (2005). Dd-pref: A language for expressing preferences over sets. In *AAAI*, pages 620–626.

[Donsbach et al., 2009] Donsbach, J. S., Tannenbaum, S. I., Alliger, G. M., Mathieu, J. E., Salas, E., Goodwin, G. F., and Metcalf, K. A. (2009). Team composition

optimization: The team optimal profile system (tops). Technical Report ARI TR 1249, U.S. Army Research Institute for the Behavioral and Social Sciences.

[Doyle and McGeachie, 2003] Doyle, J. and McGeachie, M. (2003). Exercising qualitative control in autonomous adaptive survivable systems. In *Self-Adaptive Software: Applications*, chapter 8. Springer Berlin Heidelberg.

[Doyle and Thomason, 1999] Doyle, J. and Thomason, R. H. (1999). Background to qualitative decision theory. *AI magazine*, 20:55–68.

[Dubois et al., 2002] Dubois, D., Fargier, H., Prade, H., and Perny, P. (2002). Qualitative decision theory: from savage's axioms to nonmonotonic reasoning. *Journal of the ACM*, 49(4):455–495.

[Dustdar and Schreiner, 2005] Dustdar, S. and Schreiner, W. (2005). A survey on web services composition. *International Journal on Web and Grid Services*, 1(1):1–20.

[Fishburn, 1970] Fishburn, P. (1970). *Utility Theory for Decision Making*. John Wiley and Sons.

[Fishburn, 1985] Fishburn, P. (1985). *Interval Orders and Interval Graphs*. J. Wiley, New York.

[French, 1986a] French, S. (1986a). Decision theory: An introduction to the mathematics of rationality.

[French, 1986b] French, S. (1986b). *Decision theory: an introduction to the mathematics of rationality*. Halsted Press, New York, USA.

[Goldsmith et al., 2008] Goldsmith, J., Lang, J., Truszczynski, M., and Wilson, N. (2008). The computational complexity of dominance and consistency in cp-nets. *J. Artif. Intell. Res. (JAIR)*, 33:403–432.

[Hamadi and Benatallah, 2003] Hamadi, R. and Benatallah, B. (2003). A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference*, pages 191–200. Australian Computer Society, Inc.

[Hendler et al., 1990] Hendler, J., Tate, A., and Drummond, M. (1990). Ai planning: systems and techniques. *AI Mag.*, 11(2):61–77.

[Hristidis and Papakonstantinou, 2004] Hristidis, V. and Papakonstantinou, Y. (2004). Algorithms and applications for answering ranked queries using ranked views. *The VLDB Journal*, 13(1):49–70.

[Huhns and Singh, 2005] Huhns, M. P. and Singh, M. P. (2005). Service-oriented computing: Key concepts and principles. *Internet Computing*, 9(1):75–81.

[Jain, 2009] Jain, R. (2009). Handling worst case in skyline. Masters thesis, York University, Department of Computer Science and Engineering.

[Jung et al., 2010] Jung, H., Han, H., Yeom, H. Y., and Kang, S. (2010). A fast and progressive algorithm for skyline queries with totally- and partially-ordered domains. *Journal of Systems and Software*, 83(3):429 – 445.

[Kahlon et al., 2009] Kahlon, V., Wang, C., and Gupta, A. (2009). Monotonic partial order reduction: An optimal symbolic partial order reduction technique. In *Proc. of Intl. Conf. on Computer Aided Verification*, pages 398–413. Springer-Verlag.

[Keeney and Raiffa, 1993] Keeney, R. L. and Raiffa, H. (1993). Decisions with multiple objectives: Preferences and value trade-offs.

[Kiessling, 2002] Kiessling, W. (2002). Foundations of preferences in database systems. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 311–322. VLDB Endowment.

[Kiessling and Kostler, 2002] Kiessling, W. and Kostler, G. (2002). Preference sql: design, implementation, experiences. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 990–1001. VLDB Endowment.

[Lago et al., 2002] Lago, U. D., Pistore, M., and Traverso, P. (2002). Planning with a language for extended goals. In *Eighteenth national conference on Artificial intelligence*, pages 447–454, Menlo Park, CA, USA. American Association for Artificial Intelligence.

[Lappas et al., 2009] Lappas, T., Liu, K., and Terzi, E. (2009). Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 467–476, New York, NY, USA. ACM.

[Liu et al., 2005] Liu, F., Zhang, L., Shi, Y., Lin, L., and Shi, B. (2005). Formal analysis of compatibility of web services via ccs. In *Proceedings of the International Conference on Next Generation Web Services Practices*, page 143. IEEE Computer Society.

[Martens et al., 2006] Martens, A., Moser, S., Gerhardt, A., and Funk, K. (2006). Analyzing compatibility of bpel processes. In *International Conference on Internet and Web Applications and Services*.

[Mas-Colell et al., 1995] Mas-Colell, A., Whinston, M. D., and Green, J. R. (1995). *Microeconomic Theory*. Oxford University Press.

[Morgenstern and Von Neumann, 1944] Morgenstern, O. and Von Neumann, J. (1944). *Theory of Games and Economic Behavior*. Princeton University Press.

[Papazoglou, 2003] Papazoglou, M. (2003). Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pages 3–12. IEEE Computer Society.

[Passerone et al., 2002] Passerone, R., de Alfaro, L., Henzinger, T. A., and Sangiovanni-Vincentelli, A. L. (2002). Convertibility verification and converter synthesis: two faces of the same coin. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 132–139, New York, NY, USA. ACM.

[Pathak et al., 2007] Pathak, J., Basu, S., and Honavar, V. (2007). On context-specific substitutability of web services. In *Proceedings of the International Conference on Web Services*, pages 192–199. IEEE Computer Society.

[Pathak et al., 2008a] Pathak, J., Basu, S., and Honavar, V. (2008a). Assembling composite web services from autonomous components. In *Emerging Artificial Intelligence Applications in Computer Engineering, Maglogiannis, I., Karpouzis, K., and Soldatos, J. (ed). IOS Press. In press.*

[Pathak et al., 2008b] Pathak, J., Basu, S., and Honavar, V. (2008b). Composing web services through automatic reformulation of service specifications. In *Proceedings of the 5th IEEE International Conference on Services Computing. To appear.*

[Pathak et al., 2006a] Pathak, J., Basu, S., Lutz, R., and Honavar, V. (2006a). Moscoe a framework for modeling web service composition and execution. In *Proceedings of the 22nd International Conference on Data Engineering Workshops.*

[Pathak et al., 2006b] Pathak, J., Basu, S., Lutz, R., and Honavar, V. (2006b). Selecting and composing web services through iterative reformulation of functional specifications. In *Proceedings of the 18th Intl. Conference on Tools with Artificial Intelligence*, pages 445–454. IEEE Computer Society.

[Pistore et al., 2005a] Pistore, M., Marconi, A., Bertoli, P., and Traverso, P. (2005a). Automated composition of web services by planning at the knowledge level. In *Nineteenth International Joint Conference on Artificial Intelligence*, pages 1252–1259.

[Pistore et al., 2005b] Pistore, M., Traverso, P., and Bertoli, P. (2005b). Automated composition of web services by planning in asynchronous domains. In *Fifteenth International Conference on Automated Planning and Scheduling*, page 211.

[Preisinger, 2009] Preisinger, T. (2009). *Graph-Based Algorithms for Pareto Preference Query Evaluation*. BoD Books on Demand.

[Queille and Sifakis, 1982] Queille, J.-P. and Sifakis, J. (1982). Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming*, pages 337 – 351. Springer Verlag.

[Rausand and Høyland, 2003] Rausand, M. and Høyland, A. (2003). *System Reliability Theory: Models, Statistical Methods and Applications Second Edition*. Wiley-Interscience.

[Sacharidis et al., 2009] Sacharidis, D., Papadopoulos, S., and Papadias, D. (2009). Topologically sorted skylines for partially ordered domains. In *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 1072–1083, Washington, DC, USA. IEEE Computer Society.

[Santhanam et al., 2008] Santhanam, G. R., Basu, S., and Honavar, V. (2008). Tcp-compose⋆ - a tcp-net based algorithm for efficient composition of web services using qualitative preferences. In Bouguettaya, A., Krger, I., and Margaria, T., editors, *Procceedings of the Sixth International Conference on Service-Oriented Computing*, volume 5364 of *Lecture Notes in Computer Science*, pages 453–467.

[Santhanam et al., 2009a] Santhanam, G. R., Basu, S., and Honavar, V. (2009a). A dominance relation for unconditional multi-attribute preferences. Technical Report 09-24, Department of Computer Science, Iowa State University.

[Santhanam et al., 2009b] Santhanam, G. R., Basu, S., and Honavar, V. (2009b). Web service substitution based on preferences over non-functional attributes. In *Proceedings of International Conference on Services Computing*, pages 210–217.

[Santhanam et al., 2010a] Santhanam, G. R., Basu, S., and Honavar, V. (2010a). Dominance testing via model checking. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, pages 357–362. AAAI Press.

[Santhanam et al., 2010b] Santhanam, G. R., Basu, S., and Honavar, V. (2010b). Efficient dominance testing for unconditional preferences. In *Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, pages 590–592. AAAI Press.

[Schneider and Shanteau, 2003] Schneider, S. L. and Shanteau, J. (2003). *Emerging perspectives on judgment and decision research*, page 207. Cambridge University Press.

[Schropfer et al., 2007] Schropfer, C., Binshtok, M., Shimony, S. E., Dayan, A., Brafman, r., Offermann, P., and Holschke, O. (2007). Introducing preferences over nfps into service selection in soa. In *Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*.

[Shaparau et al., 2006] Shaparau, D., Pistore, M., and Traverso, P. (2006). Contingent planning with goal preferences. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence*. AAAI Press.

[Sirin et al., 2004] Sirin, E., Parsia, P., Wu, D., Hendler, J., and Nau, D. (2004). Htn planning for web service composition using shop2. *Journal of Web Semantics*, 1(4):377–396.

[Smythe and Mahmoud, 1995] Smythe, R. T. and Mahmoud, H. M. (1995). A survey of recursive trees. *Theor Prob Math Stat*, (51):1–27.

[Sohrabi et al., 2006] Sohrabi, S., Prokoshyna, N., and McIlraith, S. (2006). Web service composition via generic procedures and customizing user preferences. In *Fifth International Semantic Web Conference (ISWC2006)*, pages 597–611.

[Traverso and Pistore, 2004] Traverso, P. and Pistore, M. (2004). Automated composition of semantic web services into executable processes. In *Proceedings of ISWC 2004*, pages 380–394. Springer-Verlag.

[Wang, 2000] Wang, F. (2000). Efficient data structure for fully symbolic verification of real-time software systems. In *Proc. of Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS 1785*, pages 157–171. Springer-Verlag.

[Wilson, 2004a] Wilson, N. (2004a). Consistency and constrained optimisation for conditional preferences. In *ECAI*, pages 888–894.

[Wilson, 2004b] Wilson, N. (2004b). Extending cp-nets with stronger conditional preference statements. In *AAAI*, pages 735–741.

[Yu and Lin, 2005] Yu, T. and Lin, K. J. (2005). Service selection algorithms for composing complex services with multiple qos constraints. In *Service-Oriented Computing - ICSOC 2005*, pages 130–143. Springer Berlin / Heidelberg.

[Zeng et al., 2003] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q. Z. (2003). Quality driven web services composition. In *Proceedings of the 12th International Conference on World Wide Web*, pages 411–421. ACM.

[Zeng et al., 2004] Zeng, L., Benatallah, B., Ngu, A. H. H., Dumas, M., Kalagnanam, J., and Chang, H. (2004). Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327.

[Zhang and Malik, 2002] Zhang, L. and Malik, S. (2002). The quest for efficient boolean satisfiability solvers. In *CADE-18: Proceedings of the 18th International Conference on Automated Deduction*, pages 295–313, London, UK. Springer-Verlag.