Utility Functions for *Ceteris Paribus* Preferences

by

Michael McGeachie

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2002

Author .....
Department of Electrical Engineering and Computer Science
May 24, 2002

Certified by .....
Jon Doyle
SAS Institute Distinguished Professor of Computer Science,
North Carolina State University
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Utility Functions for *Ceteris Paribus* Preferences

by

Michael McGeachie

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2002, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Ceteris paribus preference statements concisely represent preferences over outcomes or goals in a way natural to human thinking. Many decision making methods require an efficient method for comparing the desirability of two arbitrary goals. We address this need by presenting an algorithm for converting a set of qualitative *ceteris paribus* preferences into a quantitative utility function. Our algorithm is complete for a finite universe of binary features. Constructing the utility function can, in the worst case, take time exponential in the number of features. Common forms of independence conditions reduce the computational burden. We present heuristics using utility independence and constraint based search to achieve efficient utility functions.

Thesis Supervisor: Jon Doyle
Title: SAS Institute Distinguished Professor of Computer Science,
North Carolina State University

Acknowledgments

I owe many thanks and expressions of gratitude to the people I have known over the last two years, who have helped me to learn something about computer science, helped me to finish this thesis, helped me to appreciate the environment presented at MIT, and discover Boston and Cambridge as an enjoyable place to spend two years. I will mention as many of these people as I can here, and though I surely omit a name or two, the contributions of those people are dearly received.

It has been a rare privilege to work under the supervision of Professor Jon Doyle. His knowledge, leadership, and advice on technical and practical matters has enabled this work and shaped it fundamentally. For this I am profoundly grateful.

I thank the Clinical Decision Making Group at the Laboratory for Computer Science for providing advice and encouragement throughout my work on this thesis. Particularly Professor Peter Szolovits, Dr. Mojdeh Mohtashemi, and Lik Mui have provided insight into the current work and a larger perspective on the corpus of computer science research and the society of professional researchers.

My family's support is both an irreplaceable and intrinsic part of anything I do. For everything, I thank my parents Robert and Catherine, and my brothers Patrick and Daniel.

I must mention some of the many people in the world whose continued wit, ingenuity, perception, and gumption enrich the lives of those around them. More than any others whom I have had the good fortune of knowing, Surj Patel, Stan Bileschi, Tracy Hammond, and Andrea Boehland have helped me enjoy Boston, Cambridge, and MIT.

And finally, without the support of a training grant from the National Library of Medicine or the support of the Pfizer Corporation, none of this work would have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Decision theory studies how to make good decisions. Choosing carefully among possible actions is most valuable in a situation where there are many possible actions, and only one or a limited number of them can be executed. Initially decision theory was used to help human beings make a difficult and important decision. This application involves a "decision expert" interviewing a decision maker, and making careful study of their desires toward outcomes and attitudes toward risk. See [KR76] for an excellent discussion of this application and process. More recent applications of decision theory involve programming computer, software, or robot agents to make rational decisions based upon their given desires, goals, intentions, beliefs, etc. (for example, see [BRST00],[BDH$^+$01].

1.1.1 Investigating Alternative Decision Theories

Decision theory analyzes actions by their outcomes. A common formulation of decision theory models these outcomes as being describable by some type of primitive properties. These properties allow us to make statements describing "possible worlds." According to this model of outcomes, the possible worlds must vary in predictable ways, along dimensions definable *a priori*, such as the amount of money a person has, the address of their home, the color of their suit, or the presence or absence of a hospitalization-requiring illness. In addition, the relationship between actions and the variables of outcomes must be known, at least probabilistically. For example, an action "put on left shoe" leads to the world with the left shoe on, but may have a small chance of altering other aspects of that world, such as causing an injury leading to hospitalization.

Such a system allows a decision maker to reason about which actions lead to which possible worlds. If we know which of the possible worlds is most desirable or advantageous, we can choose actions according to which leads to the best outcome or world. We use a preorder to rank outcomes of possible actions based on their desirability. This is formalized this as a complete preorder, a reflexive and transitive relation, over all possible outcomes. There can be many different rankings of the

desirability of the same set of outcomes. Different decision makers may well rank outcomes differently, according to their individual human tastes and preferences.

Outcomes can be assigned numeric utilities. We use "utility" to refer to a numeric measure of the desirability of a particular outcome, such that if m_1, m_2 are two outcomes, and $u(m_1)$ is the utility of the outcome, then if m_1 is preferred to m_2 then $u(m_1) > u(m_2)$. A utility for each outcome can be elicited from human decision makers via methods described in [KR76]. Then probabilities of every outcome occurring as a result of every event can be assessed, and the decision maker can then choose the action with the highest "expected utility." The expected utility of a decision is the sum of the utilities of each possible outcome times the probability of the outcome, or $\sum_{i=1}^{n} p_i u(m_i)$. Note that the terms "utility" and "expected utility" are sometimes used with different meanings in the literature. We do not consider probabilistic actions in this thesis, and therefore we do not deal with expected utility.

Such models have been successfully applied in several domains, in particular Bayes Nets [Pea88] are widely used. However, such a treatment of utility is not without its problems. Chief among these problems is that it can be difficult or impossible to ascertain all the required preferences and probabilities. Decomposing outcomes into constituent properties and expressing preferences and utility function over the properties of outcomes can reduce this burden.

Decision theory formalisms that decompose outcomes into their component qualities or features, and then reason about these features are called "Qualitative Decision Theories" [DT99]. While there are several different languages and logics of qualitative preference rankings [DSW91][MS99][BG96] (discussed in more detail later), they all simply try to state which possible outcomes are preferred to which other outcomes (in reference to the total preorder over the outcomes). For many domains, qualitative rankings are more natural and appropriate than quantitative rankings [WD91]. For example, when outcomes differ by qualitative attributes, when chocolate is preferred to vanilla; or when quantitative scales are unimportant, when fast is better than slow, but exact speeds are not important or entirely controllable. Note that the tradeoffs between qualitative attributes can be quantitative: chocolate could be five times as desirable as vanilla. Conversely, quantitative comparisons are useful when outcomes differ in respect to some easily measurable quantity, such as the utility of money being proportional to the log of the amount of money. Of course, quantitative comparisons also naturally lend themselves to computation. Qualitative representations lend themselves to logical reasoning, graph algorithms, and sometimes no algorithm of less than exponential complexity.

1.1.2 *Ceteris Paribus* Preferences

Doyle and Wellman [WD91] have observed that qualitative representations of preferences are a succinct and reasonable approximation of at least one type of common human preferences. Doyle, Shoham, and Wellman [DSW91] present a theoretical formulation of human preferences of generalization in terms of *ceteris paribus* preferences, *i.e.,* all-else-equal preferences. *Ceteris paribus* relations express a preference over sets of possible worlds. We consider all possible worlds (or outcomes) to be

Feature \ **Tutor**	p	q	r
Graduated	false	false	true
A in Software Engineering	true	false	false
A in Computer Systems	true	true	false
Cambridge resident	true	true	true
Will work Tuesdays	false	false	true
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 1.1: Properties of possible computer science tutors

describable by some (large) set of binary features F. Then each *ceteris paribus* rule specifies some features of outcomes, and a preference over them, while ignoring the remaining features. The specified features are instantiated to either true or false, while the ignored features are "fixed," or held constant. A *ceteris paribus* rule might be "we prefer programming tutors receiving an A in Software Engineering to tutors not receiving an A, other things being equal." In this example, we can imagine a universe of computer science tutors, each describable by some set of binary features F. Perhaps $F = \{$*Graduated, SoftwareEngineering_A, ComputerSystems_A, Cambridge_resident, Willing_to_work_on_Tuesdays,* $\ldots\}$. The preferences expressed above state that, for a particular computer science tutor, they are more desirable if they received an A in the Software Engineering course, all other features being equal. Specifically, this makes the statement that a tutor p, of the form shown in table 1.1, is preferred to another tutor q, also in table 1.1, assuming the elided features are identical. The *ceteris paribus* preference makes no statement about the relationship between tutor p and tutor r because they differ with regard to other features. With persons p and q, they differ only on the feature we have expresses a preference over (grade in Software Engineering), and all other features are equal. Concerning tutors p and r however, the *ceteris paribus* statement asserts no preference, because all else is not equal (they differ on Graduation Status, Computer Systems grade, and willingness to work on Tuesdays).

1.1.3 Thesis Goal

Other researchers have developed logics of preference representation [DW94, MS99, BG96]. Each of these representations has its strengths and weaknesses. An essential dimension of analysis of these representations is the reasoning methods or computational methods the representation facilitates. Some of the representations do not yet have explicit reasoning or computational methods, the work in this thesis will provide an algorithm for reasoning in the preference representation of Doyle, Shoham, and Wellman [DSW91, DW94].

Once we have any representation of specific preferences over possible outcomes, this restricts the number of possible preorders consistent with the stated preferences.

In decision theory, we must decide which outcome is most desirable, or which of some set of easily achievable outcomes is most desirable. Determining which of two outcomes is more preferred is generally termed a "preference query." A query is a search for a justification among the given preferences over outcomes, and the characteristics of this search are dependent on the characteristics of the computational structures defined. Numerical utility functions offer the possibility of doing comparisons with numerical calculations, rather than reasoning directly with the preferences. For example, to decide a preference query, we can just compute the utility of two outcomes and compare them to decide which outcome is most preferred. With qualitative representations of utility, we must first perform substantial translations to be able to answer queries.

We now give a formal account of the *ceteris paribus* formulation we will use, then a statement of the thesis task using that formalism.

1.2 A Formal "All Else Equal" Logic

We employ a restricted logical language $\mathcal{L}$, patterned after [DSW91] but using only the standard logical operators $\neg$ (negation) and $\wedge$ (conjunction) to construct finite sentences over a set of atoms $\mathcal{A}$.[1] Each atom $a \in \mathcal{A}$ corresponds to a feature $f \in F$, a space of binary features describing possible worlds. We write $f(a)$ for the feature corresponding to atom a. By $literals(\mathcal{A})$ we denote the atoms of $\mathcal{A}$ and their negations; $literals(\mathcal{A}) = \mathcal{A} \cup \{\neg a \mid a \in \mathcal{A}\}$. A complete consistent set of literals m is a *model*. That is, m is a model iff exactly one of a and $\neg a$ are in m, for all $a \in \mathcal{A}$. We use $\mathcal{M}$ for the set of all models of $\mathcal{L}$.

A model of $\mathcal{L}$ assigns truth values to all atoms of $\mathcal{L}$, and therefore to all formula in $\mathcal{L}$ and all features in F. We write $f_i(m)$ for the truth value assigned to feature f_i by model m. A model *satisfies* a sentence p of $\mathcal{L}$ if the truth values m assigns to the atoms of p make p true. We write $m \models p$ when m satisfies p. We define a *proposition* expressed by a sentence p, by $[p] = \{m \in \mathcal{M} \mid m \models p\}$.

A *preference order* is a complete preorder (reflexive and transitive relation) $\succsim$ over $\mathcal{M}$. When $m \succsim m'$, we say that m is *weakly preferred* to m'. If $m \succsim m'$ and $m' \not\succsim m$, we write $m \succ m'$ and say that m is *strictly preferred* to m'. If $m \succsim m'$ and $m' \succsim m$, then we say m is *indifferent* to m', written $m \sim m'$.

The *support* of a sentence p is the minimal set of atoms determining the truth of p, denoted $s(p)$. The support of p is the same as the set of atoms appearing in an irredundant sum-of-products sentence logically equivalent to p. Two models m and m' are *equivalent modulo* p if they are the same outside the support of p. Formally, $m \equiv m' \bmod p$ iff

$$m \backslash (literals(s(p))) = m' \backslash (literals(s(p)))$$

Model modification is defined as follows. A set of *model modifications of* m *making* p *true*, written $m[p]$, are those models satisfying p which assign the same truth values

[1] We disallow the operators $\vee, \rightarrow, \leftrightarrow$ in $\mathcal{L}$. Logical sentences using disjunction, implication, and equivalence can be translated into (possibly larger) equivalent logical sentences in $\mathcal{L}$.

to atoms outside the support of p as m does. That is,

$$m[p] = \{m' \in [p] \mid m \equiv m' \bmod p\}.$$

A statement of desire is an expression of *ceteris paribus* preferences. Desires are defined in terms of model modification. We write $p \trianglerighteq q$ when p is desired at least as much as q. Formally, we interpret this as $p \trianglerighteq q$ if and only if for all m in $\mathcal{M}$, $m' \in m[p \wedge \neg q]$ and $m'' \in m[\neg p \wedge q]$, we have $m' \succsim m''$. This is just a statement that p is desired over q exactly when any model making p true and q false is weakly preferred to any model making p false and q true, whenever the two models assign the same truth values to all atoms not in the support of p or of q. If p is weakly preferred to q and there is some pair of models m, m', where m makes p true and q false, m' makes p false and q true, m, m' assign the same truth values to atoms outside the support of p and q, and m is strictly preferred to m', we instead have a strict preference for p over q, written $p \triangleright q$. That is p is strictly preferred *ceteris paribus* over q holds if,

$$\begin{array}{l} \forall m \in \mathcal{M}, \\ \forall (m', m''), m' \in m[p \wedge \neg q] \wedge m'' \in m[\neg p \wedge q] \rightarrow m' \succsim m'', \\ \exists (m', m''), m' \in m[p \wedge \neg q] \wedge m'' \in m[\neg p \wedge q] \rightarrow m' \succ m''. \end{array} \tag{1.1}$$

1.2.1 Preference Specification

Ceteris paribus preferences are specified by a set of *preference rules* using the language $\mathcal{L}$. A preference *rule* is any statement of the form $p \trianglerighteq q$ or $p \triangleright q$, where p and q are statements in $\mathcal{L}$.

If a preference rule c implies that $m' \succ m''$, for $m', m'' \in \mathcal{M}$, then we write $m' \succ_c m''$. A set of preferences rules C is said to be *consistent* just in case for all $m', m'' \in \mathcal{M}$, it is not the case that m' is strictly preferred to m'' and m'' is strictly preferred to m'.

Given a set C of *ceteris paribus* preference rules, let $[C]$ be the set of all weak preference orders over models such that the weak preference order satisfies the preference specification C.

Conditional *Ceteris Paribus* Preferences

Conditional *ceteris paribus* preferences can be represented as well. A conditional *ceteris paribus* preference is a preference of the form: if r then $p \trianglerighteq q$, where p, q, and r are a statements in $\mathcal{L}$. This is taken to mean that $p \triangleright q$, a *ceteris paribus* preference, holds just in case r holds. Such a preference is equivalent to the unconditional *ceteris paribus* preference $r \wedge p \trianglerighteq r \wedge q$ whenever the support of r is disjoint from that of p and q. Further, we suggest that when r has overlapping support with p or q, the conditional preference if r then $p \trianglerighteq q$ is best expressed as:

$$\begin{array}{l} \forall (m', m''), \forall m \\ m' \in m[p \wedge \neg q] \wedge m'' \in m[\neg p \wedge q], \\ \forall (\mu', \mu'') \\ \mu' \in m'[r], \mu'' \in m''[r], \\ \mu' \succsim \mu''. \end{array}$$

This representation has the intuitive meaning that r must be true for the preference to have meaning, and that the preference regarding p and q must be satisfied as much as possible, while allowing r to be true. Note that this definition of conditional preference with overlapping support reduces to the definition of conditional preference with disjoint support, in the case that support of r is actually disjoint from support of p and of q. That is, if r, p, q have disjoint support, then

$$\mu' \in m'[r], m' \in m[p \wedge \neg q] \Rightarrow \mu' \in m[r \wedge p \wedge \neg q]$$

and similarly,

$$\mu'' \in m''[r], m'' \in m[\neg p \wedge q] \Rightarrow \mu'' \in m[r \wedge \neg p \wedge q].$$

Other researchers, for example [BBHP99], consider conditional *ceteris paribus* preferences an important extension of unconditional *ceteris paribus* preferences. Boutilier *et. al.* do not, however, provide semantics for conditional preferences where the condition and the preference are not logically independent.

Since there is this convenient correspondence between conditional *ceteris paribus* preferences and our simplified *ceteris paribus* in $\mathcal{L}$, the results established in the remainder of the thesis apply to both types of *ceteris paribus* preferences.

1.2.2 Utility Functions

We have described the language $\mathcal{L}$ in the preceding section. A *utility function* $u : \mathcal{M} \rightarrow \mathbb{R}$, maps each model in $\mathcal{M}$ to a real number. Each utility function implies a particular preorder over the models $\mathcal{M}$. We denote the preorder implied by u with $p(u)$. Given a finite set C of *ceteris paribus* preferences in a language $\mathcal{L}$ over a set of atoms $\mathcal{A}$ representing features in F, we define a utility function as follows.

A *utility function* is a function u that maps each model to a real number such that u represents the preference set C.

We are now in a position to formally state our task.

Given a set of ceteris paribus *preference statements C, find a utility function u such that $p(u) \in [C]$.*

We will carry out this task by demonstrating several methods for constructing ordinal utility functions, u; this involves demonstrating two things. First, we show what the function u computes: how it takes a model and produces a number. Second, we show and how to compute u given a finite set of *ceteris paribus* preferences C. These methods are sound and complete. We then demonstrate heuristic means of making the computation of u more efficient.

Before we discuss the function u, we define another representation of *ceteris paribus* preferences, which we refer to as *feature vector representation*. We discuss this in the following chapter.

Chapter 2

Feature vector representation

We now define a new representation of *ceteris paribus* preferences. This representation makes the preference statements more explicit; thus it will be simpler to conceive some types of algorithms for computation using the feature vector representation.

Let C be a finite set of *ceteris paribus* preference statement in $\mathcal{L}$. With F our space of binary features, we note that F may be infinite. We define $F(C) \subseteq F$ to be the set of features corresponding to the atoms in the union of the support of each rule in C. That is:

$$F(C) = \{f(a) : a \vee \neg a \in \bigcup_{c \in C} s(c)\}$$

Those features not specified in any rule in C are not relevant to compute the utility of a model, since there is no preference information about them in the set C. Because we require C finite, $F(C)$ is also finite, we write $|F(C)| = N$, and $F(C) = \{f_1, ..., f_N\}$.

2.1 Definition

The "feature vector representation" is several related languages and definitions, presented in this section.

We define a family of languages $\mathcal{L}^*$. Each $\mathcal{V}$, we call a *feature vector*, enumerates the domain of a particular language $\mathcal{L}^*(\mathcal{V})$. The feature vector is an ordered list of features taken from the set $F(C)$. We write $\mathcal{V} = \langle f_1, f_2, ..., f_N \rangle$, where $f_i \in F(C)$. Thus, $\mathcal{L}^*(\cdot)$ is a function that takes feature vectors and produces a language. We write $v(\mathcal{L}^*(\mathcal{V}))$ for the feature vector of $\mathcal{L}^*(\mathcal{V})$, this defines $v(\mathcal{L}^*(\mathcal{V})) = \mathcal{V}$. We will usually assume the feature vector of $\mathcal{L}^*(\mathcal{V})$ is the set of relevant features $F(C)$ of the set C of *ceteris paribus* rules. However, we will at times discuss different or even overlapping languages $\mathcal{L}^*(\mathcal{V})$, such as $\mathcal{L}^*(\langle f_2, f_3, f_5 \rangle)$ and $\mathcal{L}^*(\langle f_1, f_3, f_4 \rangle)$. When the feature vector is clear from the context, we will suppress the vector and just write $\mathcal{L}^*$.

A statement in $\mathcal{L}^*(\langle f_1, f_2, ..., f_N \rangle)$ is an ordered vector of N elements drawn from the alphabet $\Gamma = \{0, 1, *\}$. Thus, $\langle *, 1, * \rangle \in \mathcal{L}^*(\langle f_1, f_2, f_3 \rangle)$, and $\langle 0, 0, 0, 1, * \rangle \in \mathcal{L}^*(\langle f_1, f_2, f_3, f_4, f_5 \rangle)$. When discussing a particular statement p of $\mathcal{L}^*(\mathcal{V})$ we drop the vector notation and concatenate the elements of p into one string, so that, *e.g.*,

$\langle 1, *, 0\rangle$ becomes $1*0$. We write $f_i(p)$ for the value in Γ assigned to $f_i \in v(\mathcal{L}^*(\mathcal{V}))$ for a sentence p in $\mathcal{L}^*(\mathcal{V})$.

A model of $\mathcal{L}^*(\langle f_{i_1}, f_{i_2}, ..., f_{i_M}\rangle)$ is a concatenation of truth values, $v_1 v_2 \dots v_M$ where $v_i \in \Gamma' = \{0, 1\}$, for each of the M features, $f_{i_1}, f_{i_2}, ..., f_{i_M}$. When a model uses the alphabet Γ' to represent *true* and *false*, 0 indicates *false* and 1 indicates *true*. The intuition is that each model of $\mathcal{L}^*(\mathcal{V})$ assigns truth values to the features $\mathcal{V}$. If m is a model of $\mathcal{L}^*(V)$, we write $f_i(m)$ to denote the value in Γ' assigned to f_i by m, for any feature $f_i \in F$. For a given feature vector $\mathcal{V}$, we denote the set of all models of $\mathcal{L}^*(\mathcal{V})$ by $\mathcal{M}^*(\mathcal{V})$. As with the notation $\mathcal{L}^*(\mathcal{V})$, we will sometimes suppress the $\mathcal{V}$ in $\mathcal{M}^*(\mathcal{V})$ when it is clear from context. We also use $v(\mathcal{M}^*(\mathcal{V})) = \mathcal{V}$.

There exists a correspondence between models in $\mathcal{M}$ and models in $\mathcal{M}^*(\mathcal{V})$. We define the following translation on models:

We project models in $\mathcal{M}$ to models in $\mathcal{M}(\mathcal{V})$ by a mapping α :

Definition 2.1 (Model Projection) *The translation $\alpha : \mathcal{M} \to \mathcal{M}^*$ is defined for each $m \in \mathcal{M}$ and $f \in F(C)$ by $\alpha(m) = m'$, $m \in \mathcal{M}^*$. For all $f_i \in v(\mathcal{M}^*)$,*

- $f(\alpha(m)) = 1$ *if* $f \in m$
- $f(\alpha(m)) = 0$ *if* $\neg f \in m$

This projection induces an equivalence relation on $\mathcal{M}$, and we write $[m]$ to mean the set of models in $\mathcal{M}$ mapped to the same model in $\mathcal{M}(\mathcal{V})$ as m:

$$[m] = \{m' \in \mathcal{M} \mid \alpha(m') = \alpha(m)\} \tag{2.1}$$

The translation α specifies that m and m' must assign the same truth values to features that appear in $(\mathcal{L}^*)$, but that on features not appearing therein, there is no restriction. When the feature vector of $\mathcal{L}^*$ is the set of features F, there is a one-to-one correspondence of models in $\mathcal{L}^*$ and $\mathcal{L}$.

We introduce the concept of a model *satisfying* a statement s in $\mathcal{L}^*$. A model m *satisfies* s, written $m \models s$, if $\forall f_i \in s$ with $f_i \neq$ '$*$', m assigns the same truth value to f_i as s does. For example, $m = 0011$ satisfies $*0*1$; $m = 0011$ also satisfies $00**$. Note the similarity to the concept of satisfying defined over *ceteris paribus* formula in section 1.2.

Since we may talk about models of languages parameterized by different vectors, we need a way to discuss models mapping between languages. We define model *restriction.* If m is a model of $s \subseteq F(C)$, and $s' \subseteq s$, a *restriction* of m to the features s', denoted $m \restriction s'$, is the set of values m assigns to features in s'. We say that one model satisfies another model, written $m \models m'$, when m' is a model of s', m is a model of s, $s' \subseteq s$, and $m' = m \restriction s'$.

A language for rules in the feature vector representation is required. We denote this language with $\mathcal{L}^*_{\mathcal{R}}(\mathcal{V})$. A statement in this language is composed of a pair of statements (p, q) in $\mathcal{L}^*(\mathcal{V})$, separated by "$\succ$" or "$\succsim$," and subject to the following. All statements are of the following two forms:

$$v'_1 v'_2 \dots v'_N \succ v''_1 v''_2 \dots v''_N$$
$$v'_1 v'_2 \dots v'_N \succsim v''_1 v''_2 \dots v''_N$$

where each of $v'_1, v'_2, \ldots, v'_N, v''_1, v''_2, \ldots, v''_N \in \Gamma$. If $v'_i = *$, then we require $v''_i = *$, and vice versa. We generally write rules r like "$p \succ q$" and "$p \succsim q$" where $p, q \in \mathcal{L}^*(\mathcal{V})$. We say that a statement in $\mathcal{L}^*_{\mathcal{R}}(\mathcal{V})$ is also a *rule* in $\mathcal{L}^*_{\mathcal{R}}(\mathcal{V})$. Note that rules are of the form $p \succ q$ rather than $p \prec q$. We refer to the left-hand side of the rule as the statement in $\mathcal{L}^*$ left of the "$\succ$" or "$\succsim$" operator, and we write $LHS(r)$. We define right-hand side and $RHS(r)$ analogously. Thus, if $r = p \succ q$, $LHS(r) = p$, $RHS(r) = q$. An example of a rule in $\mathcal{L}^*_{\mathcal{R}}(\langle f_1, f_2, f_3, f_4\rangle)$ is: $**01 \succ **10$.

Let p be a statement in $\mathcal{L}^*(F(C))$. We need a concept of a statement p's *support features*, similar to the support of a statement in $\mathcal{L}$. The support features of a statement p is the minimal set of features needed to determine if a model of $\mathcal{L}^*(F(C))$ satisfies p. These are exactly those features in p that are assigned value either 0 or 1. This only excludes those features assigned value '*'. Given a statement p, we write $s(p)$ to denote the set of support features of p. Similarly, when r is a rule in $\mathcal{L}^*_{\mathcal{R}}$, r's *support features*, $s(r)$, are exactly those features in $s(LHS(r))$. For example, suppose $p \in \mathcal{L}^*(\langle f_1, f_2, f_3, f_4, f_5\rangle)$, and $p = *10*0$, then $s(p) = \{f_2, f_3, f_5\}$. Note that $s(LHS(r)) = s(RHS(r))$, as a consequence of the definition of $\mathcal{L}^*_{\mathcal{R}}$.

We say that a pair of models (m_1, m_2) of $\mathcal{L}^*$ *satisfies* a rule r if m_1 satisfies $LHS(r)$, m_2 satisfies $RHS(r)$, and m_1, m_2 have the same value for those features represented by '*' in r. Formally, with $V = v(\mathcal{L}^*)$, we have $\forall f_i \in V \backslash s(r)$, m_1 assigns the same truth value to f_i as m_2. We write $(m_1, m_2) \models r$ when (m_1, m_2) satisfies r. For example, $(100, 010) \models 10* \succ 01*$, but $(101, 010) \not\models 10* \succ 01*$.

The meaning of a rule r in $\mathcal{L}^*_{\mathcal{R}}$ is a preference order over $\mathcal{M}$. For a strict rule r using "$\succ$", we have $\forall (m_1, m_2) \models r$, $[m_1] \succ_r [m_2]$. For a weak rule r using "$\succsim$", we have $\forall (m_1, m_2) \models r$, $[m_1] \succsim_r [m_2]$. Thus the rule $**01 \succ **10$ in $\mathcal{L}^*_{\mathcal{R}}(\langle f_1, f_2, f_3, f_4\rangle)$ represents four specific preferences over models of $\mathcal{L}^*(\langle f_1, f_2, f_3, f_4\rangle)$. These are:

$$0001 \succ 0010$$
$$0101 \succ 0110$$
$$1001 \succ 1010$$
$$1101 \succ 1110$$

Note that this says nothing at all about the preference relationship between, *e.g.*, 0101 and 1010.

2.2 $\mathcal{L}^*$ compared to $\mathcal{L}$

These two languages of *ceteris paribus* preference have similar expressive power. Many of the terms used in the definition of the feature vector representation have direct analogs in the *ceteris paribus* representation of Doyle and Wellman. We now show a translation from a *ceteris paribus* rule to an equivalent set of possibly many feature vector representation rules. We then give the opposite translation.

In the following, we assume the set of relevant features $F(C)$ is known. This can be computed from a set of *ceteris paribus* rules (as discussed in Chapter 4), or it can be given *a priori*.

2.2.1 Forward Translation

A set of rules R of feature vector representation is compatible with a rule $c = p_c \rhd q_c$ in the *ceteris paribus* representation just in case

$$m_1 \succ_c m_2 \Rightarrow \exists r \in R \mid (m_1', m_2') \models r \tag{2.2}$$

where m_1, m_2 model $\mathcal{L}$, m_1', m_2' model $\mathcal{L}^*$, such that $m_1 \in [m_1']$ and $m_2 \in [m_2']$. We will show that such an r can always be constructed. Similarly, a set R is compatible with a rule $c = p_c \trianglerighteq q_c$ if

$$m_1 \succsim_c m_2 \Rightarrow \exists r \in R \mid (m_1', m_2') \models r. \tag{2.3}$$

That is, we define equivalence in the intuitive fashion, that each set of weak or strict rules implies the same set of preferences over models as the *ceteris paribus* rules. Before delving into the details of our construction, we define two important translation functions.

Definition 2.2 (Characteristic Statement σ) *Let $\mathcal{M}'$ be the set of models of a set of literals, literals($\mathcal{A}'$), where $\mathcal{A}' \subseteq \mathcal{A}$. σ is a function: $\sigma : \mathcal{M}' \rightarrow \mathcal{L}^*(F(C))$. Let m be a model of literals($\mathcal{A}'$). We set $f_i(\sigma(m))$ as follows:*

- $f_i(\sigma(m)) = 1$ *iff* $f_i(m) = true$
- $f_i(\sigma(m)) = 0$ *iff* $f_i(m) = false$
- $f_i(\sigma(m)) = *$ *iff* $f_i \notin \{f(a) \mid a \in \mathcal{A}'\}$

An important property of σ is as follows. If a is a model of $literals(\mathcal{A}')$, m a model in $\mathcal{M}^*(F(C))$, then m satisfies a implies that m satisfies $\sigma(a)$. More formally,

$$m \models a \rightarrow m \models \sigma(a). \tag{2.4}$$

This can be seen by considering each of the N elements of $F(C)$ one at a time. Let $F' = \{f(a) \mid a \in \mathcal{A}'\}$. If $f_i \in F'$, then a assigns a truth value to f_i, which is consistent with $f_i(m)$ and $f_i(\sigma(a))$. If $f_i \notin F'$, then a does not assign f_i a truth value, nor does $\sigma(a)$ require a particular truth value for $f_i(m)$ for a model m to satisfy $\sigma(a)$. Note that if $f_i \in (F' \backslash F(C))$, it is (again) not required to be of a particular truth value for a model to satisfy $\sigma(a)$.

Definition 2.3 (Characteristic Model μ) *Let p be a statement in $\mathcal{L}^*(\mathcal{V})$, and let $\mathcal{M}'$ be the set of models of $s(p)$ in $\mathcal{L}$. Then μ is a function: $\mu(p, \mathcal{L}^*(\mathcal{V})) : \mathcal{L}^*(\mathcal{V}) \rightarrow \mathcal{M}'$. To construct $m' = \mu(p, \mathcal{L}^*(\mathcal{V}))$, we set $m' = \{\}$, the empty set of literals, then include literals (from the set literals($\mathcal{A}$)) in m' for each feature $f_i \in \mathcal{V}$ as follows:*

- $a_i \in m'$ *iff* $f_i(p) = 1$
- $\neg a_i \in m'$ *iff* $f_i(p) = 0$

- $a_i \notin m' \wedge \neg a_i \notin m'$ *iff* $f_i(p) = *$

An important property of μ is as follows. Let $\mathcal{A}'$ be a set of atoms such that $\mathcal{A}' \subseteq \mathcal{A}$, let a be a model of $literals(\mathcal{A}')$, and m a model in $\mathcal{M}^*(F(C))$. Then m satisfies a implies that m satisfies $\mu(a, \mathcal{L}^*(F(C)))$. More formally,

$$m \models a \rightarrow m \models \mu(a, \mathcal{L}^*(F(C))). \tag{2.5}$$

This can be seen by considering each $f_i \in F(C)$ one at a time. Let $F' = \{f(a) \mid a \in \mathcal{A}'\}$. If $f_i \in F'$, then a assigns a truth value to f_i, which is consistent with $f_i(m)$ and $f_i(\mu(a, \mathcal{L}^*(F(C))))$. If $f_i \notin F'$, then a does not assign f_i a truth value, nor does $\mu(a, \mathcal{L}^*(F(C)))$ require a particular truth value for $f_i(m)$ for a model m to satisfy $\mu(a, \mathcal{L}^*(F(C)))$. Note that if $f_i \in (F' \backslash F(C))$, it is (again) not required to be of a particular truth value for a model to satisfy $\mu(a, \mathcal{L}^*(F(C)))$.

We translate a single *ceteris paribus* rule c into a set of feature vector representation rules R by the support of c. If c is of the form $p_c \rhd q_c$, where p_c, q_c are sentences in $\mathcal{L}$, then it is models that satisfy $p_c \wedge \neg q_c$ which are preferred to models that satisfy $\neg p_c \wedge q_c$, other things being equal. For brevity, let $s_c = s(p_c \wedge \neg q_c) \cup s(\neg p_c \wedge q_c)$. We let w be a set of truth values for the features in s_c. Note that

$$\mu(w, \mathcal{L}^*(F(C))) = m \mid m \models w$$

for m a model of $\mathcal{L}^*(F(C))$. Then let W_l be the set of all w as follows, where w is a model of s_c:

$$W_l = w \mid \mu(w, \mathcal{L}^*(F(C))) \models p_c \wedge \neg q_c.$$

Thus, we have each $w_i \in W_l$ is a different set of truth values for s_c making $p_c \wedge \neg q_c$ true. Similarly, we define W_r as the set of all w with

$$W_r = w \mid \mu(w, \mathcal{L}^*(F(C))) \models \neg p_c \wedge q_c.$$

Note that the set of features s_c is the set of support features for each rule $r \in R$. We construct a new set W_l' from W_l by augmenting each member. Let W_l' be a set of statements in $\mathcal{L}^*(F(C))$.

We define a set W_l' by applying σ to W_l. That is,

$$W_l' = \sigma(w_l) \mid w_l \in W_l.$$

Similarly, we define W_r' by

$$W_r' = \sigma(w_r) \mid w_r \in W_r.$$

Note that the members of W_l' and W_r' are of length $|F(C)|$, while those of W_l and W_r are of size $|s_c|$.

We now construct a set of rules R composed of rules in $\mathcal{L}^*_{\mathcal{R}}(F(C))$. We define one $r \in R$ for all pairs $(w_l', w_r'), w_l' \in W_l', w_r' \in W_r'$ as follows: $r = w_l' \succ w_r'$ (or $r = w_l' \succsim w_r'$ if c was a weak preference). This completes the translation.

Consider a simple example. In the following, we assume $F(C) = \langle f_1, f_2, f_3, f_4 \rangle$. A *ceteris paribus* rule might be of the form $a_2 \wedge \neg a_4 \rhd a_3$, with each a_i in the set $literals(\mathcal{A})$. This expresses the preference for models m:

$$m \models (a_2 \wedge \neg a_4) \wedge \neg a_3$$

over models m':

$$m' \models \neg(a_2 \wedge \neg a_4) \wedge a_3$$

other things being equal (see equation 1.1). Note $s_c = \{a_2, a_3, a_4\}$. We note $W_l = \{\{a_2, \neg a_3, \neg a_4\}\}$, and $W_r = \{\{a_2, a_3, a_4\}, \{\neg a_2, a_3, a_4\}, \{\neg a_2, a_3, \neg a_4\}\}$. We then translate this into (three in this case) feature vector representation rules using σ:

$$*100 \succ *111$$
$$*100 \succ *011$$
$$*100 \succ *010$$

The above exposition and construction gives us the necessary tools to state and prove the following lemma.

Lemma 2.2.1 (Feature vector representation of *ceteris paribus* rules) *For all* ceteris paribus *rules c over the language $\mathcal{L}$ and the features $F(C)$, there exists a set of rules R in $\mathcal{L}^*_{\mathcal{R}}(F(C))$ such that $\exists r \in R$ whenever $m_1 \succ_c m_2$ with $(m'_1, m'_2) \models r$ where m_1, m_2 are models of $\mathcal{L}$, m'_1, m'_2 are models of $\mathcal{L}^*$, such that $m_1 \in [m'_1]$ and $m_2 \in [m'_2]$.*

Proof. Without loss of generality, we take $c = p_c \succ q_c$. The case of $c = p_c \succsim q_c$ is the same. Using the definition of a *ceteris paribus* rule, we note that lemma 2.2.1 is satisfied when

$$\begin{array}{l} \forall m \in \mathcal{M}, m' \in m[p_c \wedge \neg q_c], m'' \in m[\neg p_c \wedge q_c] \rightarrow \\ \exists r \in R, m' \models LHS(r), m'' \models RHS(r). \end{array} \tag{2.6}$$

We show that if m, m', m'' are such that $m' \in m[p_c \wedge \neg q_c], m'' \in m[\neg p_c \wedge q_c]$, then our translation above constructs an r such that $m' \models LHS(r), m'' \models RHS(r)$. Given such m, m', m'', from the definition of $m' \in m[p_c \wedge \neg q_c]$, we have $m' \models p_c \wedge \neg q_c$. Let w_a be a member of W_l such that $w_a = m' \restriction s_c$ and w_b be a member of W_r such that $w_b = m'' \restriction s_c$. By definition of restriction, $m' \models w_a$, and $m'' \models w_b$.

Let w'_a be in W'_l such that $w'_a = \sigma(w_a)$. Since $m' \models w_a$ then by equation 2.4 $m' \models w'_l$. We also have $m' \models w'_a$. Since $w'_a = LHS(r)$ for some r in R according to the construction, we have $m' \models LHS(r)$ for some r . A similar argument shows that $m'' \models RHS(r)$ for some (possibly different) r. Since $\forall a \in W'_l, \forall b \in W'_r, \exists r : a = LHS(r), b = RHS(r)$, we choose r such that $m' \models LHS(r) = w'_a$ and $m'' \models RHS(r) = w'_b$. This completes the proof of Lemma 2.2.1. □

Thus, we have shown, for a given rule $c = p_c \rhd q_c$ in the *ceteris paribus* representation, there exists a set of rules R in the feature vector representation such that $(m_1, m_2) \models r \in R$ iff there exists m such that $m_1 \in m[p_c \wedge \neg q_c], m_2 \in m[\neg p_c \wedge q_c]$. Thus, the construction preserves the meaning of a rule in the *ceteris paribus* representation.

2.2.2 Reverse Translation

We give a translation from one rule in $\mathcal{L}^*_{\mathcal{R}}(F(C))$ to one rule in $\mathcal{L}$. This is a construction for a rule r in $\mathcal{L}^*_{\mathcal{R}}$ into a *ceteris paribus* rule c in $\mathcal{L}$, such that if $(m_1, m_2) \models r$, there exists $c = p_c \succ q_c$ and m, such that $m_1 \in m[p_c \wedge \neg q_c]$ and $m_2 \in m[\neg p_c \wedge q_c]$; and similarly for rules using "$\succsim$". Suppose we have a general feature vector representation rule $r = LHS(r) \succ RHS(r)$. This means that models satisfying $LHS(r)$ are preferred to those satisfying $RHS(r)$, all else equal. A *ceteris paribus* rule is a comparison of formulae $a \rhd b$ where a, b are formulae in the language $\mathcal{L}$. Thus, we must look for some formulae a, b such that

$$a \wedge \neg b \equiv \mu(LHS(r), \mathcal{L}^*(F(C))) \tag{2.7}$$

and

$$\neg a \wedge b \equiv \mu(RHS(r), \mathcal{L}^*(F(C))).$$

In the following, m denotes a model in $\mathcal{M}^*(F(C))$. Consider the sets of models $[LHS(r)] = \{m' \mid \alpha(m') \models LHS(r)\}$, and $[RHS(r)] = \{m' \mid \alpha(m') \models RHS(r)\}$. Note that $[LHS(r)]$ and $[RHS(r)]$ are disjoint. Disjointness follows from the support features of $LHS(r)$ equal to the support features of $RHS(r)$, and that $LHS(r) \neq RHS(r)$. $[LHS(r)] \subseteq \overline{[RHS(r)]}$ follows from disjointness, where $\overline{[RHS(r)]} = \{m' \mid \alpha(m') \not\models RHS(r)\}$ is the complement of $[RHS(r)]$. Thus,

$$[LHS(r)] \cap \overline{[RHS(r)]} = [LHS(r)]$$

and

$$[RHS(r)] \cap \overline{[LHS(r)]} = [RHS(r)].$$

This suggests a solution to equation (2.7). We let $a = \mu(LHS(r), \mathcal{L}^*(F(C)))$ so we have $[a] = [LHS(r)]$, and similarly, $b = \mu(RHS(r), \mathcal{L}^*(F(C)))$, and $[b] = [RHS(r)]$. Then we have

$$[a \wedge \neg b] = [a] \cap \overline{[b]} = [a]$$

and

$$[\neg a \wedge b] = \overline{[a]} \cap [b] = [b],$$

as required. Thus, our *ceteris paribus* rule c is $a \rhd b$.

Lemma 2.2.2 (*ceteris paribus* rule expressed by $\mathcal{L}^*_{\mathcal{R}}$) *Given a rule r in $\mathcal{L}^*_{\mathcal{R}}(f)$, for each pair of models $(m_1, m_2) \models r$, there exists a* ceteris paribus *rule $c = p_c \rhd q_c$, or $c = p_c \trianglerighteq q_c$, with p_c, q_c in $\mathcal{L}$, such that there exists an m such that $m_1 \in m[p_c \wedge \neg q_c], m_2 \in m[\neg p_c \wedge q_c]$.*

Proof of Lemma 2.2.2. The proof follows from the construction. We must show that for each pair $(m_1, m_2) \models r$ in $\mathcal{L}^*(F(C))$, our construction creates a rule $c = p_c \rhd q_c$, with p_c, q_c in $\mathcal{L}$, such that there exists m such that $m_1 \in m[p_c \wedge \neg q_c]$, $m_2 \in m[\neg p_c \wedge q_c]$. The argument for $c = p_c \trianglerighteq q_c$ is similar.

Let m'_1, m'_2 model $\mathcal{L}$, and be such that $\alpha(m'_1) = m_1$, and $\alpha(m'_2) = m_2$. Then $(\alpha(m'_1), \alpha(m'_2)) \models r$. We know that $\alpha(m'_1) \models LHS(r)$ and $\alpha(m'_2) \models RHS(r)$.

This implies that m_1' is in $[LHS(r)]$ and m_2' is in $[RHS(r)]$. Since we define $a = \mu(LHS(r), \mathcal{L}^*(F(C)))$, $[a] = [LHS(r)]$ follows from the definition of μ. Similarly, $[b] = [RHS(r)]$, so we have $m_1' \in [a]$, $m_2' \in [b]$. $m_1' \in [a \wedge \neg b]$ and $m_2' \in [\neg a \wedge b]$ follows from $[a] = [a \wedge \neg b]$, $[b] = [\neg a \wedge b]$.

Now we show that m_1, m_2 are the same outside the support of $[a \wedge \neg b]$ and $[\neg a \wedge b]$, respectively. Note that any two logically equivalent statements have the same support. This is a consequence of how we define support. Thus, $s(LHS(r)) = s(a) = s(a \wedge \neg b)$, and $s(RHS(r)) = s(b) = s(\neg a \wedge b)$. Note also, from the definition of the feature vector representation, we have $s(LHS(R)) = s(RHS(R))$, as a consequence of the requirement that the support features of $LHS(r)$ be the same as the support features of $RHS(r)$. Thus $s([a \wedge \neg b]) = s([\neg a \wedge b])$. We are given that $(m_1, m_2) \models r$, this implies m_1, m_2 are the same outside the support of $LHS(r)$, $RHS(r)$. Thus, m_1, m_2 are the same outside support of $[a \wedge \neg b]$. This implies there exists m such that $m_1' \in m[a \wedge \neg b]$, $m_2' \in m[\neg a \wedge b]$, which completes the proof. We note that $m = m_1'$ or $m = m_2'$ satisfies the condition on m. $\square$

2.2.3 Summary

We have shown translations from the feature vector representation to the *ceteris paribus* representation, and a similar translation from the *ceteris paribus* representation to the feature vector representation. Combining Lemma 2.2.1 and Lemma 2.2.2 we can state the following theorem:

Theorem 2.1 (Expressive Equivalence) *Any preference over models representable in either the* ceteris paribus *representation or the feature vector representation can expressed in the other representation.*

Using this equivalence and the construction established in section 2.2.1, we define an extension of the translation function σ (defined in definition 2.2) to sets of *ceteris paribus* preference rules in $\mathcal{L}$. This translation converts a set of *ceteris paribus* preference rules to a set of feature vector representation statements. That is, $\sigma : C \rightarrow C^*$, where C is a set of *ceteris paribus* rules, and C^* is a set of rules in $\mathcal{L}^*_{\mathcal{R}}(F(C))$. This translation is accomplished exactly as described in section 2.2.1.

Despite the demonstrated equivalence, the two preference representations are not equal in every way. Clearly, the translation from the *ceteris paribus* representation can result in the creation of more than one feature vector representation rule. This leads us to believe that the language of the *ceteris paribus* representation is more complicated than the other. This is naturally the case, since we can have complicated logical sentences in the language $\mathcal{L}$ in our *ceteris paribus* rules. Thus, it can take many more rules in $\mathcal{L}^*$ to represent a rule in $\mathcal{L}$. This gives us some further insight.

The length, in characters, of a *ceteris paribus* rule in the logical language $\mathcal{L}$, is arbitrary. The length of a rule in $\mathcal{L}^*_{\mathcal{R}}(\mathcal{V})$ is always $2 * |\mathcal{V}|$. Generally, we are concerned with the case of $\mathcal{V} = |F(C)|$, the relevant feature set. We might also ask how quickly we can tell if a model m of $\mathcal{L}^*(\mathcal{V})$ satisfies a statement r in $\mathcal{L}^*_{\mathcal{R}}(\mathcal{V})$. This verification is essentially a string parsing problem, where we check elements of m against elements

of r. The determination of satisfaction takes time proportional to the length of the formula.

Chapter 3

Some simple ordinal utility functions

In this chapter we illustrate simple utility functions, $u : \mathcal{M}^*(F(C)) \rightarrow \mathbb{R}$ consistent with an input set of feature vector representation preferences C^* where each rule $r \in C^*$ is a statement in $\mathcal{L}^*_{\mathcal{R}}(F(C))$. It is possible to combine this with the translation defined in the previous chapter. We may take as input a set of *ceteris paribus* preference rules C, and then let $C^* = \sigma(C)$. Similarly, one can use these functions to define utility functions over models in $\mathcal{L}$ by composition with the model-projection mapping α. Specifically, one finds the utility of a model $m \in \mathcal{M}$ by computing the projection $\alpha(m) \in \mathcal{M}(\mathcal{V})$ and using one of the functions defined in the following.

We consider a *model graph*, $G(C^*)$, a directed graph which will represent preferences expressed in C^*. Each node in the graph represents one of the possible models over the features $F(C)$. The graph $G(C^*)$ always has exactly $2^{|F(C)|}$ nodes. The graph has two different kinds of directed edges. Each directed edge in $G(C^*)$, $e_s(m_1, m_2)$ from source m_1 to sink m_2, exists if and only if $(m_1, m_2) \models r$ for some strict preference rule $r \in C^*$. Similarly, an edge $e_w(m_1, m_2)$ from source m_1 to sink m_2, exists if and only if $(m_1, m_2) \models r$ for some weak preference rule $r \in C^*$. Each edge, e_s and e_w, is an explicit representation of a preference for the source over the sink. It is possible to consider each rule r and consider each pair of models $(m_1, m_2) \models r$, and create an edge e for each such pair.

After this process is completed, we can determine if m_i is preferred to m_j according to C^* by looking for a path from m_i to m_j in $G(C^*)$ and a path from m_j to m_i. If both pathes exist, and are composed entirely of weak edges e_w, then we can conclude that $m_i \sim m_j$. If only a path exists from m_i to m_j, or a path exists using at least one strict edge, then we can conclude that $m_i \succ m_j$ according to C^*. Similarly, if only a path exists from m_j to m_i, or one using at least one strict edge, then we can conclude that $m_j \succ m_i$ according to C^*. If neither path exists, then a preference has not been specified between m_i and m_j in C^*. Cycles in the model graph are consistent if all the edges in the cycle are weak edges (e_w). In this case, all the models in the cycle are similarly preferred to each other, and should be assigned the same utility. If there are cycles using strict edges (e_s), this would indicate that some model is preferred to itself according to C^*, which we consider inconsistent. Thus, it is possible to construct the

Utility Function	Symbol	Method
Minimizing GUF	u_M	longest path from m
Descendent GUF	u_D	number of descendants of m
Maximizing GUF	u_X	longest path ending at m
Topological GUF	u_T	rank of m in topological-sorted order

Table 3.1: Four Different Ordinal Graphical Utility Functions (GUFs)

graph, and then look for cycles, with the presence of strict-cycles indicating that the input preferences are inconsistent.

3.1 Graphical Utility Functions

In this section we will define four different Graphical Utility Functions (GUFs). Each GUF will use the same graph, a model graph $G(C^*)$, but will define different measures of utility using this graph. We will define GUFs that compute the utility of a model m by checking the length of the longest path originating at m, counting the number of descendants of m in $G(C^*)$, or the number of ancestors, or the length of the longest path originating elsewhere and ending at m, or the number of nodes following it in the topological-sorted order of $G(C^*)$. We present a summary of each GUF in table 3.1.

Definition 3.1 (Minimizing Graphical Utility Function) *Given the input set of* ceteris paribus *preferences C^*, and the corresponding model graph $G(C^*)$, the Minimizing Graphical Utility Function is the utility function u_M where $u_M(m)$ is equal to the number of unique nodes on the longest path, including cycles, originating from m in $G(C^*)$.*

The intuition is that in a graph, the distance between nodes, measured in unique nodes, can indicate their relative utilities. Note that if neither $m_1 \succ m_2$ or $m_2 \succ m_1$ in C^*, then we do not require $u(m_1) = u(m_2)$. Only when $m_1 \succsim m_2$ and $m_2 \succsim m_1$ in C^*, do require $u(m_1) = u(m_2)$. This implies members of a cycle should receive the same utility. However, in a graph with cycles the concept of longest path needs more discussion. Clearly several infinite long pathes can be generated by looping forever on different cycles. We mean the longest non-backtracking path wherein all nodes on any cycle the path intersects are included on the path. It is important to include each of the members of a cycle as counted once among the "unique nodes" on this longest path, however, since this assures that a set of nodes on the same cycle will receive the same utility. For example, suppose the rules C^* are such that only the

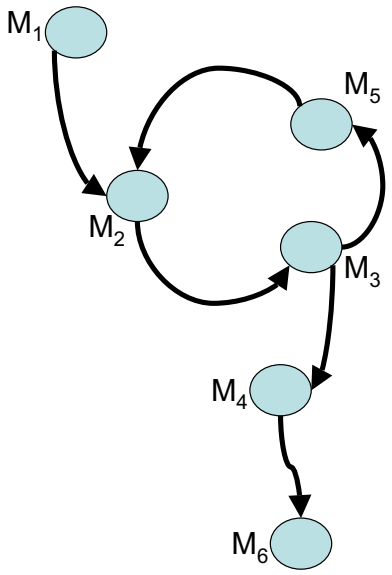

Figure 3-1: Model graph for preferences in equations 3.1

following relations are implied over models:

$$\begin{aligned} m_1 &\succsim m_2 \\ m_2 &\succsim m_3 \\ m_3 &\succ m_4 \\ m_3 &\succsim m_5 \\ m_5 &\succsim m_2 \\ m_4 &\succ m_6. \end{aligned} \tag{3.1}$$

It is clear that m_2, m_3, m_5 form a cycle (see figure 3.1). The longest path from m_1 clearly goes through this cycle, in fact this path visits nodes m_1, m_2, m_3, m_4, m_6. However, since this path intersects the cycle $\{m_2, m_3, m_5\}$, we add all nodes on the cycle to the path, in this case only the node m_5. Thus, the "longest path" we're interested in from m_1 passes though all six nodes, and we have $u_M(m_1) = 6$. Similarly, the other nodes receive utility as follows:

$$\begin{aligned} u_M(m_1) &= 6 \\ u_M(m_2) &= 5 \\ u_M(m_3) &= 5 \\ u_M(m_4) &= 2 \\ u_M(m_5) &= 5 \\ u_M(m_6) &= 1. \end{aligned}$$

Consider the relation between unrelated models provided by the minimizing graphical utility function. Our different graphical utility functions define different relationships for such models, in fact, the relationship is somewhat arbitrary. Consider the following example. Suppose the rules in C^* are such that only the following pairs of

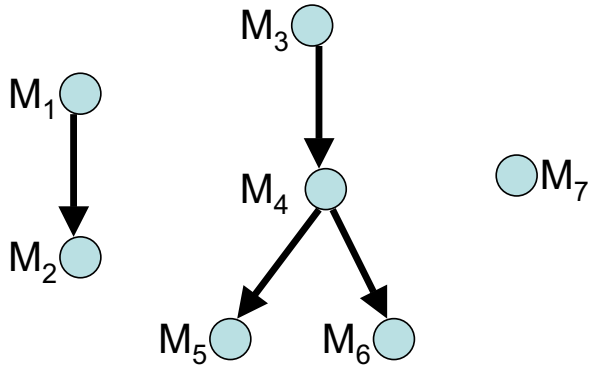

Figure 3-2: Model graph for preferences in equations 3.2

models satisfy any rule in C^*:

$$\begin{aligned} m_1 &\succ m_2 \\ m_3 &\succ m_4 \\ m_4 &\succ m_5 \\ m_4 &\succ m_6 \end{aligned} \tag{3.2}$$

The relationships between several models (see figure 3.1), for example m_1 and m_3, is unspecified. A utility function is free to order these two any way convenient, and the utility function will still be consistent with the input preferences. The minimizing graphical utility function gives the following values to the models:

$$\begin{aligned} u_M(m_1) &= 2 \\ u_M(m_2) &= 1 \\ u_M(m_3) &= 3 \\ u_M(m_4) &= 2 \\ u_M(m_5) &= 1 \\ u_M(m_6) &= 1. \end{aligned}$$

An interesting property of this utility function is that it is *minimizing*, that is, it assigns minimal utility to models that are not explicitly preferred to other models according to the preference set. Thus, suppose there exists an m_7 in the above domain, about which no preferences are specified. This model will receive minimal utility (1) from the utility function.

Definition 3.2 (Descendent Graphical Utility Function) *Given the input set of* ceteris paribus *preferences C^*, and the corresponding model graph $G(C^*)$, the Descendent GUF is the utility function u_D where $u_D(m)$ is equal to the total number of unique nodes on any paths originating from m in $G(C^*)$.*

Definition 3.3 (Maximizing Graphical Utility Function) *Given the input set of* ceteris paribus *preferences C^*, and the corresponding model graph $G(C^*)$, we let*

$\max(G(C^*))$ *be the length of the longest path in* $G(C^*)$*. The maximizing GUF is the utility function* u_X *where* $u_X(m)$ *is equal to* $\max(G(C^*))$ *minus the number of unique nodes on the longest path originating at any node other than* m *and ending at* m *in* $G(C^*)$*.*

Definition 3.4 (Topological Sort Graphical Utility Function) *Given the input set of strict* ceteris paribus *preferences* C^**, and the corresponding model graph* $G(C^*)$*, let* $n = 2^{|F(C)|}$ *be the number of nodes in* $G(C^*)$*. The topological sort GUF is the utility function* u_T *where* $u_T(m)$ *is equal to* n *minus the rank of* m *in the topological-sorted order of* $G(C^*)$*.*

Each of the above utility functions is ordinal. Each function except the topological sort GUF handles both weak and strict preferences. Our Maximizing Graphical Utility Function must make use of the same technique for the path intersecting a cycle that we use for the Minimizing Graphical Utility Function. The Descendent function has no such difficulty with cycles, but note that a node must be one of its own descendants.

Each function behaves differently toward models not mentioned by the preferences. We have already discussed the "minimizing" property, where a utility function gives very low or zero utility to models not mentioned by the preferences. These are models that are neither preferred to other models or have other models preferred to them. We therefore have no information about them, and are free to order them as convenient, while preserving the ordinal property of the utility function. In contrast to "minimizing," we call the above functions "maximizing" when they assign high utility to models about which we have no information.

Consider the example given in equations 3.2. Under the Descendent Graphical Utility Function (definition 3.2), we get the following utilities for $m_1, ..., m_7$:

$$\begin{aligned} u_D(m_1) &= 2 \\ u_D(m_2) &= 1 \\ u_D(m_3) &= 4 \\ u_D(m_4) &= 3 \\ u_D(m_5) &= 1 \\ u_D(m_6) &= 1 \\ u_D(m_7) &= 1. \end{aligned}$$

We give slightly higher utility to m_3, m_4 than the under function 3.1, since the former counts all descendants, while the latter counts only the longest path.

The maximizing graphical utility function, definition 3.3, gives the following values:

$$\begin{aligned} u_X(m_1) &= 3 \\ u_X(m_2) &= 2 \\ u_X(m_3) &= 3 \\ u_X(m_4) &= 2 \\ u_X(m_5) &= 1 \\ u_X(m_6) &= 1 \\ u_X(m_7) &= 3. \end{aligned}$$

Which is remarkable chiefly because it assigns $u(m_7) = 3$. This the speculative characteristic of the function.

Topological sort gives different answers depending on how it is implemented. However, each model gets a unique utility value. That is, the topological utility function is a one-to-one function from models to $\mathbb{R}$. It could give the following values for the example:

$$\begin{aligned} u_T(m_1) &= 7 \\ u_T(m_2) &= 6 \\ u_T(m_3) &= 5 \\ u_T(m_4) &= 4 \\ u_T(m_5) &= 3 \\ u_T(m_6) &= 2 \\ u_T(m_7) &= 1. \end{aligned}$$

This is a good example of how the utility of models can vary hugely under different utility function, and still have both function be consistent.

Now that we have discussed some of the properties and behaviors of each of the GUFs defined above, we prove their consistency.

Theorem 3.1 (Consistency of Minimizing GUF) *Given the input set of* ceteris paribus *preferences C^*, and the corresponding model graph $G(C^*)$, the utility function that assigns $u_M(m)$ equal to the number of unique nodes on the longest path originating from m in $G(C^*)$ is consistent with C^*.*

Proof. Let $u_M(m)$ be equal to the number of nodes on the longest path originating from m in $G(C^*)$. For u_M to be consistent with C^*, we require $p(u_M) \in [C^*]$. Specifically, this requires that $u_M(m_1) \geq u_M(m_2)$ whenever $m_1 \succsim_{C^*} m_2$, and $u_M(m_1) > u_M(m_2)$ whenever $m_1 \succ_{C^*} m_2$. Choose a pair m_1, m_2 such that $m_1 \succ_{C^*} m_2$ according to C^*. By construction of $G(C^*)$, there exists an edge from m_1 to m_2 in $G(C^*)$. Thus, $u_M(m_1) \geq u_M(m_2)$ because the there exists a path from m_1 that contains m_2, and therefore contains the longest path from m_2, plus at least one node, namely the node m_1. If $m_1 \succsim_{C^*} m_2$ then it is possible that there is both a path from m_1 to m_2 and a path from m_2 to m_1. In this case, these two pathes define a cycle containing both m_1 and m_2. Since m_1 and m_2 lie on the same cycle, $u_M(m_1) = u_M(m_2)$ since any longest path accessible from one model is also the longest path from the other. □

Theorem 3.2 (Consistency of Descendent GUF) *Given the input set of* ceteris paribus *preferences C^*, and the corresponding model graph $G(C^*)$, the utility function that assigns $u_D(m)$ equal to the total number of unique nodes on any pathes originating from m in $G(C^*)$ is consistent with C^*.*

Proof. Following the proof of theorem 3.1, if we know that $m_1 \succ m_2$ according to C^*, then by construction of $G(C^*)$, there exists an edge from m_1 to m_2 in $G(C^*)$. Since m_2 is on a path from m_1, m_1 has at least one more descendent than m_2, namely, m_1. Therefore $u_D(m_1) > u_D(m_2)$. If it is a weak edge from m_1 to m_2, then there might also be a path from m_2 to m_1, in which case, both models have the same set of descendants, and $u_D(m_1) = u_D(m_2)$. □

Theorem 3.3 (Consistency of Maximizing GUF) *Given the input set of* ceteris paribus *preferences C^*, and the corresponding model graph $G(C^*)$, we let* $\max(G(C^*))$ *be the length of the longest path in $G(C^*)$. The utility function that assigns$u_X(m)$ equal to* $\max(G(C^*))$ *minus the number of unique nodes on the longest path originating at any node other than m and ending at m in $G(C^*)$ is consistent with C^*.*

Proof. Omitted.

Theorem 3.4 (Consistency of Topological Sort GUF) *Given the input set of strict* ceteris paribus *preferences C^*, and the corresponding model graph $G(C^*)$, let $n = 2^{|F(C)|}$ be the number of nodes in $G(C^*)$. The utility function that assigns $u_T(m)$ equal to n minus the rank of m in the topological-sorted order of $G(C^*)$ is consistent with C^*.*

Proof. Omitted.

3.2 Complexity

The utility functions outlined in the previous section, while conceptually simple, have poor worst-cast complexity.

In order to construct $G(C^*)$, we need to add edges to the graph; since adding an edge is a constant time operation, counting the number of edges is the logical measure of complexity for this task. At worst, there are edges between every node in $G(C^*)$, so constructing $G(C^*)$ takes time quadratic in the number of nodes of $G(C^*)$. The number of nodes is exponential in the size of $F(C)$. The input is only of size $|C^*| * |F(C)|$, so graph construction is exponential in the size of the problem description. Thus, we anticipate a faster solution.

The computation of $u(m)$, after $G(C^*)$ is constructed, is linear in the number of descendants (for the Descendent Graphical Utility Function) of m in $G(C^*)$. The number of descendants is bounded by the number of nodes of in the graph. The other utility functions measure the number of ancestors, or the longest path from or to a node. Clearly counting the number of ancestors is the same computational burden as counting the number of descendants. Computing the longest path originating at a node and ending elsewhere is also the same, since all descendants must be searched so that a longest path maybe discovered. All of these functions are exponential in the size of $F(C)$.

There is a tradeoff to be made between the construction of $G(C^*)$ and the evaluation of $u(m)$. The computation of $u(m)$ can be reduced by significant preprocessing. Clearly each value of $u(m)$ could be cached at the node m in $G(C^*)$, using, for example, Dykstra's all-paths algorithm. This is still exponential in $|F(C)|$, but a much bigger exponent. Alternatively, a few values of $u(m)$ can be cached for different m's. If we are using the Descendent Graphical Utility Function, then computation of $u(m)$ need only proceed until each branch of the search from node m reaches a node with a cached utility value. If there are k separate branches of the search from m, $u(m_i)$ for $i = 1, ..., k$ is the cached utility value for a node m_i that is the terminus of the i^{th}

branch from m, and a total of t nodes found in all of the branches before reaching cached nodes, we assign

$$u(m) = 1 + t + \sum_{i=1}^{k} u(m_i).$$

If we keep a static set of nodes m_i with cached utilities, this is a consistent utility function. It is not exactly the same as the Descendent Graphical Utility Function, since the descendants of two cached nodes might overlap. However, since the important property is that an ancestor node has a higher utility than its descendants, this function is still consistent. In order to relieve ourselves of exponential dependence on $|F(C)|$, we need to cut the number of nodes searched before reaching cached nodes to $\log(|G(C^*)|)$. Since all nodes in $G(C^*)$ could be, for example, in one long path of length $|G(C^*)|$, we need to cache $|G(C^*)|/\log|G(C^*)|$ nodes. Thus, the preprocessing work is still linear in $|G(C^*)|$ and still exponential in $|F(C)|$.

On the other hand, $G(C^*)$ can be computed on-demand, when the utility of a model is required. We can do so in the following manner. We search for a path from a model m_0 by finding rules in $r \in C^*$ such that $(m_0, m_1) \models r$, where m_1 is arbitrary. An edge $e(m_0, m_1)$ in $G(C^*)$ exists if and only if there exists some $(m_0, m_1) \models r$ for any $r \in C^*$ (this follows from the proof of theorem 3.1). Thus, searching the list of rules in C^* for a pair $(m_0, m_1) \models r$ for some $r \in C^*$ is equivalent to following an edge $e(m_0, m_1)$ in $G(C^*)$. Then we recursively look for rules r such that $(m_1, m_2) \models r$, and then (m_2, m_3), and so on, such that our search becomes a traversal of the graph $G(C^*)$. Each branch of the search terminates when $(m_{k-1}, m_k) \models r$ for some rule $r \in C^*$, but $(m_k, m_{k+1}) \not\models s$ for all rules s in C. We know that there exists a path from m_0 with length k; if k is the length of the longest such path, we can then assign $u(m_0) = k$,

Since these methods are theoretically exponential, we might well ask if there is a simple statement of preferences that result in an exponential number of edges in $G(C^*)$. The answer is yes, and it has a pleasing form. For $|F(C)| = 4$, we choose set C^* equal to :

$$***1 \succ ***0 \tag{3.3}$$

$$**10 \succ **01 \tag{3.4}$$

$$*100 \succ *011 \tag{3.5}$$

$$1000 \succ 0111 \tag{3.6}$$

This set orders all models lexicographically (the same ordering we give models if we interpret them as binary representations of the integers from 0 to 15). Note that $|C^*| = |F(C)|$, and the length of the longest path through $G(C^*)$ is exactly $2^{|F(C)|}$ nodes. In fact, this is a complete linear ordering over the models on $F(C)$, with no ties. Thus, this gives one example of a small number of preference statements that suffice to give the Descendent Graphical Utility Function and its peers exponential time bounds in the size of the input.

Note that the suffix-fixing heuristics presented by [BBHP99] do not apply because our preference rules are of a quite different form, as seen in equations 3.3 to 3.6.

Boutilier *et. al.*'s methodology admits logical conditions determining when a preference holds, but the preference itself is limited to only one feature. Our preference rules can reference more than one feature at a time. Thus a rule such as $**10 \succ **01$ is not possible.

In the following sections, we present a number of heuristics using the concept of *utility independence* that reduce the complexity of evaluating $u(m)$.

Chapter 4

Utility Independence

Utility independence (UI) is the idea that the contribution to utility of some features can be determined without knowledge of the values assigned to other features. More precisely, if a set of features S_i is utility independent of a disjoint set of features S_j, then the utility given to S_i, does not depend on the values the features of S_j assume. We give some general background regarding utility independence here. In the following subsections, we demonstrate how this expedites our calculation of utility from *ceteris paribus* preferences. For these discussions, we assume that $C^* = \sigma(C)$, for C a set of strict *ceteris paribus* preference rules. We use S_i, $1 \leq i \leq k$, for sets of features $S_i \subseteq F(C)$, which are UI of other sets of features.

For a given set S_i, define a *partial utility function* $\hat{u_i} : \mathcal{M}^*(S_i) \rightarrow \mathbb{R}$ that assigns a utility to all models m in $\mathcal{M}^*(F(C))$ based on the restriction of m to S_i. For a given set S_i, and given that we have a function $\hat{u_i}$ defined, we will call a *subutility function* a function $u_i : \mathcal{M}^*(F(C)) \rightarrow \mathbb{R}$ such that $\forall m, u_i(m) = \hat{u_i}(m \restriction S_i)$. We will write $u_i(\cdot)$ for the subutility function for S_i, when it is clear that the subscript i refers to a particular set of features with the same subscript. We will also write $u_S(\cdot)$ for the subutility function for the features S when S is a set of features.

We use the logical connective "$\wedge$" between models $m_1 \wedge m_2$ as shorthand for $\mu(m_1, \mathcal{L}^*(F(C))) \cup \mu(m_2, \mathcal{L}^*(F(C)))$. That is, $m_1 \wedge m_2$ is a combined model of the support features of m_1 and m_2. To talk about properties of the values assigned by m_1 over the set S_1 and the values assigned by m_2 to S_2, we would discuss $m_1 \restriction S_1 \wedge m_2 \restriction S_2$.

A set of features S_i is UI of a disjoint set of features S_j if and only if, for all $m_1 \restriction S_j$, $m_2 \restriction S_i$, $m_3 \restriction S_i$, and $m_4 \restriction S_j$

$$\begin{array}{l}((m_2 \restriction S_i \wedge m_1 \restriction S_j) \succ (m_3 \restriction S_i \wedge m_1 \restriction S_j)) \Rightarrow \\ ((m_2 \restriction S_i \wedge m_4 \restriction S_j) \succ (m_3 \restriction S_i \wedge m_4 \restriction S_j))\end{array} \tag{4.1}$$

which is the general definition of utility independence [KR76]. [1]

In general UI is not symmetric; if S_i is UI of S_j that does not imply that S_j is UI of S_i. However, when a particular pair of feature sets are UI of each other, we say they are *mutually utility independent*. If this is the case, and $S_i \cup S_j = F(C)$, then

[1] Keeney and Raiffa actually term this *preferential independence*. Our terminology and notation differ from this source.

there exists a utility function u consistent with C^*, of the form:

$$u(m) = t_i u_i(m) + t_j u_j(m)$$

where $t_i, t_j > 0$ are scaling constants [KR76]. This is known as an *additive decomposition*. This theory generalizes to several sets of mutually UI features. If we have a collection of disjoint sets $S_1, S_2, \ldots S_Q \subset F(C)$, where each S_i is UI of its complement, $\overline{S_i}$, then there exists a utility function of the form

$$u(m) = \sum_{i=1}^{Q} t_i u_i(m). \tag{4.2}$$

This result is proven in [Deb59]. This is the form of the utility function that we will use throughout, and that we will attempt to construct.

4.1 *Ceteris Paribus* Preferences and UI Assumptions

As we have hinted before, and as we will see later, having a large number of utility-independent feature sets of small cardinality greatly speeds the computation of a utility function. The task is then to establish utility independence between sets of features and their complements, wherever possible. One common method of determining if two sets of features are utility independent is to ask the user whose preferences are being modelled. The burden is placed on the user of the preference system. In the decision theory literature, there is a tradition of assuming that utility independence of feature sets must be communicated along with a preference. Keeney and Raiffa give substantial treatment to the best phrasing of such questions to human decision makers [KR76]. Even recent work, for example, [BG96] require that utility independence statements accompany each preference statement before they analyze the preference.

Our approach is slightly different. We assume that each feature is utility independent of all other features, then try to discover for which features this assumption is invalid. When we compute which features are UI of which other features, for each $f \in F(C)$, we assume that f is UI of $F(C)\backslash f$. We then look for evidence that demonstrates two feature sets are not UI of each other. These evidences are rule pairs for which implication (4.1) does not hold.

Two rules can demonstrate that implication (4.1) does not hold for two feature sets. The intuitive idea is to find a pair of rules that demonstrate the preference for some features depends on the value of other features. To use a very simple example, consider two rules in $\mathcal{L}^*_{\mathcal{R}}(\langle f_1, f_2, f_3\rangle)$; the rules $01* \succ 11*$ and $10* \succ 00*$. It is easy to see that the preference expressed over f_1 switches depending on the value of f_2. *I.e.*, $\{f_1\}$ is not UI of $\{f_2\}$. Linguistically, such a preference can be interpreted as follows. If f_2 is 1, then I prefer $f_1 = 0$ to $f_1 = 1$. However, if instead $f_2 = 0$, then I prefer $f_1 = 1$ to $f_1 = 0$. Perhaps f_1 indicates that I carry an umbrella, and f_2 indicates that

it is a sunny day. Then this statement represents the utility of the umbrella being dependent on the current weather. In good weather, the umbrella is a burden; in inclement weather, the umbrella is beneficial.

To establish utility dependence, we require the demonstration of models in $\mathcal{M}^*(F(C))$: m_1, m_2, m_3, m_4; for some feature sets S_i, S_j and a pair of rules r_1, r_2 such that implication (4.1) does not hold. The correspondence is analogous to implication (4.1): we want all of the following conditions to hold

$$\begin{array}{l}(m_2 \restriction S_i \wedge m_1 \restriction S_j) \models LHS(r_1),\\ (m_3 \restriction S_i \wedge m_1 \restriction S_j) \models RHS(r_1),\\ (m_2 \restriction S_i \wedge m_4 \restriction S_j) \models RHS(r_2),\\ (m_3 \restriction S_i \wedge m_4 \restriction S_j) \models LHS(r_2).\end{array} \tag{4.3}$$

Although this may seem hopelessly abstract, it is easy to notice such m_1, m_2, m_3, m_4, S_i, S_j by looking at the support feature sets of each rule. Since S_j is a subset of the support features of r_1, and m_1 restricted to S_j satisfies both $LHS(r_1)$ restricted to S_j and $RHS(r_1)$ restricted to S_j, we can look for rules of this form. Lexically, these rules are easy to recognize: some of the features specified in the rule have the same value on the left- and right-hand sides of the rule. Then we look for another rule, r_2, with the same property: there is a set of features S'_j that is a subset of the support features of r_2 and there is an m_4 satisfies both $LHS(r_2)$ restricted to S'_j and $RHS(r_2)$ restricted to S'_j, with the additional restriction that S'_j is a subset of S_j. Again, we are looking for a rule that specifies the same values for the same features on the left- and right-hand sides, but these features must be a subset of those features we found for r_1. If the previous conditions hold, we have fixed m_1, m_4, and S_j used in conditions (4.1) and (4.3).

We then check that r_1, r_2 are such that an S_i can be found that is disjoint with S_j and a subset of the support features of both r_1 and r_2, and an m_2 can be found that satisfies both $LHS(r_1)$ restricted to S_i and $RHS(r_2)$ restricted to S_i. Here we are looking for a preference over models restricted to S_i that switches with the values assigned to S_j. Again, this is easy to check for, by doing lexical comparisons on the left- and right-hand sides of the support feature sets of rules. If all the preceding holds for some S_i, S_j, then S_i is utility dependent on S_j. We are assured of this since the condition (4.1) is violated.

An example. Consider the rules in $\mathcal{L}^*_{\mathcal{R}}(\langle f_1,f_2,f_3,f_4,f_5\rangle)$ as follows:

$$r_1 = *0001 \succ *1101 \tag{4.4}$$

$$r_2 = **100 \succ **000 \tag{4.5}$$

Thus we can choose $S_j = f_4, f_5, m_1 \restriction S_j = 01$. This satisfies our condition on m_1, that $m_1 \models LHS(r_1) \restriction S_j$ and $m_1 \models RHS(r_1) \restriction S_j$. We choose S_j to be the features of the intersection of $\mu(LHS(r_1), \mathcal{L}^*(F(C)))$ and $\mu(RHS(r_1), \mathcal{L}^*(F(C)))$. Then we consider r_2. We again consider the features of intersection of $\mu(LHS(r_2), \mathcal{L}^*(F(C)))$ and $\mu(RHS(r_2), \mathcal{L}^*(F(C)))$, which is $\{f_4, f_5\}$. Since this happens to be the same as S_j, we set $S'_j = S_j$. Then $m_4 \restriction S'_j = 00$, and we indeed have $m_1 \restriction S_j \neq m_4 \restriction S'_j$. We let $S_j = S_j \cup S'_j$.

Next, to choose S_i, check if there are features shared between $\mu(RHS(r_1), \mathcal{L}^*(F(C)))$ $\cap$ $\mu(LHS(r_2), \mathcal{L}^*(F(C)))$ and $\mu(RHS(r_2), \mathcal{L}^*(F(C))) \cap \mu(LHS(r_1), \mathcal{L}^*(F(C)))$. Since S_i must be disjoint from S_j, we exclude features in S_j from this check. We see that f_3 is shared in these two sets, thus we assign $S_i = f_3$. Then $m_2 \restriction S_i = 0$, $m_3 \restriction S_i = 1$, and we verify that $m_2 \restriction S_i \neq m_3 \restriction S_i$. This process constitutes evidence that S_i, S_j, m_1, m_2, m_3, m_4 cannot satisfy implication 4.1, and therefore that S_i, S_j are not UI.

Theorem 4.1 (Independence Construction) *The above method computes a partition S of the features $F(C)$ such that each set $S_i \in S$ is utility independent of the features in $F(C) \backslash S_i$, and further that no partition S' exists with $\exists S_i, S_j \in S'$ such that $S_i \cap S_j = S_k \in S$ and S' has the same properties as S.*

Proof. First note that it is clear that S is a partition because the algorithm starts out with a partition and at each step S is updated by joining two of the sets together, an operation which preserves the property of being a partition. Further, it is clear that S is a set of MUI sets, because the algorithm only halts when no two features can be found that are utility dependent.

Now we show that there does not exist another partition S' with $S_i, S_j \in S'$ such that $S_i \cap S_j = S_k \in S$. First suppose S' is such a partition. Then there exists $S_i, S_j \in S'$ such that $S_i \cup S_j = S_k \in S$, and $S_i \cap S_j = \emptyset$. Since the partition S is the result of our method, and $S_k \in S$, we know that there were some rules in C that caused S_i and S_j to be joined into S_k. Without loss of generality, assume $a \subseteq S_i$ was shown to be utility dependent on $b \subseteq S_j$. Thus, there is some pair of rules r_1, r_2, and some models m_1, m_2, m_3, m_4 such that equations 4.3 are satisfied. That is, we have:

$$
\begin{aligned}
&(m_2 \restriction a \wedge m_1 \restriction b) \models LHS(r_1),\\
&(m_3 \restriction a \wedge m_1 \restriction b) \models RHS(r_1),\\
&(m_2 \restriction a \wedge m_4 \restriction b) \models RHS(r_2),\\
&(m_3 \restriction a \wedge m_4 \restriction b) \models LHS(r_2).
\end{aligned}
$$

However, this is in direct conflict with the definition of utility independence, as given in implication 4.1. Thus we have reached a contradiction, and conclude that our supposition, that S' is such a partition, is false. $\square$

4.2 Utility-Independent Feature Decomposition

We are interested in computing a partition $S = \{S_1, S_2, ..., S_Q\}$ of the features $F(C)$ such that each set S_i is utility independent of its complement, $F(C) \backslash S_i$. That is, the feature sets are all mutually utility independent. This partition allows us to use an additive utility decomposition as described in equation 4.2.

Since our general method starts with each feature assumed to be utility independent of all the others, and we then establish utility dependence among certain features, we must mention some properties of utility dependence. Even though we have stated that utility independence is not symmetric [KR76], mutual utility independence is symmetric. If S_i is utility dependent on S_j, but not S_j on S_i, it is no

longer the case that S_i is utility independent of $F(C)\backslash S_i$. In fact, the feature set of minimal cardinality containing S_i and independent of its complement is the set $S_i' = S_i \cup S_j$. Thus, utility dependence, as we use it, is a symmetric property.

Similarly, mutual utility independence requires that utility dependence is transitive. Consider the following example. If S_i is utility dependent on S_j and S_i is utility dependent on S_k, then the set with minimal cardinality that is utility independent of its complement and contains S_i is the set $S_i' = S_i \cup S_j \cup S_k$.

4.3 A Method for Finding Partitions

By combining the methods discussed above, we can compute a partition of $F(C)$ into mutually independent feature sets. When our set of features is $F(C)$, we calculate a set of feature vector representation rules C^*, according to section 2.2.1. We outline an algorithm below. The main idea is that sets of features $S_i, S_j \subseteq F(C)$, which are not UI of each other can be detected by doing preprocessing on C^*. We simply check each pair of rules to see if a preference for one feature changes with the value assigned to another feature (by checking if the pair of rules satisfies the conditions laid out in section 4.1). If this is the case, we note that these features are not UI of each other.

We would like to compute a set S of mutually utility-independent feature sets. Since we must behave as if utility dependence is transitive, we will look for disjoint sets. We initialize S to a set of singleton sets, where each set contains only one feature from $F(C)$. That is, we have $S_1 = \{f_1\}$, $S_2 = \{f_2\}$, ..., $S_N = \{f_N\}$. As we consider rule pairs and decide that a feature in one set is utility dependent on another feature, we join these two sets containing these features together. Computationally, this is an all-pairs process on the preference rules C^*. For each pair of rules r_1, r_2, the overlap of $s(r_1)$ and $s(r_2)$ must be computed. Then we check if this overlap admits models of $\mathcal{L}^*(F(C))$ such that implication 4.1 does not hold. The following pseudo-code performs this check, for rules r_1, r_2:

1. $s_{same1} \leftarrow \{f_i \mid f_i(s(LHS(r_1))) = f_i(s(RHS(r_1))), \forall f_i \in s(r_1)\}$
2. $s_{same2} \leftarrow \{f_i \mid f_i(s(LHS(r_2))) = f_i(s(RHS(r_2))), \forall f_i \in s_{same1}\}$
3. $S_u \leftarrow s_{same1} \cap s_{same2}$
4. $s_{cross1} \leftarrow \{f_i \mid f_i(s(LHS(r_1))) = f_i(s(RHS(r_2))), \forall f_i \in s(r_1)\}$
5. $s_{cross2} \leftarrow \{f_i \mid f_i(s(RHS(r_1))) = f_i(s(LHS(r_2))), \forall f_i \in s(r_1)\}$
6. $s_{intersect} \leftarrow s_{cross1} \cap s_{cross2}$
7. $S_d \leftarrow \{f_i \mid f_i(LHS(r_1)) \neq f_i(RHS(r_1)), \forall f_i \in s_{intersect}\}$

If S_d and S_u are both non-empty, and we discover that features S_d must be dependent on features S_u. We update our sets $S_i \in S$ as follows. For each pair $f', f'' \in (S_d \cup S_u)$, find the set S_i which contains f', and the set S_j which contains f''. Then remove both of S_i, S_j from the partition S and add a set $S_i' = S_i \cup S_j$ to S.

After performing such overlap calculations, we have computed a partition of $F(C)$, S, where each $S_i \in S$ is UI of its complement.

Theorem 4.2 (Independence Algorithm) *The above algorithm computes a set S of feature sets S_i such that each set S_i is utility independent of the features in $F(C)\backslash S_i$.*

Proof. We check each pair of rules using the above pseudo code. Performing this on each pair of rules performs the same check described in section 4.1. Then by theorem 4.1, the pseudo code produces a partition S such that each set $S_i \in S$ is utility independent of the features in $F(C)\backslash S_i$. We show that this is the case.

The first variable, s_{same1}, holds the support features that are the are assigned the same value on the LHS and RHS of r_1. s_{same2} performs the same calculation for r_2. S_u is the intersection of these two. Thus, S_u is a set of features that have the same value on the LHS and RHS of r_1, and on r_2. S_u is the same as the set S_j used in implication 4.1 and 4.3. The variable s_{cross1} holds the support features which are assigned the same value on the LHS of r_1 and the RHS of r_2. s_{cross2} holds the support features which are assigned the same value on the RHS of r_1 and the LHS of r_2. $s_{intersect}$ holds the intersection of s_{cross1} and s_{cross2}. S_d is the subset of $s_{intersect}$ that switches values from one rule to the other. Thus S_d is the set of features over which r_1 and r_2 express a preference over models that switches depending on the value of the features in S_u. □

Theorem 4.3 (Utility Decomposition) *Using the partition S provided in section 4.3 permits a utility function of the form*

$$u(m) = \sum_{i:S_i \in S} t_i u_i(m) \tag{4.6}$$

where u_i is a subutility utility function for the features S_i and t_i is a constant parameter.

Proof. By theorem 4.2, the partition S is a partition of $F(C)$ into feature sets utility independent of their complements. A general result in utility theory guarantees that any partition of into utility-independent sets permits a additive utility decomposition of the form given here [KR76]. □

Chapter 5

Utility Construction

We now describe how to compute one utility function consistent with the set of input *ceteris paribus* preferences, C. We take the partition of $F(C) : S_1, S_2, \ldots, S_Q$ discussed in section 4.1, where each set of features S_i is utility independent of its complement. We have shown that there is an additive utility function that is a linear combination of subutilities, each subutility a separate function of a particular S_i. We associate a *scaling parameter* $t_i > 0$ with each S_i such that the utility of a model is

$$u(m) = \sum_{i=1}^{Q} t_i u_i(m). \tag{5.1}$$

We will argue that this is consistent with a set of preference rules C^* where $C^* \in [C]$ is a set of rules in $\mathcal{L}^*_{\mathcal{R}}(F(C))$. We have two tasks: to craft the subutility functions u_i, and to choose the scaling constants t_i. We will accomplish these two tasks in roughly the following way. We will show how a rule can be *restricted* to a set S_i. This is essentially a shortening of a rule to a particular set of features, in the same manner as we have defined restriction of models to a set of features. By restricting rules to the sets S_i, we can use these shortened forms of the rules to make GUFs for the partial utility functions u_i. Rules that do not conflict in general can conflict when restricted to different feature sets. These conflicts can be phrased as constraints and resolved using a boolean constraint satisfaction solver (SAT). To assign values to scaling parameters t_i, we will define a set of linear inequalities which constrain the variables t_i. The linear inequalities can then be solved using standard methods for solving linear programming problems. The solutions to the inequalities are the values for the scaling parameters t_i. Along the way we will show some helpful heuristics.

We first describe the subutility functions, and then their linear combination into a full utility function.

5.1 Subutility Functions

Our task is to define subutility functions $u_{S_i}(m) = \hat{u}_{S_i}(m \restriction S_i)$ that take as input the values for the features $F(C)$, and return an integer. We define such a function relative to some rules $r \in C^*$. We say that a subutility function u_{S_i} is ϵ-*consistent* with $r \in C^*$

if $u_{S_i}(m_1) \geq \epsilon + u_{S_i}(m_2)$ whenever $(m_1, m_2) \models r$ and $(m_1 \restriction S_i) \neq (m_2 \restriction S_i)$, where $\epsilon > 0$. In the following, we generally let $\epsilon = 1$ and prove results for 1-consistency. In general, it is not necessary to use $\epsilon =$, but using 1 will make some calculations simpler. We show that 1-consistency implies the consistency of u with r.

Theorem 5.1 (Subutilities) *Given rules C^* and a mutually utility-independent partition S of $F(C)$, if each u_i is 1-consistent with r, then*

$$u(m) = \sum_{i=1}^{Q} t_i u_i(m)$$

where $t_i > 0$, is consistent with r.

Proof. By definition of consistency of u, u is consistent with r if $u(m_1) > u(m_2)$ whenever $(m_1, m_2) \models r$. Using equation 5.1, $u(m_1) > u(m_2)$ is true when

$$\sum_{i=1}^{Q} t_i u_i(m_1) > \sum_{i=1}^{Q} t_i u_i(m_2).$$

Rearranging the summations gives

$$\sum_{i=1}^{Q} t_i (u_i(m_1) - u_i(m_2)) > 0.$$

For each utility-independent set S_i, clearly either $(m_1 \restriction S_i) \neq (m_2 \restriction S_i)$ or $(m_1 \restriction S_i) = (m_2 \restriction S_i)$, so we can split the summation into these two parts.

$$\sum_{i:(m_1 \restriction S_i)=(m_2 \restriction S_i)}^{Q} t_i (u_i(m_1) - u_i(m_2)) + \sum_{i:(m_1 \restriction S_i)\neq(m_2 \restriction S_i)}^{Q} t_i (u_i(m_1) - u_i(m_2)) > 0.$$

By definition of a subutility function u_i, if $(m_1 \restriction S_i) = (m_2 \restriction S_i)$ then $u_i(m_1 \restriction S_i) = u_i(m_2 \restriction S_i)$, so the first summation is a sum of zeros, giving:

$$\sum_{i:(m_1 \restriction S_i)\neq(m_2 \restriction S_i)}^{Q} t_i (u_i(m_1) - u_i(m_2)) > 0.$$

By definition of 1-consistency of u_i with r, $u_{S_i}(m_1) - u_{S_i}(m_2) \geq 1$ when $(m_1, m_2) \models r$ and $(m_1 \restriction S_i) \neq (m_2 \restriction S_i)$. Since $t_i > 0$, we have so

$$\sum_{i:(m_1 \restriction S_i)\neq(m_2 \restriction S_i)}^{Q} t_i (u_i(m_1) - u_i(m_2)) > 0.$$

Which establishes that $\sum_{i=1}^{Q} t_i (u_i(m_1) - u_i(m_2)) > 0$, and therefore that $u(m_1) > u(m_2)$. This finishes the proof. □

We show how to generate a 1-consistent subutility function by constructing a *restricted model graph*, $G_i(R)$ for a set of rules R in $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$. $G_i(R)$ is a multi-graph with directed edges, where each model m in $\mathcal{M}(S_i)$ is a node in the graph; recall we have defined $\mathcal{M}(\mathcal{V})$ to be the set of models of the features listed in $\mathcal{V}$. Let m_1, m_2 be models in $\mathcal{M}(S_i)$ with $m_1 \neq m_2$. There exists one directed edge in the restricted model graph $G_i(R)$ from node m_1 to node m_2 for each distinct rule $r \in R$ such that $(m_1', m_2') \models r$ and $(m_1' \restriction S_i) = m_1$ and $(m_2' \restriction S_i) = m_2$. We label the edges with the rule r that causes the edge. The interpretation of an edge $e(m_1, m_2)$ in $G_i(S_i)$ is of a strict preference for m_1 over m_2. The construction of this graph $G_i(R)$ parallels the construction of the general model graph G, as described in chapter 3.

Note that if a rule $r \in R$ has $S_i \backslash s(r) \neq \emptyset$, then the rule makes more than one edge in the model graph $G_i(R)$. Specifically, a rule r makes

$$2^{|S_i \backslash s(r)|}$$

edges.

With a restricted model graph $G_i(R)$ defined, there is still the matter of defining a utility function on this graph. It appears we could choose any of the Ordinal Utility Functions described in chapter 3 to be the utility function for $\hat{u}_i$, and preserve further results in this chapter. For example, we could use the Minimizing Graphical Utility Function for $\hat{u}_i$. Recall that in this ordering, each node has utility equal to the length of the longest path originating at the node. If we use the Minimizing GUF, then we want $\hat{u}_i(m \restriction S_i)$ to return the length of the longest path starting at node $(m \restriction S_i)$ in graph $G_i(R)$.

Lemma 5.1.1 (Cycle-Free Subutility) *Given a set of rules $R \subseteq C^*$ and a set of features $S_i \subseteq F(C)$ such that the restricted model graph $G_i(R)$ is cycle-free, and $\hat{u}_i(m \restriction S_i)$ is the minimizing graphical utility function over $G_i(R)$, the subutility function $u_i(m) = \hat{u}_i(m \restriction S_i)$ for S_i is 1-consistent with all $r \in R$.*

Proof. We must show that $u_i(m_1) \geq 1 + u_i(m_2)$ whenever $(m_1, m_2) \models r$ and $m_1 \restriction S_i \neq m_2 \restriction S_i$. First let $\hat{u}_i$ be the minimizing graphical utility function (from chapter 3) defined over $G_i(R)$. Then pick some $r \in R$ and some pair $(m_1, m_2) \models r$ where $(m_1 \restriction S_i) \neq (m_2 \restriction S_i)$. By definition of $G_i(R)$, there exists an edge $e(m_1 \restriction S_i, m_2 \restriction S_i)$ thus, $\hat{u}_i(m_1 \restriction S_i) \geq 1 + \hat{u}_i(m_2 \restriction S_i)$ because the there exists a path from $(m_1 \restriction S_i)$ that contains $(m_2 \restriction S_i)$, and therefore contains the longest path from $(m_2 \restriction S_i)$, plus at least one node, namely the node $(m_1 \restriction S_i)$. So we have

$$u_i(m_1) = \hat{u}_i(m_1 \restriction S_i) \geq 1 + u_i(m_2) = \hat{u}_i(m_2 \restriction S_i).$$

Since this holds for all $(m_1, m_2) \models r$ and for all $r \in R$, this proves the lemma. □

We claim but do not prove that similar results hold when $\hat{u}_i$ is any of the other three graphical utility functions defined in chapter 3.

5.2 Conflicting Rules

Although we assume that the input preferences C^* are all strict preferences, and have no conflicting rules, it is possible that a restricted model graph $G_i(C^*)$ for $S_i \subset F(C)$ has strict-edge cycles in it even though a model graph $G(C^*)$ for $F(C)$ does not. Since we are creating subutility functions for use in an additive composition utility function, we cannot use cycles. Using cycles would break many of the heuristics we will define.

Consider a set of rules $R \subseteq C^*$, and a mutually utility-independent partition S of the features $F(C)$. The rules R *conflict* if there exists some feature set $S_i \in S$ such that no subutility function u_i for S_i can be 1-consistent with all $r \in R$. When such a feature set S_i exists, we say that R *conflict on* S_i or that R *conflict on* u_i.

Lemma 5.2.1 (Conflict Cycles) *Given a set of strict preference rules $R \subseteq C^*$ and a mutually utility-independent partition S of $F(C)$, the rules R conflict if and only if the rules R imply a cycle in any restricted model graph $G_i(R)$ of some set of features $S_i \in S$.*

Proof. If the rules R imply a cycle in $G_i(R)$, then there exists some cycle of nodes m_j in $G_i(R)$. Call this cycle $Y = (m_1, m_2, ..., m_k, m_1)$. By the definition of a model graph, an edge $e(m_j, m_{j+1})$ exists if and only if $m_j \succ_R m_{j+1}$. Thus, for a subutility function u_i to be 1-consistent with R, the following would have to hold:

$$u_i(m_1) > u_i(m_2) > ... > u_i(m_k) > u_i(m_1)$$

Which is not possible, because $u_i(m_1) > u_i(m_1)$ is impossible. Thus, if R implies a cycle in some $G_i(R)$, then there is no subutility function u_i consistent with R.

In the other direction, we show that if the rules R conflict on S_i, then they imply a cycle in $G_i(R)$. We assume the opposite and work toward a contradiction. Suppose R conflict on S_i but they do not imply a cycle in $G_i(R)$. By the definition of a restricted model graph, $G_i(R)$ is cycle-free. Then we define the subutility function u_i to be the minimizing graphical utility function based on the graph $G_i(R)$. By lemma 5.1.1, u_i is 1-consistent with R. This is a contradiction, so we have shown that when R conflict on S_i they imply a cycle in $G_i(R)$. This proves the lemma. □

Note that if the rules R imply a cycle in the (unrestricted) model graph $G(R)$ over all features, $F(C)$, then the rules R represent contradictory preferences.

We define the *restriction* of a rule r to a set of features S, written $r \restriction S$, similar to restricting a model to a set. The restriction of a rule r to S is another rule in the feature vector representation, but over fewer features:

$$r \restriction S = RHS(r) \restriction S \succ LHS(r) \restriction S$$

Consider rules x, y in $\mathcal{L}_r(\mathcal{V})$, with $\mathcal{V} = (f_1, f_2, f_3, f_4)$, with $x = *01* \succ *10*$, $y = **01 \succ **10$. If we examine the restriction to feature three: $x \restriction \{f_3\}$ and $y \restriction \{f_3\}$, then we have $x \restriction \{f_3\} = 1 \succ 0$ and $y \restriction \{f_3\} = 0 \succ 1$. Thus, when restricted to f_3 it is inconsistent to assert both x and y. In these cases, we say that x, y conflict on f_3.

We call a set of rules R_i the *relevant rule set for* S_i if R_i is the set of rules $r \in C^*$ such that $s(r) \cap S_i \neq \emptyset$. The rules R_i may or may not conflict on S_i. If the rules R_i do not conflict on S_i, we say that R_i is *conflict-free*. If some subset of the rules R_i do conflict on S_i, we resolve conflicts by choosing a subset of rules $\overline{R}_i \subseteq R_i$ that is conflict-free. We choose some of the conflicting rules of R_i and exclude them from the set $\overline{R}_i$. How to choose which rule to exclude from $\overline{R}_i$ is a central point of discussion in the rest of the chapter. For now, just note that our choice of rules to remove influences our choice of scaling parameters t_i.

We define two properties of conflicting rule sets on R_i. Suppose there are sets of conflicting rules $Y_1, Y_2, ..., Y_K$, where each is composed of rules $r \in R_i$, and a feature set S_i. A set of conflicting rules Y_k is *minimal* if for any $r \in Y_k$, $Y_k \backslash r$ implies a subset of the cycle implied by Y_k on S_i. The set of conflicting rule sets $Y = \{Y_1, Y_2, ..., Y_K\}$ is *minimal and complete* if each conflicting subset R_i' of R_i is represented by some $Y_k \in Y$ such that for any $r \in Y_k$, $Y_k \backslash r$ implies a subset of the cycles implied by R_i'.

Theorem 5.2 (Conflict-free Rule Set) *Given a relevant rule set R_i for S_i, and a minimal, complete set of conflicting rule sets Y for R_i, and*

$$\overline{R}_i = R_i \backslash \{r_1, r_2, ..., r_K\} \tag{5.2}$$

where $r_1 \in Y_1$, $r_2 \in Y_2$, ..., $r_K \in Y_K$, then $\overline{R}_i$ is conflict-free.

Proof. We will assume the opposite, and arrive at a contradiction. Suppose that $\overline{R}_i$ is not conflict-free, then by lemma 5.2.1 $\overline{R}_i$ implies a cycle in $G_i(\overline{R}_i)$ of S_i. Since $\overline{R}_i \subset R_i$, the same cycle is implied by R_i. By minimality and completeness of Y, there is some set of rules $Y_j \in Y$ such that Y_j implies the cycles implied by R_i. By minimality of Y_j, if $\overline{R}_i$ implies the same cycle as Y_j, then $\overline{R}_i$ must contain the rules Y_j. But, the rules Y_j cannot be in $\overline{R}_i$, because $r_j \in Y_j$ and we have defined $r_j \notin \overline{R}_i$ in equation 5.2. Thus, we have arrived at a contradiction, and therefore shown that $\overline{R}_i$ is conflict-free. □

Lemma 5.2.1 states that conflict sets in R are equivalent to cycles in the restricted model graph $G_i(R)$. Thus, we can find minimal sets of conflicting rules in R by using a cycle detection algorithm on the graph $G_i(R)$. We can annotate the edges in $G_i(R)$ with the rule $r \in R$ that implies the edge. This way, it is simple to translate a cycle to a set of rules. Since $G_i(R)$ is a multi-graph, we know that if we find all cycles of $G_i(R)$, we have found a minimal and complete set of conflicting rules in R. Since one set of rules R may cause more than one cycle in a given $G_i(R)$, there may be multiple identical conflicting sets of rules. Thus, when translating a set of cycles to a set of sets of conflicting rules, we may end up pruning away duplicate cycles.

We can construct $\overline{R}_i$ satisfying equation 5.2 given minimal conflict sets Y_k and relevant rule sets R_i. Note that there are several possible values for $\overline{R}_i$ satisfying this equation, since we can choose any $r_1, r_2, ..., r_K$ such that $r_1 \in Y_1$, $r_2 \in Y_2, ..., r_K \in Y_K$. Since r_k can be a member of more than one rule conflict set Y_k, it is possible that the cardinality of $\overline{R}_i$ varies with different choices of r_k. It is generally advantageous to have $\overline{R}_i$ have as high a cardinality as possible, but not necessary.

We can construct the restricted model graph $G_i(\overline{R}_i)$. This is a model graph $G_i(\overline{R}_i)$ that is cycle-free. With no cycles, we can construct a subutility function for S_i using any of the graphical utility function given in chapter 3 over the restricted model graph $G_i(\overline{R}_i)$.

Theorem 5.3 (Subutility 1-Consistency) *Given a set of features $S_i \subseteq F(C)$ and corresponding conflict-free relevant rule set $\overline{R}_i$, the subutility function u_i using a graphical utility function from chapter 3 based on the restricted model graph $G_i(\overline{R}_i)$ is 1-consistent with all rules $r \in \overline{R}_i$*

Proof. Since $\overline{R}_i$ is conflict-free, by lemma 5.2.1, $G_i(\overline{R}_i)$ is cycle-free. Then by lemma 5.1.1, u_i is 1-consistent with $\overline{R}_i$. □

We call a subutility function satisfying theorem 5.3 a *graphical subutility function for S_i*. Note that $\hat{u}_i$ and u_i may or may not be 1-consistent with $r \in (R_i \backslash \overline{R}_i)$, and in general we assume that these functions are not 1-consistent with such r. For notational convenience, we say that a subutility function u_i as described in theorem 5.3 *agrees* with rules $r \in \overline{R}_i$, and that u_i *disagrees* with rules $r \in (R_i \backslash \overline{R}_i)$. 1-Consistency is implied by agreement, but not the converse, because agreement is a relation between a specific type of subutility function and a rule.

5.3 Consistency Condition

Recall that we have defined what it means for a utility function u to be consistent with a set of strict preferences C^* as $p(u) \in [C^*]$. This is equivalent to having $u(m_1) > u(m_2)$ whenever $m_1 \succ_{C^*} m_2$. This translates into the following condition on the subutility functions u_i. For each rule $r \in C^*$ and each pair of models $(m_1, m_2) \models r$, we have the following:

$$\sum_{i=1}^{Q} t_i u_i(m_1) > \sum_{i=1}^{Q} t_i u_i(m_2) \tag{5.3}$$

that is, if we have a rule r implying that $m_1 \succ m_2$, then the utility given by u to m_1 must be greater than that given to m_2. For it to be otherwise would mean our utility function u was not faithfully representing the set of preferences C. Let R_i be the relevant rule set for S_i. Examine one subutility function, u_i, and pick some $r \notin R_i$. By the definition of $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$, if $(m_1, m_2) \models r$, both m_1, m_2 assign the same truth values to the features in S_i. That is, $(m_1 \restriction S_i) = (m_2 \restriction S_i)$. This implies $u_i(m_1) = u_i(m_2)$. So we can split the summations of equation (5.3), giving:

$$\sum_{i:r\in R_i} t_i u_i(m_1) + \sum_{i:r\notin R_i} t_i u_i(m_1) > \sum_{i:r\in R_i} t_i u_i(m_2) + \sum_{i:r\notin R_i} t_i u_i(m_2) \tag{5.4}$$

then simplify:

$$\sum_{i:r\in R_i} t_i u_i(m_1) > \sum_{i:r\in R_i} t_i u_i(m_2). \tag{5.5}$$

Given a partition $S = \{S_1, ..., S_Q\}$ of $F(C)$, corresponding relevant rule sets R_i for each $S_i \in S$, conflict-free rule sets $\overline{R}_i$ for each R_i, and subutility functions u_i

1-consistent with $\overline{R}_i$, define $S_a(r)$ and $S_d(r)$ as follows. For a rule r, $S_a(r)$ is the set of indices i such that $r \in \overline{R}_i$, and $S_d(r)$ is the set of indices i such that $r \in (R_i \backslash \overline{R}_i)$.

Theorem 5.4 (Rule Consistency) *Given a mutually utility-independent partition $S = \{S_1, ..., S_Q\}$ of $F(C)$, corresponding relevant rule sets R_i for each $S_i \in S$, conflict-free rule sets $\overline{R}_i$ for each R_i, and subutility functions u_i 1-consistent with $\overline{R}_i$, a additive decomposition utility function u is consistent with a rule r if for all $(m_1, m_2) \models r$, we have*

$$\sum_{i \in S_a(r)} t_i(u_i(m_1) - u_i(m_2)) > \sum_{j \in S_d(r)} t_j(u_j(m_2) - u_j(m_1)). \tag{5.6}$$

Proof. If u is consistent with r then $u(m_1) > u(m_2)$ whenever $m_1 \succ_r m_2$. Inequality 5.6 is equivalent to:

$$\sum_{i \in S_a(r) \cup S_d(r)} t_i(u_i(m_1) - u_i(m_2)) > 0.$$

Note that the set of indices $S_a(r) \cup S_d(r)$ is the set of all indices i for which $r \in R_i$. Thus, the above is equivalent to

$$\sum_{i : r \in R_i} t_i(u_i(m_1) - u_i(m_2)) > 0$$

and we can split the summation so we have

$$\sum_{i : r \in R_i} t_i u_i(m_1) > \sum_{i : r \in R_i} t_i u_i(m_2).$$

The above is the same as inequality 5.5, which is equivalent to inequality 5.4, which in turn is equivalent to inequality 5.3, and since we define an additive decomposition utility function u as:

$$u(m) = \sum_{i}^{Q} t_i u_i(m)$$

we can substitute this definition of $u(m)$ for the summations in inequality 5.3 to obtain:

$$u(m_1) > u(m_2)$$

which is what we have set out to prove. □

There are two important consequences of this theorem.

Corollary 5.3.1 (Total Agreement) *A partial utility function u_i for the features S_i* must *be 1-consistent with r on S_i if $s(r) \subseteq S_i$.*

In this case we have exactly one index i such that $r \in R_i$. Thus, we have either $|S_a(r)| = 1$ and $|S_d(r)| = 0$ or the opposite, $|S_a(r)| = 0$ and $|S_d(r)| = 1$. It is easy to see in the second case that inequality 5.6 cannot be satisfied, since t_i is a positive quantity and for $i \in S_d(r)$, $u_i(m_2) - u_i(m_1)$ is also a positive quantity. Thus, if a rule's specified features overlap with only one subutility, then that subutility must be consistent with the rule.

Corollary 5.3.2 (Minimal Agreement) *In an additive decomposition utility function u consistent with C^*, each rule $r \in C^*$ must be 1-consistent with some subutility function u_i.*

By assumption, u satisfies theorem 5.4, so u must satisfy inequality 5.6. If some rule r is not 1-consistent with any subutility function u_i, then we have $|S_a(r)| = 0$ and $|S_d(r)| \geq 1$, and inequality (5.6) will not be satisfied.

Using the above theorem and corollaries, a number of algorithms are possible to produce a consistent utility function. We discuss these in the following section.

5.4 Choosing Scaling Parameters

Once we have a collection of utility-independent sets S_i, we create subutility functions as described in section 5.1. We can then choose scaling parameters t_i based on which rules disagree with each partial utility function $\hat{u}_i$. There are several possible strategies for choosing these parameters. We will first give the most simple scenario, then the most general, and then present an optimization.

5.4.1 No Conflicting Rules

Given a set of rules C^*, a mutually utility-independent partition S of $F(C)$, and corresponding relevant rule sets R_i, the sets R_i may or may not have conflicting rules. The simplest possible scenario is when there are no conflicting rules on any sets $S_i \in S$, and each u_i is 1-consistent with each rule $r \in R_i$. If this is the case, consistency of u becomes easy to achieve. Consider that inequality 5.6 becomes:

$$\sum_{i \in S_a(r)} t_i(u_i(m_1) - u_i(m_2)) > 0$$

since the set of inconsistent subutility indices, $S_d(r)$, is empty. By the definition of consistency with r, for $i \in S_a(r)$, $u_i(m_1) - u_i(m_2)$ is always a positive quantity. Thus, we are free to choose any positive values we wish for t_i. For simplicity, choose $t_i = 1, 1 \leq i \leq Q$.

Consider the following example. Take the usual partition S of $F(C)$ and additive decomposition utility function u. Then choose a rule r and suppose that for all i such that $r \in R_i$, u_i is a subutility function for S_i 1-consistent with r. We have: $u_i(m_1) > u_i(m_2)$ for each u_i, and each pair $(m_1, m_2) \models r$. We are unconstrained in our choices of t_i, since $t_i u_i(m_1) > t_i u_i(m_2)$ holds for any $t_i > 0$.

This brief inquiry lets us state the following theorem.

Theorem 5.5 (Simple Scaling Parameters) *Given a mutually utility-independent decomposition $S = \{S_1, S_2, ..., S_Q\}$ and a set of feature vector rules C^* in L^*, with R_i the set of relevant rules for S_i, and each u_i is 1-consistent with each $r \in R_i$, then*

$$u(m) = \sum_{i=1}^{Q} u_i(m) \tag{5.7}$$

is an ordinal utility function consistent with C^*.

Proof. Fix a rule $r \in C^*$. If each u_i is 1-consistent with all rules in R_i, then if $r \in R_i$, for all $(m_1, m_2) \models r$ we have $u_i(m_1) - u_i(m_2) > 0$. So we have

$$\sum_{i:r \in R_i} (u_i(m_1) - u_i(m_2)) > 0$$

and since for $r \notin R_i$ $u_i(m_1) = u_i(m_2)$ for all $(m_1, m_2) \models r$, we have:

$$\sum_{i=1}^{Q} (u_i(m_1) - u_i(m_2)) > 0$$

or

$$\sum_{i=1}^{Q} u_i(m_1) - \sum_{i=1}^{Q} u_i(m_2) > 0$$

Substituting $u(m)$ for the definition in equation 5.7 we have

$$u(m_1) - u(m_2) > 0.$$

Since we chose r arbitrarily, this holds for any $r \in C^*$. Thus, u as defined in equation 5.7 is a utility function consistent with C^*. □

As a corollary, under the same conditions, each of the graphical utility functions can be made consistent with C^*.

Corollary 5.4.1 (No Conflicting Rules) *Given a mutually utility-independent decomposition $S = \{S_1, S_2, ..., S_Q\}$ and a set of feature vector rules C^* in L^*, with R_i the set of relevant rules for S_i and R_i conflict-free for all i, then*

$$u(m) = \sum_{i=1}^{Q} u_i(m) \tag{5.8}$$

where each u_i is one of the graphical utility functions given in chapter 3 over the restricted model graph $G_i(R_i)$, is an ordinal utility function consistent with C^*.

Proof. By assumption, R_i is conflict-free, so the restricted model graph $G_i(R_i)$ is cycle-free. By theorem 5.3 each u_i is 1-consistent with each $r \in R_i$. By theorem 5.5, u is consistent with C^*. □

5.4.2 Simple Conflicting Rules

Given a mutually utility-independent partition S of $F(C)$ and corresponding relevant rule sets $R_i \subseteq C^*$ containing conflicting rules, the conflicts can interact in different ways. In some cases, the conflicts are such that they admit an elegant solution. The intuitive idea is as follows: if r disagrees with some set of subutility functions $S_d(r)$, then the weight (scaling parameters t_i) assigned to those subutility functions must be

less than the weight assigned to some subutility function u_r that agrees with r. The weight of that one agreement must be larger than the sum of all the disagreements.

The following example illustrates a configuration of relevant rule sets R_i that can cause this to occur, and how to assign the scaling parameters t_i. Given the usual partition S, relevant rule sets R_i, conflict-free rule sets $\overline{R}_i$, and subutility functions u_i 1-consistent with $\overline{R}_i$, choose some $r \in C^*$. Suppose there is exactly one subutility function 1-consistent with r, u_r, and let $S_d(r)$ be the set of indices of subutility functions u_i such that $s(r) \cap S_i \neq \emptyset$ and u_i is not 1-consistent with r. We are interested to find assignments to scaling parameters t_i such that the additive decomposition utility function u is consistent with r. To this end, we examine the values of $u_i(m_1) - u_i(m_2)$ when $(m_1, m_2) \models r$.

We have $u_r(m_1) > u_r(m_2)$; and for all $i \in S_d(r)$ we have $u_i(m_1) < u_i(m_2)$ for all pairs $(m_1, m_2) \models r$. Let $\max(u_i)$ represent the maximum value returned by u_i for any models m. Clearly, $\max(u_i) \geq u_i(m_2)$ for all $i \in S_d(r)$ and all $(m_1, m_2) \models r$. Similarly, $u_i(m_1) \geq 0$. Thus, we can bound the expression $u_i(m_1) - u_i(m_2)$ as follows:

$$u_i(m_1) - u_i(m_2) \geq -\max(u_i) \tag{5.9}$$

Recall theorem 5.4 states that for u to be consistent with r, we must have:

$$\sum_{i \in S_a(r)} t_i(u_i(m_1) - u_i(m_2)) > \sum_{j \in S_d(r)} t_j(u_j(m_2) - u_j(m_1)).$$

In this case, we have $\sum_{i \in S_a(r)} t_i(u_i(m_1) - u_i(m_2)) = t_r(u_r(m_1) - u_r(m_2)$, and for $i \in S_d(r)$, $u_i(m_2) - u_i(m_1) \leq \max(u_i)$. Substituting this in the above condition implies that u is consistent with r if

$$t_r(u_r(m_1) - u_r(m_2)) > \sum_{u_i \in S_d} t_i \max(u_i).$$

By assumption of r 1-consistent with u_r, $u_r(m_1) - u_r(m_2) \geq 1$, so u is consistent with r if

$$t_r > \sum_{u_i \in S_d} t_i \max(u_i).$$

Note that this provides an explicit condition on the value of t_r for u to be consistent with r.

We can distill the intuition behind the above example into a technique appropriate when there are few rules per utility-independent set, S_i. Given the mutually utility-independent partition S and rules C^*, we define an ordering of S called $\vec{S}$. The ordering $\vec{S} = (\vec{S}_1, \vec{S}_2, ..., \vec{S}_Q)$ and each $\vec{S}_i \in S$ has corresponding relevant rule set R_i. $\vec{S}$ is *conflict-free* if R_1 is conflict free and for all $i : 2 \leq i \leq Q$,

$$R_i \backslash \left(\bigcup_{j=1}^{i-1} R_j \right)$$

is conflict-free. That is, $\vec{S}$ is conflict free if the first set is conflict-free, and subsequent sets are either conflict-free or conflict-free disregarding rules appearing in earlier sets.

Lemma 5.4.1 (Conflict-free Ordering) *Given rules $r \in C^*$, a mutually utility-independent partition S of $F(C)$, a conflict-free ordering $\vec{S}$ of S, relevant rule sets R_i for $\vec{S}_i \in \vec{S}$, each u_i a subutility function 1-consistent with $\{R_i \setminus \cup_{j=1}^{i-1} R_j\}$, $S_a(r)$ the set of indices of subutility functions u_i 1-consistent with r, $S_d(r)$ the set of indices of subutility functions u_i such that $s(r) \cap \vec{S}_i \neq \emptyset$ and u_i is not 1-consistent with r, then, for all r, there exists $j \in S_a(r)$ such that $j < k$, for all $k \in S_d(r)$.*

Proof. Choose some rule r. r is in relevant rule sets R_i for $i \in \{S_a(r) \cup S_d(r)\}$. Let j be the first such rule set according to $\vec{S}$, that is, for all $k \neq j$, $k \in \{S_a(r) \cup S_d(r)\}$ implies $j < k$. We are given that u_j is 1-consistent with $\{R_j \setminus \cup_{l=1}^{j-1} R_l\}$, so by definition of $S_a(r)$, $j \in S_a(r)$. Since $j < k$ for all $k \in \{S_a(r) \cup S_d(r)\}$, $j < k$ for all $k \in S_d(r)$. This proves the lemma. □

Theorem 5.6 (Lexicographic Ordering) *Given rules $r \in C^*$, a mutually utility-independent partition S of $F(C)$, a conflict-free ordering $\vec{S}$ of S, relevant rule sets R_i for $\vec{S}_i \in \vec{S}$, each u_i a subutility function 1-consistent with $\{R_i \setminus \cup_{j=1}^{i-1} R_j\}$, $S_a(r)$ the set of indices of subutility functions u_i 1-consistent with r, $S_d(r)$ the set of indices of subutility functions u_i such that $s(r) \cap \vec{S}_i \neq \emptyset$ and u_i is not 1-consistent with r, then*

$$u(m) = \sum_{i}^{Q} t_i u_i(m) \tag{5.10}$$

where $t_Q = 1$ and for all $i : 1 \leq i < Q$,

$$t_i = 1 + \sum_{j=i+1}^{Q} t_i \max(u_i), \tag{5.11}$$

u is a utility function consistent with C^*.

Proof. Pick some $r \in C^*$. We are given that r is 1-consistent with subutility functions indicated by $S_a(r)$ and inconsistent with subutility functions indicated $S_d(r)$. For any pair of models $(m_1, m_2) \models r$, we must show $u(m_1) > u(m_2)$. We will do so by showing that $u(m_1) - u(m_2) > 0$. From equation 5.10, we have

$$u(m_1) - u(m_2) = \sum_{i=1}^{Q} (t_i u_i(m_1)) - \sum_{i=1}^{Q} (t_i u_i(m_2))$$

For $i \notin \{S_a(r) \cup S_d(r)\}$, we have $u_i(m_1) = u_i(m_2)$, for any models $(m_1, m_2) \models r$. Thus, we can remove these terms from our summations, and we have:

$$u(m_1) - u(m_2) = \sum_{i \in S_a(r) \cup S_d(r)} (t_i u_i(m_1)) - \sum_{i \in S_a(r) \cup S_d(r)} (t_i u_i(m_2))$$

since $S_a(r)$ is disjoint from $S_d(r)$, we can rearrange the terms in the summations:

$$u(m_1) - u(m_2) = \sum_{i \in S_a(r)} t_i(u_i(m_1) - u_i(m_2)) + \sum_{i \in S_d(r)} t_i(u_i(m_1) - u_i(m_2)).$$

By lemma 5.4.1, there exists x in $S_a(r)$, such that $x < y$ for all $y \in S_d(r)$. Since $u_i(m_1) - u_i(m_2) > 0$ for all i in $S_a(r)$, we can bound $u(m_1) - u(m_2)$ by

$$u(m_1) - u(m_2) \geq$$
$$t_x(u_x(m_1) - u_x(m_2)) + \textstyle\sum_{i \in S_d(r)} t_i(u_i(m_1) - u_i(m_2)).$$

Similarly, $u_i(m_1) - u_i(m_2) \geq -\max(u_i)$ for all i in $S_d(r)$, thus the above can be bounded by:

$$u(m_1) - u(m_2) \geq$$
$$t_x(u_x(m_1) - u_x(m_2)) - \textstyle\sum_{i \in S_d(r)} t_i \max(u_i).$$

Since u_x is 1-consistent with r, $u_x(m_1) - u_x(m_2) \geq 1$. Then it is true that

$$u(m_1) - u(m_2) \geq t_x - \sum_{i \in S_d(r)} t_i \max(u_i).$$

Since $x < y$ for all $y \in S_d(r)$, $S_d(r) \subseteq \{x+1, x+2, ..., Q\}$, it follows that

$$u(m_1) - u(m_2) \geq t_x - \sum_{i=x+1}^{Q} t_i \max(u_i).$$

Substituting the definition of t_x from equation 5.11 we have:

$$u(m_1) - u(m_2) \geq 1 + \sum_{i=x+1}^{Q} t_i \max(u_i) - \sum_{i=x+1}^{Q} t_i \max(u_i).$$

And simplifying provides that

$$u(m_1) - u(m_2) \geq 1.$$

Since we chose r arbitrarily, this derivation holds for all $r \in C^*$. Thus, u as given in equation 5.10 is consistent with C^*. This proves the lemma. $\square$

It should be clear that we can look for a conflict-free ordering $\vec{S}$ in the following manner. Given a partition S and corresponding rule sets R_i, we can check if any of the R_i are conflict-free. If so, S_i is the first set in the conflict-free ordering, $\vec{S}_1$. Let the relevant rule set for $\vec{S}_1$ be $\overline{R}_1$. We can then check if any $R_i \backslash \overline{R}_1$ are conflict-free. If so, pick one such S_i and let this be $\vec{S}_2$. We can then proceed iteratively finding further $\vec{S}_i$, until either we have found a conflict-free ordering, or we have not. If we do find a conflict-free ordering, we can then define graphical subutility functions based on that ordering, and assign scaling parameters t_i as given in equation 5.11. The result is a utility function consistent with C^* (by theorem 5.6).

If we cannot find a conflict-free ordering of the partition S, then we must use a more complicated strategy for assigning scaling parameters t_i. This is discussed in the following section.

5.4.3 Scaling Parameter Assignment by Constraint Satisfaction

When we have a mutually utility-independent partition S of $F(C)$ without a conflict-free ordering, we can construct a constraint satisfaction problem from the set of conflicts and use a general boolean SAT-solver to find a solution. The satisfiability problem will be constructed so that it represents the rule conflicts on the UI feature sets, and a solution to the constraint problem will determine which subutility functions agree and disagree with each rule. Constraint solvers are fast and quickly handle many problems with tens of thousands of terms [SLM92]. Although all general constraint satisfiability algorithms have worst-case exponential running time, we only need solve one satisfiability problem, when the utility function is constructed. The evaluation of the utility function on models of $\mathcal{L}^*$ is independent of the utility function construction cost.

Given rules C^* and a mutually utility-independent partition S of $F(C)$ with corresponding relevant rule sets R_i, we define a boolean satisfiability problem $P(C^*, S)$ in conjunctive normal form (CNF), a conjunction of disjunctions. Let Y_{ik} be a minimal set of conflicting rules on R_i, and let $Y_i = \{Y_{i1}, Y_{i2}, ..., Y_{iK}\}$ be a minimal and complete set of conflicts for S_i. Let Y be the set of all such Y_{ik}. And let

$$R_c = \bigcup_i \bigcup_k Y_{ik}$$

the set of all rules involved in any conflict. The boolean variable z_{ij} in $P(C^*, S)$ represents a pair (i, r), where $r \in C^*$ and i is an index of $S_i \in S$. The truth value of z_{ij} is interpreted as stating whether or not subutility function u_i agrees with rule $r_j \in R_c$. Let $X_j = \{l \mid S_l \cap s(r_j) \neq \emptyset\}$ denote the set of indices of UI sets that overlap with r_j.

Conjuncts in $P(C^*, S)$ are of one of two forms: those representing the subutility functions in a set X_j; and those representing conflicts between rules of a particular cycle Y_{ik}.

Definition 5.1 (Rule-Set Clauses) *Given rules C^*, a mutually utility-independent partition S of $F(C)$ with corresponding relevant rule sets R_i, R_c of all rules involved in any conflict and X_j the set of indices of UI sets overlapping with $r_j \in R_c$, then all clauses of the form*

$$\bigvee_{i \in X_j} z_{ij} \tag{5.12}$$

are the Rule-Set Clauses of $P(C^*, S)$.

The rule-set clauses of $P(C^*, S)$ represent the possible subutility functions a rule might agree with. Corollary 5.3.2 states each rule $r \in C^*$ must be 1-consistent with some subutility function u_i, accordingly in a solution to $P(C^*, S)$, one of these variables must be true.

Definition 5.2 (Conflict Clauses) *Given rules C^*, a mutually utility-independent partition S of $F(C)$ with corresponding relevant rule sets R_i, $Y_i = \{Y_{i1}, Y_{i2}, ..., Y_{iK}\}$ be a minimal and complete set of conflicts for S_i, R_c of all rules involved in any conflict and X_j the set of indices of UI sets overlapping with $r_j \in R_c$, then all clauses of the form*

$$\bigvee_{j:r_j \in Y_{ik}} \neg z_{ij} \tag{5.13}$$

are the Conflict Clauses of $P(C^*, S)$.

The conflict clauses of $P(C^*, S)$ represent a particular conflicts set of rules on a particular UI set S_i. At least one of the rules in Y_{ik} must disagree with S_i.

Combining rule-set clauses and conflict clauses, $P(C^*, S)$ is the conjunction of all such clauses. Thus

$$P(C^*, S) = \left(\bigwedge_{j=1}^{Q} \left(\bigvee_{i \in X_j} z_{ij} \right) \right) \bigwedge \left(\bigwedge_{Y_{ik} \in Y} \left(\bigvee_{j:r_j \in Y_{ik}} \neg z_{ij} \right) \right)$$

Consider an example. Suppose we have sets S_1, S_2 with the following rule conflicts. Further suppose that $r_1, r_2 \in R_1, R_2$, and that $s(r_1) = s(r_2) = S_1 \cup S_2$. Suppose that $Y_{11} = \{r_1, r_2\}$ and $Y_{21} = \{r_1, r_2\}$, that is, r_1, r_2 conflict on both of S_1, S_2. There are two rule-set clauses:

$$(z_{11} \vee z_{21}) \wedge (z_{12} \vee z_{22})$$

and two conflict clauses:

$$(\neg z_{11} \vee \neg z_{12}) \wedge (\neg z_{22} \vee \neg z_{21}).$$

Combining these gives our value for $P(C^*, S)$,

$$P(C^*, S) = (z_{11} \vee z_{21}) \wedge (z_{12} \vee z_{22}) \wedge (\neg z_{11} \vee \neg z_{12}) \wedge (\neg z_{22} \vee \neg z_{21})$$

Once we have computed $P(C^*, S)$, we can use a solver to arrive at a solution Θ. This solution Θ is an assignment of *true* or *false* to each variable z_{ij} in $P(C^*, S)$. Note that there need not be a solution Θ.

Given a solution Θ to $P(C^*, S)$, we need to translate that back into a construction for the subutility functions over S_i. Using the definition of the propositional variables z_{ij}, we look at the truth values assigned to z_{ij} in Θ. This shows which subutility functions disagree with which rules. Let Θ_f be the set of all z_{ij} assigned value *false* in Θ. We define rule sets $\overline{R}_i \subseteq R_i$ for each relevant rule set R_i, as follows. For all $S_i \in S$,

$$\overline{R}_i = R_i \backslash \{r_j \mid z_{ij} \in \Theta_f\} \tag{5.14}$$

Simply, we construct conflict-free relevant rule sets $\overline{R}_i \subseteq R_i$, from the solution to $P(C^*, S)$, then we can construct u_i 1-consistent with $\overline{R}_i$.

We show that $\overline{R}_i$ constructed according to Θ are conflict-free.

Lemma 5.4.2 (Conflict Clause Solution) *Given a solution Θ to $P(C^*, S)$, and relevant rule sets R_i for each $S_i \in S$, rule sets $\overline{R}_i \subseteq R_i$,*

$$\overline{R}_i = R_i \backslash \{r_j \mid z_{ij} \in \Theta_f\}$$

are conflict free.

Proof. We proceed with a proof by contradiction. Choose some S_i. Assume $\overline{R}_i$ has a conflict, $Y_{\overline{R}} \subseteq \overline{R}_i$. Since Y is a complete set of conflicts, $Y_{\overline{R}_i} \in Y$. Thus, there exists a conflict clause $\vee_{j:r_j \in Y_{\overline{R}_i}} \neg z_{ij}$ in $P(C^*, S)$, so one of these variables $z_{ij} \in \Theta_f$. Call this variable z_{ix}. By definition of $\overline{R}_i$, when $z_{ix} \in \Theta_f$, rule $r_x \notin \overline{R}_i$. This contradicts $r_x \in \overline{R}_i$, so we have arrived at a contradiction and must conclude that $\overline{R}_i$ is conflict-free. $\square$

Theorem 5.7 (Correctness of SAT-Formulation) *Given a mutually utility-independent partition S of $F(C)$, if $P(C^*, S)$ has no solution then the preferences C^* are not consistent with a utility function u of the form $u(m) = \sum_i t_i u_i(m)$.*

Proof. If there is no solution to $P(C^*, S)$, it means that in all possible assignments of truth values to variables, some clause is not satisfied. By definition of the propositional variables in (PC^*, S), the truth assignments correspond directly to rule-subutility function 1-consistency. The clauses in $P(C^*, S)$ are of two forms: rule-set clauses (definition 5.1), and conflict clauses (definition 5.2). By assumption, there is no solution to $P(C^*, S)$. Thus, in each possible assignment of truth values to the variables in $P(C^*, S)$, either there exists some rule-set clause that is not satisfied, or there exists some conflict clause that is not satisfied. In terms of the interpretation given to variables z_{ij}, if a rule-set clause is not satisfied then there is some rule r_j that is not 1-consistent with any subutility function u_i for each S_i. In the other case, if a conflict clause is not satisfied, then there exists some cycle Y_{ik} where each $r_j \in Y_{ik}$ is included in the set $\overline{R}_i$ (by definition of $\overline{R}_i$, equation 5.14); and thus $\overline{R}_i$ is not conflict-free.

Thus we must treat two cases. Suppose the first case holds: in any assignment of truth values to variables z_{ij} in $P(C^*, S)$, there is some rule r_j that is not 1-consistent with any subutility function u_i. Thus, by corollary 5.3.2, a utility function $u(m) = \sum_i t_i u_i(m)$ is not consistent with r_j. By definition of consistency, u is not consistent with C^*. This holds for any additive decomposition utility function u based on the mutually utility-independent partition S.

The second case is that in any assignment of truth values to the variables z_{ij}, there is some conflict clause $C_{\overline{R}_i} = \vee_{j:r_j \in Y_{ik}} \neg z_{ij}$ where all z_{ij} are true. We show that this case is equivalent to the first case. By definition of $P(C^*, S)$, for all $r_j \in C_{\overline{R}_i}$, there is a rule-set clause $C_j = \vee_{i \in X_j} z_{ij}$. If there were some $z_{xj} \in C_j$ for $x \neq i$, where $z_{xj} = \textit{true}$, we could construct a solution to $P(C^*, S)$ by making z_{ij} *false* and leaving z_{xj} *true*. But by assumption, there is no solution to $P(C^*, S)$, so we must conclude that for all $z_{xj} \in C_j$ with $x \neq i$, $z_{xj} = \textit{false}$. Therefore, we make z_{ij} *false*, and then for all $z_{xj} \in C_j$, $z_{xj} = \textit{false}$, so we now have an assignment of truth values of the first case.

This shows that if there is no solution to the SAT problem, $P(C^*, S)$, then there is no additive decomposition utility function, consistent with the stated preferences C^*. $\square$

However, we are not guaranteed that a solution to $P(C^*, S)$ leads to a consistent utility function. There is still the matter of setting the scaling parameters t_i. In doing so, we must be careful to satisfy inequality (5.6). Therefore, for each rule $r \in R_i \backslash \overline{R}_i$, we keep a list of linear inequalities, I, that must be satisfied by choosing the appropriate scaling parameters t_i. We discuss this in the following section.

5.4.4 Linear Inequalities

The solution to a satisfiability problem $P(C^*, S)$ determines how to construct conflict-free rule sets $\overline{R}_i \subseteq R_i$ for each utility-independent feature set $S_i \in S$. However, we must then set the scaling parameters, t_i, so that inequality 5.6 is satisfied. We can do so by building a list of constraints on the values that the scaling parameters may assume while still making inequality 5.6 true. These constraints are linear inequalities relating the relative importance of the parameters t_i.

Our list I of inequalities should assure that each pair (m_1, m_2) that satisfies $r \in C^*$ is consistent with the total utility function. By inequality 5.6, given rules C^*, a mutually utility-independent partition S of $F(C)$ with corresponding relevant rule sets R_i, and conflict-free $\overline{R}_i \subseteq R_i$, let $S_a(r)$ be the set of indices for which $r \in \overline{R}_i$, and let $S_d(r)$ be the set of indices for which $r \in (R_i \backslash \overline{R}_i)$, for each $(m_1, m_2) \models r$:

$$\sum_{i \in S_a(r)} t_i u_i(m_1) + \sum_{i \in S_d(r)} t_i u_i(m_1) > \sum_{i \in S_a(r)} t_i u_i(m_2) + \sum_{i \in S_d(r)} t_i u_i(m_2) \tag{5.15}$$

That is, the utility function u must assign higher utility to m_1 than to m_2. We can simplify these inequalities as follows:

$$\sum_{i \in \{S_a(r) \cup S_d(r)\}} t_i (u_i(m_1) - u_i(m_2)) > 0 \tag{5.16}$$

Note that the inequality need not contain terms for $i \notin \{S_d(r) \cup S_a(r)\}$, since, by definition of $(m_1, m_2) \models r$, we have $(m_1 \restriction S_i) = (m_2 \restriction S_i)$ and $u_i(m_1) - u_i(m_2) = 0$. Furthermore, note that many of the linear inequalities are redundant. A simple intuition tells us the important inequalities are the boundary conditions: the constraints that induce maximum or minimum values for the parameters t_i. Consider the set of model pairs (m_1, m_2) such that $(m_1, m_2) \models r$. Some pairs provide the minimum and maximum values for $u_i(m_1) - u_i(m_2)$ for some $i \in \{S_a(r) \cup S_d(r)\}$. Call $minmax(r, i)$ the set of maximum and minimum values of $u_i(m_1) - u_i(m_2)$ for model pairs satisfying r. This set has at most two elements and at least one. Let $\mathcal{M}(r)$ be the set of model pairs as follows:

$$\begin{array}{l} \mathcal{M}(r): \\ (m_1, m_2) \models r, \\ u_i(m_1) - u_i(m_2) \in minmax(r, i) \forall i \in \{S_a(r) \cup S_d(r)\} \end{array}$$

Each model pair in the set $\mathcal{M}(r)$ causes $u_i(m_1) - u_i(m_2)$ to achieve either its minimum or its maximum value on every subutility function either agreeing or disagreeing with r. For concreteness, stipulate that each $(m_1, m_2) \in \mathcal{M}(r)$ has $f_j(m_1) = 0$ and $f_j(m_2) = 0$ for $j \notin \{S_a(r) \cup S_d(r)\}$. We can now define the irredundant system of linear inequalities as an inequality of the form given in equation 5.16 for each rule r and each model pair $(m_1, m_2) \in \mathcal{M}(r)$.

Definition 5.3 (Inequalities) *Given rules C^*, a mutually utility-independent partition S of $F(C)$ with corresponding relevant rule sets R_i, conflict-free $\overline{R}_i \subseteq R_i$, $\overline{R}_i \in \overline{R}$, where $S_a(r)$ is the set of indices for which $r \in \overline{R}_i$, $S_d(r)$ is the set of indices for which $r \in (R_i \backslash \overline{R}_i)$, and each subutility function u_i is 1-consistent with all $r \in \overline{R}_i$, then $I(C^*, S, \overline{R})$ is the set of linear inequalities*

$$\sum_{i \in \{S_a(r) \cup S_d(r)\}} t_i(u_i(m_1) - u_i(m_2)) > 0$$

for all r in some $(R_i \backslash \overline{R}_i)$ and for all model pairs in $\mathcal{M}(r)$.

By definition of the *scaling parameters* t_i of an additive decomposition utility function u, $t_i > 0$ for all $1 \geq i \geq Q$. Thus, a solution to $I(C^*, S, \overline{R})$ is an assignment of positive numbers to each t_i.

Lemma 5.4.3 (Sufficiency of $I(C^*, S, \overline{R})$) *Given $I(C^*, S, \overline{R})$ and a solution to it, for all r in some $(R_i \backslash \overline{R}_i)$ and for all model pairs $(m_1, m_2) \models r$,*

$$\sum_{i \in \{S_a(r) \cup S_d(r)\}} t_i(u_i(m_1) - u_i(m_2)) > 0$$

Proof. Let x_i be the maximum value of $u_i(m_1) - u_i(m_2)$ and y_i be the minimum value of $u_i(m_1) - u_i(m_2)$, then the following inequalities are members of $I(C^*, S, \overline{R})$:

$$\begin{array}{l} x_i t_i + \sum_{j=1, j \neq i}^{Q} t_j x_j > 0 \\ x_i t_i + \sum_{j=1, j \neq i}^{Q} t_j y_j > 0 \\ y_i t_i + \sum_{j=1, j \neq i}^{Q} t_j x_j > 0 \\ y_i t_i + \sum_{j=1, j \neq i}^{Q} t_j y_j > 0. \end{array}$$

Since we have a solution to $I(C^*, S, \overline{R})$, we have an assignment of values to $t_1, ..., t_Q$ such that the above hold. Clearly the following hold for any $(u_i(m_1) - u_i(m_2)) : x \geq z \geq y$:

$$\begin{array}{l} (u_i(m_1) - u_i(m_2))t_i + \sum_{j=1, j \neq i}^{Q} t_j x_j > 0 \\ (u_i(m_1) - u_i(m_2))t_i + \sum_{j=1, j \neq i}^{Q} t_j y_j > 0. \end{array}$$

Since this holds for any choice of $i : 1 \geq i \geq Q$, this proves the lemma. □

Theorem 5.8 (Inequalities) *If the system of linear inequalities, $I(C^*, S, \overline{R})$, has a solution, this solution corresponds to a utility function u consistent with C^*.*

Proof. Each subutility function, u_i, $1 \leq i \leq Q$, is fixed. By lemma 5.4.3, for a given rule r in some $(R_i \backslash \overline{R}_i)$, we are assured that if $(m_1, m_2) \models r$ then $\sum_i t_i u_i(m_1) > \sum_i t_i u_i(m_2)$, for all $i \in \{S_a(r) \cup S_d(r)\}$. Since $S_a(r)$ and $S_d(r)$ are disjoint, we can rearrange the summations and obtain:

$$\sum_{i \in S_a(r)} t_i(u_i(m_1) - u_i(m_2)) > \sum_{j \in S_d(r)} t_j(u_j(m_2) - u_j(m_1)).$$

Then by theorem 5.4, u is consistent with r.

If r is not in any $(R_i \backslash \overline{R}_i)$, then for all i such that $r \in R_i$, u_i is 1-consistent with r. For j such that $r \notin R_j$, $(m_1, m_2) \models r$ implies that $(m_1 \restriction S_j) = (m_2 \restriction S_j)$, so all u_j are 1-consistent with r. Since all subutility functions are 1-consistent with r, we have:

$$\sum_{i}^{Q} t_i(u_i(m_1) - u_i(m_2)) > 0$$

and u is consistent with r. Thus, u is consistent with all r in some $(R_i \backslash \overline{R}_i)$, and all r not in some $(R_i \backslash \overline{R}_i)$, so u is consistent with all $r \in C^*$. This proves the lemma. □

We can solve the system of linear inequalities $I(C^*, S, \overline{R})$ using any linear inequality solver. We note that it is possible to phrase this as a linear programming problem, and use any of a number of popular linear programming techniques to find scaling parameters t_i. If we rewrite the inequalities in the form

$$\sum_{i=1}^{Q} t_i(u_i(m_1) - u_i(m_2)) \geq \epsilon \tag{5.17}$$

where ϵ is small and positive, then we can solve this system of inequalities by optimizing the following linear program. The task of the linear program is to minimize a new variable, t_0, subject to the following constraints:

$$t_0 + \sum_{i=1}^{Q} t_i(u_i(m_1) - u_i(m_2)) \geq \epsilon$$

for each (m_1, m_2) as described above, and the constraint that $t_0 \geq 0$ [Chv83]. We must be careful that ϵ is sufficiently smaller than 1, or we may not find solutions that we could otherwise discover. Given ϵ small enough, this linear program has the property that it always has a solution, by making t_0 very large. However, only when the optimal value of t_0 is 0, is there a solution to the original system of linear inequalities.

We can bound the total number of linear inequalities in $I(C^*, S, \overline{R})$ as follows. Given that $r \in (R_i \backslash \overline{R}_i)$ for some i, it contributes at most

$$2^{|S_a(r) \cup S_d(r)|} \tag{5.18}$$

inequalities to the set of inequalities. This is a simple combinatoric argument. r contributes exactly one inequality for each combination of minimum and maximum

value on each set S_i. For each set a rule can have both a minimum and a maximum (they may not be distinct). This suffices to give us the bound stated in condition 5.18.

We have mentioned that $I(C^*, S, \overline{R})$ might have no solution. we discuss this situation in the following section.

5.5 Assurance of Linearity

In this section we discuss how to deal with nonlinear tradeoffs between satisfaction of one feature set and satisfaction of another feature set.

The basic problem is as follows. A basic result from utility theory [KR76] states mutual utility independence implies the existence of a utility function defined as an additive combination of subutility functions (given in equation 4.2). However, this result assumes that the subutility functions are non-linear. And, in fact, closely tuned to one another in such a manner that the term "independent" is almost ironic. Our subutility functions are actually linear (shown below). Thus, there are certain types of preferences, representable with *ceteris paribus* preference statements, that our utility decomposition cannot represent.

We call this problem the *non-linear tradeoff ratio* problem, because we view it as a problem of taking two linear functions, u_i and u_j, and specifying a *tradeoff ratio* between them. A tradeoff ratio is the amount of satisfaction of S_i a decision maker is willing to sacrifice in order to gain an improvement of one unit of satisfaction of S_j. It may be that a decision maker will always sacrifice three units of S_i to gain one unit of S_j; this implies a utility function of the form

$$u(m) = 3 * u_i(m) + u_j(m).$$

On the other hand, it may be the case that the amount of S_i a decision maker will sacrifice for one unit of S_j depends on the amount of S_i. For example, in economics it is common to model the utility of money as proportional to the logarithm of the amount of the money. It is intuitive to think that the more money a person has (satisfaction of S_i), the more they are willing to spend \$100 on a meal (satisfaction of S_j).

It is perhaps remarkable that qualitative *ceteris paribus* preference statements can concisely imply particular non-linear tradeoffs between subutility functions. We give here a simple example of such preferences. Suppose set $S_i = \{f_1, f_2\}$ and set $S_j = \{f_3\}$ are utility independent. The following preferences

$$\begin{aligned} 010 &\succ 000 \\ 100 &\succ 010 \\ 001 &\succ 000 \end{aligned}$$

imply that $10 \succ 01 \succ 00$ on S_i, and on S_j we have $1 \succ 0$. To establish a trade-off ratio between features S_i and features S_j, we can state

$$010 \succ 001 \tag{5.19}$$

implying that satisfaction of S_i is more important than satisfaction of S_j (it pairs a desirable value of S_i with an undesirable value of S_j and states this is preferred to the opposite.) However, it is possible to present several other preferences that specify different satisfaction ratios, for different levels of satisfaction of S_i and S_j. Consider

$$011 \succ 100 \tag{5.20}$$

where we pair a desirable value for S_i with an undesirable value for S_j and compare it to a desirable value for S_j combined with an undesirable value for S_i. This specifies that increasing S_j from 0 to 1 is more important than increasing S_i from 01 to 10. Asserting the preferences in both equation 5.19 and equation 5.20 at the same time is consistent, but it is not representable by our utility functions. This is because we have a linear combination of linear subutility functions.

The graphical subutility functions are linear in the following sense. For the minimizing graphical utility functions using strict preferences, for each integer $n \in \{1, 2, \ldots \max(u_i)\}$, there exists some model m such that $u_i(m) = n$. The other graphical utility functions have similar problems (briefly, that if $m_1 \succ m_2$ the size of $(u_i(m_1) - u_i(m_2))$ is arbitrary.) There is no way of specifying, for example, that some model is "much more preferred" to another model, or that some model is "ten times as desirable" as another model. This is a natural shortcoming based on our formalism of qualitative *ceteris paribus* preference statements.

This problem of non-linear tradeoff ratios between feature sets S_i, S_j, can be fixed by defining $S_k = S_i \cup S_j$ as a new utility-independent set in, and removing S_i and S_j from, the partition of UI sets S. A simple way to determine when this is necessary is to check each preference to see if it:

- specifies values over at least two UI feature sets S_i, S_j, and
- specifies $(m_1' \wedge m_2'') \succ (m_1'' \wedge m_2')$ where $u_i(m_1') < u_i(m_1'')$ and $u_j(m_2') < u_j(m_2'')$.

This involves calculating several subutility values u_i for each preference statement. This is no more costly computationally than other aspects of our algorithm. Calculating u_i is potentially $O(2^{|S_i|})$, and we might have to calculate this $O(2^{|S_i \setminus s(r)|})$ times for each rule r. The standard assumption is our algorithm is efficient if $|S_i|$ is of a reasonable size. When $|S_i|$ is of size $\log N$, where N is the total number of features, then this step is on the order $O(|C^*| * N^2)$, where $|C^*|$ is the number of input preferences.

Thus, without severely impacting performance of utility construction, we can check rules to determine when non-linearity occurs. By joining UI sets, the computation of $u(m)$ becomes less efficient, but is able to represent more complicated preferences. Concurrently with our discovery of this problem, Boutilier *et. al.*, [BBB01] encounters exactly the same problem and proposes the same solution. We leave it to future work to find more efficient ways of solving the non-linear tradeoff ratio problem.

5.6 Complexity

The running time of the SAT algorithm and the Linear Inequality solver steps can be prohibitive. We consider here the time taken for both steps, for some different preference structures. We will discuss the following cases: when each feature $f_i \in F(C)$ is UI of every other feature, when non are UI of any other feature, when each UI feature set is of size equal to the log of the number of features in $F(C)$. There are two different complexity questions to ask. One is the time and space required to construct u. The other is the time and space required to evaluate $u(m)$ on a particular model m.

In the following, we assume that $F(C) = N$ and that the number of utility-independent sets in the partition S is k. The time required for the SAT solver depends on both the number of clauses and the number of boolean variables appearing in the SAT problem. The number of boolean variables is bounded by $|C^*| * k$, and the number of clauses depends on the number of conflicts among rules, which we do not have a good bound upon. For a rule r the number of conflicts can be estimated by $|s(r)| * \frac{N}{k}$. The SAT clauses are of two forms. The first is to ensure that every rule agrees with some UI set. Thus, these clauses have one variable per UI set S_i such that $s(r) \cap S_i \neq \emptyset$. We upper-bound the quantity $|\{S_i \mid s(r) \cap S_i \neq \emptyset\}|$ by k. The second is to ensure at least one rule of a conflict is ignored. These clauses have a number of boolean variables equal to the number of rules involved in the conflict. The number of rules in each conflict is bounded by $|C^*|$ and by $2^{|S_i|}$, since there can only be $2^{|S_i|}$ different rules per utility-independent set. In pathological cases, the number of conflicts could be as high as $|C^*|!$, if every rule conflicts with every other set of rules. However, it is difficult to imagine such scenarios. Particularly, we assume the preferences C^* are consistent, and will therefore have a low number of conflicts. Clearly, we have no good bound on the number of conflicts. However, since there is no known polynomial-time algorithm for the satisfiability of our CNF formula, the SAT portion of the algorithm will have worst-case exponential time cost unless we can bound the number of conflicts and rules by $O(\log(|F(C)|))$. Simple counterexamples exist that have $O(|F(C)|)$ conflicts. Thus, the efficiency of our algorithm depends on heuristic methods to both 1) reduce the number of conflicts and 2) quickly solve SAT problems. We have mentioned earlier the use of partial lexicographic orderings to set some of the scaling parameters t_i, and the use of fast randomized SAT-solvers to quickly solve our constraint problem.

We have already shown that the number of linear inequalities contributed by a rule is less than $2^{|S_a(r) \cup S_d(r)|}$. More precisely, for each rule, there is one inequality for each combination of the maximum and minimum of $u_i(m_1) - u_i(m_2)$ for each subutility function. For the special case where i is such that $|S_i \; s(r)| = 0$ i contributes exactly one term, since the minimum and maximum are the same. Thus each rule contributes

$$\prod_{i \in S_a(r) \cup S_d(r)} \left\{ \begin{array}{lll} 2 & if & |s_i \backslash s(r)| \geq 1 \\ 1 & if & |s_i \backslash s(r)| = 0 \end{array} \right\}$$

First we consider two extreme cases. Suppose that every feature is found to be utility dependent upon every other feature, so that the partition S is $S_1 = \{f_1, f_2, ..., f_N\}$,

and $k = 1$. Then there can be no conflicts, since every rule must agree with u_1. Thus, there are no SAT clauses and no linear inequalities needed. However, the evaluation time of $u(m)$ is the time taken to traverse a model graph of all N features. We argued in chapter 3 that this is always worst-case exponential in N.

On the other hand, suppose that every feature is found to be utility independent of every other feature, and we have $k = N$. The number of inequalities per rule is exactly one, we have the total number of inequalities equal to $|C^*|$. The evaluation time of $u(m)$ is order N, since we evaluate N graphs of size 2.

Now suppose we have a more balanced utility decomposition. Suppose each utility-independent set is of size $\log N$. Then $k = N/(\log N)$. The number of inequalities per rule is less than $2^{|S_a(r) \cup S_d(r)|}$, and since $|S_a(r) \cup S_d(r)| \leq N/\log N$, this is less than $2^{N/\log N}$. Thus the total number of inequalities is less than $|C^*| * 2^{N/\log N}$. The evaluation of $u(m)$ requires traversing k graphs each of size $2^{\log N}$. Thus evaluating $u(m)$ is order $N^2/(\log N)$.

If the size of the utility-independent sets gets larger than $\log N$, then we will not have polynomial time evaluation of $u(m)$, In addition, the number of possible linear inequalities goes up exponentially with the number of utility-independent sets each rule overlaps with.

Thus, given a favorable MUI decomposition and few numbers of conflicts, we can create a utility function in time $O(\max_r(2^{|S_a(r) \cup S_d(r)|}) + |C^*|^2 + \max_i 2^{|S_i|})$. The first term is for the number of inequalities, the second term is for the utility independence computation, and the third term is for actually constructing the subutility functions u_i. We can then evaluate $u(m)$ in time $O(\max_i 2^{|S_i|})$.

5.7 Extending with Weak Preferences

The techniques outlined above work when the input set of preferences are entirely strict, and we can assume that no two models must have the same utility in a consistent utility function. The graphical utility function can include cycles, but using GUFs with cycles as subutility functions within the presented framework can create complications.

The use of strict preferences in the feature vector language is motivated by the desires to avoid degenerate preferences and to use our heuristics, such as a lexicographic ordering. If all preferences are weak, then there is a (class of) trivial utility functions consistent with the weak preferences in which every model receives the same utility. In the linear inequality step, it is the desire to avoid the trivial solution $t_i = 0$, $1 \leq i \leq Q$, that requires us to use linear inequalities of the form:

$$\sum_{i \in \{S_a(r) \cup S_d(r)\}} t_i u_i(m_1) - \sum_{i \in \{S_a(r) \cup S_d(r)\}} t_i u_i(m_2) \geq \epsilon \tag{5.21}$$

rather than of the form:

$$\sum_{i \in \{S_a(r) \cup S_d(r)\}} t_i u_i(m_1) - \sum_{i \in \{S_a(r) \cup S_d(r)\}} t_i u_i(m_2) \geq 0 \tag{5.22}$$

However, we could alter this to allow specific weak preference rules to generate linear inequalities of the form in equation 5.22. If there is a mix of weak and strict preferences generating a mix of inequalities of the forms in equations 5.21 and 5.22, then the system of linear inequalities is unlikely, in general, to have the trivial solution $t_i = 0$ for all t_i.

Allowing weak preferences makes us unable to use the lexicographic ordering system presented in section 5.4.2 on UI sets that overlap a weak preference rule. That is, if we have a weak preference rule r, we cannot use the lexicographic ordering on any set S_i if $S_i \cap s(r) \neq \emptyset$.

5.8 Summary

In this chapter we have presented a number of methods of assigning scaling parameters to the utility function $u(m) = \sum_i t_i u_i(m)$ and choosing which subutility functions should be 1-consistent with which rules. Some methods are faster than others. However, the methods presented here are not complete - they may fail to produce a utility function from *ceteris paribus* preferences C^*. When the methods fail, it is always possible to join partition elements $S_i, S_j \in S$ together to perform constraint satisfaction and linear programming on a smaller problem. Work in progress addresses optimal choice of which feature sets to join together.

We note that the combination of a lexicographic ordering over a subset of the features is a strong heuristic. It is possible to arrange as many as possible of the features into a lexicographic ordering, then use the SAT-conflict resolution system and linear inequalities to assign the remaining utility-independent sets and scaling parameters. However, this method remains to be analyzed. We should not assume this method is entirely correct, it places unnecessary constraints on the linear inequalities, and make finding a correct solution tricky.

Chapter 6

A Complete Algorithm

In the preceding, we have described several parts of an algorithm for computing with qualitative *ceteris paribus* preference statements. This has given us enough tools to accomplish the goal of the thesis, to find a utility function u such that $p(u) \in [C]$ for a set of *ceteris paribus* preferences C. In the following, we reiterate how these techniques can be woven together into a complete algorithm.

Our algorithm has several steps. Firstly, the algorithm takes as input a set C of *ceteris paribus* preference statements in the language $\mathcal{L}$, presented in section 1.2, and a type of graphical utility function g (from chapter 3). These statements are logical statements over a space of features F. The algorithm outputs a utility function u consistent with C. The steps of the algorithm are as follows:

1. Compute the set of relevant features $F(C)$, that is the support of C.
2. Compute $C^* = \sigma(C)$. Translation $\sigma(C)$ is defined in section 2.2.1 and converts a set of preferences C in language $\mathcal{L}$ to a C^* of preferences in the language $\mathcal{L}^*(F(C))$.
3. Compute a partition $S = \{S_1, S_2, ..., S_Q\}$ of $F(C)$ into mutually utility-independent feature sets, by the pairwise comparison of rules in C^*, as discussed in section 4.3.
4. Construct relevant rule sets R_i for each S_i. R_i is defined to be the set of rules $r \in C^*$ such that $s(r) \cap S_i \neq \emptyset$.
5. Construct the restricted model graph $G_i(R_i)$ for each set of rules R_i.
6. Compute a minimal and complete set Y_i of cycles Y_{ik} for each graph $G_i(R_i)$ such that each cycle Y_{ik} is a set of rules from R_i.
7. Construct the satisfiability problem $P(C^*, S)$ from all cycles Y_{ik}.
8. Find a solution Θ to $P(C^*, S)$.
9. Choose conflict-free rule sets $\overline{R}_i \subseteq R_i$ sets using solution Θ of $P(C^*, S)$, as given in equation 5.2.

10. Construct cycle-free restricted model graphs $G'_i(\overline{R}_i)$

11. Define each subutility function u_i to be the graphical subutility function of type g based on $G'_i(\overline{R}_i)$.

12. Use definition 5.3 to construct $I(C^*, S, \overline{R})$, a system of linear inequalities relating the parameters t_i.

13. Solve $I(C^*, S, \overline{R})$ for each t_i using linear programming.

    (a) If $I(C^*, S, \overline{R})$ has a solution, pick a solution, and use the solution's values for u_i and t_i to construct and output a utility function $u(m) = \sum_i t_i u_i(m)$.

    (b) If $I(C^*, S, \overline{R})$ has no solution, construct u to be the graphical utility function of type g based on $G(C^*)$, and output u.

This algorithm is correct and complete, further it is possible to prove as much by reference to the previous theorems presented in the thesis. We present this here.

Theorem 6.1 (Completeness) *Given a set of* ceteris paribus *preferences C, the above algorithm produces a utility function u such that $p(u) \in [C]$.*

Proof. Theorem 2.1 shows that $C^* = \sigma(C)$ represents the same preference as C. Therefore, any utility function u such that $p(u) \in [C^*]$ satisfies $p(u) \in [C]$. By theorem 4.2, the partition S is a partition of $F(C)$ into mutually utility-independent feature subsets. Lemma 5.2.1 implies that a minimal and complete set of conflicts Y_i for rule set R_i can be computed by performing cycle-detection on the restricted model graph $G_i(R_i)$. By definitions 5.1 and 5.2, S, R, Y, are enough to construct a satisfiability problem $P(C^*, S)$. By lemma 5.4.2, the solution Θ to $P(C^*, S)$ allows choosing of $\overline{R}_i$ conflict-free. By Lemma 5.2.1, each restricted model graph $G_i(\overline{R}_i)$ is cycle-free. It is then possible to build and solve a set of linear inequalities $I(C^*, S, \overline{R})$, as given in definition 5.3. If $I(C^*, S, \overline{R})$ has a solution, then this solution provides values for each t_i. By theorem 5.8 $u(m) = \sum_i t_i u_i(m)$ is a utility function consistent with C^*. If $I(C^*, S, \overline{R})$ has no solution, then the algorithm outputs u the graphical utility function of type g based on $G(C^*)$. By theorems 3.1, 3.2, 3.3 and 3.4, this is a utility function consistent with C^*. □

Chapter 7

A Detailed Example

We illustrate the above methodology with an example. Suppose a system administrator sets preferences about the relative performance and security of different processes on a digital animation company's computers. The system administrator identifies the following processes running on the company's computers:

1. login
2. mail
3. web server
4. animation
5. database
6. backup
7. SETI at home

The computers do the normal tasks of authenticating the users of the system, serving electronic mail, displaying a promotional website about the company's work, actually doing the animation computations, keeping a database of business records, backup of the files and data, and, when the machines aren't otherwise busy, the company aids astronomers in the search for extraterrestrial life (seti). For the purpose of this illustration we assume each of these processes can be optimized either for security or performance but not for both. The system administrator then specifies the following preferences over the processes, shown in figure 7-1. In addition, there are two situations that effect the preferences over process performance and security. These are an upcoming payday, which requires massive database accesses to compute payrolls, and the threat of a mail-worm virus, which requires massive mail security to prevent damage to the company's system. The preferences among different processes and involving these two conditions are given in figure 7-2.

Since we assume that a given computer process can be optimized either for security or for performance, we can combine each process into a single logical feature. We define the features $\{f_1, f_2, ..., f_9\}$ as follows:

Service	*Service-specific preferences*
login	login_security $\succ$ login_performance
mail	mail_security $\succ$ mail_performance
web	web_performance $\succ$ web_security
animation	animation_performance $\succ$ animation_security
backup	backup_performance $\succ$ backup_security
seti	seti_performance $\succ$ seti_security

Figure 7-1: Pure service preferences

Cross-service preferences
login_security $\succ$ web_security
web_performance $\succ$ seti_performance
animation_performance $\succ$ seti_performance
animation_performance $\succ$ mail_performance
backup_performance $\succ$ seti_performance
payday $\wedge$ db_security $\succ$ payday $\wedge$ db_performance
payday $\wedge$ db_performance $\succ$ payday $\wedge$ animation_performance
$\neg$ payday $\wedge$ animation_performance $\succ$ $\neg$ payday $\wedge$ db_performance
worm $\wedge$ mail_security $\succ$ worm $\wedge$ login_security
$\neg$ worm $\wedge$ login_security $\succ$ $\neg$ worm $\wedge$ mail_security

Figure 7-2: Mixed service preferences

Preference	*logical preference*
login_security $\succ$ login_performance	$\neg f_1 \succ f_1$
mail_security $\succ$ mail_performance	$\neg f_2 \succ f_2$
web_performance $\succ$ web_security	$f_3 \succ \neg f_3$
a_performance $\succ$ a_security	$f_4 \succ \neg f_4$
payday $\wedge$ db_security $\succ$ payday $\wedge$ db_performance	$f_8 \wedge \neg f_5 \succ f_8 \wedge f_5$
backup_performance $\succ$ backup_security	$f_6 \succ \neg f_6$
seti_performance $\succ$ seti_security	$f_7 \succ \neg f_7$
login_security $\succ$ web_security	$\neg f_1 \succ \neg f_3$
web_performance $\succ$ seti_performance	$f_3 \succ f_7$
a_performance $\succ$ seti_performance	$f_4 \succ f_7$
a_performance $\succ$ mail_performance	$f_4 \succ f_2$
backup_performance $\succ$ seti_performance	$f_6 \succ f_7$
payday $\wedge$ db_perf $\succ$ payday $\wedge$ a_perf	$f_8 \wedge f_4 \succ f_8 \wedge f_5$
$\neg$ payday $\wedge$ a_perf $\succ$ $\neg$ payday $\wedge$ db_perf	$\neg f_8 \wedge f_5 \succ \neg f_8 \wedge f_4$
worm $\wedge$ mail_security $\succ$ worm $\wedge$ login_security	$f_9 \wedge \neg f_2 \succ f_9 \wedge \neg f_1$
$\neg$ worm $\wedge$ login_security $\succ$ $\neg$ worm $\wedge$ mail_security	$\neg f_9 \wedge \neg f_1 \succ \neg f_9 \wedge \neg f_2$

Figure 7-3: *Ceteris paribus* preferences

$f_1 =$ *true* iff login_performance iff $\neg$ login_security
$f_2 =$ *true* iff mail_performance iff $\neg$ mail_security
$f_3 =$ *true* iff web_performance iff $\neg$ web_security
$f_4 =$ *true* iff animation_performance iff $\neg$ animation_security
$f_5 =$ *true* iff database_performance iff $\neg$ database_security
$f_6 =$ *true* iff backup_performance iff $\neg$ backup_security
$f_7 =$ *true* iff seti_performance iff $\neg$ seti_security
$f_8 =$ *true* iff payday iff $\neg$ payday
$f_9 =$ *true* iff mail_worm iff $\neg$ mail_worm

We take the preferences expressed in figures 7-1 and 7-2 and translate them into *ceteris paribus* preferences in the formalism of section 1.2, and present the result in figure 7-3.

Then we convert the *ceteris paribus* preferences in figure 7-3 into preferences in $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$, where $\mathcal{V} = (f_1, f_2, ..., f_9)$. The preferences converted to $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$ are presented in figure 7-4.

The next step is to compute a partition of $F(C)$ into mutually utility-independent features. We check each pair of rules in C^*. Rules #13 and #14 establish dependence between features f_4, f_5, and f_8. Similarly, rules #15 and #16 establish the dependence

Rule	Ceteris paribus *preference, C*	$\mathcal{L}_{\mathrm{r}}(\mathcal{V})$ *preference,* C^*
r_1	$\neg f_1 \succ f_1$	$0******* \succ 1*******$
r_2	$\neg f_2 \succ f_2$	$*0****** \succ *1******$
r_3	$f_3 \succ \neg f_3$	$**1***** \succ **0*****$
r_4	$f_4 \succ \neg f_4$	$***1**** \succ ***0****$
r_5	$f_8 \wedge \neg f_5 \succ f_8 \wedge f_5$	$****0**1* \succ ****1**1*$
r_6	$f_6 \succ \neg f_6$	$*****1*** \succ *****0***$
r_7	$f_7 \succ \neg f_7$	$******1** \succ ******0**$
r_8	$\neg f_1 \succ \neg f_3$	$0*1***** \succ 1*0*****$
r_9	$f_3 \succ f_7$	$**1***0** \succ **0***1**$
r_{10}	$f_4 \succ f_7$	$***1**0** \succ ***0**1**$
r_{11}	$f_4 \succ f_2$	$*0*1**** \succ *1*0****$
r_{12}	$f_6 \succ f_7$	$*****10** \succ *****01**$
r_{13}	$f_8 \wedge f_4 \succ f_8 \wedge f_5$	$***10**1* \succ ***01**1*$
r_{14}	$\neg f_8 \wedge f_5 \succ \neg f_8 \wedge f_4$	$***01**0* \succ ***10**0*$
r_{15}	$f_9 \wedge \neg f_2 \succ f_9 \wedge \neg f_1$	$10******1 \succ 01******1$
r_{16}	$\neg f_9 \wedge \neg f_1 \succ \neg f_9 \wedge \neg f_2$	$01******0 \succ 10******0$

Figure 7-4: *Ceteris paribus* preference rule definitions for rules $r_1, ..., r_{16}$, preferences shown in $\mathcal{L}_{\mathrm{r}}$ and $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$

of f_1, f_2, and f_9. So we have our partition S of $F(C)$ as follows:

$$\begin{aligned} S &= \{S_1, S_2, S_3, S_4, S_5\} \\ S_1 &= \{f_1, f_2, f_9\} \\ S_2 &= \{f_3\} \\ S_3 &= \{f_4, f_5, f_8\} \\ S_4 &= \{f_6\} \\ S_5 &= \{f_7\} \end{aligned} \tag{7.1}$$

The corresponding relevant rule-sets R_i for each S_i are given below, where each rule r_j is defined in figure 7-4:

$$\begin{aligned} R_1 &= \{r_1, r_2, r_8, r_{11}, r_{15}, r_{16}\} \\ R_2 &= \{r_3, r_8, r_9\} \\ R_3 &= \{r_4, r_5, r_{10}, r_{11}, r_{13}, r_{14}\} \\ R_4 &= \{r_6, r_{12}\} \\ R_5 &= \{r_7, r_9, r_{10}, r_{12}\} \end{aligned} \tag{7.2}$$

Now we construct restricted model graphs $G_i(R_i)$ for each set $S_i \in S$. We present the graph $G_1(R_1)$ in figure 7-5. The nodes in the graph represent models, accordingly are labelled with the letter "M", and subscripted with the decimal integer representation of their binary truth assignments to the features in S_1, with low-order bits corresponding to higher numbered features. For example, "M_3" represents the model 011 over features $\{f_1, f_2, f_9\}$, where $f_1 = \mathit{false}$, $f_2 = \mathit{true}$, and $f_9 = \mathit{true}$. The restricted rules for S_1 are as follows:

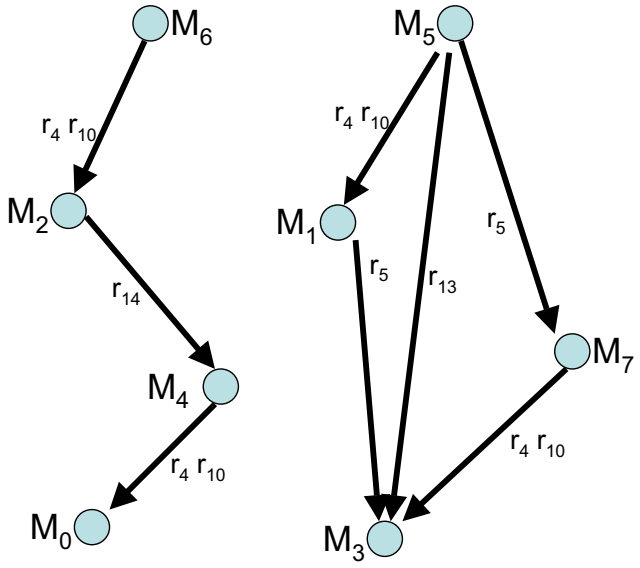

Figure 7-5: Restricted model graph $G_1(R_1)$ for the set $S_1 = \{f_1, f_2, f_9\}$, R_1 is defined in equation 7.2. Edges between models are labelled with the rule(s) that cause the edge.

Rule	*Restriction to* $S_1 = \{f_1, f_2, f_9\}$
r_1	$0** \succ 1**$
r_2	$*0* \succ *1*$
r_8	$0** \succ 1**$
r_{11}	$*0* \succ *1*$
r_{15}	$101 \succ 011$
r_{16}	$010 \succ 100$

The restricted model graph $G_2(R_2)$ is shown in figure 7-6. The restricted rules for S_2 are shown below:

Rule	*Restriction to* $S_2 = \{f_3\}$
r_3	$1 \succ 0$
r_8	$1 \succ 0$
r_9	$1 \succ 0$

It is merely a coincidence that two of these rules are redundant.

The restricted model graph $G_3(R_3)$ is shown in figure 7-7, and the rules restricted to $S_3 = \{f_4, f_5, f_8\}$ are shown below:

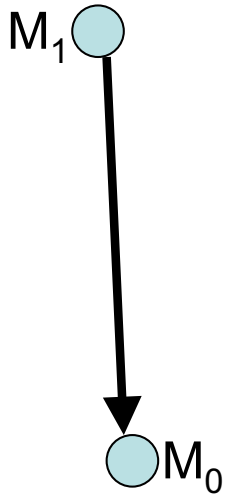

Figure 7-6: Restricted model graph $G_2(R_2)$ for the set $S_2 = \{f_3\}$, R_2 is defined in equation 7.2. This graph is, coincidentally, identical to $G_4(R_4)$ for the set $S_4 = \{f_6\}$.

Rule	*Restriction to* $S_3 = \{f_4, f_5, f_8\}$
r_4	$1^{**} \succ 0^{**}$
r_5	$^*01 \succ ^*11$
r_{10}	$1^{**} \succ 0^{**}$
r_{11}	$^*1^* \succ ^*0^*$
r_{12}	$101 \succ 011$
r_{13}	$010 \succ 100$

The restricted model graph $G_r(R_4)$ is identical to the graph $G_2(R_2)$. However, the rules of R_4 restricted to $S_4 = \{f_6\}$ are somewhat different, and shown here:

Rule	*Restriction to* $S_4 = \{f_6\}$
r_6	$1 \succ 0$
r_{12}	$1 \succ 0$

The restricted model graph $G_5(R_5)$ is shown in figure 7-8. The rules R_5 restricted to $S_5 = \{f_7\}$ are shown below:

Rule	*Restriction to* $S_5 = \{f_7\}$
r_7	$1 \succ 0$
r_9	$0 \succ 1$
r_{10}	$0 \succ 1$
r_{12}	$0 \succ 1$

From the model graphs $G_1(R_1), ..., G_5(R_5)$, we compile sets of rules that cause cycles in each restricted model graph. These conflicts between rules will be resolved via a satisfiability problem $P(C^*, S)$. As is evident from the figures, $G_1(R_1), G_2(R_2), G_4(R_4)$ have no cycles. However, $G_3(R_3)$ has two cycles. We annotate the edges of $G_3(R_3)$

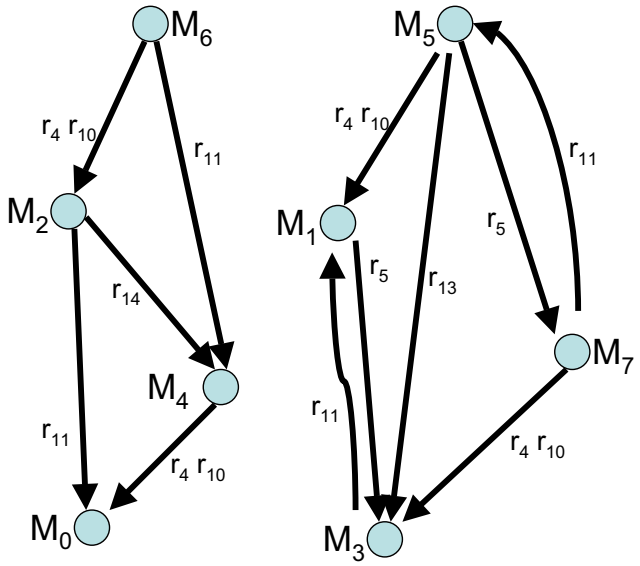

Figure 7-7: Restricted model graph $G_3(R_3)$ for the set $S_3 = \{f_4, f_5, f_8\}$, R_3 is defined in equation 7.2. Edges between models are labelled with the rule(s) that cause the edge.

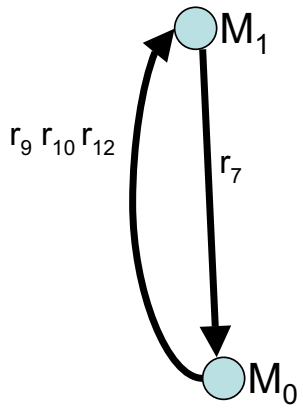

Figure 7-8: Restricted model graph $G_5(R_5)$ for the set $S_5 = \{f_7\}$, R_5 is defined in equation 7.2. Edges between models are labelled with the rule(s) that cause the edge.

with the rule or rules that imply the edge, so that we can easily determine from the graph which rules cause cycles. Therefore, set S_3 has the following conflict:

$$Y_{3,1} = \{r_5, r_{11}\} \tag{7.3}$$

Note that the two cycles are redundant. The rules r_5 and r_{11} cause two separate cycles, but this translates into only one set of conflicting rules. On the set S_5 we have cycles:

$$\begin{aligned} Y_{5,1} &= \{r_7, r_9\} \\ Y_{5,2} &= \{r_7, r_{10}\} \\ Y_{5,3} &= \{r_7, r_{12}\} \end{aligned} \tag{7.4}$$

The rule r_7 conflicts with each of the other rules in R_5.

We now build a satisfiability problem, $P(C^*, S)$, to resolve the cycles found in the restricted model graphs. We use the boolean variable z_{ij} to represent the rule r_j restricted to set S_i. We first generate conflict clauses for each cycle. These are as follows:

Cycle	*CNF-clause*
$Y_{3,1}$	$\neg z_{3,5} \vee \neg z_{3,11}$
$Y_{5,1}$	$\neg z_{5,7} \vee \neg z_{5,9}$
$Y_{5,2}$	$\neg z_{5,7} \vee \neg z_{5,10}$
$Y_{5,3}$	$\neg z_{5,7} \vee \neg z_{5,12}$

We then generate rule-set clauses for each rule r involved in any conflict. These are as follows:

Rule	*CNF-clause*
r_5	$z_{3,5}$
r_7	$z_{5,7}$
r_9	$z_{2,9} \vee z_{5,9}$
r_{10}	$z_{3,10} \vee z_{5,10}$
r_{11}	$z_{1,11} \vee z_{3,11}$
r_{12}	$z_{3,12} \vee z_{4,12} \vee z_{5,12}$

Each clause must be satisfied, or we know that there is no additive decomposition utility function consistent with the input preferences. We use any SAT-solver to find

a solution for $P(C^*, S)$. One such solution is as follows:

$$\begin{aligned}
z_{3,5} &= true \\
z_{3,11} &= false \\
z_{5,7} &= true \\
z_{5,9} &= false \\
z_{5,10} &= false \\
z_{5,12} &= false \\
z_{2,9} &= true \\
z_{5,9} &= true \\
z_{3,10} &= true \\
z_{1,11} &= true \\
z_{3,12} &= true \\
z_{4,12} &= true
\end{aligned}$$

We then construct the rule-sets $\overline{R}_i \subseteq R_i$ according to equation 5.14:

$$\begin{aligned}
\overline{R}_1 &= \{r_1, r_2, r_8, r_{11}, r_{15}, r_{16}\} \\
\overline{R}_2 &= \{r_3, r_8, r_9\} \\
\overline{R}_3 &= \{r_4, r_5, r_{10}, r_{13}, r_{14}\} \\
\overline{R}_4 &= \{r_6, r_{12}\} \\
\overline{R}_5 &= \{r_7\}.
\end{aligned} \tag{7.5}$$

Note that $\overline{R}_1 = R_1, \overline{R}_2 = R_2$, and $\overline{R}_4 = R_4$. We compute restricted model graphs $G_3(\overline{R}_3)$ and $G_5(\overline{R}_5)$, shown in figures 7-9 and 7-10, respectively. We need not compute graphs $G_1(\overline{R}_1)$, $G_2(\overline{R}_2)$, or $G_4(\overline{R}_4)$, since they are the same as the corresponding graphs $G_i(R_i)$.

We define a graphical subutility function u_i for each set S_i based on the restricted model graphs $G_i(\overline{R}_i)$. For each subutility function, we use the descendent graphical utility function. We present these subutility functions in the following tables.

Model of S_1	$u_1(m \upharpoonright S_1)$
000	1
001	2
010	3
011	1
100	2
101	4
110	4
111	2

Model of S_2	$u_2(m \upharpoonright S_2)$
0	1
1	2

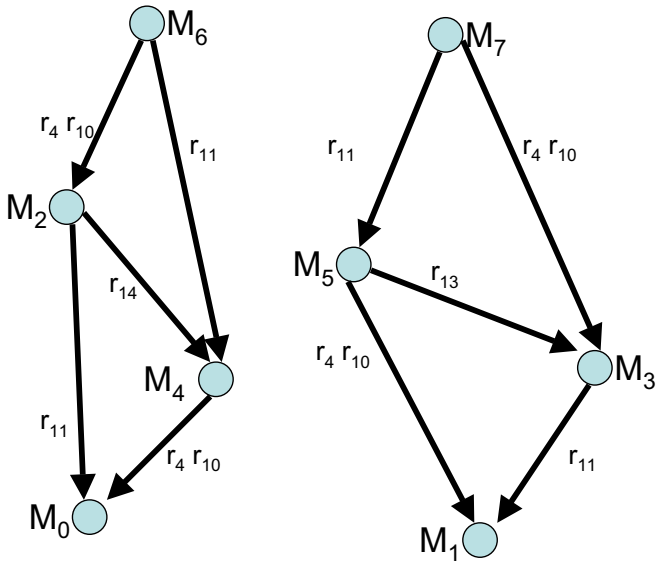

Figure 7-9: Restricted model graph $G_3(\overline{R}_3)$ for the set $S_3 = \{f_4.f_5, f_8\}$, $\overline{R}_3$ is defined in equation 7.5. Edges between models are labelled with the rule(s) that cause the edge.

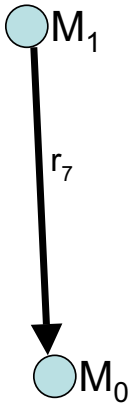

Figure 7-10: Restricted model graph $G_5(\overline{R}_5)$ for the set $S_5 = \{f_7\}$, $\overline{R}_5$ is defined in equation 7.5. Edges between models are labelled with the rule(s) that cause the edge.

Model of S_3	$u_3(m \upharpoonright S_3)$
000	1
001	1
010	3
011	2
100	2
101	3
110	4
111	4

Model of S_4	$u_4(m \upharpoonright S_4)$
0	1
1	2

Model of S_5	$u_5(m \upharpoonright S_5)$
0	1
1	2

With the subutility functions u_i defined above, we define the system of linear inequalities $I(C^*, S, \overline{R})$. We need to add linear inequalities for any rule that disagrees with any subutility function. In our example, these are rules r_9, r_{10}, r_{11}, and r_{12}. Since we solve this with a linear program, for each rule, we add linear inequalities of the form given in equation 5.17.

Since r_9 has preferences over S_2 and S_5, we need to add linear inequalities for all model pairs (m_1, m_2) in $\mathcal{M}(r_9)$, or, all combinations of the minimum and maximum values of $u_2(m_1) - u_2(m_2)$ and $u_5(m_1) - u_5(m_2)$ for model pairs satisfying r_9. Below, we enumerate the members of sets $minmax(r_9, 2)$, $minmax(r_9, 5)$, and $\mathcal{M}(r_9)$. The members of $\mathcal{M}(r_9)$ are written using the notation $((m'_1, m''_1), (m'_2, m''_2))$, where m'_1 and m'_2 are models of S_2, and m''_1 and m''_2 are models of S_5.

Set	*Members*
$minmax(r_9, 2)$	$\{1\}$
$minmax(r_9, 5)$	$\{-1\}$
$\mathcal{M}(r_9)$	$\{((1,0),(0,1))\}$

Rule r_{10} intersects with sets S_3 and S_5. Below are the members of sets $minmax(r_{10}, 3)$, $minmax(r_{10}, 5)$, and $\mathcal{M}(r_{10})$. The members of $\mathcal{M}(r_{10})$ are written using the notation $((m'_1, m''_1), (m'_2, m''_2))$, where m'_1 and m'_2 are models of S_3, and m''_1 and m''_2 are models of S_5.

Set	*Members*
$minmax(r_{10}, 3)$	$\{1, 2\}$
$minmax(r_{10}, 5)$	$\{-1\}$
$\mathcal{M}(r_{10})$	$\{((100,0),(000,1)),$ $((101,0),(001,1))\}$

Rule	u_1	u_2	u_3	u_4	u_5	
r_9		$+t_2$			$-t_5$	≥ 1
r_{10}			$+t_3$		$-t_5$	≥ 1
r_{10}			$+2t_3$		$-t_5$	≥ 1
r_{11}	$+t_1$		$+2t_3$			≥ 1
r_{11}	$+t_1$		$-2t_3$			≥ 1
r_{11}	$+2t_1$		$+2t_3$			≥ 1
r_{11}	$+2t_1$		$-2t_3$			≥ 1
r_{12}				$+t_4$	$-t_5$	≥ 1

Figure 7-11: Linear inequalities in $I(C^*, S, \overline{R})$, for setting scaling parameters $t_1, ..., t_5$. These are computed from rules that disagree with subutility functions u_i. We list the rule that contributes each of the linear inequalities

Below is a table for sets involving r_{11}. We show the members of sets $minmax(r_{11}, 1)$, $minmax(r_{11}, 3)$, and $\mathcal{M}(r_{11})$. The members of $\mathcal{M}(r_{11})$ are written using the notation $((m'_1, m''_1), (m'_2, m''_2))$, where m'_1 and m'_2 are models of S_1, and m''_1 and m''_2 are models of S_3.

Set	*Members*
$minmax(r_{11}, 1)$	$\{1, 2\}$
$minmax(r_{11}, 3)$	$\{-2, 2\}$
$\mathcal{M}(r_{11})$	$\{((000, 010), (010, 000)),$ $((000, 111), (010, 101)),$ $((001, 010), (011, 000)),$ $((001, 111), (011, 101))\}$

Finally we show members of sets $minmax(r_{12}, 4)$, $minmax(r_{12}, 5)$, and $\mathcal{M}(r_{12})$. The members of $\mathcal{M}(r_{12})$ are written using the notation $((m'_1, m''_1), (m'_2, m''_2))$, where m'_1 and m'_2 are models of S_4, and m''_1 and m''_2 are models of S_5.

Set	*Members*
$minmax(r_{12}, 4)$	$\{1\}$
$minmax(r_{12}, 5)$	$\{-1\}$
$\mathcal{M}(r_{12})$	$\{((1, 0), (0, 1))\}$

From the above four tables, we generate a list of linear inequalities $I(C^*, S, \overline{R})$, and solve for the scaling parameters $t_1, ..., t_5$. The inequalities in $I(C^*, S, \overline{R})$ are shown in figure 7.

We also require that each $t_i > 0$. We use linear programming to solve $I(C^*, S, \overline{R})$. One solution is

$$t_1 = 5$$
$$t_2 = 2$$
$$t_3 = 2$$
$$t_4 = 2$$
$$t_5 = 1$$

We can, finally, write an additive decomposition utility function for the set of preferences given in figure 7-3. The function is:

$$u(m) = 5u_1(m) + 2u_2(m) + 2u_3(m) + 2u_4(m) + u_5(m)$$

where the u_i are the graphical subutility functions defined in previously. The complexity of evaluating $u(m)$ for a model is now time $O(|S|)$, if we implement the subutility functions as constant-time lookup hash tables of values for each model. Thus, we have to store hash tables of size dominated by the largest utility-independent feature set, S_1 or S_3. In this case, $|S_1| = 3$, so we require space $O(2^3)$. However, this represents a large improvement in performance over a simple graph method (as presented in chapter 3), where the running time (or space) required is $O(2^9)$.

We can then use the utility function to compute utilities of some models. For example, we compute the utilities of the model receiving lowest utility (000000000), and the model receiving highest utility (101111111). These can be computed by finding the minimums and maximums, respectively, of the subutility functions.

Model of F(C)	u_1	u_2	u_3	u_4	u_5	$u(m)$
000000000	$+5*1$	$+2*1$	$+2*1$	$+2*1$	$+1*1$	$=12$
101111111	$+5*4$	$+2*2$	$+2*4$	$+2*2$	$+1*2$	$=38$

We note that the highest utility corresponds to a situation where the processes for logic, web server, animation, database, backup, and SETI at Home, are all optimized for performance. The mail process is optimized for security. Also, both payday occurs and an email worm attack is threatened. Naturally, it is not always payday and it is not always the case that the company is preparing for an email worm attack. Suppose that it is not payday and there is no attack, then the model with highest utility is 111111100, which corresponds to the configuration where each process is optimized for performance.

Chapter 8

Implementation

We implemented the general form of the algorithm laid out in this thesis. The implementation is done in C# and available from www.mit.edu/~mmcgeach. The implementation has an interface allowing atoms, models, rules, and preference operators to be defined. The internal representations of models are strings or sometimes bit vectors packed into integers when its know the models will be less than 32 bits long (as is appropriate for a model restricted to a subutility function - if the subutility functions have more than 32 features, the algorithms are entirely intractable anyway). We translated java code for solving SATs in CNF and for solving linear programming problems. We note that much translation between Java and C# can be done automatically. Our SAT solver was a translation into C# of a translation into Java by Daniel Jackson[1], of the original WalkSat algorithm written by Henry Kautz and Bart Selman in C. Similarly, we translated a Linear Programming package called JLAPACK[2], which is a Java translation of the popular C library CLAPACK, which is itself a translation of the popular Fortran LAPACK.

8.1 Application Programming Interface

We describe here an interface to the preference algorithms implemented. We call this the "iPreferenceLibrary", an interface to the preference library. There are five important interfaces in the preference library: IAtom, IModel, IOperator, IRelation, and IRelationSet. IAtom is for defining the atoms of the feature space, those atoms over which a preference is expressed. IModel allows models to be defined, where a model is a collection of boolean truth assignments to Atoms. There is currently one type of IOperator, the "strictly preferred to" operator. A Relation is composed of an Operator and two Models, where the LHS-Model is the left operand of the Operator, and the RHS-Model is the right operand of the Operator. A Relation is a preference or a preference rule. IRelationSet allows creations of a collection of relations. Once a IRelationSet is defined, a utility function can be constructed on that set, and utilities

[1] http://sdg.lcs.mit.edu/ dnj/walksat/

[2] Siddhartha Chatterjee, The Harpoon Project, http://www.cs.unc.edu/Research/HARPOON/jlapack/

of models over the set of atoms used therein can be computed.

Each interface has a corresponding implementation class. These are CPAtom, CPModel, CPOperator, CPRelation, and CPRelationSet, respectively. We describe the API for the interaction of these in the following.

8.1.1 CPAtom Class

A CPAtom inherits from IAtom. The IAtom declaration is as follows.

```
public interface IAtom
{
    string          Name { get; }
    object          Id { get;}
}
```

The Id of an IAtom must be unique. This is used in determining if one atom is the same as another atom. A CPAtom is defined by its name, and the Id of a CPAtom is the hashcode of the atom's name. CPAtom overrides both Equals(Object o) and GetHashCode() to refer to the Id of the atom.

8.1.2 CPModel Class

A CPModel is a collection of CPAtoms paired with truth values. The interpretation is that a model is a collection of features (atoms), and the features are either true or false. Features not included in the model are assumed to be "don't care", in the *ceteris paribus* sense, frequently represented with '$*$' in earlier sections of this paper.

A CPModel inherits from IModel. IModel is defined as follows

```
public interface IModel
{
    void            Add(IAtom atom, bool assignment);
    void            Remove(IAtom atom);
    bool            Contains(IAtom atom);
    bool            ValueOf(IAtom atom);
    int             Count();
    IEnumerator     GetEnumerator();
}
```

Methods

```
void Add(IAtom atom, bool assignment);
```

Adds atom to the model with truth value assignment. If atom was already a member of the model, it overwrites the previous truth value of that atom.

```
public void Remove(IAtom atom);
```

Removes atom from the model, if it exists.

```
public bool Contains(IAtom atom);
```

Checks if the atom is contained in the model. If so, returns true, otherwise false. Atoms are compared using Object.Equals().

```
public bool ValueOf(IAtom atom);
```

Returns the truth value assigned to atom when atom was added into the model. It is an error to call this on an atom that is not in the model.

```
public int Count();
```

Returns the number of different atoms in the model.

```
public IEnumerator GetEnumerator;
```

Returns an enumeration of all the atoms contained in the model.

8.1.3 CPOperator Class

A CPOperator inherits from IOperator. The IOperator declaration is as follows.

```
public interface IOperator
{
    string Name {get;}
}
```

Operators are largely left for future use. As it is, the only operator is a "strict preference" operator, declared in CPOperator:

```
public static readonly CPOperator StrictPref =
      new CPOperator("StrictPref");
```

8.1.4 CPRelation Class

A CPRelation inherits from IRelation. The IRelation declaration is as follows.

```
public interface IRelation
{
    IModel LHSModel {get; set;}
    IModel RHSModel {get; set;}
    IOperator Operator {get; set;}
}
```

A CPRelation is a *ceteris paribus* preference statement. The CPRelation is composed of two CPModels and one CPOperator. The semantics of the preference is "LHSModel IOperator RHSModel." Since, at this time, there is only one IOperator, the "strict preference" operator, the CPRelation implies that the LHSModel CPModel is strictly preferred to the RHSModel CPModel.

The CPRelation constructor and the LHSModel and RHSModel *set* method will throw exceptions if the LHSModel and RHSModel do not contain the same atoms. This is the same restriction we defined earlier on rules r in $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$: if $f_i(LHS(r)) = *$ then $f_i(RHS(r)) = *$ and vice versa.

8.1.5 CPRelationSet Class

A CPRelationSet inherits from IRelationSet. The IRelationSet declaration is as follows.

```
public interface IRelationSet
{
    void Add(IRelation rel);
    void Remove(IRelation rel);
    bool Contains(IRelation rel);
    bool ContainsAtom(IAtom atom);
    IModel Concretify(IModel model);
    IEnumerator GetEnumerator();

    void InitEngine();
    double GetUtility(IModel model);
    IEnumerator GetTopN(int n);
}
```

A CPRelationSet is a set of CPRelations. As a set of CPRelations, the CPRelationSet supports the usual set operations: add, remove, and contains. The CPRelationSet also has functions for actually invoking the *Ceteris Paribus* Preference Engine and computing the utility of models in relation to the constituent CPRelations. The CPRelations contained within the CPRelationSet need not have all the same Atoms within.

```
void Add(IRelation rel);
```

Adds the given CPRelation to the CPRelationSet. If the relation is already in the CPRelationSet, nothing is changed.

```
void Remove(IRelation rel);
```

Removes the CPRelation from the CPRelationSet. If the relation is not in the CPRelationSet, nothing is changed.

```
bool Contains(IRelation rel);
```

Returns true if the given CPRelation is already in the CPRelationSet. For two CPRelations to be equal, they must be the same object (*i.e.* two pointers pointing at the same object).

```
bool ContainsAtom(IAtom atom);
```

This returns false if InitEngine has not yet been called on the CPRelationSet. This function returns true if the input atom is a member of any CPModel in any CPRelation in the CPRelationSet.

```
IModel Concretify(IModel model);
```

This returns the input CPModel if InitEngine has not yet been called on the CPRelationSet. The CPRelationSet maintains a list of all atoms used in all of the constituent CPRelations. Concretify returns a CPModel that is an extension of the input CPModel in the following way. For any atom in the CPRelationSet but not in the CPModel, the new model now has a value of "false" for that atom. The returned CPModel is otherwise the same as the input CPModel.

```
IEnumerator GetEnumerator();
```

Returns an enumeration of all the CPRelations contained in the CPRelationSet.

```
void InitEngine();
```

Builds a *ceteris paribus* utility function from the preferences entailed by the CPRelations in the CPRelationSet. Other functions of CPRelationSet can then be used. If changes to the CPRelations in the CPRelationSet after calling InitEngine(), and need to be incorporated into the utility function, then InitEngine should be called again. This function can be computationally expensive.

```
double GetUtility(IModel model);
```

Throws an exception if called before InitEngine has been called. Otherwise returns the utility of the input CPModel according to the utility function entailed by the CPRelations in the CPRelationSet.

```
IEnumerator GetTopN(int n);
```

Throws an exception if called before InitEngine has been called. Otherwise returns an enumeration of N CPModels that receive the top N utility values according to the utility function entailed by the CPRelations in the CPRelationSet.

8.1.6 Simple Example

The Following is a simple example of how to use the API presented in the preceding sections.

```
CPAtom a1 = new CPAtom("male");
CPAtom a2 = new CPAtom("tall");
CPAtom a3 = new CPAtom("dark");
CPAtom a4 = new CPAtom("handsome");

CPModel m1 = new CPModel();
m1.Add(a1, true);
m1.Add(a2, true);
m1.Add(a4, true);

CPModel m2 = new CPModel();
m2.Add(a1, false);
m2.Add(a2, false);
m2.Add(a4, false);

CPRelation r1 = new CPRelation(m1,m2);

CPAtom a5 = new CPAtom("rich");

CPModel m3 = new CPModel();
m3.Add(a5, true);
m3.Add(a1, false);
m3.Add(a2, true);

CPModel m4 = new CPModel();
m4.Add(a5, false);
m4.Add(a1, false);

CPRelation r2 = new CPRelation(m3,m4);

CPRelationSet cprs = new CPRelationSet();
cprs.Add(r1);
cprs.Add(r2);
cprs.InitEngine();
cprs.GetTopN(5);
```

Chapter 9

Related Work

Our work has some general characteristics that set it apart from other work in the field of Artificial Intelligence dealing with utility theory. In general, the literature usually assumes that some human-centric method is used to assess a human decision maker's preferences and utilities over a feature space (such as described in [KR76]). Our work produces such a function from preference statements. Further, our work contains a method for automatically computing utility independence, which is generally assumed or gathered directly from the decision maker in the literature. And finally, no other work actually computes a numeric utility function from purely qualitative preferences. On the other hand, our work does not treat probabilistic actions, which complicates things immensely. Researchers who are concerned to treat probability frequently spend most of their effort on such issues.

Many researchers have defined related logics of desire [vdTW98, MS99, Sho97], several of them even define logics of *ceteris paribus* statements [TP94a, TP94b, BG96, BG95, BBHP99]. We mention these in the following sections.

9.1 Related Systems

Leendert van der Torre and Emil Weydert [vdTW98], have an interpretation of goals which is generally different from the *ceteris paribus* system mentioned here. They define a conditional goal as $a \rightarrow b$ meaning if a obtains, then try to achieve b. This puts them in close similarity to Bacchus and Grove's conditional *ceteris paribus* utilities [BG96], rather than the work of Doyle and Wellman [DW94]. The system of van der Torre and Weydert is concerned to elevate goals to the semantic level of desires, and as such propose a system wherein they assign each goal an *a priori* utility. They can then add the utilities of the goals together, when one or more goals are accomplished simultaneously. They allow goals to have negative impact if they are not satisfied. The idea of giving goals semantics in terms of desires is natural, [WD91] gives an interpretation of goals in terms of the same *ceteris paribus* representation used in this thesis.

La Mura and Shoham use a very different representation, called expected utility networks [MS99, Sho97], wherein they reason about utilities the way researchers

have previously reasoned about probabilities. They discusses additive utility decomposition, wherein a subutility is called a "factor" which is a constant utility amount contributed to a situation if the situation makes the factor true. With a utility decomposition in hand, they define "Bayes Nets for utilities," called u-nets [Sho97] or eu-nets [MS99] for "expected utility" nets, and show some powerful reasoning methods using these nets. These reasoning methods work similar to Bayesian network methods, where an expected utility can be computed from the topology of the network and the probability and utility functions present at each node. This work differs in two important aspects from the work proposed in this thesis. Firstly, they use a vastly different representation, necessitating different computation structures. Secondly, they take the utility factors as primitives, known *a priori*, rather than constructing these from some more basic preference information.

9.2 *Ceteris Paribus* Reasoning Systems

Tan and Pearl use conditional *ceteris paribus* comparatives [TP94a, TP94b]. While these do use a *ceteris paribus* interpretation of preferences, they condition the statement on some set of worlds. Thus the *ceteris paribus* rule only holds if some other logical condition (or set of worlds) is also true. They assume each desire is quantified, $\alpha \succ \beta$ by ϵ, such that the utility of α exceeds that of β by at least ϵ. This quantification aside, their unconditional *ceteris paribus* semantics are equivalent to those used here. They are concerned in [TP94a] to deal with the specificity of the rules, and allow a more specific rule to override a more general rule. They do not deal with the computation of preference queries: which outcomes are preferred to which other outcomes. Thus this work does not address computation, nor the construction of subutilities, which are the main two aims of this thesis.

Bacchus and Grove have presented a slightly different conception of *ceteris paribus* preference specification [BG96], and algorithms for computing with it[BG95]. Similar to La Mura and Shoham [MS99], their computation paradigm is an adaptation of Bayesian Networks to utility. Their use of conditional additive utility independence of sets of different features to define graphical representations is similar to our use of unconditional additive utility independence. However, their graph represents dependencies between features, they then compute aa simplified form of the utility function from the cliques of the graph. Our work uses the preferences themselves to compute utility independence, and the preferences to define the form of the utility function.

Bacchus and Grove present their preference representation in [BG96], where they contrast their own approach with that of Doyle, Shoham, and Wellman [DSW91]. Bacchus and Grove use a conditional expected utility measure, the expected utility divided by the total probability of occurrence, where if the sum of expected utility of states satisfying ϕ is greater than those satisfying $\neg\phi$ then there is a preference for ϕ over $\neg\phi$. Their representation requires *a priori* knowledge of all the utilities and all the probabilities of states satisfying ϕ and $\neg\phi$.

The main differences between Bacchus and Grove's work and our own are as follows. Firstly, there are no qualitative preferences. Secondly, they do not discuss

how to get utility independence, they assume it is given. Thirdly, they don't talk about how to construct the subutilities. Fourthly, their work is complicated by their use of conditional additive independence (independence may fail sometimes) and use of Bayes nets for probability computation. However, it is still similar in the sense that they try to compute utility functions with *ceteris paribus* preferences.

Boutilier [BBHP99, BBGP97], also proposes a system of conditional *ceteris paribus* preference statements. They use the conditions on the *ceteris paribus* statements to define a graph structure, called a *Ceteris Paribus* Network. Each node is a feature, and is annotated with a table describing the user's (human decision maker's) qualitative preferences given every combination of parent values.

Boutilier's work allows complicated conditions on a *ceteris paribus* preference, but the preference itself must be restricted to a single utility-independent attribute. Our method allows both complicated conditions (section 1.2.1) and complicated preferences spanning several utility-independent feature sets. Again, our work address constructing the individual subutility functions, while other work does not. Boutilier's nodes in the cp-net are annotated with tables of utilities exponential in the size of the number of features involved. This is equivalent to our (worst-case) exponential time requirement to evaluate a subutility function for a UI feature set. The nodes in the cp-net must then be traversed to compute a dominance query. Boutilier proves some (weak) search methods for this traversal, and argues for some strong heuristics. In many cases, their heuristics suffice to assure the search is backtrack free. However, the number of nodes in the graph may be exponential in the size of the feature space. When the heuristics apply, the search is generally linear in the number of features. Thus, the complexity of comparing two models is similar to the complexity of computing the utility of a model in our system, if we compile each subutility function into an (exponential-size) look-up table.

Boutilier *et. al.*'s recent work allows combining their cp-net with a quantitative utility net and then computing the utility of a model with the new structure [BBB01]. Therein they give substantial treatment of how to elicit preferences from human decision makers, and how to massage these preferences to conform to the "ucp-net" representation. Interestingly, they obtain a series of constraints on the trade-off weights between utility-independent feature sets, then use linear programming to solve for the trade-off parameters.

Chapter 10

Future Work

Future work extending this thesis can be divided into two groups. The first is work using the current *ceteris paribus* representation and formalism. The second is work that alters or augments the representation presented in some way.

There are a few avenues of future research that we mentioned briefly in the main body of this thesis. We can attempt to synthesize the treatment of weak preferences in chapter 3 with the way we build subutility functions. It may be possible to just pretend all weak preferences are actually strict preferences, and use constraint satisfaction to choose one or the other weak preference when we have two asserting $m \succsim m'$ and $m' \succsim m$. Or we can allow these to both be satisfied by allowing a cycle in u_i from m to m'. If we allow a cycle from m to m', that is, $m \sim m'$, we must define this u_i to be disagreeing with any strict preference that assert either that $m \succ m'$ or that $m' \succ m$.

We might explore the possible uses of partial conflict-free ordering of mutually utility-independent feature sets. It seems unlikely in general that entire conflict-free orderings will be found, but more likely that partial conflict-free orderings would arise. It may be straightforward to assign a partial lexicographic ordering to a partial conflict-free ordering, then order the remaining sets using constraint satisfaction. This approach is likely to complicate the definitions and use of linear inequalities.

Non-linear scaling could be addressed in a more intricate manner. Composing non-linear functions with the subutility functions could remedy the issue. Or, conditioning the value of the scaling parameter by the value of the subutility function would achieve the same result. In either case, the important task will be determining the exact form of the nonlinearity required.

We plan to extend the representation of qualitative preferences given herein in several ways. Firstly, we might give a treatment of non-binary discrete valued features. For this, the work of Wellman on Qualitative Probabilistic Networks [Wel90] might be appropriate. The natural extension of that is to treat continuous-valued features. From a purely speculative standpoint, treating continuous valued variables could require a richer (and less qualitative) preference specification language. One extension could be a representation similar to partial differential equations. The decision maker might specify a partial relation between two features, such as "A is 10-times as desirable as B." This has a natural representation in terms of partial

differential equations.

Alternatively, our representation could be fractured into several representations of various decision theoretic concepts. Our feature vector language mixes together representations of utility independence, tradeoff ratios between mutually utility-independent feature sets, and preferences within a feature set. Clearly some of the power of the representation presented herein comes from this combination, but it may be possible to achieve faster reasoning methods by splitting the representations. However, this direction would bring our work much closer to the work of Bacchus and Grove and Boutilier *et. al.* as described in section 9.2

Chapter 11

Conclusions

Although much work remains to be done in the area of reasoning with *ceteris paribus* preferences, the work demonstrated herein has provided some answers. We have shown a translation between the logical representation of *ceteris paribus* preferences and a new, feature-vector representation of *ceteris paribus* preferences. The feature-vector representation makes it simple to construct a model graph representing a particular set of preferences. This graph makes the definition of numeric utility functions simple, although of worst-case exponential complexity. We have then shown how mutual utility independence between features can be inferred from the structure of preference statements in the feature-vector representation. Finally we demonstrated some heuristic methods that can be used in the presence of utility independence to achieve better performance on computing the numeric utility of a model, according to a fixed set of preferences.

The work herein has provided several reasoning methods using the *ceteris paribus* representation of [DSW91], for which there previously were no explicit methods developed in detail. There have been other representations of preferences, some of which have well-developed reasoning methods. Future applications involving the automatic specification of preferences, and computation with them, may now choose to use the representation method of Doyle, Shoham, and Wellman and the reasoning methods presented in this thesis.

Appendix A

Notation Index

In Order of Appearance

Chapter 1

$\mathcal{L}$: Restricted logical language over the set of atoms $\mathcal{A}$.
F : Set of binary features describing possible worlds or outcomes of interest.
$\mathcal{A}$: Set of atoms. Each corresponds to a feature in the universe of discourse.
$f(a)$: The feature in F corresponding to atom a in $\mathcal{A}$.
$literals(\mathcal{A})$: Set of atoms $\mathcal{A}$ and their negations.
$\mathcal{M}$: Set of all models of $\mathcal{L}$.
m, m', m_i : Any or a particular model of $\mathcal{M}$ or another set of models.
$f_i(m)$: The value assigned to feature f_i in F by model m.
p, q : Any or a particular sentence in a logical language, such as $\mathcal{L}$.
$m \models p$: m satisfies p, or, the assignment of truth values to atoms by m makes formula p true.
$[p]$: Proposition p, the set of models making p true.
$\succsim$: Weak preference.
$\succ$: Strict preference.
$s(p)$: Minimal set of atoms determining the truth of p.
$m \equiv m' \mod p$: Equivalence modulo p, models m, m' are the same outside $s(p)$.
$m[p]$: Model modifications of m making p true, the set of models satisfying p that are all the same outside $s(p)$. $p \trianglerighteq q$: p is desired *ceteris paribus* at least as much as q.
$p \rhd q$: p is strictly desired *ceteris paribus* over q.
c : A *ceteris paribus* preference rule of the form $p \rhd q$ or $p \trianglerighteq q$ where p, q are statements in $\mathcal{L}$.
$\mathcal{F}(\mathcal{C})$: x.
C : A set of preference rules.
$[C]$: Set of all weak preference orders consistent with C.
u : A utility function that maps each model of $\mathcal{L}$ to a real number.
$p(u)$: Preorder over models implied by the utility function u.

Chapter 2

$F(C)$: Set of features corresponding to atoms in the union of supports of $c \in C$.
N : $N = |F(C)|$.
$\mathcal{L}^*$: Family of languages used in the feature vector representation.
$\mathcal{V}$: Feature Vector, an ordered subset of $F(C)$.
Γ : The alphabet $\{0, 1, *\}$.
$\mathcal{L}^*(\mathcal{V})$: Particular language of ordered $|\mathcal{V}|$-tuples taken from Γ.
$v(\mathcal{L}^*(\mathcal{V}))$: The feature vector of $\mathcal{L}^*(\mathcal{V})$, or $\mathcal{V}$.
Γ' : The alphabet $\{0, 1\}$.
$f_i(p)$: The value in Γ assigned by the statement p to the ith feature of F.
$f_i(m)$: The truth value assigned by the model m to the ith feature of F.
$\mathcal{M}^*(V)$: Set of all models of $\mathcal{L}^*(\mathcal{V})$.
$v(\mathcal{M}^*(\mathcal{V}))$: $\mathcal{V}$.
α : Function $\mathcal{M}^*(F(C)) \to \mathcal{M}$.
$[m]$: equivalence class of $m \in \mathcal{M}^*(F(C))$, $= \{m' \mid m' = \alpha(m)\}$.
$m \models s$: $s \in \mathcal{L}^*$, m assigns the same truth values as s, except where s assigns a '$*$'.
$m \upharpoonright s$: Restriction of m to set of features s, the truth values m assigns to the features of s.
$m \models m'$: m' assigns truth values to a set of features that is a subset of the features m assigns values. And these values are the same.
$\mathcal{L}^*_{\mathcal{R}}(\mathcal{V})$: language of rules in the feature vector language. Each of the form $p \succ q$, $p, q \in \mathcal{L}^*(\mathcal{V})$.
$LHS(r)$: If $r = p \succ q$, $LHS(r) = p$.
$RHS(r)$: If $r = p \succ q$, $LHS(r) = q$.
$s(p)$: Support features of a statement p, those not marked with a '*'.
$s(r)$: Support features of a rule r, those not marked with a '*'.
$(m_1, m_2) \models r$: $m_1 \models LHS(r)$, $m_2 \models RHS(r)$, and the two models are the same outside $s(r)$.
$\sigma(m)$: function taking a truth assignment for a partial set of literals and extending to a model of $\mathcal{L}^*(F(C))$ by padding with '*'.
$\mu(k, \mathcal{L}^*(F(C)))$: A characteristic model of k in $\mathcal{M}^*(F(C))$, any model m such that $m \models k$.
$[LHS(r)]$: subset of models of $\mathcal{L}$ defined by $\{m' \mid m' = \alpha(m) \upharpoonright F(C), m \models LHS(r)\}$.
$\sigma(C)$: $\sigma : C \to C^*$, translation taking a set of *ceteris paribus* rules and converting them to rules in $\mathcal{L}^*_{\mathcal{R}}(F(C))$.

Chapter 3

C^* : An input set of preferences, each in $\mathcal{L}^*_{\mathcal{R}}(F(C))$.
$G(C^*)$: Model Graph.
$e_s(m_1, m_2)$: Directed edge in $G(C^*)$, indicating that there is a strict preference for m_1 over m_2.
$e_w(m_1, m_2)$: Directed edge in $G(C^*)$, indicating that there is a weak preference for

m_1 over m_2.
u_M : Minimizing GUF, longest path from m.
u_D : Descendant GUF, number of unique descendants under m.
u_X : Maximizing GUF, longest path ending at m.
u_T : Topological GUF, rank of m in topological-sorted order.

Chapter 4

S_i : A subset of $F(C)$.
$\hat{u}_i$: A *partial utility function* $m \restriction S_i \rightarrow \mathbb{R}$.
u_i : A *subutility function*: $m \rightarrow \mathbb{R}$ such that $u_i(m) = \hat{u}_i(m \restriction S_i)$.
u_S : Subutility function for the feature set S.
$m_1 \wedge m_2$: shorthand for combination of models, $\mu(m_1, \mathcal{L}(\mathcal{V})(F(C))) \cup \mu(m_2, \mathcal{L}(\mathcal{V})(F(C)))$.
$\overline{S_i}$: complement of features S_i, that is $F(C) \backslash S_i$.
S : partition of $F(C)$ into mutually utility-independent feature sets $\{S_1, S_2, ..., S_Q\}$.

Chapter 5

t_i : Scaling parameter.
u_{S_i} consistent with r : if $u_{S_i}(m_1) \geq 1 + u_{S_i}(m_2)$ whenever $(m_1, m_2) \models r$ and $(m_1 \restriction S_i) \neq (m_2 \restriction S_i)$ $G_i(R)$: Restricted model graph. Restrict features to S_i.
Conflicting Rules R: R conflict if there is no subutility function consistent with all $r \in R$.
$r \restriction S_i$: Rule restricted to a feature set. $RHS(r) \restriction S_i$ *succ* $LHS(r) \restriction S_i$.
R_i : Set of rules from C^*, where each has $s(r) \cap S_i \neq \emptyset$.
Conflict-Free R: No subset of rules R conflict.
R is a minimal conflict set: Set of rules R conflict but no subset of R conflict.
Complete set of conflict sets: Contains all sets of conflicting rules over some feature space.
$\overline{R}_i$: Subset of rules from R_i such that $\overline{R}_i$ is conflict-free.
Graphical Subutility Function : subutility function based on $G_i(\overline{R}_i)$ and consistent with rules $\overline{R}_i$.
u_i *agrees* with r : u_i is a graphical subutility function based on $G_i(\overline{R}_i)$ and $r \in \overline{R}_i$.
u_i *disagrees* with r : u_i is a graphical subutility function based on $G_i(\overline{R}_i)$ and $r \in (R_i \backslash \overline{R}_i)$.
$S_a(r)$: Set of indices i such that $r \in \overline{R}_i$.
$S_d(r)$: Set of indices i such that $r \in (R_i \backslash \overline{R}_i)$.
$\max(u_i)$: The maximum value returned by u_i for any model m.
$\vec{S}$: Ordering of the utility-independent feature sets S.
$\vec{S}$ conflict-free : If R_1 is conflict free and so is $R_i \backslash (\cup_{j=1}^{i-1} R_j)$ for all $i > 1$.
$P(C^*, S)$: Boolean CNF-form problem based on rules C^* and decomposition S.
Y_{ik} : Minimal set of conflicting rules on S_i.
Y_i : Minimal and complete set of Y_{ik} for S_i.

Y : Set of all Y_i.
R_c : Set of rules appearing in any sets in Y. z_{ij} : Boolean variable where i is an index of $S_i \in S$, and $r_j \in R_c$.
Θ : Solution to $P(C^*, S)$.
Θ_f : Set of variables z_{ij} assigned value *false* by solution Θ.
$minmax(r, i)$: Set of minimum and maximum values of $(u_i(m_1) - u_i(m_2))$ for all $(m_1, m_2) \models r$.
$\mathcal{M}(r)$: Set of model pairs (m_1, m_2) where for all i, $(u_i(m_1) - u_i(m_2)) \in minmax(r, i)$
$\overline{R}$: set of all $\overline{R}_i$.
$I(C^*, S, \overline{R})$: set of linear inequalities based on $C^*, S, \overline{R}$.

Bibliography

[BBB01] Craig Boutilier, Fahiem Bacchus, and Ronen L. Brafman. Ucp-networks: A directed graphical representation of conditional utilities. In *Proceedings of Seventeenth Conference on Uncertainty in Artificial Intelligence*, Seattle, 2001. To Appear.

[BBGP97] Craig Boutilier, Ronen Brafman, Christopher Geib, and David Poole. A constraint-based approach to preference elicitation and decision making. In Jon Doyle and Richmond H. Thomason, editors, *Working Papers of the AAAI Spring Symposium on Qualitative Preferences in Deliberation and Practical Reasoning*, pages 19–28, Menlo Park, California, 1997. AAAI.

[BBHP99] Craig Boutilier, Ronen I. Brafman, Holger H. Hoos, and David Poole. Reasoning with conditional ceteris paribus preference statements. In *Proceedings of Uncertainty in Artificial Intelligence 1999 (UAI-99)*, 1999.

[BDH+01] J. Broersen, M. Dastani, Z. Huang, J. Hulstijn, and L. van der Torre. The boid architecture. In *Proc. of 5th International Conference on Autonomous Agents*, pages 9–16. Association of Computing Machinery, 2001.

[BG95] Fahiem Bacchus and Adam Grove. Graphical models for preference and utility. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 3–19. Morgan Kaufmann, 1995.

[BG96] Fahiem Bacchus and Adam Grove. Utility independence in a qualitative decision theory. In *Proceedings of the Fifth International Conference on Knowledge Representation and Reasoning*, pages 542–552. Morgan Kaufmann, 1996.

[BRST00] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In H. Kautz and B. Porter, editors, *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 355–362, Austin, TX, July 2000. AAAI.

[Chv83] Vašek Chvátal. *Linear Programming*. W.H. Freeman and Company, New York, 1983.

[Deb59] G. Debreu. *Topological methods in cardinal utility theory*. Mathematical Methods in the Social Sciences. Stanford University Press, Stanford, California, 1959.

[DSW91] Jon Doyle, Yoav Shoham, and Michael P. Wellman. A logic of relative desire (preliminary report). In Zbigniew Ras, editor, *Proceedings of the Sixth International Symposium on Methodologies for Intelligent Systems*, Lecture Notes in Computer Science, Berlin, 1991. Springer-Verlag.

[DT99] Jon Doyle and Richmond H. Thomason. Background to qualitative decision theory. *AI Magazine*, 20(2):55–68, Summer 1999.

[DW94] J. Doyle and M. P. Wellman. Representing preferences as *ceteris paribus* comparatives. In *Working Notes of the AAAI Symposium on Decision-Theoretic Planning*. AAAI, March 1994.

[KR76] R. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley and Sons, New York, 1976.

[MS99] Piero La Mura and Yoav Shoham. Expected utility networks. In *Proc. of 15th conference on Uncertainty in Artificial Intelligence*, pages 366–373, 1999.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, California, 1988.

[Sho97] Yoav Shoham. A mechanism for reasoning about utilities (and probabilities): Preliminary report. In Jon Doyle and Richmond H. Thomason, editors, *Working Papers of the AAAI Spring Symposium on Qualitative Preferences in Deliberation and Practical Reasoning*, pages 85–93, Menlo Park, California, 1997. AAAI, AAAI.

[SLM92] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.

[TP94a] Sek-Wah Tan and Judea Pearl. Qualitative decision theory. In *AAAI94*, Menlo Park, CA, July 1994. AAAI Press.

[TP94b] Sek-Wah Tan and Judea Pearl. Specification and evaluation of preferences for planning under uncertainty. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *KR94*, San Francisco, CA, May 1994. Morgan Kaufmann.

[vdTW98] L. van der Torre and E. Weydert. Goals, desires, utilities and preferences. In *Proceedings of the ECAI'98 Workshop on Decision Theory meets Artificial Intelligence*, 1998.

[WD91] Michael Wellman and Jon Doyle. Preferential semantics for goals. In Thomas Dean and Kathleen McKeown, editors, *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 698–703, Menlo Park, California, 1991. AAAI Press.

[Wel90] Michael P. Wellman. Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, 44:257–303, 1990.