To appear in *Proceedings of the Eighteenth National Conference on Artificial Intelligence* (AAAI-2002), Edmonton, Alberta, Canada, July 28-August 1, 2002.

Efficient Utility Functions for Ceteris Paribus Preferences

Michael McGeachie
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
mmcgeach@mit.edu

Jon Doyle
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535
Jon_Doyle@ncsu.edu

Abstract

Ceteris paribus (other things being equal) preference provides a convenient means for stating constraints on numeric utility functions, but direct constructions of numerical utility representations from such statements have exponential worst-case cost. This paper describes more efficient constructions that combine analysis of utility independence with constraint-based search.

Introduction

Work on qualitative decision theory (Doyle & Thomason 1999) has explored methods for logical specification of abstract and generic preference information as a way of overcoming limitations suffered by direct numerical specifications of utility, especially in supporting automated adaptation of utility functions to changes in preferences. Purely logical specifications of utility functions, however, do not support the needs of automated decision-making methods, which must typically evaluate numerical utility functions repeatedly in computing expected utilities of actions. Such computations demand methods for automatically constructing numeric utility functions that satisfy logical specifications.

McGeachie and Doyle (2002) use direct graph-theoretic methods to construct several numeric utility-function representations of sets of preference rules expressed in a language related to the *ceteris paribus* preference logic of Doyle, Shoham, and Wellman (1991). Those direct methods construct a directed graph over salient model features that represents the initial preferences, and then use standard graph-theoretic functions, such as the number of nodes reachable from a starting node, to define utility representations. Such direct constructions can prove costly because small sets of *ceteris paribus* preference rules can specify very large graphs. In particular, one can express a lexicographic (total linear) order using one rule per model feature.

The following describes provably correct heuristic methods that improve on these constructions by exploiting the notion of utility independence to reduce the problem to one of constructing and combining utility functions over smaller sets of model features. The methods described here transform *ceteris paribus* preference rules into constraints on possible combinations of these subutility functions, and then apply standard constraint-satisfaction methods to find specific combinations that yield overall utility representations of the original *ceteris paribus* preferences.

Ceteris paribus preference and utility

We employ a restricted language $\mathcal{L}$, patterned after (Doyle, Shoham, & Wellman 1991) but using only the logical operators $\neg$ and $\wedge$, over a set of atoms F corresponding to propositional features mentioned in preference statements. By $literals(F)$ we denote the atoms of F and their negations; $literals(F) = F \cup \{\neg f \mid f \in F\}$. We call a complete consistent set m of literals a *model*. That is, a set of features m is a model iff m contains, for each $f \in F$, exactly one of f and $\neg f$. We write $\mathcal{M}$ 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 formulae in $\mathcal{L}$. We write $f(m)$ for the truth value assigned to feature f by model m, and say m *satisfies* a sentence p of $\mathcal{L}$, written $m \models p$, if the truth values m assigns to the atoms of p make p true. We define the *proposition* expressed by a sentence p, denoted $[p]$ 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 $m \sim m'$. A *utility function* $u : \mathcal{M} \rightarrow \mathbb{R}$ maps each model to a real number. A utility function u represents a preference order $\succsim$ just in case $u(m) \geq u(m')$ whenever $m \succeq m'$.

We define a new language $\mathcal{L}_{\mathrm{r}}$ of *preference rules* or *preference constraints* to consist of statements of the form $p \trianglerighteq q$, for $p, q \in \mathcal{L}$, meaning p is desired at least as much as q, and $p \triangleright q$, meaning p is desired more than q. Models of $\mathcal{L}_{\mathrm{r}}$ consist of preference orders over models of $\mathcal{L}$. We define the meaning of preference rules in terms of the notions of $\mathcal{L}$-model equivalence and modification.

The *support* of a sentence $p \in \mathcal{L}$ 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 to atoms outside the support of p as m does. That is,

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

Formally, we say that a preference order $\succsim$ *satisfies* $p \trianglerighteq q$ 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 means that when two models assign the same truth values to all features not in the support of either p or q, one making p true and q false is weakly preferred to one making p false and q true. The preference order satisfies a strict *ceteris paribus* preference $p \triangleright q$ iff in addition it strictly prefers some model making p true and q false to some model making p false and q true.

For a preference rule c, we write $[c]$ to denote the set of preference orders over $\mathcal{M}$ that satisfy c, that is, consistent with the constraint expressed by c. We write $[C]$ for a set of preference rules to denote the set of orders consistent with each $c \in C$, that is, $[C] = \bigcap_{c \in C} [c]$. Consistent rules and rule sets admit at least one consistent preference order. If a preference rule c implies that $m' \succ m''$, for $m', m'' \in \mathcal{M}$, we write $m' \succ_c m''$. For a set C of preference rules, we write $m' \succ_C m''$ to mean that $m' \succ_c m''$ for each $c \in C$.

We define the support of C, denoted $F(C)$, to be the set of features in $\mathcal{L}$ present in the support of statements of $\mathcal{L}$ appearing in constraints in C. Formally, $F(C)$ contains those features f such that either f or $\neg f$ appears in $\bigcup_{c \in C} s(c)$.

The following examines the problem of constructing, for a finite set C of *ceteris paribus* preference rules, a utility function u that represents some order in $[C]$ in an efficient manner.

Intermediate representation

The utility construction methods developed here employ an intermediate representation in terms of rules that relate paired patterns of specified and "don't care" feature values.

Let C be a finite set of preference rules. Because each preference rule mentions only finitely many features, $F(C)$ is also finite, and we write N to mean $|F(C)|$.

We construct utility functions representing the constraints in C in terms of model features. 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. Accordingly, we focus our attention on $F(C)$.

We define the intermediate representation relative to an enumeration $\mathcal{V} = (f_1, \ldots, f_N)$ of $F(C)$.

We define the language $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$ of intermediate rules in terms of a language $\mathcal{L}(\mathcal{V})$ of intermediate propositions over the ternary alphabet $\Gamma = \{0, 1, *\}$.

A statement in $\mathcal{L}(\mathcal{V})$ consists of a sequence of N letters drawn from the alphabet Γ, so that $\mathcal{L}(\mathcal{V})$ consists of words of length N over Γ. For example, if $\mathcal{V} = (f_1, f_2, f_3)$, we have $*10 \in \mathcal{L}(\mathcal{V})$. Given a statement $p \in \mathcal{L}(\mathcal{V})$ and a feature $f \in F(C)$, we write $f(p)$ for the value in Γ assigned to f in p. In particular, if $f = \mathcal{V}_i$, then $f(p) = p_i$.

An intermediate rule in $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$ consists of a triple $p \succ q$ in which $p, q \in \mathcal{L}(\mathcal{V})$ have matching $*$ values. That is, $p \succ q$ is in $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$ just in case $p_i = *$ if and only if $q_i = *$ for all $1 \leq i \leq N$. For example, if $\mathcal{V} = (f_1, f_2, f_3)$, $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$ contains the expression $*10 \succ *00$ but not the expression $*10 \succ 0{*}0$. We refer to the statement in $\mathcal{L}(\mathcal{V})$ left of the $\succ$ symbol in a rule r as the left-hand side of r, and denote it $LHS(r)$. We define right-hand side $RHS(r)$ analogously. Thus $p = LHS(p \succ q)$ and $q = RHS(p \succ q)$.

We regard statements of $\mathcal{L}(\mathcal{V})$ containing no $*$ letters as *models* of $\mathcal{L}(\mathcal{V})$, and write $\mathcal{M}(\mathcal{V})$ to denote the set of all such models. We say a model m *satisfies* s, written $m \models s$, just in case m assigns the same truth value to each feature as s does for each non $*$ feature in s. That is, $m \models s$ iff $f(m) = f(s)$ for each $f \in F(C)$ such that $f(s) \neq *$. For example, 0011 satisfies both $*0{*}1$ and $00{*}{*}$.

We project models in $\mathcal{M}$ to models in $\mathcal{M}(\mathcal{V})$ by a mapping $\alpha : \mathcal{M} \to \mathcal{M}(\mathcal{V})$ defined, for each $m \in \mathcal{M}$ and $f \in F(C)$, so that $f(\alpha(m)) = 1$ if $f \in m$ and $f(\alpha(m)) = 0$ 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, namely $[m] = \{m' \in \mathcal{M} \mid \alpha(m') = \alpha(m)\}$.

We say that a pair of models (m, m') of $\mathcal{L}(\mathcal{V})$ *satisfies* a rule r in $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$, and write $(m, m') \models r$, if m satisfies $LHS(r)$, m' satisfies $RHS(r)$, and m, m' have the same value for those features represented by $*$ in r, that is, $m_i = m'_i$ for each $1 \leq i \leq N$ such that $LHS(r)_i = *$. For example, $(100, 010) \models 10* \succ 01*$, but $(101, 010) \not\models 10* \succ 01*$.

The meaning $[r]$ of a rule r in $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$ is the set of all preference orders $\succ$ over $\mathcal{M}$ such that for each $m, m' \in \mathcal{M}$, if $(\alpha(m), \alpha(m')) \models r$, then $m \succ m'$. Thus a rule $**01 \succ **10$ represents the four specific preferences

$$\begin{array}{ll} 0001 \succ 0010 & 0101 \succ 0110 \\ 1001 \succ 1010 & 1101 \succ 1110 \end{array}$$

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

Constructing subutility functions over subsets of features requires the ability to consider models restricted to these subsets. We write $\mathcal{M}[S]$ to denote the set of models over a feature set $S \subseteq F$, so that $\mathcal{M} = \mathcal{M}[F]$. If $m \in \mathcal{M}[S]$ and $S' \subseteq S$, we write $m \upharpoonright S'$ to denote the *restriction* of m to S', that is, the model $m' \in \mathcal{M}[S']$ assigning the same values as m to all features in S'. We say that a model $m \in \mathcal{M}[S]$ satisfies a model $m' \in \mathcal{M}[S']$, written $m \models m'$ just in case $S' \subseteq S$ and $m' = m \upharpoonright S'$.

The *support features* of a statement p in $\mathcal{L}(\mathcal{V})$, written $s(p)$, are exactly those features in p that are assigned value either 0 or 1, which represent the least set of features needed to determine if a model of $\mathcal{L}(\mathcal{V})$ satisfies p. The support features of a rule r in $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$, denoted $s(r)$, are the features in $s(LHS(r))$. The definition of $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$ implies that $s(LHS(r))$ = $s(RHS(r))$.

One can show (McGeachie 2002) that, given a set C of *ceteris paribus* rules we can convert them into a set C^* of intermediate representation rules equivalent in the sense that both sets denote the same set of preference orders $[C] = [C^*]$. Thus, we can use the language $\mathcal{L}(\mathcal{V})$ in the following and know that our conclusions hold over statements in $\mathcal{L}$.

A basic utility function

We now define one utility function, $u : \mathcal{M}(\mathcal{V}) \to \mathbb{R}$ consistent with a set C^* of rules in the intermediate representation $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$. Composition of this function with the projection $\alpha : \mathcal{M} \to \mathcal{M}(\mathcal{V})$ yields a utility function on $\mathcal{M}$.

To construct this utility function, we first use the rules in C^* to define a directed graph $G(C^*)$ over $\mathcal{M}(\mathcal{V})$, called a *model graph*, that represents the preferences expressed in C^*. Each node in the graph represents one of the 2^N possible models $\mathcal{M}(\mathcal{V})$. The model graph $G(C^*)$ contains an edge $e(m_1, m_2)$ from source m_1 to sink m_2 if and only if $(m_1, m_2) \models r$ for some rule $r \in C^*$. Each edge represents a preference for the source over the sink. If C^* is consistent, then $G(C^*)$ is acyclic; a cycle would indicate the inconsistency of C^*. We can determine whether m is preferred to m' by looking for a path from m to m' in $G(C^*)$. The existence of such a path means $m \succ m'$.

The utility function used here assigns to a model $m \in \mathcal{M}(\mathcal{V})$ a value $u(m)$ equal to the number of nodes on the longest path originating from m in $G(C^*)$. We call this the *minimizing* utility function and elsewhere show it consistent with the preferences expressed in C^* (McGeachie & Doyle 2002).

Utility independence

Utility independence (UI) exists when the contribution to utility of some features is independent of the values of other features. In the following subsections, we demonstrate how this expedites our calculation of utility from preferences in the intermediate representation.

The general definition for utility independence of features S_i of features S_j is

$$[(m_2 \upharpoonright S_i) \cup (m_1 \upharpoonright S_j) \succ (m_3 \upharpoonright S_i) \cup (m_1 \upharpoonright S_j)] \Rightarrow \\ [(m_2 \upharpoonright S_i) \cup (m_4 \upharpoonright S_j) \succ (m_3 \upharpoonright S_i) \cup (m_4 \upharpoonright S_j)] \tag{1}$$

for all m_1, m_2, m_3, m_4 (Keeney & Raiffa 1976). The idea is that values m_2 for S_i are preferred to values m_3 for S_i, no matter what values S_j happens to assume. We are interested in computing a partition $S = \{S_1, S_2, ..., S_Q\}$ of $F(C)$ such that each set S_i is utility independent of its complement, $F(C) \backslash S_i$. That is, the feature sets S_i are all *mutually utility independent* (MUI). A basic decision theory result shows that a set of MUI features has an *additive decomposition* utility function of the form

$$u(m) = \sum_{i=1}^{k} t_i u_i(m), \tag{2}$$

for *subutility* functions $u_i : \mathcal{M}(\mathcal{V}) \to \mathbb{R}$ assigning utility values to models based on the restriction of the models to S_i. Having a large number of utility independent feature sets of small cardinality therefore greatly simplifies the computation of a utility function.

Common methods for determining utility independence assume independence relations are given *a priori* or elicited directly from a human decision maker (Keeney & Raiffa 1976; Bacchus & Grove 1996). To employ this concept in automatically computing utility functions, we use a different approach. We first assume that each feature is utility independent of all other features, and then try to discover for which features this assumption is invalid by reference to the preferences in C^*. 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.

The intuitive idea is to find a pair of rules that demonstrate that preference for some features depends on the value of other features. To use a very simple example, suppose $\mathcal{V} = (f_1, f_2, f_3)$, and consider two rules in $\mathcal{L}_{\mathrm{r}}(\mathcal{V})$; 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 $\{f_2\}$.

Sets S_i, S_j are utility dependent if we can demonstrate models m_1, m_2, m_3, and m_4 for some feature sets and a pair of rules r_1, r_2 such that the condition (1) does not hold. The correspondence is analogous to condition (1): we want all of the following

$$\begin{array}{l} (m_2 \upharpoonright S_i) \cup (m_1 \upharpoonright S_j) \models LHS(r_1), \\ (m_3 \upharpoonright S_i) \cup (m_1 \upharpoonright S_j) \models RHS(r_1), \\ (m_2 \upharpoonright S_i) \cup (m_4 \upharpoonright S_j) \models RHS(r_2), \\ (m_3 \upharpoonright S_i) \cup (m_4 \upharpoonright S_j) \models LHS(r_2). \end{array} \tag{3}$$

A method for finding such r_1, r_2, S_i, S_j is as follows. Since S_j must be a subset of the support features of r_1, and m_1 restricted to S_j must satisfy 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 must be 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 (1) and (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 (1) is violated.

We require a partition $S = \{S_1, S_2, ..., S_Q\}$ of $F(C)$, into MUI feature sets. A partition can be computed by starting with a set S of singleton sets; $S_1 = \{f_1\}$, $S_2 = \{f_2\}$, ..., $S_N = \{f_N\}$. We consider each pair of rules (r_1, r_2) in

C^*, and check if a preference for one feature changes with the value assigned to another feature by using the preceding method. If so, we have discovered sets of utility dependent features, and we update our partition S by joining these two sets together.

After performing such overlap calculations, we have computed a partition of $F(C)$ into MUI subsets. This method clearly produces a partition by starting with one, by combining utility-dependent subsets, and by stopping when no two subsets exhibit utility dependence on each other, making the partition elements MUI.

Utility computation

We now describe how to compute one utility function consistent with the set C^* of transformed input *ceteris paribus* preferences expressed in $\mathcal{L}_r(\mathcal{V})$. Using the methods of the previous section, we find a partition of $F(C)$ into MUI feature sets $S_1, S_2, \ldots, S_Q$ and seek an additive utility function of the form given in Equation (2) consisting of a linear combination of subutilities, with each subutility function a separate function of a particular S_i.

We have two tasks: to craft the subutility functions u_i and to choose the scaling constants t_i. We accomplish these two tasks in roughly the following way. We construct subutility functions by *restricting* rules to a set S_i. This is essentially a projection of a rule onto a particular set of features, in the same manner as we have defined restriction of models to a set of features. We then apply the graph-theoretic utility function definition to these restricting rules to obtain subutility functions u_i. However, rules that do not conflict in general can conflict when restricted to different feature sets. We use a boolean constraint satisfaction solver to resolve these conflicts. To assign values to scaling parameters t_i, we define linear inequalities that constrain the variables t_i and use standard linear programming methods to solve the inequalities for suitable scaling values t_i.

Subutility functions

We first define subutility functions $u_i(m)$ that take as input the values for the features S_i, and return an integer.

We examine the structure of the rules in the intermediate representation. Since we are only concerned with the features S_i, we can ignore rules that do not have members of S_i in their support feature set. We construct a *restricted model graph*, $G_i(C^*)$, where we consider each model $m \in \mathcal{M}[S_i]$ to be a node in the graph. Each rule r and pair $(m_1, m_2) \models r$, with $m_1, m_2 \in \mathcal{M}[S_i]$, indicate a directed edge from m_1 to m_2 in $G_i(C^*)$. Note that if a rule $r \in C^*$ has $S_i \backslash s(r) \neq \emptyset$, then the rule makes more than one edge in the model graph $G_i(C^*)$. Specifically, a rule r makes

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

edges. The construction of this graph $G_i(C^*)$ exactly parallels the construction of the general model graph G, as described earlier. We then use the minimizing utility function for u_i, wherein each node has utility equal to the length of the longest path originating at the node.

Conflicting rules Although we assume that the input preferences C^* have no conflicting rules, it is possible that a restricted model graph $G_i(C^*)$ has cycles in it even though a model graph $G(C^*)$ for $F(C)$ does not. Since our definition of graph theoretic utility functions does not handle cycles, we must fix this difficulty before we can use such a function for a subutility function.

Let R_i represent the set of rules $r \in C^*$ such that $s(r) \cap S_i \neq \emptyset$. Rules in a set R_i *conflict* if they imply a cycle in any model graph $G_i(C^*)$ of some utility independent set of features S_i. Consider now a minimal set of conflicting rules R_i, where for each r in R_i, r models $\mathcal{L}_r(\mathcal{V})$, so that for any $r \in R_i$, $R_i \backslash \{r\}$ implies no cycles or implies a different cycle in the model graph $G_i(C^*)$ of S_i. Since we assume the input preferences are not contradictory, a set of rules can only conflict when the rules are restricted to a particular subset of their support features. The restriction of a rule to a set of features S is another rule in the intermediate representation, but over fewer features:

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

Consider rules x, y in $\mathcal{L}_r(\mathcal{V})$, with $\mathcal{V} = (f_1, f_2, f_3, f_4)$, and $x = {*}01{*} \succ {*}10{*}$, $y = {*}{*}01 \succ {*}{*}10$. If we examine the restriction to feature three: $x \upharpoonright \{f_3\}$ and $y \upharpoonright \{f_3\}$, then we have $x \upharpoonright \{f_3\} = 1 \succ 0$ and $y \upharpoonright \{f_3\} = 0 \succ 1$. 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. If we let $S = \{f_3\}$, then we say x, y *conflict on the set* S.

Let $r_a, r_b, r_c \in R_i$. Suppose r_a, r_b, and r_c conflict on S_i. *E.g.* we might have $r_a \upharpoonright S_i = 001 \succ 100, r_b \upharpoonright S_i = 100 \succ 010, r_c \upharpoonright S_i = 010 \succ 001$. We resolve conflicts by choosing one of the conflicting rules and removing the rule, temporarily, from the set R_i. The following section explains how to choose which rule to remove.

Suppose there are sets of conflicting rules $Y_1, Y_2, ..., Y_K$, where each is composed of rules $r \in R_i$. By definition, each set Y_k represents a cycle in $G_i(C^*)$. We require that all distinct cycles in $G_i(C^*)$ are represented by (perhaps duplicate) conflicting rule sets Y_k. Since Y_k is a minimal set implying a particular cycle in $G_i(C^*)$, for any $r \in Y_k$ we know $Y_k \backslash \{r\}$ does not imply the same cycle in $G_i(C^*)$. All cycles in $G_i(C^*)$ are represented by some $Y_k \in \{Y_1, Y_2, ..., Y_K\}$. We will choose a rule $r \in Y_k$ for each set Y_k, and remove r from R_i (discussed below). Since we stipulate that all $r' \in Y_k$ are taken from the set R_i, when we remove r from R_i this also removes r from all sets Y_k which might contain r. After removal of r, we rebuild $G_i(C^*)$ with the remains of R_i. We now have a model graph $G_i(C^*)$ that is cycle-free. Note that, after this removal process, $|R_i| \geq 1$, since one rule cannot imply a cycle. With no cycles, we can define u_i to be the subutility function for S_i based on the minimizing utility function of $G_i(C^*)$. We say that the resulting function u_i *agrees* with rules $r \in R_i$, and that u_i *disagrees* with those rules r that were removed from R_i. When u_i agrees with r, then for all $(m_1, m_2) \models r$, $u_i(m_1) > u_i(m_2)$.

If a utility function u is consistent with a set of preferences C^*, then u represents some order 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 property of the subutility functions u_i relating utility contributions due to the subutility functions agreeing and disagreeing with an intermediate rule. To state the result, we define $S_a(r)$ and $S_d(r)$ to be the sets of indices of subutility functions respectively agreeing and disagreeing with r.

Theorem 1 *A utility function* u *is consistent with a rule* r *if for all* $(m_1, m_2) \models r$*, we have*

$$\sum_{i \in S_a(r)} t_i(u_i(m_1) - u_i(m_2)) > \sum_{j \in S_d(r)} t_j(u_j(m_2) - u_j(m_1)) \tag{4}$$

We omit the proof.

This theorem has the following important consequence.

Corollary 1 *In a utility function* u *of the form (2) over a partition of* $F(C)$ *consistent with* C^**, each rule* $r \in C^*$ *must agree with some subutility function* u_i*.*

Otherwise we have $|S_a(r)| = 0$ and $|S_d(r)| \geq 1$, and Equation (4) will not be satisfied.

Choosing scaling parameters

Once we have a collection of utility independent sets S_i, we create subutility functions as described above. We can then choose scaling parameters t_i based on which rules disagree with each subutility function u_i. There are several possible strategies for choosing these parameters. Due to space constraints, we mention briefly the simplest case, then present the most general method.

If no subutility functions disagree with any rules, we have the following case. For any rule r and all i such that $r \in R_i$, u_i agrees with r. We have: $u_i(m_1) > u_i(m_2)$ for each u_i, and each pair $(m_1, m_2) \models r$. Then 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 $t_i > 0$.

Constraint satisfaction If we have several UI sets S_i with conflicts between rules, then we must use a more complicated strategy for assigning scaling parameters t_i, in which case we construct and solve a constraint satisfaction problem from the set of conflicts. Constraint solvers exist that can quickly handle problems with tens of thousands of terms (Selman, Levesque, & Mitchell 1992). Although this may be costly, this price is paid only when the utility function is constructed. The evaluation of the utility function on models of $\mathcal{L}(\mathcal{V})$ is independent of the construction cost.

Corollary 1 states that each rule r must agree with some subutility function u_i. For each utility independent feature set S_i, let $Y_{i1}, Y_{i2}, ..., Y_{iK}$ be sets of conflicting rules, where each is composed of rules $r \in R_i$. Each set Y_{ik} represents a cycle in $G_i(C^*)$; and let Y be the set of all such conflicts. Let $R_c = \cup_{i,k} Y_{ik}$ be the set of all rules involved in any conflict.

Our satisfiability problem uses a conjunction of disjunctions (conjunctive normal form) of propositional variables z_{ij}, with z_{ij} interpreted as stating that 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 the formula are of one of two forms: those representing the subutility functions indicated by X_j; and those representing conflicts between rules of a particular cycle Y_{ik}. Those of the first category are of form $\vee_{i \in X_j} z_{ij}$. This represents the possible subutility functions a rule might agree with, accordingly in a solution to the formula, one of these variables must be true.

The other conjuncts represent a particular cycle on a particular UI set. If Y_{ik} is a conflict set on S_i, then the conjunct is of the form $\vee_{j: r_j \in Y_{ik}} \neg z_{ij}$. That is, u_i must disagree with at least one of the rules in Y_{ik}. Combining conjunctions of both types, we can write the entire satisfiability formula as

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

Consider an example. Suppose we have sets S_1, S_2 with 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. We make a satisfiability formula as so:

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

Once we have this formula, we use a solver to arrive at a solution. It can be shown that the satisfiability problem is always satisfiable if the preferences C^* admit an additive decomposition utility function (McGeachie 2002). A solution to the satisfiability problem can be translated back into a construction for the subutility functions using the translation defined above, we simply look at which subutility functions disagree with which rules. We then remove those rules from the relevant sets R_i, then construct u_i according to R_i, which is now conflict-free.

Linear inequalities The satisfiability solver tells us which rules disagree with which subutility functions. However, we must then set the scaling parameters, t_i, so as to satisfy Equation (4). We can do so by building a list of constraints on the values that the scaling parameters may assume.

Suppose subutility functions for the utility-independent sets $S_d(r)$ disagreed with r, while subutility functions for the utility-independent sets $S_a(r)$ agreed with r. Then we add an inequality of the form

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

to a list I of inequalities for each pair $(m_1, m_2) \models r$. The inequalities I constrain the total utility function to assign higher utility to m_1 than to m_2. Note that the inequality need not contain terms for u_j for j not in $S_d(r)$ or $S_a(r)$, since, by definition of $(m_1, m_2) \models r$, we have $m_1 \upharpoonright S_j = m_2 \upharpoonright S_j$. The total number of linear inequalities is far less than $2^{F(C)}$. Specifically, each rule r contributes

$$\prod_{S_i: s(r) \cap S_i \neq \emptyset} 2^{|S_i \setminus s(r)|} \tag{5}$$

inequalities to the set I of inequalities. One obtains (5) by noticing the number of inequalities contributed by a rule is the number of different values of $u_i(m_1)$ - $u_i(m_2)$ for $i \in (S_d(r) \cup S_a(r))$ and $(m_1, m_2) \models r$.

We solve the system of linear inequalities I 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. For example, see (Chvátal 1983).

It can be shown that if the system of linear inequalities, I, has a solution, this solution corresponds to a utility function u consistent with C^*. If the inequalities prove unsatisfiable, it may be because the preferences do not admit a linear scaling between subutility functions. We can either follow (McGeachie & Doyle 2002) and use the minimizing utility function over the degenerate partition $S = \{F(C)\}$ or, less drastically, join partition elements together to perform constraint satisfaction and linear programming on a smaller problem (a solution used in (Boutilier, Bacchus, & Brafman 2001) to solve the same problem). Work in progress addresses optimal choice of which feature sets to join together.

Conclusions

We have summarized a method for constructing numeric utility-function representations of preferences specified using the *ceteris paribus* preference logic of (Doyle, Shoham, & Wellman 1991). We sketch the complexity of our methods, then compare to similar *ceteris paribus* reasoning systems.

There are two different efficiency questions to ask. One is the time and space required to construct u, the other is the time and space required to evaluate u on a particular model m. Constructing u involves first computing the utility independent partition of features, which takes time $O(|C^*|^2)$. Then we solve a satisfiability problem, where time required depends on the number of rules involved in conflicts on each UI set. In pathological cases, the number of conflicts could be as high as $|C^*|!$, if every rule conflicts with every other set of rules. Finally we solve a system of linear inequalities, where the number of inequalities is

$$\sum_{r \in C^*} \prod_{S_i : s(r) \cap S_i \neq \emptyset} 2^{|S_i \setminus s(r)|}$$

Computing $u(m)$ involves computing $u_i(m)$, where u_i is a minimizing utility function, for all the UI feature sets. Each of the subutility functions can be exponential in the size of the UI set, $|S_i|$; in the worst case, this means exponential in $|F(C)|$.

Since we assume our input preferences $|C|$ are not inherently contradictory, we anticipate in practice there will be manageable numbers of conflicting rules. If in addition, each set S_i is of size $\log N$, our algorithm requires polynomial time and space. There are less than $|C^*| * \prod_{i=1}^{Q} 2^{\log N}$ linear inequalities, so construction of u takes polynomial time. Each model graph $G_i(C^*)$ has at most $2^{\log N}$ nodes, so evaluating $u(m)$ takes time $O(k * N)$.

Other researchers have proposed methods of computing with different representations of *ceteris paribus* preferences. The work of (Bacchus & Grove 1995) and (Mura & Shoham 1999) use quantified preference statements, in contrast to our qualitative preference statements. Our work is similar to the work of Boutilier, Bacchus, Brafman (2001), who compute an additive decomposition utility function from quantified preferences. Their method uses subutility functions that are exponential-size lookup tables, which is similar to our exponential-size graph theoretic subutility functions. Earlier work by (Boutilier *et al.* 1999) uses qualitative conditional *ceteris paribus* preferences, and presents some strong heuristics for computing dominance queries, but uses a very different representation for *ceteris paribus* preferences.

Acknowledgements

This work was supported in part by DARPA under contract F30602-99-1-0509. Michael McGeachie is supported in part by a training grant from the National Library of Medicine, and a grant from the Pfizer corporation.

References

Bacchus, F., and Grove, A. 1995. Graphical models for preference and utility. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 3–19. Morgan Kaufmann.

Bacchus, F., and Grove, A. 1996. Utility independence in a qualitative decision theory. In *Proceedings of the Fifth International Conference on Knowledge Representation and Reasoning*, 542–552. Morgan Kaufmann.

Boutilier, C.; Bacchus, F.; and Brafman, R. L. 2001. Ucp-networks: A directed graphical representation of conditional utilities. In *Proceedings of Seventeenth Conference on Uncertainty in Artificial Intelligence*. To Appear.

Boutilier, C.; Brafman, R. I.; Hoos, H. H.; and Poole, D. 1999. Reasoning with conditional ceteris paribus preference statements. In *Proceedings of Uncertainty in Artificial Intelligence 1999 (UAI-99)*.

Chvátal, V. 1983. *Linear Programming*. New York: W.H. Freeman and Company.

Doyle, J., and Thomason, R. H. 1999. Background to qualitative decision theory. *AI Magazine* 20(2):55–68.

Doyle, J.; Shoham, Y.; and Wellman, M. P. 1991. A logic of relative desire (preliminary report). In Ras, Z., ed., *Proceedings of the Sixth International Symposium on Methodologies for Intelligent Systems*, Lecture Notes in Computer Science. Berlin: Springer-Verlag.

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

McGeachie, M., and Doyle, J. 2002. Utility functions for ceteris paribus preferences. Submitted for publication.

McGeachie, M. 2002. Utility functions for ceteris paribus preferences. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts. In preparation.

Mura, P. L., and Shoham, Y. 1999. Expected utility networks. In *Proc. of 15th conference on Uncertainty in Artificial Intelligence*, 366–373.

Selman, B.; Levesque, H. J.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In Rosenbloom, P., and Szolovits, P., eds., *Proceedings of the Tenth National Conference on Artificial Intelligence*, 440–446. Menlo Park, California: AAAI Press.