A Constraint-Based Approach to Preference Elicitation and Decision Making

Craig Boutilier and **Ronen Brafman** and **Chris Geib** and **David Poole**
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z4 Canada

Abstract

We investigate the solution of constraint-based *configuration problems* in which the preference function over outcomes is unknown or incompletely specified. The aim is to configure a system, such as a personal computer, so that it will be optimal for a given user. The goal of this project is to develop algorithms that generate the most preferred feasible configuration by posing preference queries to the user. In order to minimize the number and the complexity of preference queries posed to the user, the algorithm reasons about the user's preferences while taking into account constraints over the set of feasible configurations. We assume that the user can structure their preferences in a particular way that, while natural in many settings, can be exploited during the optimization process. We also address in a preliminary fashion the trade-offs between computational effort in the solution of a problem and the degree of interaction with the user.

1 Introduction

There has been considerable interest in the area of automated decision analysis and the development of automated decision making aids. Such tools can be used to act on behalf of a user, make decisions which are communicated to the user, or simply help a user in the process of formulating and solving a particular *decision problem*. The specification of a decision problem generally requires four components: an estimate of the *current state* or conditions under which the decision is to be made; a set of *actions* or decisions that can be taken; a *model of the system dynamics* which describes the potential outcomes of any action; and a set or *preferences* qualifying the relative goodness of particular outcomes.

Extracting these four types of information can be extremely difficult and time consuming, and human decision analysts have developed sophisticated techniques to help elicit this information from decision makers [8]. For many application domains, however, once a model of the system is known, it is unlikely to change in substantial ways. This can provide considerable leverage in the construction of automated decision making agents (DAs) for a given application; the models can be "hard-wired" into the DA program. This is precisely the type of situation we wish to address, namely, the development of DAs for particular applications that can be used by any of a number of different users to solve their decision problems within that domain.

More difficult to deal with in such a model are user preferences. While many users may be faced with the "same" decision scenario, these users will not generally have the same preferences over decision outcomes.[1] In order to make the right choices, a DA must interact with the user to determine their preferences over outcomes. This form of *preference elicitation* is yet another role often filled by human decision analysts, who rely on sophisticated, intuitive techniques for eliciting preferences. Any intelligent DA must have the ability to extract user preferences on a case by case basis. This type of information cannot be hard-wired or precompiled, since the aim is to act on the behalf of, or advise, a *particular* user.

Preference elicitation is a very difficult task in general and is a key focus in work on decision analysis [10, 8, 6]. Automating the process of preference extraction can be very difficult. Straightforward approaches involving the direct comparison of all pairs of outcomes are generally infeasible for a number of reasons, including the exponential number of outcomes (in the number of relevant variables or attributes for which preferences are indicated) and the complexity of the questions that are asked (the comparison of complete outcomes). There has been considerable work on exploiting the structure of preferences and utility functions in a way that allows them to be appropriately decomposed [10, 1, 2]. For instance, if certain attributes are preferentially independent of others [10], one can assign degrees of preference to individual attribute values without worrying about other attribute values. Furthermore, if one assumes more stringent conditions, often one can construct an additive value function in which each attribute contributes to overall preference to a certain "degree" (the *weight* of that attribute) [10]. For instance, it is common in engineering design problems to make such assumptions and simply require users to assess the

[1] By the "same decision problem" we mean simply that the underlying system (i.e., the actions, dynamics, and initial state) is the same.

weights [4]. This allows the direct tradeoffs between values of different attributes to be assessed concisely.

Models such as these make the preference elicitation process easier by imposing specific requirements on the form of the value function. In general, one must allow users to specify the structure of their preferences in an automated decision making context. One of the problems we address in the paper is the development of a particular graphical model for structuring preferences based on the dependence of attribute value preference on a certain set of attributes and their conditional preferential independence with respect to others. This model will be fairly natural and concise in many settings, and has good computational properties.

Even with good models of preference structure, assessment of an entire preference ranking can involve considerable effort: there will still be a large number of parameters to specify. Furthermore, the effort involved can be "wasted" if we assess preferences (in detail) over outcomes that are infeasible (given the set of actions available), or that are dominated by other feasible outcomes. In general, one would like to explore only regions of the search space (space of possible outcomes) that are feasible and assess preferences over this region. For instance, this is the intuition underlying much of *goal programming* [9, 5] and certain approaches to problem of engineering design [4]. Algorithms that generate nondominated (e.g., Pareto optimal) outcomes that are feasible have been developed; a user is then presented with this set and asked to rank them or determine the most preferred alternative from this set. The second contribution of this paper is the development of an algorithm that serves a similar purpose using our particular graphical model of preferences. We assume that the set of possible actions (and implicitly the set of outcomes) is represented by a set of constraints that determine feasible configurations. We then explore the set of feasible outcomes using a constraint-based optimizatizing (branch-and-bound) search algorithm to identify nondominated outcomes given the constraints on preferences imposed by the model. Then preferences over these feasible, nondominated alternatives can be assessed directly. As part of this, we use a novel algorithm for testing dominance given preference information (essentially statements of conditional preferential independence) contained in our model.

Finally we note that while search can determine the set of feasible outcomes, thus minimizing the degree of intrusion upon a user required for preference assessment, the converse effect also exists: preference information can be used to drastically prune the search space. It is often easy to tell that all outcomes below a particular branch is a search tree are dominated by an already enumerated feasible outcome. Thus there is a tension between the two desiderata of minimizing user effort and reducing computation time. The search algorithm we develop does prune the search space to whatever extent possible given the information in the graphical preference model (which can often be considerable). However, we suggest that this points to a clear need for *interactive search algorithms* in which the objective function in a particular region of the search space can be obtained through user queries in an effort to minimize computational effort. This observation is not novel (see work, e.g., on interactive goal programming [5]). There are clear tradeoffs involved between the number and complexity of the user queries required to prune part of the search space and the amount of search time (or size of the search space) expected to be pruned. The final contribution of this paper, somewhat more speculative, is a set of suggestions for how such a process might be realized in the context of our search algorithm, and what sorts of preference structure and queries might best support this goal. The ultimate aim of this line of research is the development of on-line optimization procedures in which the underlying objective is initially unknown (or partially known), where the objective function can be "filled in" through interaction with a user, and where user queries about preferences are minimized (in number and complexity) subject to the need for effective search.

The paper is structured as follows. In Section 2, we describe the necessary background on preference functions. In Section 3 we define constraint-based configuration problems in an effort to make our task more concrete. We define our graphical preference model, *CP-networks*, in Section 4 and describe its semantics in terms of *ceteris paribus* or *conditional preferential independence* statements. We also present an algorithm for domination testing using such a model. Section 5 deals with optimization using CP-models and describes a branch-and-bound algorithm for determining the set of nondominated feasible outcomes in an efficient manner. Section 6 offers some suggestions for making the search algorithm interactive, and we offer some thoughts on future research in Section 7.

2 Preferences and Optimization

We focus our attention on single-stage decision problems with complete information, ignoring in this paper any issues that arise in multi-stage, sequential decision analysis and any considerations of risk that arise in the context of uncertainty.[2] We begin with an outline of the relevant notions from decision theory. We assume that the world can be in one of a number of *states* $\mathcal{S}$ and at each state s there are a number of *actions* $\mathcal{A}_s$ that can be performed. Each action, when performed at a state, has a specific *outcome* (we do not concern ourselves with uncertainty in action effects or knowledge of the state). The set of all outcomes is denoted $\mathcal{O}$. A *preference ranking* is a total preorder $\succeq$ over the set of outcomes: $o_1 \succeq o_2$ means that outcome o_1 is equally or more preferred to the decision maker than o_2. The aim of decision making under

[2] Such issues include assigning preferences to *sequences* of outcome states, assessing uncertainty in beliefs and system dynamics, and assessing the user's attitude towards risk.

certainty is, given knowledge of a specific state, to choose the action that has the most preferred outcome. We note that the ordering $\succeq$ will be different for different decision makers. For instance, two different customers might have radically different preferences for different types of computer systems that a sales program is helping them configure. Often, for a state s, certain outcomes in $\mathcal{O}$ cannot result from any action $a \in \mathcal{A}_s$: those outcomes that can obtain are called *feasible outcomes* (given s).

What makes the decision problem difficult is the fact that outcomes of actions and preferences are not usually represented so directly. We focus here on preferences. We assume a set of *features* (or variables or attributes) $F = \{F_1, \cdots F_n\}$ over which the decision maker has preferences. Each feature F_i is associated with a domain of *feature values* $\mathcal{F}_i = \{f_1^i, \cdots f_{n_i}^i\}$ it can take. The product space $\mathcal{F} = \mathcal{F}_1 \times \cdots \times \mathcal{F}_n$ is the set of outcomes. Thus direct assessment of a preference function is usually infeasible due to the exponential size of $\mathcal{F}$. We denote a particular assignment of values to a set $X \subseteq F$ as $\overline{x}$, and the concatenation of two such partial assignments to X and Y ($X \cap Y = \emptyset$) by $\overline{xy}$. If $X \cup Y = F$, $\overline{xy}$ is a (complete) outcome.

Fortunately, a preference function can be specified (or partially specified) concisely if it exhibits sufficient structure. We describe certain types of structure here, referring to [10] for a detailed description of these (and other structural forms) and a discussion of their implications. These notions are standard in multi-attribute utility theory. A set of features X is *preferentially independent* of its complement $Y = F - X$ iff, for all $\overline{x}_1, \overline{x}_2, \overline{y}_1, \overline{y}_2$, we have

$$\overline{x}_1\overline{y}_1 \succeq \overline{x}_2\overline{y}_1 \text{ iff } \overline{x}_1\overline{y}_2 \succeq \overline{x}_2\overline{y}_2$$

In other words, the structure of the preference relation over assignments to X, when all other variables are held fixed, is the same no matter what values these other variables take. If the relation above holds, we say $\overline{x}_1$ is preferred to $\overline{x}_2$ *ceteris paribus*. Thus, one can assess the relative preferences over assignments to X once, knowing these preferences do not change as other attributes vary. We can define conditional preferential independence analogously. Let X, Y and Z partition F (each set is nonempty). X and Y are *conditionally preferentially independent* given $\overline{z}$ iff, for all $\overline{x}_1, \overline{x}_2, \overline{y}_1, \overline{y}_2$, we have

$$\overline{x}_1\overline{y}_1\overline{z} \succeq \overline{x}_2\overline{y}_1\overline{z} \text{ iff } \overline{x}_1\overline{y}_2\overline{z} \succeq \overline{x}_2\overline{y}_2\overline{z}$$

In other words, the preferential independence of X and Y only holds when Z is assigned $\overline{z}$. If this relation holds for all assignments $\overline{z}$, we say X and Y are *conditionally preferentially independent* given Z.

This decomposability of a preference functions often allows one to identify the most preferred outcomes rather readily. Unfortunately, the *ceteris paribus* component of these definitions ensures that the statements one makes are relatively weak. In particular, they do not imply a stance on specific value tradeoffs. For instance, suppose two variables A and B are preferentially independent so that the preferences for values of A and B can be assessed separately; e.g., suppose $a_1 \succ a_2$ and $b_1 \succ b_2$. Clearly, a_1b_1 is the most preferred outcome and a_2b_2 is the least; but if feasibility constraints make a_1b_1 impossible, we must be satisfied with one of a_1b_2 or a_2b_1. We cannot tell which is most preferred using these separate assessments. However, under stronger conditions (e.g., *mutual preferential independence*) one can construct an additive value function in which weights are assigned to different attributes (or attribute groups). This is especially appropriate when attributes take on numerical values. We refer to [10] for a discussion of this problem.

Given such a specification of preferences, a number of different techniques can be used to search the space of feasible outcomes for a most preferred outcome.

3 The Configuration Problem

For concreteness, we focus on *configuration problems* with full information. A configuration problem is one in which the decision to be taken consists of a number of aspects, each of which must be decided on—that is, one chooses a *configuration*—and which interact in potentially complex ways to determine the outcome of the decision.

Intuitively, one can view such a decision problem as the problem of "configuring a system." The decision maker is forced to choosing a number of different components, from a variety of options, which when put together determine just how good the resulting system is. For example, one could configure a computer system by choosing a processor from a particular set of options, a specific amount of memory, certain peripherals, and so on. Just how good this system is depends on the needs and preferences of the user for whom the system is being configured. These preferences will often be specified with respect to set of *features* or attributes (as discussed in the previous section), these being determined by the chosen configuration. This induces *indirect* preferences over configurations.

We formalize the problem as follows:

Definition We assume the existence of a finite set of *component attributes* $C_1, \cdots C_n$. With each attribute C_i we associate a domain, or finite set of *component values*, $\mathcal{C}_i = \{c_1^i, \cdots c_{n_i}^i\}$. The *configuration space* for a given problem is the set $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$. Each element of $\mathcal{C}$ is a *configuration*.

For instance, the domain of component *processor* might be $\{486, pent75, pent100, pent133\}$, while the component *games* might have values $none$, $basic$, and $advanced$.

Definition We assume the existence of a finite set of *feature attributes* $F_1, \cdots F_n$. With each attribute F_i we associate a domain, or finite set of *feature values*, $\mathcal{F}_i = \{f_1^i, \cdots f_{n_i}^i\}$. The *outcome space* or *feature space* for a given problem is the set $\mathcal{F} = \mathcal{F}_1 \times \cdots \times \mathcal{F}_n$. Each element of $\mathcal{F}$ is an *outcome*.

As an example, we might have the feature *desk-top-publ* with values $\{rudmntry, basic, advncd, highvol\}$. Again we emphasize that the distinction between features and components is that components represent the controllable aspects of the decision problem (essentially the action space), while features represent those qualities of an overall configuration for which a user can readily (or more easily) articulate direct preferences. There is no reason that $C_i = F_j$ is forbidden—a user may have direct preferences over certain component values; but generally, we expect these sets to have little overlap.

Configurations are bound to satisfy certain constraints. We will take a logical approach to the specification of constraints and their solution—in particular treating them as satisfiability problems (see [11])—though more classic CSP formulations could also be used directly.

Definition The set of *configuration constraints* $Cons_{\mathcal{C}}$ is a set of logical constraints over the possible assignments of values to different components. The set of *feasible configurations* $\mathcal{C}_F \subseteq \mathcal{C}$ consists of those configurations that satisfy all constraints in $Cons_{\mathcal{C}}$.

An example of a configuration constraint might be

$$C_1 = c_1^1 \vee C_3 \in \{c_1^3, c_5^3\}$$

Definition A *causal model* M is a mapping $M : \mathcal{C} \to \mathcal{F}$. We assume that M is total and deterministic. Generally, the model M will be represented by a set of *causal rules* (i.e., logical rules) $\mathcal{M}$.

Two examples of such causal rules are:

$$C_1 = c_1^1 \supset F_1 = f_1^1$$

$$C_1 \in \{c_2^1, c_3^1\} \wedge C_3 \neq c_5^3 \supset F_1 = f_1^1$$

We will assume that the set of causal rules given is consistent and complete. We note that $Cons_{\mathcal{C}}$ and M together determine which elements of feature space are "reachable." We define the set of *feasible outcomes* $\mathcal{F}_F \subseteq \mathcal{F}$ to be the image of $\mathcal{C}_F$ under the mapping M; that is, $M(\mathcal{C}_F) = \mathcal{F}_F$.

Given a preference ranking $\succeq$ over feature space, an *optimal configuration* is any $c \in \mathcal{C}_F$ such that $M(c) \succeq f$ for any $f \in \mathcal{F}_F$. In other words, we would like a feasible configuration that determines a (necessarily feasible) outcome that is at least as good as any other feasible outcome.

Our configuration problem can be stated concisely as follows: given a set of components and features, a set of configuration constraints and a causal model; given the ability to ask preference queries over $\mathcal{F}$; determine an optimal feasible configuration.

4 CP-Networks

We now define a representation for user preferences that is somewhat graphical in nature, and exploits conditional preferential independence in structuring a user's preferences. The model is similar to a Bayesian network on the surface, however, the nature of the relation between nodes within a network will generally be quite weak (e.g., compared with the probabilistic relations that exist in Bayes nets). Others have defined graphical representations of preference relations; for instance Bacchus and Grove [1, 2] have shown some strong results pertaining to undirected graphical representations of additive independence. Our representation and semantics is rather distinct, and our main aim in using the graph is to capture statements of conditional preferential independence.

For each feature F, we ask the user to identify a set of *parent* features $P(F)$ that can affect her preference over various F values. That is, given a particular value assignment to $P(F)$, the user should be able to determine a preference order for the values of F, all other things being equal. Formally, denoting all other features aside from F and $P(F)$ by $\overline{F}$, we have that F and $\overline{F}$ are conditionally preferentially independent given $P(F)$. Given this information, we ask the user to explicitly specify her preferences over F values for all possible $P(F)$ values. We use the above information to create an annotated graph in which each feature F has $P(F)$ as its set of parents. The node F is annotated with a table describing the user's preferences over her values given every combination of parent values. We call these structures *conditional preference networks* (or CP-networks).

Example 1 Asking the user to describe her preference over feature B, we are told that this preference depends on the value for A and on that value alone (*ceteris paribus*). We then make A a parent of B and ask about her preference on B for each value of A. She may say that, when a holds, she prefers b over $\overline{b}$, and when $\overline{a}$ holds she prefers $\overline{b}$ over b, *ceteris paribus*. This is written here as:

$$a : b \succ \overline{b}$$
$$\overline{a} : \overline{b} \succ b$$

In this paper, we show how to exploit the information contained in CP-networks when the dependency graph is a polytree (a DAG with at most one path between any two nodes when the arcs are treated as undirected edges) and features are binary. We note that nothing in the semantics forces the graph to be acyclic.

Example 2 Suppose we have two features A and B, where A is a parent of B and A has no parents. Assume the following conditional preferences:

$$a \succ \overline{a}; \quad a : b \succ \overline{b}; \quad \overline{a} : \overline{b} \succ b$$

Somewhat surprisingly, this information is sufficient to totally order the outcomes:

$$ab \succ a\overline{b} \succ \overline{a}\overline{b} \succ \overline{a}b.$$

Notice that we can judge each outcome in terms of the conditional preferences it violates. The ab outcome violates none of the preference constraints. Outcome $a\overline{b}$ violates the conditional preference for B. Outcome $\overline{a}\overline{b}$ violates the preference for A. Outcome $\overline{a}b$ violates both. What is surprising is that the *ceteris paribus* semantics implies that violating the A constraint is worse than violating the B constraint (we have $a\overline{b} \succ \overline{a}\overline{b}$). That is, the parent preferences have higher priority than the child preferences.

Example 3 Suppose we have three features A, B, and C, and suppose that the preference dependency graph is disconnected. Let's assume that $a \succ \overline{a}$, $b \succ \overline{b}$, and $c \succ \overline{c}$. Given this information we can conclude that abc is the most preferred outcome, then comes $\overline{a}bc$, $a\overline{b}c$, and $ab\overline{c}$. These three cannot be ordered based on the information provided. Less preferred than the last two is $a\overline{b}\overline{c}$, and so on. The least preferred outcome is $\overline{a}\overline{b}\overline{c}$.

Example 4 Suppose we have three features A, B, and C, and the conditional preference graph forms a chain with A having no parents, A the parent of B, and B the parent of C. Suppose we have the following dependence information:

$$a \succ \overline{a}; \quad a : b \succ \overline{b}; \quad \overline{a} : \overline{b} \succ b; \quad b : c \succ \overline{c}; \quad \overline{b} : \overline{c} \succ c$$

These preference constraints imply the following ordering:

$$abc \succ ab\overline{c} \succ a\overline{b}\overline{c} \succ a\overline{b}c \succ \overline{a}\overline{b}c \succ \overline{a}bc \succ \overline{a}b\overline{c},$$

which totally orders all but one of the outcomes. Notice how we get from one outcome to the next in the chain: we *flip* (or exchange) the value of exactly one feature according to the preference dependency information. The element not in this chain is $\overline{a}\overline{b}\overline{c}$, and we can derive the ordering $a\overline{b}\overline{c} \succ \overline{a}\overline{b}\overline{c} \succ \overline{a}\overline{b}c$. Thus, the only two outcomes not totally ordered are $\overline{a}\overline{b}\overline{c}$ and $a\overline{b}c$. From example 2, we saw that violations of preference constraints for parent features are worse than violations of constrains over child feature preferences. In one of the two unordered outcomes we violate the preference over the most important feature (namely A), while in the other outcome we violate preference over two less important features (B and C). The semantics of the CP-networks does not specify which of these tuples has preference over the other.

There are two important things to notice about these examples. First, a chain of "flipping feature values" can be used to show that one outcome is better than another. Second, violations are worse (i.e., have a larger negative impact on preference) the higher up they are in the network, although we cannot compare two (or more) lower level violations to violation of a single ancestor constraint. These observations are exploited in the algorithm described below for deciding whether one outcome is preferred to another, given the preference constraints represented in a CP-networks. Consider now the following more general example.

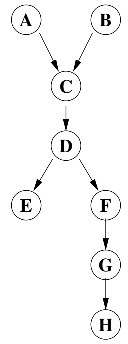

Figure 1: An Example Conditional Preference Graph

Example 5 Consider the preference graph of Figure 1. Suppose that the conditional preferences are:

$$\begin{gathered}
a \succ \overline{a}; \quad b \succ \overline{b}; \\
(a \wedge b) \vee (\overline{a} \wedge \overline{b}) : c \succ \overline{c}; \quad (a \wedge \overline{b}) \vee (\overline{a} \wedge b) : \overline{c} \succ c \\
c : d \succ \overline{d}; \quad \overline{c} : \overline{d} \succ d; \quad d : e \succ \overline{e}; \quad \overline{d} : \overline{e} \succ e; \\
d : f \succ \overline{f}; \quad \overline{d} : \overline{f} \succ f; \quad f : g \succ \overline{g}; \quad \overline{f} : \overline{g} \succ g \\
g : h \succ \overline{h}; \quad \overline{g} : \overline{h} \succ h
\end{gathered}$$

We illustrate the intuitions behind out algorithm for dominance testing through a sequence of examples.

Suppose we want to compare outcome $abcdef\overline{g}\overline{h}$ (which violates the g preference) and outcome $\overline{a}b\overline{c}\overline{d}\overline{e}\overline{f}\overline{g}\overline{h}$ (which violates the a preference). In order to show that the first is preferred, we generate the sequence: $\overline{a}b\overline{c}\overline{d}\overline{e}\overline{f}\overline{g}\overline{h} \prec ab\overline{c}\overline{d}\overline{e}\overline{f}\overline{g}\overline{h} \prec abc\overline{d}\overline{e}\overline{f}\overline{g}\overline{h} \prec abcde\overline{f}\overline{g}\overline{h} \prec abcdef\overline{g}\overline{h} \prec abcdef\overline{g}\overline{h}$. Intuitively, we constructed a sequence of increasingly preferred outcomes, using only valid conditional independence relations represented in the CP-network, by *flipping* values of variables. We are allowed to change the value of a "higher priority" variable (higher in the network) to its preferred value, even if this introduces a new preference violation for some lower priority variable (a descendent in the network). For instance, the first flip of A's value in this sequence to its preferred state repairs the violation of A's preference constraint, while introducing a preference violation with respect to C (the value $\overline{c}$ is dispreferred when ab holds). This process is repeated (e.g., making C take its conditionally most preferred value at the expense of violating the preference for D) until the single preference violation of F (in the "target" outcome) is shown to be preferred to the single preference violation of A (in the initial outcome). This demonstrates how the violation of conditional preference for a specific value of

Input: (1) CP-network P for features $F_1, \ldots, F_n$ (topologically sorted).
(2) Two feature-value vectors $c_1 = (f_1^1, \ldots, f_n^1), c_2 = (f_1^2, \ldots, f_n^2)$.
Output: YES – c_1 is preferred to c_2 according to P.
NO – c_1 is not preferred to c_2 according to P.

For $i = 1$ to n {
  Let $\overline{F} = (F_{l_1}, \ldots, F_{l_k})$ be F_i's parents in P.
  Let $c_{j|\overline{F}} = (f_{l_1}^j, \ldots, f_{l_k}^j)$ for $j = 1, 2$.
  Let $X_i =$ Flips$(F_i, (X_{l_1}, \ldots, X_{l_k}), c_{1|\overline{F}}, c_{2|\overline{F}})$ +
      (X_i-violation in c_2) − (F_i-violation in c_1).
  If $X_i < 0$ Return NO }
  Return YES.

Figure 2: Algorithm for Dominance Checking.

some variables is dispreferred to the violation of one of its descendent's preferences.

Suppose we compare $abcdef\bar{g}h$ (which violates the G preference and the H preference) and $\bar{a}b\bar{c}\bar{d}\bar{e}\bar{f}\bar{g}\bar{h}$ (which violates the A preference). These turn out not to be comparable (neither is preferred to the other). The sequence of flips above cannot be extended to change the values of both G and H so that their preference constraints are violated. The sole violation of the A constraint cannot be dominated by the violation of two (or more) descendents *in a chain*.

If we want to compare $abcd\bar{e}f\bar{g}\bar{h}$ (which violates the E preference and the G preference) and $\bar{a}b\bar{c}\bar{d}\bar{e}\bar{f}\bar{g}\bar{h}$ (which violates the A preference), we can use the following sequence: $\bar{a}b\bar{c}\bar{d}\bar{e}\bar{f}\bar{g}\bar{h} \prec ab\bar{c}\bar{d}\bar{e}\bar{f}\bar{g}\bar{h} \prec abc\bar{d}\bar{e}\bar{f}\bar{g}\bar{h} \prec abcd\bar{e}\bar{f}\bar{g}\bar{h} \prec abcd\bar{e}f\bar{g}\bar{h}$. The violation of E and G is preferred to the violation of A: intuitively, the A violation can be absorbed by violation in *each path starting at* D.

Now consider the comparison of $abcdef\bar{g}h$ (which violates the G and H preferences) and $\bar{a}\bar{b}cdefgh$ (which violates the A and B preferences). We can use the following sequence of flips to show preference: $\bar{a}\bar{b}cdefgh \prec a\bar{b}cdefgh \prec a\bar{b}\bar{c}defgh \prec a\bar{b}\bar{c}\bar{d}efgh \prec a\bar{b}\bar{c}\bar{d}e\bar{f}gh \prec a\bar{b}\bar{c}\bar{d}e\bar{f}\bar{g}h \prec ab\bar{c}\bar{d}e\bar{f}\bar{g}h \prec abc\bar{d}e\bar{f}\bar{g}h \prec abcde\bar{f}\bar{g}h \prec abcdef\bar{g}h$. This shows how two violations in ancestor variables can be used to cover two violations in their descendents.

These examples illustrate how certain preference violations have priority over others in determining the relative ordering of two outcomes. Intuitively, dominance is shown by constructing a sequence of legal flips from the initial outcome to the target. However, we can see that we merely need to count the number of flips needed in a downward "sweep" through the network to verify that such a sequence exists. The algorithm in Figure 2 makes these intuitions precise.

Figure 2 presents an algorithm for checking whether one tuple of feature values c_1 is preferred to another tuple c_2 given a CP-network whose graph is singly connected. The algorithm returns YES whenever c_1 is preferred to c_2 and it returns NO if this relation cannot be determined. We note that it is a trivial matter, given c_1 and c_2 to rule out one (or sometimes both) of the possibilities $c_1 \succ c_2$ or $c_2 \succ c_1$.[3]

The algorithm works as follows: using a topological sort of the CP-network, we assign X values to each of the features. Intuitively, the X value of feature F_i is the maximal number of *exchanges* (or flips) in the value of this feature along any sequence of increasingly preferred outcomes (feature tuples) commencing with c_2 and terminating at c_1. In order to calculate this value, we use the `Flips` function. This function has the following inputs: (1) the (initial) value of F_i's parents at c_2, (2) the (final) value of F_i's parents at c_1, (3) the number of flips that F_i's parents can make (i.e., their X values), and, implicitly (4) the dependence of F_i's value on its parents' values, as recored in the CP-network. This function calculates the maximal number of value changes that F_i's parents can induce in F_i given that they start in their c_2 values, they end in their c_1 values, and *en route*, each parent F_{l_j} cannot change its values more than X_{l_j} times. We can calculate the `Flips` value by enumerating all possible "paths" that satisfy these constraints and counting the number of times the preferred value of F_i changes as a function of its parents. In order to calculate the X value of F_i, we must also consider whether or not its initial c_2 value is a preference violation (in which case we can add another flip, as sanctioned by the *ceteris paribus* semantics) and whether or not is final c_1 value is a preference violation (in which case we must subtract one flip).

Finally, if the X value of some node is negative, then clearly, no increasing sequence of tuples of the above form exists, and c_1 is not preferred to c_2.

Our approach can be generalized to handle multiply-connected graphs, but as in Bayesian nets, the cost of computation can increase substantially. In particular, we cannot only consider a single families in computing the X values. Rather, we must consider all children that share parents together. In that case, we generate maximal (i.e., undominated) vectors of X values for these children. These values depend on all of these children's parents. Then, we must check whether for any of these undominated tuples, the algorithm would return YES. Only when NO is returned for all possible tuples, can we return a negative answer.

We note that this is not the only possible semantics one could use for "networks" representing preferences. For instance, the conditional preference for some attribute A given its parents could be taken to be stronger than indicated by the *ceteris paribus* interpretation—a strict lexicographic notion of importance, such as that discussed in Section 6, could be used. Moreover, there are a number of other ways in which one could structure a preference function. We conjecture,

[3] We can find the first features (in a topological sort of the network), F_1 and F_2 whose preference constraints are violated by c_1 and c_2, respectively. If F_1 is a descendent of F_2, we rule out $c_1 \succ c_2$; if F_2 is a descendent of F_1, we rule out $c_2 \succ c_1$; if neither, then neither outcome is preferred and the dominance algorithm need not be consulted.

however, that such conditional *ceteris paribus* assertions are quite natural in many settings. We expect that models combining different forms of preference statements, including *ceteris paribus* statements, will offer greater expressive power and more natural models.

5 Finding the Best Configuration

We assume that the set of configuration constraints has been given and that a CP-network of the user's preferences over outcome space has been specified. We present a (more or less) non-interactive approach to the problem of finding an optimal configuration in this section, and defer discussion of making the model more interactive to the following section.

Our approach consists of four distinct phases:

1. Compile $Cons_{\mathcal{C}}$ into a set of constraints $Cons_{\mathcal{F}}$ that implicitly represent $\mathcal{F}_F$, the set of feasible outcomes.

2. Determine the set of nondominated feasible outcomes in $f \in \mathcal{F}_F$ (given the specified CP-network).

3. Ask the user to select a most preferred outcome f from the nondominated set.

4. Determine a feasible configuration c for outcome f

We do not address Phase 1 or Phase 4 of the algorithm in detail here. Translating a set of constraints $Cons_{\mathcal{C}}$, via the causal model, into constraints $Cons_{\mathcal{F}}$ over the outcome space, is a straightforward task, as is the abductive process required to obtain a configuration that determines a most preferred feasible outcome. This is not to say there are not interesting issues pertaining to this process (e.g., can a compact set of constraints $Cons_{\mathcal{C}}$ be mapped into a more unwieldy set $Cons_{\mathcal{F}}$); but this is not our central focus here. The one assumption we make is that the set $Cons_{\mathcal{F}}$ of constraints on feasible outcomes is represented in conjunctive normal form.

In this section we describe Phase 2 of the algorithm in detail, and Phase 3 in less detail. The search algorithm is shown in Figure 3. The steps of the algorithm work as follows. We first note that the search for nondominated outcomes proceeds by searching separately for assignments to disconnected components of a particular *problem graph*. More precisely, given a set of variables, a set of CP-arcs denoting conditional independence relations among them and a set of feasibility constraints over those variables, we define the problem graph to be the undirected graph obtained by removing directionality information from the CP-arcs, and adding an undirected arc between any two variables that occur in the same constraint. It is not hard to see:

Proposition 1 *Let G_i be some disconnected components of a problem graph G. The variables within component G_i can be optimized without regard to the values of variables outside that component.*

Input: Connected graph G, context K, constraints C
Output: Feasible, nondominated outcomes given K

Choose any variable V with no parents in G
Let $v_1, v_2, \ldots v_k$ be the preference ordering for the values of V given K
Let $Result = \emptyset$
FOR $i = 1 \ldots k$ DO:
  Assign $V = v_i$
  Reduce constraints C by $V = v_i$ to obtain C_i
  Let Sat_i be elements of C satisfied by $V = v_i$
  IF $Sat_i \subseteq Sat_j$ for some $j < i$
  OR C_i is inconsistent THEN
    Go to end of loop
  ELSE
    Reduce graph G to get G_i
    Let $G_i^1, \cdots G_i^m$ be the connected components of G_i
    FOR $j = 1 \ldots m$ DO:
      Search $G_i^j, K \cup \{V = v_i\}, C_i$
    IF $Search(G_i^j) \neq \emptyset$, for all $j \leq m$ THEN
      Add $v_i \times Search(G_1^j) \times \cdots Search(G_m^j)$ to $Result$
Return $Result$

Figure 3: Search Algorithm for Non-dominated Outcomes

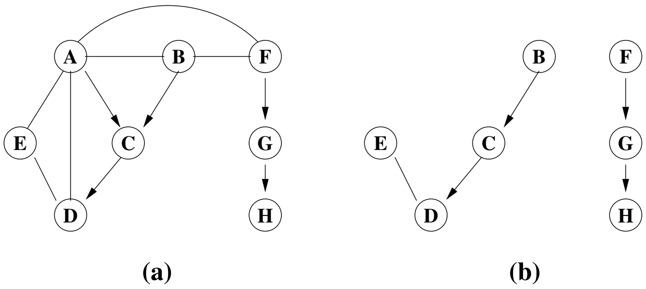

Figure 4: (a) Problem Graph; and (b) Reduced Problem Graph with disconnected components

This gives us a good way of decomposing the search problem into distinct searches over smaller variable sets: the solutions can be pieced together to obtain all nondominated outcomes.

This decomposition can be conditional as well. For instance, consider the graph shown in Figure 4(a), where undirected arcs indicate constraint relations and directed arcs are CP-arcs. Suppose at some point in the search variable A has been instantiated with value a_1 (i.e., we are searching for outcomes where $A = a_1$). In all subsequent search steps (i.e., under that node in the search tree), the CP-arcs that emanate from A can be removed (since there is no longer any choice in the assignment to A, it cannot influence the preference for other child values).[4] Furthermore, all constraints arcs can be removed if they involve A, and any constraint arc between two variables, distinct from A, corresponding to a constraint that is satisfied by $A = a_1$, can also be removed. This can result in new disconnected fragments being obtained, each of which can be optimized independently *given* $A = a_1$. For instance, Figure 4(b) shows such a reduced graph. It is as-

[4] We will see below that A can only be instantiated once all of its CP-parents have been instantiated and removed from the graph.

sumed: that a constraint involving variables A, B and F has been satisfied by $A = a_1$, disconnecting B from F (they can now be optimized independently); and that a constraint involving A, D and E was not satisfied by $A = a_1$, so D and E remain coupled in the reduced constraint.

Our search is a straightforward, depth-first, branch-and-bound style algorithm [7, 14, 13]. The algorithm proceeds by assigning values to variables in a depth-first fashion, using a variable ordering that is consistent with the ordering constraints imposed by CP-arcs (i.e., no child can be assigned before its parents). The value ordering for a variable A is that determined by the preferences for A values given the current partial variables assignment (which must include an assignment to all of A's parents). Whenever a variable A is assigned a value a_i, the set of constraints passed on to the next search node is *reduced*: any constraint satisfied by the assignment is removed from the constraint set, and any constraint involving a disjunct that requires $A \neq a_i$ is made tighter by removing that disjunct. If this assignment causes the current graph (or subgraph) to become disconnected, each fragment invokes an independent search.[5]

We note that there is some pruning that can take place in the search tree. In particular, suppose that the values of A are ordered according to preference as $a_1, \cdots a_k$. If assignment $A = a_j$ satisfies an equal or smaller set of constraints (in a fixed context) than was satisfied by $A = a_i$, $i < j$, then we do not continue to search under $A = a_j$: we can show that any feasible outcome given involving $A = a_j$ is dominated by some feasible outcome involving $A = a_i$, and whether or not this dominating outcome is in the set of nondominated outcomes, any outcome involving $A = a_j$ (and the given context) cannot belong to that set. In essence, the search is a depth-first branch-and-bound, where the set of nondominated alternatives so far generated correspond (in some sense) to our current lower bound.

Finally, we note that when potential nondominated alternatives for a particular subgraph are returned involving some assignment $A = a_j$ (and a given context C), we compare the alternative to all nondominated alternatives (for context C) involving more preferred assignments $A = a_i$, $i < j$. If the nondomination test described in the previous section is passed, the alternative is added to the nondominated set for the current subgraph and context. We note that the domination algorithm is run only on the subgraph, not on all variables in the original problem.

The output of Phase 2 is the set of all outcomes that are potentially most preferred: it contains all and only outcomes that are nondominated given the CP-network specified by the user. Phase 3 then presents these alternatives to the user for selection. We generally expect, for significant problems, that users will have a hard time assessing complicated outcomes involving all variables. There are several strategies one might adopt to make this task more manageable. Apart from dependence, our DA might ask a user to rate variables according to *importance* (we discuss importance in the next section). In addition, the same graph decomposition techniques used in the search algorithm can be used structure the questions about tradeoffs into questions involving much smaller sets of variables.

We note that an algorithm very similar to this one has been proposed by D'Ambrosio and Birmingham [4]. They study "preference-directed design" of devices using a constraint-based approach to the enumeration of the Pareto optimal set for a given problem. The also consider the decomposition of a constraint network as variables are instantiated, rendering variables independent of one another in the optimization process. One key distinction is that their work assumes additive and mutual preferential independence among all variables. Hence they do not deal with dependencies in the preference function.

6 Interactive Search

Given a CP-network of preferences, one can enumerate the set of nondominated outcomes, which can then be presented to the user to determine which of the alternatives is most preferred. The procedure described above adopts a very specific stance on what preference queries should be asked of a user, and when they should be asked. As discussed in the introduction, however, there are tradeoffs one might address when considering whether to ask about a particular preference or to search the set of feasible outcomes to determine if this query really needs to be asked. We discuss how such tradeoffs might be addressed within our framework. These suggestions have a somewhat informal or speculative quality, and are the focus of ongoing research.

One assumption made in basic model is that a complete CP-network is specified by a user before optimization is undertaken. It certainly seems reasonable to expect a user to specify (or answer questions about) the general structure of their preferences. Arguably, a user could be quite prepared to answer questions about dependence structure—the task is not especially onerous. It may be quite another matter to specify complete CPTs associated with each node in the network. For a node A with k parents, there are d^k separate conditional preference functions to assess (where d is the expected domain size for attributes), each involving a ranking of the d values $a_1, \cdots, a_d$ of A.[6] In many circumstances, the com-

[5] We note that other constraint propagation techniques [11] can be used in a straightforward fashion if constraints are represented in classical ways. For instance, constraint propagation techniques are combined with branch-and-bound in partial constraint satisfaction algorithms [7, 14].

[6] We note that these CPTs may exhibit certain regularities that admit compact function representation and ease the assessment problem. For instance, although A may have k parents, the preference ranking over A's values may be the same if any of the A's parents is

plete ranking over all d values of A may not be needed: for instance, we could ask for only the "top" portion of the ranking associated with a certain assignment to A's parents, assuming (or hoping) that any nondominated feasible outcome will only involve these top values (and recall, search proceeds in the order of preference over these values). To be forced to consider elements near the bottom of the ranking means that the problem is very constrained. For similar reasons, we may not want to ask the user *a priori* to assess the conditional rankings associated with each assignment of values to the parents. The combinations of parent values that have low preference (say, componentwise) may never be considered, if we are fortunate and the problem is not overly constrained.

A question that needs to be addressed is just how one can tell *a priori* how much information about a particular set of conditional preferences one is expected to use during optimization. We make only a few informal suggestions here. First, since attributes are instantiated during search in an order consistent with the dependence ordering in the CP-network, we generally expect nodes "deeper" in the network to be more constrained by the time we attempt to find values for them. Thus, we should expect to require more values for these variables to be ranked. Furthermore, there are a number of measures that have been developed in the CSP literature that indicate the tightness of constraints with respect to particular variables. The more constrained a variable is, the more values we expect to have to rank. We note that the tightness of constraints about a particular attribute is a conditional notion: in one context the constraints may be severe, while in another they can be quite relaxed. This can be exploited also: if in a preferred context the constraints on an attribute are less stringent, we can expect to be able to assign that variable more preferred values.

If we do not have a complete instantiation of the CPTs in the CP-network, the search algorithm can find itself at some point lacking the information it needs to proceed. This indicates the interactive nature of the search problem. At any such point, our DA will pose certain queries to the user (e.g., asking the user to rank the values of attribute A given some instantiation of its parents). We expect a number of orthogonal, yet interesting issues to arise in the implementation of such a proposal, including those pertaining to human-computer interaction (e.g., are user's willing to answer preference queries about the same attribute at widely spaced points in time). Issues such as these suggest that asking queries about preferences only when it is assured that this information is needed may not be suitable.

So far the discussion has centered on relaxing the initial requirements of preference information from the user with the hope that this information refers only to dominated or infeasible portions of the search space. The converse question is: what information should one ask in order to *further* prune the search space? One possible way to prune the search space is to ask questions about preference tradeoffs that are not derivable from the network. For instance, imagine we have two root nodes in a network labeled with variables A and B, with $a_1 \succ a_2$ and $b_1 \succ b_2$. While the network contains no information about the relative preference of a_1b_2 and a_2b_1, this information could allow tremendous pruning of the search space. If we can determine that, say, $a_1b_2 \succ a_2b_1$, we can order the search to consider a_1b_2 first, and to prune parts of the search space under a_2b_1 that involve similar (or more stringent constraints) than the "corresponding" a_1b_2 nodes. This kind of tradeoff query can also be applied to questions of "conditional preference violation" at different points in CP-network as well.

One final notion that may be applicable, especially in an informal way, is the notion of *importance* of variables. Suppose variables A and B are preferentially independent of all other variables; let X denote the set of all other variables. We say that A is *more important* than B iff, for all a_i, a_j, $\overline{x}$, and some b_k, if $a_ib_k\overline{x} \succ a_ib_k\overline{x}$, then for all b_l, b_m we have $a_ib_l\overline{x} \succ a_ib_m\overline{x}$. Intuitively, if A is more important than B we are willing to give up any value of B in order to retain a more preferred value of A (this could be viewed as a partial lexicographic ordering applied only to variable A w.r.t. B). For instance, if we have determined that $a_1 \succ a_2 \succ a_3$ and $b_1 \succ b_2 \succ b_3$, then we automatically know that, although for a fixed value of A, this relation over B's values holds, any AB combination with a higher value of A (e.g., a_1b_3) is preferred to one with a lower value of A (e.g., a_2b_1) regardless of B's value. This notion has the obvious conditional and set-based analogs.

If a user is willing to specify that certain attributes are more important in this sense, many tradeoffs that remain unexpressed within the CP-network can be determined with little effort. As a result, certain nondominated outcomes that would be part of the output of the original search algorithm will not be given, and large parts of the search space can be pruned. Of course, the definition of importance imposes stringent requirements that may not often be met in practice. However, one can imagine looser notions of importance involving a subset of the values of some attribute that might be useful; e.g., A may be more important than B with respect to the "jump" between a_2 and a_3, meaning that the user would sacrifice any "amount" of B to prevent a_3. In addition, the notion of importance could be used less precisely to prune the search space in domains where discovery of good outcomes quickly is more important than discovery of an optimal outcome (or in domains where it is expected users have only an imprecise notion of their exact preferences over complicated outcomes). We note that in the course of search more impor-

(say) true, and a second ranking only applies when each of the parents is false. This will certainly ease assessment, and can exploit compact CPT representations such as those used in Bayesian networks [3, 12].

tant variables should be ordered before their less important counterparts, and we conjecture that importance respects dependence ordering (if a "rational" user suggests otherwise, we have not developed a proper set of features).

7 Concluding Remarks

We have sketched a framework for constraint-based optimization in settings where a user's preferences are not known by the decision agent. The key components of our proposal are a graphical model for structuring user preferences based on the notion of conditional preferential independence, and a branch-and-bound algorithm for enumerating nondominated outcomes. In addition, we examined a specific class of problems, configuration problems, in which constraints on feasible actions must first be mapped into constraints on features over which preferences are specified. We have made some preliminary suggestions for the modification of our search algorithm so that it becomes more interactive.

There are a number of very important issues that need to be addressed to make this approach to interactive optimization feasible. As noted earlier, the form of the preference statements required by CP-networks is not the only natural or concise form of preference statement a user could make. Many other types of statements could be offered (including statements of importance as described earlier) and other forms of structuring may be possible. Related to this is the extension of this framework to deal with additive value and utility functions, and continuous variables. Here models such as those proposed in [1, 2] may prove useful, as well as techniques used for goal programming [9, 5].

Within this discrete framework, the use of standard CSP formulations and constraint propagation techniques could prove quite useful in pruning search. A number of interesting human-computer interaction issues also arise. Among these, we would like to investigate: the naturalness of different types of preference statements and queries; how sensitive users are to the ordering of preference queries and the lag between "related" queries; as well as bounds on the number and complexity of the queries required to solve specific classes of problems.

References

[1] F. Bacchus and A. Grove. Graphical models for preference and utility. In *UAI-95*, pp.3–10, Montreal, 1995.

[2] F. Bacchus and A. Grove. Utility independence in qualitative decision theory. In *KR-96*, Cambridge, 1996.

[3] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *UAI-96*, pp.115–123, Portland, OR, 1996.

[4] J. G. D'Ambrosio and W. P. Birmingham. Preference-directed design. *J. Art. Intel. in Eng. Des., Anal. and Manuf.*, 1995.

[5] J. S. Dyer. Interactive goal programming. *Mgmt. Sci.*, 19:62–70, 1972.

[6] S. French. *Decision Theory*. Halsted, 1986.

[7] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Art. Intel.*, 58:21–70, 1992.

[8] R. A. Howard and J. E. Matheson, eds. *Readings on the Principles and Applications of Decision Analysis*. Strategic Decision Group, Menlo Park, CA, 1984.

[9] J. P. Ignizio. *Linear Programming in Single and Multiple Objective Systems*. Prentice-Hall, 1982.

[10] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, 1978.

[11] A. K. Mackworth. The logic of constraint satisfaction. *Art. Intel.*, 58:3–20, 1992.

[12] D. Poole. Exploiting the rule structure for decision making within the independent choice logic. In *UAI-95*, pp.454–463, Montreal, 1995.

[13] G. Verfaillie, M. Lemâitre, and T. Schiex. Russian doll search for solving constraint optimization problems. In *AAAI-96*, pp.181–187, Portland, OR, 1996.

[14] R. J. Wallace. Enhancements of branch and bound methods for the maximal constraint satisfaction problem. In *AAAI-96*, pp.188–195, Portland, OR, 1996.