Prioritized Component Systems

Gerhard Brewka
Comp. Sci. Institute
University of Leipzig
Leipzig, Germany
brewka@informatik.uni-leipzig.de

Ilkka Niemelä
Dept. of Comp. Sci. and Eng.
Helsinki University of Technology
Helsinki, Finland
Ilkka.Niemela@tkk.fi

Mirosław Truszczyński
Dept. of Comp. Sci.
University of Kentucky
Lexington, KY 40506-0046, USA
mirek@cs.uky.edu

Abstract

We introduce a flexible framework to specify problem solutions (outcomes) and preferences among them. The proposal combines ideas from answer-set programming (ASP), answer-set optimization (ASO) and CP-nets. The problem domain is structured into components. ASP techniques are used to specify values of components, as well as global (inter-component) constraints among these values. ASO methods are used to describe preferences among the values of a component and CP-net techniques to represent inter-component dependencies and corresponding preferences.

Introduction

Qualitative preferences have received considerable attention in AI lately (cf. the special issue Computational Intelligence, 20(2), 2004). A popular representation of qualitative preferences are CP-nets (Boutilier *et al.* 1999; 2004a; 2004b). They are directed acyclic graphs of variables annotated with conditional preference tables. The table associated with a variable *explicitly* describes conditional preferences among the values of that variable as a function of value assignments to the parent variables. The problem is to specify a preference ordering on *outcomes*, that is, assignments of a value to each variable. Preferences are given a *ceteris paribus* (other things being equal) interpretation. For example, the statement *red cars are preferred over blue cars* is taken to mean: if two cars differ only in colour, and one is red while the other blue, then the red one is preferred (not: each red car is better than any blue car). Preference rules express strict constraints on the preference ordering on outcomes, but are weak in the sense that they apply *only* under the *ceteris paribus* restriction.

Another approach, called *answer-set optimization* (ASO), has been developed in the context of answer-set programming (Brewka, Niemelä, & Truszczyński 2003). Although preference rules are used there as well, their meaning is quite different. Each rule expresses a ranking on answer sets. However, the rankings provided by different rules may differ, and a combination method is used (Pareto in (Brewka, Niemelä, & Truszczyński 2003)) to generate a global preference order. Hence, a rule can be viewed as a single criterion in multi-criteria decision making. ASO preference rules thus are strong but defeasible constraints on the global preference ordering on answer sets. They are strong because they apply without the *ceteris paribus* restriction. They are defeasible since they may be overridden by other rules.

Copyright © 2005, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

CP-nets allow the user to fully exploit dependency structure, both for preference representation and elicitation. However, they are restrictive in the way preferences on the values of a single variable can be specified. They basically assume that variables have a small number of known values and that a total order of these values can be explicitly given. The ASO approach uses an answer-set program to represent a single variable with complex outcomes, where outcomes are modeled by answer sets of the program. It also provides flexible means for expressing possibly conflicting and/or incomplete preferences on outcomes, as well as indifference. However, dependencies among parts of outcomes remain implicit and are not fully exploited.

Prioritized component systems, our main contribution, combine key aspects of both approaches. The setup is basically that of CP-nets. We have variables, each with its domain, and each possibly depending on its parent variables. Given a variable, for every set of values of parent variables we also have an ordering of elements of the domain of that variable. However, unlike in CP-nets, variables may have large domains, with elements known only implicitly. We refer to these "complex" variables as *components* and, following the ASO approach, represent them by answer-set programs. Preference orderings on the values for the component also are specified following the ASO approach. Each component C comes with a set of preference rules, whose bodies (and only the bodies!) may contain literals from *parent* components. Each assignment of values (answer sets) to the parent components selects a set of relevant preference rules for C. These rules express defeasible multi-criteria preference information that determines the preference ordering on the answer sets of C, given the answer sets (values) for the parent components.

The choice of answer-set programs to represent components (or outcomes, in the case of ASO programs) is not essential for our discussion. Any constraint formalism (e.g., propositional logic) could be used instead. In selecting answer-set programs we were motivated by the minimality and groundedness of their answer-set semantics, which make them useful for knowledge representation.

Our approach generalizes both CP-nets and ASO programs. CP-nets are component systems with dependency structure, but with the simplest possible components that just pick a single value for a variable. ASO programs are component systems with a single, possibly complex component but, clearly, no component dependencies. The framework we develop here allows us to model systems anywhere in between these two extremes.

Background

In most abstract terms, we are concerned with comparing the quality of elements in a set. Thus, we need (i) a formalism to specify a set of elements or, a *space of outcomes* we wish to compare, and (ii) a way to represent and reason about preferences on outcomes. The two aspects are typically separated for greater modularity, flexibility and generality.

We will now describe two formalisms for specifying and reasoning about preferences that are of primary interest later in the paper. They are: *answer-set optimization programs* (*ASO-programs*, for short) (Brewka, Niemelä, & Truszczyński 2003) and *CP-nets* (Boutilier *et al.* 1999; 2004a; 2004b).

ASO programs

In ASO programs, individual outcomes are sets of literals over a fixed set At of propositional atoms. To represent a set of outcomes, we use a logic program over the set of atoms At and assume that feasible outcomes are precisely the answer sets of P (we assume some familiarity with the logic programming terminology; we refer the reader to (Gelfond & Lifschitz 1991; Simons, Niemelä, & Soininen 2002) for relevant details). As we noted, our choice of programs to model outcomes is motivated by knowledge-representation concerns but other constraint-based formalisms could also be used.

To represent preferences in ASO programs, we use *preference rules* (or simply, *preferences*) of the form

$$\gamma_1 > \ldots > \gamma_k \leftarrow \alpha \qquad (1)$$

where the γ_is are *boolean combinations* of atoms in At, i.e., formulas built of atoms in At using disjunction ($\vee$), conjunction ($\wedge$), strong ($\neg$) and default (not) negation, with the restriction that $\neg$ can appear only in front of atoms, and not only in front of literals and where α is a conjunction of literals and expressions of the form $not\ l$ where l is a literal.

Definition 1 *An* answer-set optimization *(or* ASO*) program over the set of atoms At is a pair (C, Φ), where C is a logic program (over At), called a* generator program, *and Φ is a collection of preference rules (over At), called a* preference program. *Answer sets of C are* outcomes *of (C, Φ).*

To specify how to use preferences to compare sets of literals and, in particular, outcomes of ASO programs, we introduce a *satisfaction* relation between sets of literals and boolean combinations.

Definition 2 *Satisfaction of a boolean combination γ in a set of literals S, denoted $S \models \gamma$, is defined as:*

$$\begin{array}{lll} S \models l \ (l \text{ literal}) & \textit{iff} & l \in S \\ S \models not\ l \ (l \text{ literal}) & \textit{iff} & l \notin S \\ S \models \gamma_1 \vee \gamma_2 & \textit{iff} & S \models \gamma_1 \text{ or } S \models \gamma_2 \\ S \models \gamma_1 \wedge \gamma_2 & \textit{iff} & S \models \gamma_1 \text{ and } S \models \gamma_2. \end{array}$$

We now have the following key definition.

Definition 3 *Let S be a set of literals and r a preference rule of the form (1). The* satisfaction degree *of r in S (denoted by $s_r(S)$) is given as follows. If $S \models \alpha$ and, for some i, $1 \leq i \leq n$, $S \models \gamma_i$, then $s_r(S)$ is the smallest index of a satisfied boolean combination in the head of r. Otherwise the rule is irrelevant and $s_r(S)$ is denoted by a special symbol I.*

We assume that the smaller the index of a satisfied goal the better. For trivial reasons, irrelevant rules are not suboptimally satisfied as they either do not apply or have no goals satisfied at all (cf. (Brewka, Niemelä, & Truszczyński 2003) for a more extensive discussion of that issue). Thus, we consider I to be as good a satisfaction degree as 1. These intuitions are made formal by a preorder[1] $\preceq$ on the set of satisfaction degrees: for $\alpha, \beta \in \{I, 1, 2, \ldots\}$, $\alpha \preceq \beta$ if $\beta = 1$ or $\beta = I$, or α and β are integers and $\beta \leq \alpha$. The relation $\preceq$ is in fact a *total* preorder. We write $\prec$ for the *strict* counterpart to the relation $\preceq$ ($\alpha \prec \beta$ if $\alpha \preceq \beta$ and it is not the case that $\beta \preceq \alpha$). The relation $\prec$ is antisymmetric and transitive and so, its closure under reflexivity, is a *partial order*.

Let (C, Φ) be an ASO program and S a set of literals. The satisfaction degrees of rules from Φ in S form the basis for defining a global preference ordering on outcomes of (C, Φ). Several methods can be used for this purpose. We will restrict our discussion here to the Pareto method.

Definition 4 *Let (C, Φ) be an ASO program and let S and S' be its outcomes (answer sets). An outcome S is* Pareto-preferred *to an outcome S' of C ($S' \preceq_\Phi S$) if for every rule $r \in \Phi$, $s_r(S') \preceq s_r(S)$. If $S' \preceq_\Phi S$ and it is not the case that $S \preceq_\Phi S'$ (that is, for at least one rule $r \in \Phi$, $s_r(S') \prec s_r(S)$), then S is* strictly *Pareto-preferred to S' ($S' \prec_\Phi S$).*
An outcome S is optimal *for (C, Φ) if there is no outcome S' such that $S \prec_\Phi S'$.*

The relation $\preceq_\Phi$ is a preorder relation for the set of outcomes of (C, Φ). The relation $\prec_\Phi$, when closed under reflexivity, is a partial order on the set of outcomes.

For simplicity, we restrict the class of generator programs to normal programs (default negation in the bodies only, no classical negation symbols) with cardinality constraints (Simons, Niemelä, & Soininen 2002). Cardinality constraints allow us to specify lower and upper bounds on the number of certain atoms in an answer set. For this paper we will only need rules of the form $1\{a_1, \ldots, a_n\}1 \leftarrow \alpha$ expressing that exactly one a_i must be contained in answer sets satisfying α. Answer sets of such programs are sets of atoms.

CP-nets

CP-nets were introduced in (Boutilier *et al.* 1999; 2004a). Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a finite set of variables (attributes). For each variable $A \in \mathcal{A}$, let D_A be the *domain* of A, that is, a finite and non-empty set of *values* for A. Without loss of generality, we assume that variable domains are pairwise disjoint. An *outcome* is an n-tuple $U = (u_1, \ldots, u_n)$ such that $u_i \in D_{A_i}$, $1 \leq i \leq n$.

[1]A binary relation is a *preorder* if it is reflexive and transitive.

A *condition* is a conjunction of atomic expressions of the form $(A = a)$ and their negations, where $A \in \mathcal{A}$ and $a \in D_A$. A *conditional preference rule* (or, *conditional preference*) is an expression of the form $\alpha : \pi$ such that α is a condition and, for some variable A, π is a total preorder on the domain D_A. We write $a \leq_\pi b$ if $(a, b) \in \pi$, i.e., if b is at least as good as a in π and $a <_\pi b$ to denote the *strict preference* of b over a, i.e., if $a \leq_\pi b$ holds but $b \leq_\pi a$ does not. We note that unlike $\leq_\pi$, $<_\pi$ (when closed under reflexivity) is a partial order.

We use total preorders rather than total orders (even though the latter is a more common choice in the literature) to provide means to model *indifference*. It is important as in practice the users may be unwilling or just unable to specify a total order on the set of values of a variable in a situation when they consider different values as equally good.

A *CP-net* over a set of variables $\mathcal{A}$ is a pair (P, Φ), where P is a *parent function*, and Φ is a collection of conditional preferences such that for every conditional preference $\alpha : \pi$ in Φ, which orders values of a variable A, α involves only variables that belong to $P(A)$[2].

In a standard approach to CP-nets, preference statements are represented by conditional-preference tables — one table for each variable. Rows in the table for a variable A correspond to tuples of values from the domains D_B, where $B \in P(A)$ (exactly one row for each tuple), and each row contains a relevant ordering of values in the domain D_A. The approach we adopt here is more general.

The main contribution of CP-nets is in how they use conditional preferences to order outcomes. In what follows, given a variable A and an outcome U, by $U(A)$ we denote the value from the domain of A that appears in U.

Definition 5 *An outcome V is* one-step preferred *to an outcome W if for some variable A:*

1. *$V(B) = W(B)$ for every variable $B \in \mathcal{A} \setminus \{A\}$ (that is, for every variable other than A), and*
2. *Φ contains a conditional preference $\alpha : \pi$ for A such that W (and hence also V) satisfies α and $W(A) \leq_\pi V(A)$.*

If V is one-step preferred to W with respect to a variable A, then we say that there is an *improving A-flip* from W to V and that there is a *worsening A-flip* from V to W.

Definition 6 *The outcome V is* CP-preferred *to W, $W \preceq_{CP} V$, if V can be obtained from W by a (possibly empty) sequence of improving flips. The outcome V is* strictly CP-preferred *to W, $W \prec_{CP} V$, if $W \preceq_{CP} V$ and it is not the case that $V \preceq_{CP} W$. An outcome V is* optimal *if there is no outcome U such that $V \prec_{CP} U$.*

For every CP-net, the relation $\prec_{CP}$, when closed under reflexivity, is a partial ordering. The relation $\preceq_{CP}$ in general is not; it is a preorder only. However, for *acyclic* CP-nets (CP-nets whose parent function induces a directed acyclic graph on the set of variables) and with all preferences specifying *total orders*, $\preceq_{CP}$ is a partial order and it coincides with $\prec_{CP}$ (closed under reflexivity). The relation $\prec_{CP}$ and the notion of optimality we consider in the paper were introduced and studied in (Brafman & Dimopoulos 2004).

[2]Including a parent function in the description of a CP-net is redundant as the dependency information is contained in conditions of conditional preference rules. However, it makes the preference elicitation process more systematic.

Component systems

In an ASO program (C, Φ), the role of C is to describe the space of available outcomes and the role of Φ is to capture preferences of a user regarding these solutions. In this general form, ASO programs do not provide explicit means to handle complexity and structure present in large-scale applications, where domains and problems decompose into subdomains and subproblems with well-defined dependencies. Adding structure based on dependencies, and so implicitly expressing independence, has proven successful in knowledge representation formalisms such as Bayes nets and CP-nets. These formalisms exploit structure to ensure concise, *factored* problem representations. Our main objective is to bring structure to answer-set optimization to make the process of modeling spaces of solutions and relevant preference information more transparent and systematic. In the process, we obtain a formalism for qualitative reasoning about preferences that generalizes both ASO-programs and CP-nets.

Let $At(P)$ be the set of atoms appearing in a program P.

Definition 7 *A* component system *is a pair (Π, G), where*

1. *$\Pi = \{C_1, ..., C_n\}$ is a collection of logic programs with pairwise disjoint sets of atoms, called* components *of the component system, and*
2. *G is a set of* constraints, *that is, logic program clauses of the form*
$$\leftarrow \mathit{body}$$
such that $At(G) \subseteq \bigcup_{C \in \Pi} At(C)$.

An outcome *of (Π, G) is a tuple $\langle M(C_1), \ldots, M(C_n) \rangle$, where for every $C_i \in \Pi$, $M(C_i) \subseteq At(C_i)$ and $\bigcup M(C_i)$ is an answer set of the program $G \cup \bigcup \Pi$.*

Directly from the definition and from the properties of logic programs Π and G it follows that if M is an outcome of a component system (Π, G) then, for every component C, the set $M(C)$ is an answer sets of C.

Logic programs represent sets of literals (sets of atoms, if we restrict, as we do in this paper, to normal programs with cardinality constraints). The answer sets have no explicit structure or, are "flat". Component systems replace this flat representation with a two-level one. Elements of outcomes are no longer atomic but have structure of their own. They are collections of atoms — answer sets of system components. In other words, each component C can be thought of as a variable, its domains represented *implicitly* by a logic program (C itself) and consisting of the answer sets of C.

Every tuple of answer sets of programs forming components of a component system (Π, G) is an outcome of the component system $(\Pi, \emptyset)$. The role of G is to exclude some combinations. Adding a constraint $\leftarrow \mathit{body}$ to a logic program eliminates all answer sets of the program that satisfy body. Since there are no restrictions on atoms appearing in the clauses in G, the program G represents *global* (or inter-component) restrictions on ways, in which component values can be combined into outcomes of (Π, G).

The structure of a component system (Π, G) is implicitly present in the program $\bigcup\Pi \cup G$. Namely, there is a 1-to-1 correspondence between outcomes of the component system (Π, G) and answer sets of the program $\bigcup\Pi \cup G$: for an outcome M of (Π, G), $M^{\cup} = \bigcup_{C\in\Pi} M(C)$ is an answer set of $\bigcup\Pi \cup G$ and, for an answer set N of $\bigcup\Pi \cup G$, the tuple $\vec{N} = \langle N \cap At(C) \colon C \in \Pi\rangle$ is an outcome of (Π, G). The ability to make the structure explicit is, however, important. It allows us to model directly the structure present in applications, an issue we discuss in the last section.

We will illustrate the idea of a component system with a simple example of configuring a daily menu (more formally, describing the space of daily menus). We first establish that a daily menu consists of three components: breakfast, lunch and dinner. We then describe each of them separately, for instance, by programs listed below.

breakfast:	$1\{continental, american\}1$
	$light_b \leftarrow continental$
lunch:	$1\{soup_l, salad_l\}1$
	$1\{meat_l, fish_l, lasagne_l\}1$
	$1\{fries_l, noodles_l\}1 \leftarrow not\ lasagne_l$
	$light_l \leftarrow lasagne_l, salad_l$
dinner:	$1\{soup_d, salad_d\}1$
	$1\{meat_d, fish_d, lasagne_d\}1$
	$1\{fries_d, noodles_d\}1 \leftarrow not\ lasagne_d$
	$light_d \leftarrow lasagne_d, salad_d$

The components specify the possible choices for each case. So far there are $2\times10\times10 = 200$ possible outcomes. Global constraints G exclude certain combinations (for lack of variety or for health reasons), for instance

$\leftarrow meat_l, meat_d$
$\leftarrow lasagne_l, lasagne_d$
$\leftarrow not\ light_b, not\ light_l, not\ light_d$

While the same set of menus can be represented by a single program consisting of all these rules, component systems structure the rules in a way conceptually linked to the structure inherent in the application domain. However, a fundamental advantage of component systems is that they lend themselves well to a variety of methods to specify preferences among outcomes. We discuss that issue next.

Prioritized component systems

The main focus of this paper is on representing and reasoning about preferences pertaining to a collection of outcomes represented by a component system. To represent preferences we use preference rules as defined for ASO programs.

Definition 8 *A* prioritized component system *(PCS, for short) is a quadruple $\mathcal{C} = (\Pi, G, P, \Phi)$, where*

1. *(Π, G) is a component system;*
2. *P is a* parent function *describing dependencies between components: for every component $C \in \Pi$, $P(C)$ consists of all components in Π that C depends on; and*
3. *Φ is a set of* preference rules *of the form (1) that are subject to the following restrictions: for each preference rule $\gamma_1 > \ldots > \gamma_k \leftarrow \alpha$ in Φ there is a component C such that formulas γ_i, $1 \le i \le k$, are built of atoms in $At(C)$ and α is built of atoms in $At(C) \cup \bigcup_{D\in P(C)} At(D)$. We call such preference rules C-*preferences *and denote the set of all C-preferences by Φ_C.*

Outcomes *for a prioritized component system (Π, G, P, Φ) are the outcomes for its component system (Π, G).*

We now define the preference relation on outcomes of a PCS (Π, G, P, Φ). To this end, we first focus on the PCS $(\Pi, \emptyset, P, \Phi)$. Every outcome of (Π, G, P, Φ) is an outcome of $(\Pi, \emptyset, P, \Phi)$. Moreover, outcomes of $(\Pi, \emptyset, P, \Phi)$ are precisely those tuples M such that for every $C \in \Pi$, $M(C)$ is an answer set of C. The first step is to introduce the notion of a *flip* between outcomes of a prioritized system $(\Pi, \emptyset, P, \Phi)$.

Definition 9 *Let M and M' be outcomes for $(\Pi, \emptyset, P, \Phi)$ and $C \in \Pi$ a component. There is a C-*flip *from M to M' if $M \neq M'$ and for every $D \in \Pi$, $D \neq C$ implies $M'(D) = M(D)$.*

A flip from M to M' is improving *if $M \preceq_{\Phi_C} M'$ (that is, with respect to the Pareto-preference relation determined by preference rules in Φ_C).*

The choice of the Pareto-preference relation with respect to preferences in Φ_C is a consequence of the fact that we defined the preference relation between outcomes of ASO programs by means of that relation. If a different combination method for ASO programs is chosen, that relation has to be used here.

Definition 10 *Let (Π, G, P, Φ) be a PCS and let M and M' be outcomes for (Π, G, P, Φ). The outcome M' is* PCS-preferred *to M, $M \preceq_{PCS} M'$, if there is a sequence of improving flips from M to M' (possibly involving outcomes for $(\Pi, \emptyset, P, \Phi)$ not satisfying the constraints G as intermediate steps). It is* strictly *preferred, $M \prec_{PCS} M'$, if $M \preceq_{PCS} M'$ and it is not the case that $M' \preceq_{PCS} M$.*

Definition 11 *Let M be an answer set of a PCS $\mathcal{C}$. We say that M is an optimal outcome for $\mathcal{C}$ if there is no outcome M' for $\mathcal{C}$ such that $M \prec_{PCS} M'$.*

Let us consider again the daily menu example. To specify her preferences the user first decides the dependency structure she wants to base her preferences on. Let us assume that dinner and breakfast preferences do not depend on other components, and that lunch depends on the other two. The user first specifies her dinner and breakfast preferences, for instance, using the following preference rules:

$(d_1)\ meat_d \wedge fries_d > fish_d > meat_d \wedge noodles_d$
$(d_2)\ soup_d > salad_d \leftarrow meat_d$
$(d_3)\ salad_d > soup_d \leftarrow not\ meat_d$
$(b_1)\ light_b > not\ light_b$

She now describes conditional lunch preferences dependent on the attributes of the other meals:

$(l_1)\ meat_l > not\ meat_l \leftarrow lasagne_d, continental$
$(l_2)\ salad_l > soup_l \leftarrow soup_d$
$(l_3)\ fish_l > lasagne_l \leftarrow salad_l$
$(l_4)\ soup_l > salad_l \leftarrow american$

Now one of the optimal outcomes is:

$$\langle\{continental, light_b\}, \{salad_l, fish_l, fries_l\}, \{meat_d, fries_d, soup_d\}\rangle$$

Another optimal solution is obtained by exchanging $fries_l$ with $noodles_l$. There are also optimal solutions containing $lasagne_d$ and $salad_d$. We are aware that the full power of our approach becomes more evident in larger scale examples for which we have no space here. Nevertheless, the example illustrates the basic notions and some of the advantages of PCSs: dependencies are between components, not simple variables with unstructured values; preferences can be incomplete and conflicting (e.g. conflict between l_2 and l_4 in case of american breakfast and dinner soup), and it is possible to remain indifferent (no preference between fish/fries and fish/noodles for dinner).

The way we use improving flips to order outcomes follows (Boutilier *et al.* 2004b), and disagrees with (Prestwich *et al.* 2004). The two approaches differ in their interpretation of constraints. In (Prestwich *et al.* 2004) the constraints are viewed as specifications of *forbidden* configurations that are completely unacceptable to the user. Hence, in that approach, outcome y is preferred to an outcome x if there is a sequence of improving flips from x to y not passing through a forbidden outcome. That amounts to letting constraints override CP-preferences (as forbidden outcomes are forced to be least preferred) and appears to be a drastic change of the CP-net semantics. The alternative approach of (Boutilier *et al.* 2004b) treats constraints as simply making some outcomes unavailable. The fact that an outcome is unavailable has nothing to do with the user's preferences — should the outcome become available its quality would be measured according to user preferences specified in the CP-net. Consequently, (Boutilier *et al.* 2004b) (and we here) allow the use of unavailable outcomes to define the global preference relation by means of improving flips.

We now show that CP-nets and ASO-programs are special cases of our formalism. A CP-net (P, Φ) can be represented as a PCS in the following way. Let Z be a variable of (P, Φ) and let $z_1, \ldots, z_n$ be its values. We define a component program C_Z to consist of a single logic program rule (a choice rule in the syntax of (Simons, Niemelä, & Soininen 2002))

$$1\{Z = z_1, \ldots, Z = z_n\}1.$$

Let $\alpha : \pi$ be a conditional preference rule for a variable Z. We denote by α' the formula where every occurrence of $\neg$ is replaced with not . Next, let $L_1, \ldots, L_k$ be all maximal sets of atoms of the form $Z = z_i$ such that all z_is are equivalent with respect to the total preorder π and such that atoms in L_i are strictly preferred in π to atoms in L_j if and only if $i < j$. We set d_i, $1 \leq i \leq k$, to be the disjunction of atoms in L_i. A rule $d_1 > \ldots > d_k \leftarrow \alpha'$ is a preference in the sense of ASO programs and PCSs. We denote the set of all preferences obtained by transforming the preferences in Φ by Φ'. We have the following theorem.

Theorem 1 *Let (P, Φ) be a CP-net with the set of variables $\mathcal{V}$ and let U and U' be two of its outcomes. Then U and U' are outcomes of the PCS $(\{C_Z : Z \in \mathcal{V}\}, \emptyset, P, \Phi')$ and*

1. *$U \preceq_{CP} U'$ if and only if $U \preceq_{PCS} U'$*
2. *$U \prec_{CP} U'$ if and only if $U \prec_{PCS} U'$*

This theorem can be generalized to the case of "constrained optimization with CP-nets" approach (Boutilier *et al.* 2004a). Whenever a certain combination of variable values, say $X_1 = x_1, \ldots, X_n = x_n$, is excluded by the global constraints in a constrained net, we include a corresponding constraint of the form $\leftarrow X_1 = x_1, \ldots, X_n = x_n$ in G.

The PCS framework also generalizes answer-set optimization as described in (Brewka, Niemelä, & Truszczyński 2003). Indeed, we have the following simple result.

Theorem 2 *Let $\mathcal{C} = (C, \Phi)$ be an ASO program. Then $\mathcal{C}' = (\{C\}, \emptyset, \emptyset, \Phi)$ (in other words, both the set of global constraints and the parent function are empty) is a PCS, its outcomes are precisely the 1-tuples $\langle M \rangle$, where M is an answer set for C, and for every two outcomes M and M' of $\mathcal{C}$ (answer sets of C),*

1. *$M \preceq_{\Phi} M$ (in $\mathcal{C}$) if and only if $\langle M \rangle \preceq_{PCS} \langle M' \rangle$ (in $\mathcal{C}'$)*
2. *$M \prec_{\Phi} M$ (in $\mathcal{C}$) if and only if $\langle M \rangle \prec_{PCS} \langle M' \rangle$ (in $\mathcal{C}'$).*

The correspondences between CP-nets and PCSs can be summarized by the following table:

CP-net	PCS
variable v_i	program C_i
value of v_i	answer set of C_i
value assignment	configuration
CP-table for v_i	preference program Φ_{C_i}
v_i-flip	new answer set for C_i

Pareto and PCS ordering

Let $\mathcal{C} = (\Pi, G, P, \Phi)$ be a PCS. By disregarding the structure in $\mathcal{C}$, we obtain an ASO program $\mathcal{C}_{ASO} = (\bigcup \Pi \cup G, \Phi)$. We noted earlier that there is a 1-to-1 correspondence between outcomes of $\mathcal{C}$ (which are outcomes of (Π, G)) and outcomes of $\mathcal{C}_{ASO}$ (answer sets of $\bigcup \Pi \cup G$), given by a function that assigns to each outcome M of $\mathcal{C}$, the set $M^{\cup}$, an answer set of $\bigcup \Pi \cup G$ (an outcome of $\mathcal{C}_{ASO}$). That correspondence suggests that a natural criterion for any ordering of the outcomes of $\mathcal{C}$ is that it extends the Pareto ordering $\preceq_{\Phi}$ of the outcomes of $\mathcal{C}_{ASO}$. We will now show that our PCS-preference ordering has indeed that property for the class of *acyclic* PCSs. It is not an overly restrictive assumption since typical PCSs arising in applications are acyclic.

Theorem 3 *Let $\mathcal{C} = (\Pi, G, P, \Phi)$ be an acyclic PCS and let M and M' be two of its outcomes. If $\bigcup_{C \in \Pi} M(C) \preceq_{\Phi} \bigcup_{C \in \Pi} M'(C)$ then $M \preceq_{CP} M'$.*

Proof (sketch): Let $C_1, \ldots C_n$ be an enumeration of the components of $\mathcal{C}$ consistent with the parent function P, that is, for every component C_i, $P(C_i) \subseteq \{C_1, \ldots, C_{i-1}\}$. For every i, $1 \leq i \leq n$, let $M_i = M(C_i)$ and, similarly, $M'_i = M'(C_i)$ (that is, M_i and M'_i are answer sets of (values for) the component C_i appearing in outcomes M and M', respectively). For every i, $1 \leq i \leq n + 1$, we define

$$N_i = \langle M'_1, \ldots, M'_{i-1}, M_i, \ldots, M_n \rangle.$$

It is clear that $M = N_1$ and $M' = N_{n+1}$. Moreover, for every i, $1 \leq i \leq n$, N_{i+1} is a result of a C_i-flip applied to N_i. We can show that for each i, $1 \leq i \leq n$, the C_i-flip from N_i to N_{i+1} is an improving one. $\square$

Our proof also works for a slightly broader class of PCSs that contain preference rules whose bodies are conjunctions of formulas (not literals), each formula built of atoms of a single component.

Complexity

The computational complexity of problems related to PCSs depends on the class of programs used as components. To make the discussion concrete, we consider PCSs whose components are logic programs with cardinality and weight constraints (Simons, Niemelä, & Soininen 2002). However, all results in this section hold for PCSs whose components belong to any class of programs, for which one can check in polynomial time whether a set of literals is an answer set.

Theorem 4 *The following problem is **PSPACE**-complete: Given a PCS $\mathcal{C}$ and two of its outcomes M and M', decide whether $M \preceq_{PCS} M'$.*

Proof (sketch): Let $\mathcal{C} = (\Pi, G, P, \Phi)$. The problem can be decided using a *nondeterministic* Turing machine that starts with $M_1 = M$ and correctly guesses a sequence of improving flips from M to M'. That is, in each iteration with M_1 as the current outcome the machine correctly guesses a component C and an answer set S of C so that (1) the result M_2 of replacing $M_1(C)$ with S in M_1 is an outcome of $\mathcal{C}$, and (2) $M_1 \preceq_{\Phi_C} M_2$. Then, the machine sets M_1 to be M_2 until M' is met. It is clear that the sequence of outcomes produced by this Turing machine is an improving sequence and that the machine works in polynomial space. It follows that our problem is in the class **NPSPACE**. Since **PSPACE** $=$ **NPSPACE** (cf. (Papadimitriou 1994)), the membership part of the assertion follows.

The hardness part follows from the fact that the class of PCSs contains the class of CP-nets. In the domain of the CP-nets, the problem we are considering here is known as the *dominance* problem. It has been shown to be **PSPACE**-complete in (Lang *et al.* 2005). □

Next, we study problems related to optimal outcomes.

Theorem 5 *The problem to decide whether a PCS $\mathcal{C}$ has an optimal outcome is **NP**-complete.*

The problems concerning the existence of an outcome satisfying some property given as a boolean combination, and concerning deciding the optimality of an outcome are again much harder.

Theorem 6 *The next two problems are **PSPACE**-complete:*

1. *Given a PCS $\mathcal{C}$ and a boolean combination γ, decide whether $\mathcal{C}$ has an optimal outcome satisfying γ;*
2. *Given a PCS $\mathcal{C}$ and its outcome M, decide whether M is optimal.*

While **PSPACE**-completeness results indicate that reasoning with PCSs is inherently hard, we note that analogous problems for much simpler formalisms such as (general) CP-nets are equally complex. One way then to interpret the results in this section is that PCSs offer enhanced representational flexibility over the formalism of CP-nets at no extra cost in the computational complexity.

Discussion

Prioritized component systems support a decoupled methodology for applications combining ASP with preference elicitation (an example is product configuration: a product data model describes valid product instances and customer preferences need to be elicited for choosing a preferred instance). To *describe the space of choices*, a domain is structured into components. Each component has a generator program whose answer sets represent the valid choices for that component. For *preference elicitation* the user is guided through the components. In each step, she picks a component for which her preferences only depend on components she has already ranked. In each case she gives preference rules defining a preference order on answer sets of the current component.

A major advantage of our framework is its flexibility. The user is free to structure the domain into components in the most adequate way for a particular application. Moreover, the preference order on values of a component can be expressed conveniently using flexible rules, each one providing a criterion for assessing a value's quality. Our results show that making dependency structure explicit will, at least in the acyclic case, increase comparability and thus lead to more fine grained distinctions.

Acknowledgments

The authors acknowledge the support of EC contract IST-FET-2001-37004 (WASP), Academy of Finland grant 211025 and NSF grants IIS-0097278, IIS-0325063.

References

Boutilier, C.; Brafman, R.; Hoos, H.; and Poole, D. 1999. Reasoning with conditional ceteris paribus preference statements. In *Proceedings of UAI-99*.

Boutilier, C.; Brafman, R.; Domshlak, C.; Hoos, H.; and Poole, D. 2004a. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *JAIR* 21:135–191.

Boutilier, C.; Brafman, R.; Domshlak, C.; Hoos, H.; and Poole, D. 2004b. Preference-based constrained optimization with CP-nets. *Comp. Intelligence* 20(2):137–157.

Brafman, R., and Dimopoulos, Y. 2004. Extended semantics and optimization algorithms for CP-networks. *Comp. Intelligence* 20(2):218–245.

Brewka, G.; Niemelä, I.; and Truszczyński, M. 2003. Answer set optimization. In *Proceedings of IJCAI-03,*

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

Lang, J.; Goldsmith, J.; Truszczyński, M.; and Wilson, N. 2005. The computational complexity of dominance and consistency in CP-nets. Proceedings of IJCAI-05.

Papadimitriou, C. 1994. *Computational Complexity*. Addison-Wesley.

Prestwich, S.; Rossi, F.; Venable, K.; and Walsh, T. 2004. Constrained CP-nets. In *Proc. of the Joint Annual Workshop of ERCIM/CoLogNet on Constraint Solving and Constraint Logic Programming*.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2):181–234.