CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Pruning rules for constrained optimisation for conditional preferences
Authors	Wilson, Nic;Trabelsi, Walid
Publication date	2011-09
Original Citation	WILSON, N., TRABELSI, W. 2011. Pruning rules for constrained optimisation for conditional preferences. In: LEE, J. (ed.) Principles and Practice of Constraint Programming - CP 2011. Perugia, Italy, 12-16 Sep. Berlin, Heidelberg: Springer, pp 804-818.
Type of publication	Conference item
Link to publisher's version	http://link.springer.com/chapter/10.1007%2F978-3-642-23786-7_60 - 10.1007/978-3-642-23786-7_60
Rights	© 2011, Springer-Verlag GmbH Berlin Heidelberg. The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-642-23786-7_60
Download date	2024-09-22 00:11:13
Item downloaded from	https://hdl.handle.net/10468/1412

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Pruning Rules for Constrained Optimisation for Conditional Preferences

Nic Wilson and Walid Trabelsi

Cork Constraint Computation Centre
Department of Computer Science
University College Cork, Ireland
n.wilson@4c.ucc.ie, w.trabelsi@4c.ucc.ie

Abstract. A depth-first search algorithm can be used to find optimal solutions of a Constraint Satisfaction Problem (CSP) with respect to a set of conditional preferences statements (e.g., a CP-net). This involves checking at each leaf node if the corresponding solution of the CSP is dominated by any of the optimal solutions found so far; if not, then we add this solution to the set of optimal solutions. This kind of algorithm can clearly be computationally expensive if the number of solutions is large. At a node N of the search tree, with associated assignment b to a subset of the variables B, it may happen that, for some previously found solution α, either (a) α dominates all extensions of b; or (b) α does not dominate any extension of b. The algorithm can be significantly improved if we can find sufficient conditions for (a) and (b) that can be efficiently checked. In case (a), we can backtrack since we need not continue the search below N; in case (b), α does not need to be considered in any node below the current node N. We derive a sufficient condition for (b), and three sufficient conditions for (a). Our experimental testing indicates that this can make a major difference to the efficiency of constrained optimisation for conditional preference theories including CP-nets.

1 Introduction

Conditional preference languages, such as CP-nets and more general formalisms [4, 9, 6, 15, 2], can give a natural way for the user of a decision support system to express their preferences over multivariate options. A basic problem is: given a set of outcomes, determine which are the undominated ones, i.e., which are not considered worse than another outcome. For example, in a recommender system, one can use preference deduction techniques to infer, from the previous user inputs, which products may be preferred over others, and hence which are the undominated ones [11].

As shown in [5], one can use a depth-first search algorithm to find optimal solutions of a Constraint Satisfaction Problem (CSP) with respect to a set of conditional preferences statements (e.g., a CP-net). The algorithm in [5], as well as related algorithms in [14, 15], involve using appropriate variable and value orderings so that solutions are generated in an order compatible with

the conditional preference statements. At each leaf node we check to see if the corresponding solution of the CSP is dominated by any of the optimal solutions found so far; if not, then we add this solution to the set of optimal solutions.

The standard dominance check for CP-nets and more general languages is computationally hard, as illustrated by the PSPACE-completeness result in [8]. In this paper we follow [14, 16] in using a polynomial dominance relation, which is an upper approximation of the standard one; this enables much larger problems to be tackled (see [10] for experimental results regarding a recent implementation of the standard dominance queries).

Even so, this kind of constrained optimisation algorithm can clearly be computationally expensive if the number of solutions is large, since we have at least one dominance check (and possibly many) to make for each solution.

At a node N of the search tree, with associated assignment b to a subset of the variables B, it may happen that, for some previously found solution α, either (a) α dominates all extensions of b; or (b) α does not dominate any extension of b. The algorithm can be significantly improved if we can find sufficient conditions for (a) and (b) that can be efficiently checked (and that hold sufficiently often). In the positive case, (a), we can backtrack since we need not continue the search below N, hence pruning a possibly exponentially large part of the search tree. In the negative case, (b), α does not need to be considered in any node below the current node N, thus eliminating potentially exponentially many dominance checks involving α.

In this paper, we derive three polynomial sufficient conditions for (a), and one for (b). We have implemented and experimentally tested these in the context of a constrained optimisation algorithm, and they are seen to significantly improve the algorithm. Section 2 describes the background: the conditional preferences formalism in Section 2.1, and the polynomial notion of dominance in Section 2.2. The form of the constrained optimisation algorithm is described in Section 3. Section 4 describes the three polynomial sufficient conditions for the positive case (a), and Section 5 derives the polynomial sufficient conditions for the negative case, (b). Section 6 describes the experimental testing, and Section 7 discusses extensions.

2 Background Material

2.1 A Language of Conditional Preferences

Let V be a finite set of variables, and for each $X \in V$ let $\underline{X}$ be the set of possible values of X; we assume $\underline{X}$ has at least two elements. For subset of variables $U \subseteq V$ let $\underline{U} = \prod_{X \in U} \underline{X}$ be the set of possible assignments to set of variables U. The assignment to the empty set of variables is written $\top$. An *outcome* is an element of $\underline{V}$, i.e., an assignment to all the variables. For partial tuples $a \in \underline{A}$ and $u \in \underline{U}$, we say a *extends* u, if $A \supseteq U$ and $a(U) = u$, i.e., a projected to U gives u. More generally, we say that a *is compatible with* u if there exists outcome $\alpha \in \underline{V}$ extending both a and u, i.e., such that $\alpha(A) = a$ and $\alpha(U) = u$.

The language $\mathcal{L}$ consists of statements of the form $u : x > x' [W]$ where u is an assignment to set of variables $U \subseteq V$ (i.e., $u \in \underline{U}$), x, x' are different values of variable X, and $\{X\}$, U and W are pairwise disjoint. Let $T = V - (\{X\} \cup U \cup W)$. Such a conditional preference statement φ represents that given u and any assignment to T, x is preferred to x' irrespective of the values of W. If $W = \emptyset$ we sometimes write the statement just as $u : x > x'$.

The formal semantics is defined using total pre-orders[1] on the set $\underline{V}$ of outcomes. Formally, we say that total pre-order $\succcurlyeq$ *satisfies* $u : x > x' [W]$ if $tuxw \succcurlyeq tux'w'$ for all $t \in \underline{T}, w, w' \in \underline{W}$, since u is satisfied in both outcomes $tuxw$ and $tux'w'$, and variable X has the value x in the first, and x' in the second, and they differ at most on $\{X\} \cup W$.

If φ is the statement $u : x > x' [W]$, for $u \in \underline{U}$ and $x, x' \in \underline{X}$ then we define $u_\varphi = u$, $x_\varphi = x$, $x'_\varphi = x'$, $U_\varphi = U$, $X_\varphi = X$ and $W_\varphi = W$.

Subsets Γ of the language $\mathcal{L}$ are called *conditional preference theories (cp-theories)* [13]. For cp-theory Γ, and outcomes α and β we write $\alpha \succeq_\Gamma \beta$ when $\alpha \succcurlyeq \beta$ holds for all total pre-orders $\succcurlyeq$ satisfying each element of Γ (cf. Theorem 1 of [14]). CP-nets [3, 4] can be represented by conditional preference theories that involve statements with empty W, and TCP-nets [6] with statements involving empty or singleton W [12].

2.2 Polynomial Dominance for Conditional Preferences

In this section we describe a polynomial dominance[2] relation for conditional preferences. This polynomial dominance relation is less conservative than the standard one, leading to fewer undominated solutions, which also can be advantageous. The definitions and results in this section come from [14] (and were generalised further in [16]).

A pre-ordered search tree (abbreviated to a *pos-tree*) is a rooted directed tree (which we imagine being drawn with the root at the top, and children below parents). Associated with each node r in the tree is a variable Y_r, which is instantiated with a different value in each of the node's children (if it has any), and also a total pre-order $\succcurlyeq_r$ of the values of Y_r. A directed edge in the tree therefore corresponds to an instantiation of one of the variables to a particular value. Paths in the tree from the root down to a leaf node correspond to sequential instantiations of different variables. We also associate with each node r a set of variables A_r which is the set of all variables $Y_{r'}$ associated to nodes r' above r in the tree (i.e., on the path from the root to r), and an assignment a_r to A_r corresponding to the assignments made to these variables in the edges between the root and r. The root node r^* has $A_{r^*} = \emptyset$ and $a_{r^*} = \top$, the assignment to the empty set of variables. Hence r' is a child of r if and only

[1] A total pre-order $\succcurlyeq$ is a binary relation that is reflexive ($\alpha \succcurlyeq \alpha$), transitive and complete (i.e., for all α and β, either $\alpha \succcurlyeq \beta$ or $\beta \succcurlyeq \alpha$). If both $\alpha \succcurlyeq \beta$ and $\beta \succcurlyeq \alpha$ then we say that α and β are $\succcurlyeq$*-equivalent*.

[2] The notion of dominance in this paper is quite different from the notion of dominance as in Symmetry Breaking via Dominance Detection [7] and related work.

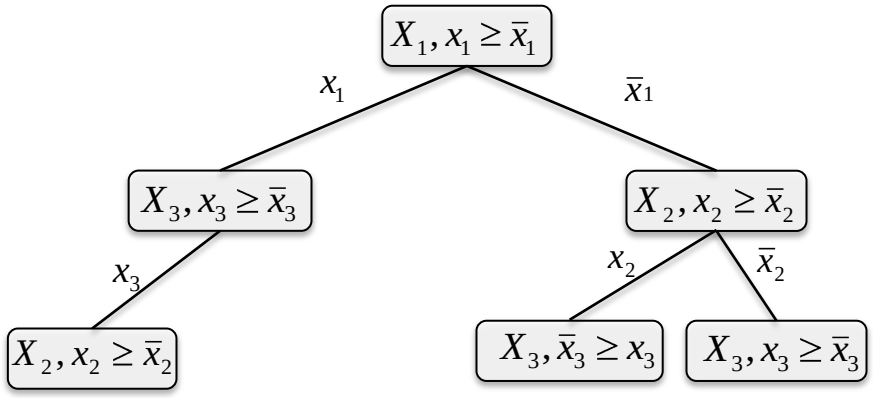
if $A_{r'} = A_r \cup \{Y_r\}$ (where $A_r \not\ni Y_r$) and $a_{r'}$ extends a_r (with an assignment to Y_r).

Formally, define a node r to be a tuple $\langle A_r, a_r, Y_r, \geqslant_r \rangle$, where $A_r \subseteq V$ is a set of variables, $a_r \in \underline{A_r}$ is an assignment to those variables, $Y_r \in V - A_r$ is another variable, and $\geqslant_r$ is a total pre-order on the set $\underline{Y_r}$ of values of Y_r. We make two restrictions on the choice of this total pre-order: firstly, it is assumed not to be the trivial complete relation on $\underline{Y}$, i.e., there exists some $y, y' \in \underline{Y}$ with $y \not\geqslant_r y'$ (so not all y and y' are $\geqslant_r$-equivalent). We also assume that $\geqslant_r$ satisfies the following condition (which ensures that the associated ordering on outcomes is transitive): if there exists a child of node r associated with instantiation $Y_r = y$, then y is not $\geqslant_r$-equivalent to any other value of Y, so that $y \geqslant_r y' \geqslant_r y$ only if $y' = y$. In particular, $\geqslant_r$ totally orders the values (of Y_r) associated with the children of r.

For outcome α, define the *path to* α to be the path from the root which includes all nodes r such that α extends a_r. To generate this, for each node r we reach, starting from the root, we choose the child associated with the instantiation $Y_r = \alpha(Y_r)$ (there is at most one such child); the path finishes when there exists no such child. Node r is said to ***decide*** *outcomes* α *and* β if it is the deepest node (i.e., furthest from the root) that is both on the path to α and on the path to β. Hence α and β both extend the tuple a_r (but they may differ on variable Y_r). We compare α and β by using $\geqslant_r$, where r is the unique node that decides α and β. Each pre-ordered search tree σ has an associated total pre-order $\succcurlyeq_\sigma$ on outcomes which is defined as follows. Let $\alpha, \beta \in \underline{V}$ be outcomes. We define $\alpha \succcurlyeq_\sigma \beta$ to hold if and only if $\alpha(Y_r) \geqslant_r \beta(Y_r)$, where r is the node that decides α and β. We therefore then have that α and β are $\succcurlyeq_\sigma$-equivalent if and only if $\alpha(Y_r)$ and $\beta(Y_r)$ are $\geqslant_r$-equivalent. This ordering is similar to a lexicographic ordering in that two outcomes are compared on the first variable on which they differ.

Example of a pos-tree. Figure 1 shows an example pos-tree. The bottom left node in the diagram represents the pos-tree node $r = \langle \{X_1, X_3\}, x_1x_3, X_2, x_2 \geqslant_r \overline{x}_2 \rangle$. The first component $A_r = \{X_1, X_3\}$ is the set of variables assigned above the node; the second component $a_r = x_1x_3$ is the assignment to A_r made in the path from the root to r. The third component $Y_r = X_2$ is the variable that is ordered next, and the fourth component, $x_2 \geqslant_r \overline{x}_2$ is the local ordering on X_2. Note that the both the local (value) orderings and the variable (importance) orderings can differ in different branches of a pos-tree. Let α and β be the outcomes $x_1\overline{x}_2x_3$ and $x_1x_2\overline{x}_3$, respectively. The path to α includes the three nodes on the left hand of the figure. The path to β contains the root node and its left hand child, $r' = \langle \{X_1\}, x_1, X_3, x_3 \geqslant \overline{x}_3 \rangle$. Node r' therefore divides α and β. Since $Y_{r'} = X_3$, and $\alpha(X_3) \geqslant_{r'} \beta(X_3)$, we have $\alpha \succcurlyeq_\sigma \beta$. ■

We say that pre-ordered search tree σ *satisfies* conditional preference theory Γ iff $\succcurlyeq_\sigma$ satisfies Γ (see Section 2.1). We give an alternative characterisation of this. Relation $\sqsupset_a^X$ on $\underline{X}$ is defined to be the transitive closure of the set of



$$x_1x_2x_3 \geq x_1\bar{x}_2x_3 \geq x_1x_2\bar{x}_3 \equiv\ x_1\bar{x}_2\bar{x}_3 \geq\ \bar{x}_1x_2\bar{x}_3 \geq\ \bar{x}_1x_2x_3 \geq\ \bar{x}_1\bar{x}_2x_3 \geq\ \bar{x}_1\bar{x}_2\bar{x}_3$$

Fig. 1. An example pos-tree σ over binary variables $\{X_1, X_2, X_3\}$, and its associated total pre-order $\succcurlyeq_\sigma$ on outcomes. For each node r we only include its associated variable Y_r and the local ordering $\geqslant_r$.

pairs (x, x') of values of X over all statements $u : x > x'\,[W]$ in Γ such that u is compatible with a.

Proposition 1 ([14]). *The following pair of conditions are necessary and sufficient for a pre-ordered search tree σ to satisfy the cp-theory Γ.*

(1) For any $\varphi \in \Gamma$ and outcome α extending u_φ: on the path to α, X_φ appears before every element of W_φ;
(2) for all nodes r in σ, $\geqslant_r \supseteq \sqsupset_{a_r}^{Y_r}$.

Condition (1) relates to the allowable variable orderings in a pre-ordered search tree satisfying Γ, and condition (2) restricts the value orderings.

For a given cp-theory Γ we define the relation $\trianglerighteq_\Gamma$ (abbreviated to $\trianglerighteq$) as follows: $\alpha \trianglerighteq \beta$ holds if and only if $\alpha \succcurlyeq_\sigma \beta$ holds for all pos-trees σ satisfying Γ (i.e, all σ such that $\succcurlyeq_\sigma$ satisfies Γ). Proposition 1 of [14] shows that if $\alpha \trianglerighteq_\Gamma \beta$ then $\alpha \trianglerighteq_\Gamma \beta$. Importantly, for any outcomes α and β it can be determined in polynomial time if $\alpha \trianglerighteq_\Gamma \beta$: see Section 4.2 of [14].

3 Constrained optimisation

In the constrained optimisation algorithms in [5, 14, 15] a search tree is used to find solutions of a CSP, where the search tree is chosen to be compatible with the cp-theory Γ, i.e., so that its associated total ordering on outcomes extends

relations $\succ_\Gamma$ and $\trianglerighteq_\Gamma$ (defined above in Sections 2.1 and 2.2, respectively). Methods for finding such search trees have been developed in [15], Sections 5 and 6. One can use a fixed variable ordering in the search tree if the cp-theory is fully acyclic—see [15], Section 5.1—i.e., there exists an ordering $X_1, \ldots, X_n$ of the variables such that for any statement $\varphi \in \Gamma$, if $X_i \in U_\varphi$ then $i < j$, where $X_\varphi = X_j$, and if $X_i \in W_\varphi$ then $i > j$.

3.1 Basic Constrained Optimisation Approach

We can make use of this compatible search tree as follows: when we find a new solution β we check if it is $\trianglerighteq$-undominated with respect to each of the current known set K of $\trianglerighteq$-undominated solutions (i.e., if it is *not* the case that there exists $\alpha \in K$ with $\alpha \trianglerighteq \beta$). If so, then β is an $\trianglerighteq$-undominated solution, since it cannot be $\trianglerighteq$-dominated by any solution found later. We add β to K, and continue the search. At the end, K will be the complete set of $\trianglerighteq$-undominated solutions (which is a subset of the set of $\succeq_\Gamma$-undominated solutions, since $\trianglerighteq \supseteq \succeq_\Gamma$).

Associated with each node of the search tree is a partial assignment b, which consists of the assignments to earlier variables B. We choose some uninstantiated variable $Y \notin B$, and assign values to Y in each child node. Also there is associated a current domain $\mathcal{D}(X)$ of each variable X. For $X \in B$, $\mathcal{D}(X) = \{b(X)\}$. For other variables X, $\mathcal{D}(X)$ is determined by constraint propagation [1], which may be done in a number of ways. The key property of $\mathcal{D}(X)$ is: for eliminated values x (i.e., $x \in \underline{X} - \mathcal{D}(X)$), there exists no solution β of the CSP extending b and such that $\beta(X) = x$. Backtracking occurs when any of the domains becomes empty, since there cannot then be any solution extending b. In the experiments described in Section 6 we enforce arc consistency to generate the current domains; however, other forms of consistency are possible, for example, global consistency where a value x is included in the domain of variable X if and only if there exists a solution β extending b and such that $\beta(X) = x$.

3.2 Incorporating Dominance and Non-dominance conditions

At any node of a depth-first search algorithm for finding solutions of a CSP, we have an associated partial assignment b to the variables B that have already been instantiated, and we have the current domain $\mathcal{D}(X)$ of each variable X. We formalise this notion of a collection of domains as follows:

Definition 1. *A* collection of domains *is a function $\mathcal{D}$ on V such that $\mathcal{D}(X) \subseteq \underline{X}$, so that $\mathcal{D}(X)$ is a set of possible values of X. For outcome β, we say that β is* of *$\mathcal{D}$ if $\beta(X) \in \mathcal{D}(X)$ for all $X \in V$.*

We say that α dominates $\mathcal{D}$ if it dominates every β of $\mathcal{D}$, and α non-dominates $\mathcal{D}$ if it doesn't dominate any β of $\mathcal{D}$:

Definition 2. *Let α be an outcome and let $\mathcal{D}$ be a collection of domains. We define:*

— α dominates $\mathcal{D}$ *if* $\alpha \trianglerighteq \beta$ *for all* β *of* $\mathcal{D}$.
— α non-dominates $\mathcal{D}$ *if for all* β *of* $\mathcal{D}$, $\alpha \not\trianglerighteq \beta$.

Suppose that α is a solution we've already found, and that we are currently at a node of the search tree with associated partial assignment b and collection of domains $\mathcal{D}$. If we can determine that α dominates $\mathcal{D}$ then there is no need to explore nodes in the search tree extending partial assignment b, so we can backtrack at this node. If, on the other hand, we can determine that α non-dominates $\mathcal{D}$ then we can eliminate α from the set of current solutions for any node below the current node, because there is no need to check again that $\alpha \not\trianglerighteq \beta$, for solutions β extending b. In Section 4, we describe sufficient conditions for α dominating $\mathcal{D}$ that can be efficiently checked, and in Section 5, an efficient sufficient condition for α non-dominating $\mathcal{D}$.

4 Sufficient Conditions for Dominance

To show, given particular assumptions, that α dominates collection of domains $\mathcal{D}$, we need to show that there cannot exist a pos-tree σ satisfying Γ that strictly prefers some element β of $\mathcal{D}$ to α. (Because then $\alpha \succcurlyeq_\sigma \beta$ for all σ satisfying Γ, and hence, $\alpha \trianglerighteq \beta$, for all β of $\mathcal{D}$.) The first rule gives conditions that imply non-existence of such a σ by just considering its root node; the second rule focuses on the node of σ that decides α and β.

4.1 The Root-Dominates Rule

We say that α root-dominates collection of domains $\mathcal{D}$ if: for all $Y \notin \bigcup_{\varphi\in\Gamma} W_\varphi$,
(i) $\alpha(Y) \sqsupset^Y_\top y$ for all $y \in \mathcal{D}(Y) - \{\alpha(Y)\}$;
(ii) if $\alpha(Y) \in \mathcal{D}(Y)$ then $\alpha(Y)$ and y are $\sqsupset^Y_\top$-equivalent for some $y \in \underline{Y} - \{\alpha(Y)\}$.

The following result states the soundness of the root-dominates rule.

Proposition 2. *If* α *root-dominates* $\mathcal{D}$ *then* α *dominates* $\mathcal{D}$, *i.e.,* $\alpha \trianglerighteq \beta$ *for all* β *of* $\mathcal{D}$.

Proof: Assume that α root-dominates $\mathcal{D}$, and consider any element β of $\mathcal{D}$, and any pos-tree σ satisfying Γ. Consider the root node r of σ with associated variable Y and local ordering $\geqslant$. Proposition 1, condition (1) implies that $Y \notin \bigcup_{\varphi\in\Gamma} W_\varphi$. If α and β differ on Y, then the root node decides α and β, and Proposition 1(2) and condition (i) imply $\alpha(Y) \geqslant \beta(Y)$, and hence $\alpha \succcurlyeq_\sigma \beta$. If, on the other hand, $\alpha(Y) = \beta(Y)$, then condition (ii) implies, using Proposition 1(2), that $\alpha(Y)$ is $\geqslant$-equivalent to some other element of $\underline{Y}$, which implies, by the definition of a pos-tree, that the root node has no children. Hence the root node again decides α and β, and so $\alpha \succcurlyeq_\sigma \beta$. Since σ was arbitrary, $\alpha \trianglerighteq \beta$, for all β of $\mathcal{D}$. ■

Example. Let V be the set of variables $\{X, Y, Z\}$ with initial domains as follows: $\underline{X} = \{x_1, x_2, x_3, x_4\}$, $\underline{Y} = \{y_1, y_2\}$ and $\underline{Z} = \{z_1, z_2\}$. Let cp-theory Γ consist of

the five statements $\top : x_1 > x_3$, $\top : x_2 > x_3$, and $\top : x_2 > x_4$ $[\{Z\}]$, $x_1 : y_1 > y_2$, and $x_2 : y_2 > y_1$. Let α be the assignment $x_2\, y_2\, z_2$, and let $\mathcal{D}(X) = \{x_3, x_4\}$, $\mathcal{D}(Y) = \underline{Y}$ and $\mathcal{D}(Z) = \underline{Z}$. Then $\bigcup_{\varphi \in \Gamma} W_\varphi = \{Z\}$, and $\alpha(X) = x_2 \sqsupset_\top^X x_3$ and $\alpha(X) \sqsupset_\top^X x_4$. Also, $\alpha(Y) = y_2 \sqsupset_\top^Y y_1 \sqsupset_\top^Y y_2$, so $\alpha(Y)$ and y_1 are $\sqsupset_\top^Y$-equivalent. Hence, α root-dominates $\mathcal{D}$. ■

The first half of the definition of *root-dominates* is actually a necessary condition for dominance:

Proposition 3. *Suppose that α dominates collection of domains $\mathcal{D}$. Then $\alpha(Y) \sqsupset_\top^Y y$ holds for all $Y \notin \bigcup_{\varphi \in \Gamma} W_\varphi$, and for all $y \in \mathcal{D}(Y) - \{\alpha(Y)\}$.*

Proof: Suppose there exists some $Y \notin \bigcup_{\varphi \in \Gamma} W_\varphi$ and $y \in \mathcal{D}(Y) - \{\alpha(Y)\}$ such that $\alpha(Y) \not\sqsupset_\top^Y y$. Then we can create a pos-tree σ with just a root node r, with associated variable $Y_r = Y$. We choose the local ordering $\geqslant_r$ so that $\geqslant_r$ contains $\sqsupset_\top^Y$ and is such that $\alpha(Y) \not\geqslant_r y$. (This is possible since $\alpha(Y) \not\sqsupset_\top^Y y$). By Proposition 1, σ satisfies Γ. Choose any β of $\mathcal{D}$ with $\beta(Y) = y$. Then, $\alpha \not\succcurlyeq_\sigma \beta$, so $\alpha \not\trianglerighteq \beta$, and hence it is not the case that α dominates $\mathcal{D}$. ■

When for all $Y \notin \bigcup_{\varphi \in \Gamma} W_\varphi$, domain $\mathcal{D}(Y)$ doesn't include $\alpha(Y)$, part (ii) of the definition of root-dominance holds vacuously, so Propositions 2 and 3 imply that root-dominance is a necessary and sufficient condition for dominance:

Proposition 4. *Suppose that $\mathcal{D}(Y) \not\ni \alpha(Y)$ for all $Y \notin \bigcup_{\varphi \in \Gamma} W_\varphi$. Then α root-dominates $\mathcal{D}$ if and only if α dominates $\mathcal{D}$.*

4.2 The Deciding-Node Dominance Rule

Let α be an outcome and let $\mathcal{D}$ be a collection of domains. Define S to be $\{Y \in V : \mathcal{D}(Y) \not\ni \alpha(Y)\}$. These are the variables that α and β differ on for all β of $\mathcal{D}$. Define Ψ to be the set of all $\varphi \in \Gamma$ such that $X_\varphi \in S$ and u_φ is compatible with $\alpha(V - S)$. (u_φ is compatible with $\alpha(V - S)$ if and only if for all $Y \in U_\varphi - S$, $\alpha(Y) = u_\varphi(Y)$.) Let $\alpha_* = \alpha(V - S)$. We will use the relation $\sqsupset_{\alpha_*}^Y$, defined in Section 2.2 as the transitive closure of all pairs (x_φ, x'_φ) such that $\varphi \in \Gamma$, $X_\varphi = Y$ and u_φ is compatible with α_*.

Definition 3. *Using the notation defined above, we say that α deciding-node-dominates $\mathcal{D}$ if $\alpha(Y) \sqsupset_{\alpha_*}^Y y$ for all $Y \notin \bigcup_{\varphi \in \Psi} W_\varphi$ and for all $y \in \mathcal{D}(Y) - \{\alpha(Y)\}$.*

The following proposition states the soundness of the deciding-node-dominates rule.

Proposition 5. *If α deciding-node-dominates $\mathcal{D}$ then α dominates $\mathcal{D}$.*

Proof: Consider any element β of $\mathcal{D}$, and any pos-tree σ satisfying Γ. Consider the node r of σ that decides α and β, with associated variable Y and tuple $a \in \underline{A}$. Firstly, $A \cap S = \emptyset$, since α and β agree on A but differ on each variable in S. If

$\varphi \in \Psi$ then $X_\varphi \in S$, and so $X_\varphi \notin A$. This implies, using Proposition 1(1), that $Y \notin W_\varphi$, so we've shown that $Y \notin \bigcup_{\varphi \in \Psi} W_\varphi$. We have that $\alpha(V-S)$ extends a (since $A \subseteq V-S$ and α extends a), which immediately implies that $\sqsupset_a^Y$ contains $\sqsupset_{\alpha_*}^Y$. If α deciding-node-dominates $\mathcal{D}$ then $\alpha(Y) \sqsupset_{\alpha_*}^Y \beta(Y)$ or $\alpha(Y) = \beta(Y)$. Therefore, by Proposition 1(2), $\alpha(Y) \geqslant_r \beta(Y)$, showing that $\alpha \succcurlyeq_\sigma \beta$, and hence $\alpha \trianglerighteq \beta$, as required. ■

Example (continued). Consider again the example in Section 4.1. Then $S = \{X\}$, α_* equals the partial assignment $y_2\, z_2$, and $\bigcup_{\varphi \in \Psi} W_\varphi = \{Z\}$. We have $\alpha(X) = x_2 \sqsupset_{\alpha_*}^X x_3$, and $x_2 \sqsupset_{\alpha_*}^X x_4$, and $\alpha(Y) = y_2 \sqsupset_{\alpha_*}^Y y_1$, showing that α deciding-node-dominates $\mathcal{D}$, and hence, by Proposition 5, α dominates $\mathcal{D}$.

If we now remove statement $x_1 : y_1 > y_2$ from Γ we still have α deciding-node-dominates $\mathcal{D}$ but we no longer have α root-dominates $\mathcal{D}$.

In the following example, α does not deciding-node-dominate $\mathcal{D}$ but α root-dominates $\mathcal{D}$, and so α dominates $\mathcal{D}$. Let $\mathcal{D}(X) = \underline{X} = \{x_1, x_2\}$, let $\mathcal{D}(Y) = \underline{Y} = \{y_1, y_2\}$, and let $\mathcal{D}(Z) = \underline{Z} = \{z_1, z_2, z_3\}$. Let Γ consist of: $z_1 : x_1 > x_2$, $z_2 : x_2 > x_1$, $\quad x_1 : y_1 > y_2$, $\quad x_2 : y_2 > y_1$, $x_1 : z_1 > z_2$, $\quad x_2 : z_2 > z_1$ and $x_1 y_2 : z_1 > z_3$. Let $\alpha = x_1\, y_1\, z_1$. Therefore, root-dominance and deciding-node-dominance are incomparable, and both are strictly stronger than dominance. ■

When, for all variables Y, $\alpha(Y)$ is not in the current domain $\mathcal{D}(Y)$ of Y, we have $S = V$, $\alpha_* = \top$ and $\Psi = \Gamma$. The definition of *deciding-node-dominates* then becomes equivalent to part (i) of the definition of *root-dominates*, with part (ii) being vacuously satisfied. Using Proposition 4, we therefore have the following result showing that these dominance definitions are then equivalent.

Proposition 6. *Suppose that $\mathcal{D}(Y) \not\ni \alpha(Y)$ for all $Y \in V$. Then α deciding-node-dominates $\mathcal{D}$ iff α root-dominates $\mathcal{D}$ iff α dominates $\mathcal{D}$.*

4.3 Projection-Dominance Condition

Let b be an assignment to set of variables B, and let $\mathcal{D}$ be a collection of domains such that $\mathcal{D}(X) = \{b(X)\}$ for $X \in B$. It follows immediately that the condition $(*)$ below is a sufficient condition for: α dominates $\mathcal{D}$. (Recall, $\alpha(B)$ means α restricted/projected to B.)

$(*)$ $\gamma \trianglerighteq \beta$ for all outcomes $\gamma \in \underline{V}$ agreeing with α on B (i.e., $\gamma(B) = \alpha(B)$), and all $\beta \in \underline{V}$ extending b (i.e., $\beta(B) = b$).

In other words, if every outcome, whose projection to B is $\alpha(B)$, dominates every outcome whose projection to B is b. Condition $(*)$ can be determined directly using the polynomial algorithm in Section 5 of [16]. (In the notation of that paper we determine if $\Gamma^* \models_{\mathcal{Y}} \psi^*$, where $\mathcal{Y}$ is the set of singleton subsets of V, and ψ is the preference statement $\alpha(B) > b \parallel \emptyset$.) However, although this check is polynomial, it's a good deal more expensive than the root-dominates rule and the deciding-node-dominates rule.

5 A Sufficient Condition for Non-Dominance

Let α be an outcome and let $\mathcal{D}$ be a collection of domains. We say that α *root non-dominates* $\mathcal{D}$ if there exists $Y \notin \bigcup_{\varphi \in \Gamma} W_\varphi$ such that $\mathcal{D}(Y) \not\ni \alpha(Y)$ and, for all $y \in \mathcal{D}(Y)$, $\alpha(Y) \not\trianglerighteq^Y_\top y$.

Proposition 7. *If α root non-dominates $\mathcal{D}$ then α non-dominates $\mathcal{D}$, i.e. for all β of $\mathcal{D}$, we have $\alpha \not\trianglerighteq \beta$.*

Proof: Consider any β of $\mathcal{D}$. Suppose α root non-dominates $\mathcal{D}$, so that there exists $Y \notin \bigcup_{\varphi \in \Gamma} W_\varphi$ such that $\mathcal{D}(Y) \not\ni \alpha(Y)$ and $\alpha(Y) \not\trianglerighteq^Y_\top \beta(Y)$. It follows, using Proposition 1, that we can define a pos-tree σ satisfying Γ with just a root node with associated variable Y and local ordering $\geqslant$ with $\alpha(Y) \not\geqslant \beta(Y)$. Then $\alpha \not\succcurlyeq_\sigma \beta$, which shows that $\alpha \not\trianglerighteq \beta$. ■

Example (continued). Let Γ be as in the example in Section 4.1, let γ be the outcome $x_1\, y_2\, z_2$, and define $\mathcal{D}'$ by $\mathcal{D}'(X) = \{x_4\}$, $\mathcal{D}'(Y) = \underline{Y}$ and $\mathcal{D}'(Z) = \underline{Z}$. Then γ root non-dominates $\mathcal{D}'$, because $X \notin \bigcup_{\varphi \in \Gamma} W_\varphi = \{Z\}$, and $\mathcal{D}'(X) \not\ni \gamma(X) = x_1$, and $\gamma(X) \not\trianglerighteq^X_\top x_4$. ■

6 Experimental Testing

6.1 Experimental setup

We performed experiments with four families of cp-theories and several sets of binary CSP instances. The CSPs were generated using Christian Bessiere's random uniform CSP generator (`www.lirmm.fr/~bessiere/generator.html`). Experiments were run as a single thread on Dual Quad Core Xeon CPU, running Linux 2.6.25 x64, with overall 11.76 GB of RAM, and processor speed 2.66 GHz. We maintain arc consistency during the search algorithm, so that the current domains $\mathcal{D}(X)$ are generated from a partial assignment b by arc consistency [1]. The conditional preferences impose sometimes strong restrictions on the variable orderings that can be used in the search tree (corresponding to the condition (1) of Proposition 1), which much reduces the potential benefit of a dynamic variable ordering; for simplicity, we used a fixed variable ordering (which is possible since in the experiments we used only fully acyclic cp-theories [15], including acyclic CP-nets).

Random Generation of Preferences: We consider four families of cp-theories, CP-nets (*CPnet*), partial conditional lexicographic orders (*Lex*), a family with varying W component (*Rand-W*), and CP-nets with local total orderings (*CPn-to*). These are generated as follows. We order the variables V as $X_1, \ldots, X_n$. For each variable X_i we randomly choose the parents set U_i to be a subset of cardinality 0, 1 or 2 of $\{X_1, \ldots, X_{i-1}\}$. For the *CPnet* family we set $W_i = \emptyset$. For the *Lex* family we set $W_i = \{X_{i+1}, \ldots, X_n\}$. For random-$W$ (*Rand-W*) problems we define W_i to be a random subset of $\{X_{i+1}, \ldots, X_n\}$.

Then, for each assignment u to U_i, we randomly choose an ordering $x^1, \ldots, x^m$ of the domain of X_i (so we'll usually have different orderings for different u). We then randomly choose a number of pairs (x^j, x^k) with $j < k$, except for the *CPn-to* family when we include all pairs (x^j, x^{j+1}), for $j = 1, \ldots, |\underline{X_i}| - 1$. For each of these pairs we include the corresponding statement $u : x^j > x^k$ $[W_i]$ in the cp-theory Γ.

We consider ten versions of the algorithm. They differ according to whether they use root-dominance (labelled **r** in the tables), deciding node-dominance (**d**), the projection-dominance condition (**p**), or the root non-dominance condition (**n**). These are compared against the basic algorithm (Section 3.1) which uses none of these additional pruning methods, and we also consider some combinations of the methods.

We performed two groups of experiments. The first group focused on comparing the different versions of the algorithm (see Tables 1 and 2). We used CSPs based on 10 four-valued variables. The second group (see Figure 2) considers how computation time—of two of the best plus the basic algorithm—varies with the number n of variables. The computation time clearly depends strongly on the number of solutions of the CSP. Because of this, we considered families of CSPs with approximately constant number of solutions, in order to obtain a clearer picture of the dependence on n. We used three-valued variables and CSPs where each constraint includes 7 of the 9 possible tuples. We then chose the number of constraints to be such that the expected number of solutions was around 1000, further filtering out CSPs differing from this by more than around 10%.

Table 1. Mean number of optimal solutions for each preference family, and running times (ms), number of visited nodes and number of dominance checks at leaves for each preference family and each method. The CSPs were based on 10 four-valued variables, and averaged around 500 solutions.

	CP-nets			**Rand-W**			**Lex**			**CPn-to**		
# opt:	87.74			38.42			24.86			13.56		
Rules	*Time*	*#nd*	*chk*	*Time*	*#nd*	*chk*	*Time*	*#nd*	*chk*	*Time*	*#nd*	*chk*
Basic	7372	1173	22430	2097	1173	9181	1134	1173	5932	2263	1173	2248
r	10637	1172	22421	3312	971	8903	1609	647	4946	2706	1148	2227
d	4104	**536**	7956	677	209	1579	236	**97**	656	223	**148**	148
r+d	4156	**536**	7956	689	206	1578	234	**97**	656	226	**148**	148
p	32572	1173	89680	2705	291	10725	620	**97**	2950	11192	1173	12745
n	818	1173	979	675	1173	1817	560	1173	2031	908	1173	560
r+n	1438	1172	978	1896	971	1729	1628	647	1797	1501	1148	545
d+n	515	**536**	**288**	371	209	445	**205**	**97**	**323**	**124**	**148**	**12**
r+d+n	**514**	**536**	**288**	**363**	**206**	**444**	206	**97**	**323**	126	**148**	**12**
p+n	5150	1173	6136	1165	291	3099	386	**97**	1363	5325	1173	4170

Table 2. Mean number of optimal solutions for each preference family, and running times (ms) for each family and each method.

	CPnet	**Rand-W**	**Lex**	**CPn-to**
10 vars, 4 values, Mean 1993 solutions				
# opt	221.2	73.0	39.5	16.4
Base	62608	14711	6496	13728
r+d	31998	3204	673	651
r+d+n	**2164**	**1557**	**509**	**445**
10 vars, 4 values, Mean 9910 solutions				
# opt	364.8	204.5	133.8	6.5
Base	564733	183710	110303	29285
r+d	278583	28666	8482	358
r+d+n	**18623**	**14595**	**5307**	**352**

6.2 Discussion of Results

All figures in the tables and graphs are the mean over 50 random instances. The experimental results confirmed that no optimal solutions were lost by the additional pruning methods (as implied theoretically by Propositions 2, 5 and 7). Table 1 shows comparisons between all the methods for CSPs with around 500 solutions. Table 2 concerns CSPs with around 2000 solutions, and with around 10,000 solutions, where, for space reasons, we only include the results for the basic algorithm plus two of the best combinations, **r+d** and **r+d+n**.

The deciding-node-dominates rule (**d**) appears to be much the most effective of the three positive pruning schemes (i.e., **r**, **d** and **p**). With this rule the number of visited nodes and the number of dominance checks are reduced significantly in comparison with the basic algorithm. It seems that root-dominates can be slightly useful when used in conjunction with the deciding-node-dominates rule (**r+d**). The projection-dominates rule was not effective; although for the *Lex* and *Rand-W* families it pruned the search tree considerably, the costliness of the dominance test—which was applied at all nodes, not just leaf nodes—was detrimental, except for the *Lex* family (see Table 1). The root non-dominance condition can improve the performance of the algorithm considerably, especially for the *CPnet* and *CPn-to* families, since it can greatly reduce the number of dominance tests. An indication of how fast it is to check conditions **r**, **d** and **p** is given by considering the average time taken per node by their corresponding algorithms. In the experiments reported in Table 1, the version of the algorithm using **p** can be seen to take much more time per node than the **r** and **d** algorithms. For example, for the CP-nets family, algorithms **r**, **d** and **p** average around 9, 8 and 28 ms per node, respectively.

The results in Figure 2 indicate that the computation time does not increase very strongly with the number of variables. (By the way, it turns out that for each preference family, the mean number of optimal solutions does not vary greatly

with n, being centred on around $160, 90, 60$ and 9 for the CPnet, Rand-W, Lex and CPn-to families, respectively.) For the Rand-W family, the new algorithms do not perform much (if at all) better than the basic algorithm. For the Lex family, the two new algorithms are mostly twice as fast as the basic one. For the CP-nets family, the **r+d** algorithm is only slightly better than the basic algorithm, but performs excellently on the CPn-to family with mostly an order of magnitude improvement, as does the **r+d+n** algorithm, which also shows more than an order of magnitude speed up for the CP-nets family.

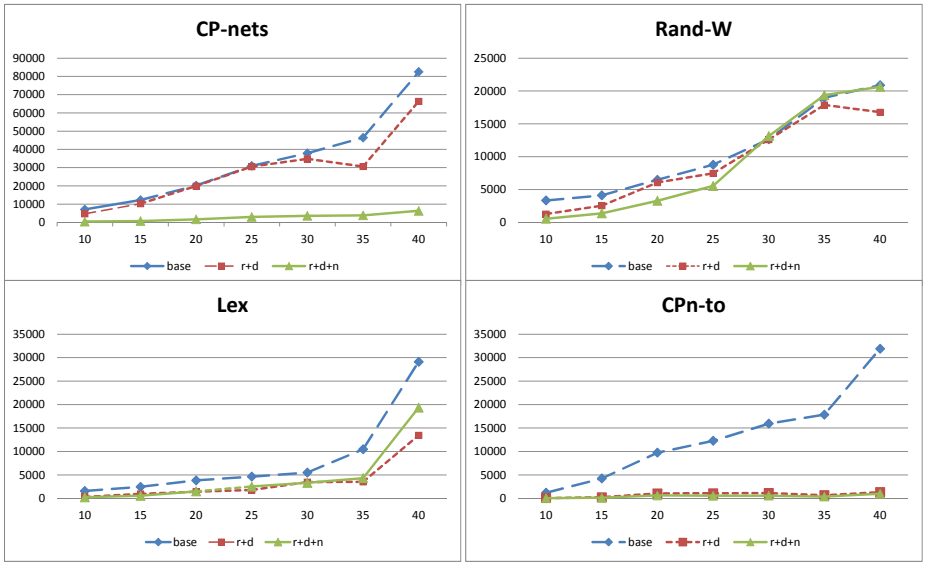


Fig. 2. Running time(ms) for each family of preferences for $n = 10, 15, \ldots, 40$ variables (having 3 values each). Each CSP has approximately 1000 solutions.

7 Discussion

The experimental results indicate that this approach to constrained optimisation for conditional preferences allows computation in a reasonable time even for problems of significant size (and problems of this kind, such as optimisation of configurable products, are not necessarily very large in practice). The additional methods developed in this paper can lead to a major improvement over the basic algorithm, often an order of magnitude improvement for the two CP-nets preference families. Interestingly, the non-dominance rule, which saves dominance checks when they are bound to fail below a node in the search tree, can be very effective, as well as one of the dominance rules. It could well be worth

constructing and testing non-dominance rules in situations involving other forms of partially ordered preferences.

There are many ways of extending the approaches. For example, we could generalise to more expressive comparative preference languages (such as that defined in [16]); we could attempt to develop the positive dominance rules for propagation, i.e., for eliminating values in future domains; we could develop an approach that uses dynamic variable orderings, making use of the consistency conditions from Section 6 of [15]; we could try to amend the projection-dominance rule to take into account the reductions in the current domains $\mathcal{D}$; furthermore, unsound pruning rules can be used, in order to find a reasonable number of solutions very fast. It would also be interesting to try applying the pruning rule from [5], which could be effective when the domains are large.

The approach in this paper was based on a polynomial dominance relation $\trianglerighteq_\Gamma$, rather the standard one $\succeq_\Gamma$. However, the results of this paper are still very relevant if one is interested in finding optimal solutions with respect to the standard dominance relation, for example, for CP-nets. Let Ω be the set of solutions of the CSP. We are computing the set Ω' of solutions α of Ω such that there does not exist $\beta \neq \alpha$ with $\beta \trianglerighteq_\Gamma \alpha$. If one uses the standard dominance relation $\succeq_\Gamma$, then the set of optimal solutions Ω^o consists of all elements α of Ω such that there does not exist $\beta \neq \alpha$ with $\beta \succeq_\Gamma \alpha$. Because $\succeq_\Gamma$ is more conservative than $\trianglerighteq_\Gamma$ (i.e., $\alpha \succeq_\Gamma \beta$ implies $\alpha \trianglerighteq_\Gamma \beta$), Ω' is always a subset of Ω^o, so any solutions generated by the approach in this paper are also optimal with respect to the standard semantics—although they will not generally be all such optimal solutions. If one wants to generate the set Ω^o precisely, one can use the depth-first search algorithm again with the dominance checking at leaf nodes being done with $\succeq_\Gamma$ rather than with $\trianglerighteq_\Gamma$. The three dominance rules from Section 4 are no longer sound, but the non-dominance rule from Section 5 is still sound, and so can be used to reduce the number of dominance checks in the search.

We focused on the constrained optimisation algorithm when we can generate outcomes using a search tree in an order that is compatible with the conditional preferences. It is possible to apply our techniques also for the case where the order of outcomes generated is not necessarily compatible with the conditional preferences. (We'd need to do this, in particular, if Γ were inconsistent, i.e., if $\succeq_\Gamma$ were not acyclic, since then there'd be no compatible search tree.) Then, at a leaf node with associated complete assignment β, we need to check also if β dominates α, as well as if α dominates β, where α is an element of K, the current set of solutions. In contrast with the standard case, K is not monotonic increasing: it can lose elements as well as gain them. Nevertheless, the dominance rules developed in this paper can again be valuable in pruning the search.

We considered the case where the set Ω of outcomes is expressed as the solutions of a CSP. In other settings, the set of outcomes, representing a set of available products, for example, is listed explicitly. The new constrained optimisation algorithms developed in this paper apply also here. Again we define dynamic variable and value orderings that determine a search tree compatible

with a set of conditional preferences; this search tree can be used to explore Ω (which is then implicitly being expressed as a decision tree), and find the optimal ones, using, as before, the positive and negative dominance rules to prune the search tree and reduce the dominance checks.

Acknowledgements

This material is based upon works supported by the Science Foundation Ireland under Grant No. 08/PI/I1912.

References

1. Bessiere, C.: Constraint propagation. In: Rossi, F., van Beek, P., T.Walsh (eds.) Handbook of Constraint Programming. Elsevier (2006)
2. Bienvenu, M., Lang, J., Wilson, N.: From preference logics to preference languages, and back. In: Proc. KR 2010 (2010)
3. Boutilier, C., Brafman, R., Hoos, H., Poole, D.: Reasoning with conditional *ceteris paribus* preference statements. In: Proceedings of UAI-99. pp. 71–80 (1999)
4. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H., Poole, D.: CP-nets: A tool for reasoning with conditional *ceteris paribus* preference statements. Journal of Artificial Intelligence Research 21, 135–191 (2004)
5. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H., Poole, D.: Preference-based constrained optimization with CP-nets. Computational Intelligence 20(2), 137–157 (2004)
6. Brafman, R., Domshlak, C., Shimony, E.: On graphical modeling of preference and importance. Journal of Artificial Intelligence Research 25, 389–424 (2006)
7. Fahle, T., Schamberger, S., Sellmann, M.: Symmetry breaking. In: Proc. CP. pp. 93–107 (2001)
8. Goldsmith, J., Lang, J., Truszczyński, M., Wilson, N.: The computational complexity of dominance and consistency in CP-nets. Journal of Artificial Intelligence Research 33, 403–432 (2008)
9. McGeachie, M., Doyle, J.: Utility functions for ceteris paribus preferences. Computational Intelligence 20(2), 158–217 (2004)
10. Santhanam, G., Basu, S., Honavar, V.: Dominance testing via model checking. In: Proc. AAAI 2010 (2010)
11. Trabelsi, W., Wilson, N., Bridge, D., Ricci, F.: Comparing approaches to preference dominance for conversational recommender systems. In: Proc. ICTAI. pp. 113–118 (2010)
12. Wilson, N.: Consistency and constrained optimisation for conditional preferences. In: Proceedings of ECAI-04. pp. 888–892 (2004)
13. Wilson, N.: Extending CP-nets with stronger conditional preference statements. In: Proceedings of AAAI-04. pp. 735–741 (2004)
14. Wilson, N.: An efficient upper approximation for conditional preference. In: Proceedings of ECAI-06. pp. 472–476 (2006)
15. Wilson, N.: Computational techniques for a simple theory of conditional preferences. Artificial Intelligence, in press, DOI 10.1016/j.artint.2010.11.018 (2011)
16. Wilson, N.: Efficient inference for expressive comparative preference languages. In: Proceedings of IJCAI 2009. pp. 961–966 (2009)