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,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

J. Lee (Ed.): CP 2011, LNCS 6876, pp. 804–818, 2011.

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 $\alpha(U) = u$.