Computing Possibly Optimal Solutions for Multi-Objective Constraint Optimisation with Tradeoffs

Nic Wilson and **Abdul Razak**
Insight Centre for Data Analytics
University College Cork, Ireland
nic.wilson@insight-centre.org
abdul.razak@insight-centre.org

Radu Marinescu
IBM Research – Ireland
radu.marinescu@ie.ibm.com

Abstract

Computing the set of optimal solutions for a multi-objective constraint optimisation problem can be computationally very challenging. Also, when solutions are only partially ordered, there can be a number of different natural notions of optimality, one of the most important being the notion of Possibly Optimal, i.e., optimal in at least one scenario compatible with the inter-objective tradeoffs. We develop an AND/OR Branch-and-Bound algorithm for computing the set of Possibly Optimal solutions, and compare variants of the algorithm experimentally.

1 Introduction

Many real-world problems involve multiple and sometimes non-commensurate objectives that need to be optimised simultaneously. Multi-objective Constraint Optimisation (MOCOP) is a general framework for such problems. Solutions are compared on a number p of (real-valued) objectives, so that each complete assignment to the decision variables has an associated multi-objective utility value, represented by a vector in $\mathbb{R}^p$. User preferences regarding tradeoffs between objectives can be expressed and reasoned about, as shown in [Marinescu *et al.*, 2013], where it is assumed that the user's preferences are represented by a weighted sum of the objectives. The authors show how to compute the undominated solutions, where a solution $\vec{x}$ is dominated by another solution $\vec{y}$ if in every consistent scenario, $\vec{y}$ is at least as good as $\vec{x}$, and in at least one scenario, $\vec{y}$ is better. However, it is arguable that this is not always the most natural notion of optimality. Alternatively, one could look to compute the *possibly optimal* solutions, which are the solutions which are optimal in at least one scenario.

In this paper we show how a MOCOP algorithm can be developed for computing the possibly optimal solutions, which will very often form a considerably smaller set than the set of undominated solutions. We analyse the properties that allow optimality operators to be computed in a modular way using branch-and-bound algorithms, using a search involving both OR-nodes and AND-nodes, and show that these properties hold for computing the possibly optimal solutions, as well as the undominated solutions. We consider several different methods for pruning search, and analyse experimentally how the performance varies, when varying the algorithm, the optimality task and the number of tradeoffs.

Section 2 gives the background for MOCOP based on tradeoffs. In Section 3 we take an abstract view of optimality operators, and give the key conditions for computation using AND/OR branch-and-bound search. Section 4 describes in more detail the techniques used in the algorithm for the computation of the possibly optimal solutions. The results of the experimental testing of the algorithm is given in Section 5, and Section 6 concludes.

Proofs are included in a longer version of the paper available online [Wilson et al.*, 2015].*

2 Preliminaries

2.1 Multi-objective Constraint Optimisation

Consider a problem with p objectives. A *utility vector* $\vec{u} = (u_1, \ldots, u_p)$ is a vector with p components where each $u_i \in \mathbb{R}$ represents the utility (or value) with respect to objective $i \in \{1, \ldots, p\}$. We assume the standard pointwise arithmetic operations, namely $\vec{u}+\vec{v} = (u_1+v_1, \ldots, u_p+v_p)$ and $q \times \vec{u} = (q \times u_1, \ldots, q \times u_p)$, where $q \in \mathbb{R}$.

A *Multi-objective Constraint Optimisation Problem* (MOCOP) is a tuple $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, where $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a set of variables taking values in finite domains $\mathbf{D} = \{D_1, \ldots, D_n\}$, respectively, and $\mathbf{F} = \{f_1, \ldots, f_r\}$ is a set of utility functions. A utility function $f_i(Y) \in \mathbf{F}$ is defined over a subset of variables $Y \subseteq \mathbf{X}$, called its *scope*, and associates a utility vector to each assignment of Y. The scopes of the functions imply a *primal graph* characterized by a certain induced width [Dechter, 2003]. The objective function is $\mathcal{F}(\mathbf{X}) = \sum_{i=1}^{r} f_i(Y_i)$, which we want to maximise. A *solution* is a complete assignment $\bar{x} = (x_1, \ldots, x_n)$ and is characterized by a utility vector $\vec{u} = \mathcal{F}(\bar{x})$. Hence, the solution space of $\mathcal{M}$ is only partially ordered and the comparison of solutions reduces to that of their corresponding p-dimensional vectors.

DEFINITION **1 (weak Pareto order)** *Let $\vec{u}, \vec{v} \in \mathbb{R}^p$ so that $\vec{u} = (u_1, \ldots, u_p)$ and $\vec{v} = (v_1, \ldots, v_p)$. We define the binary relation $\geq$ on $\mathbb{R}^p$ by $\vec{u} \geq \vec{v} \iff \forall i \in \{1, \ldots, p\}, u_i \geq v_i$.*

Given $\vec{u}, \vec{v} \in \mathbb{R}^p$, if $\vec{u} \succcurlyeq \vec{v}$ then we say that $\vec{u}$ *dominates* $\vec{v}$. As usual, the symbol $\succ$ refers to the asymmetric part of $\succcurlyeq$,

namely $\vec{u} \succ \vec{v}$ if and only if $\vec{u} \succcurlyeq \vec{v}$ and it is not the case that $\vec{v} \succcurlyeq \vec{u}$. Given finite sets $U, V \subseteq \mathbb{R}^p$, we say that U *dominates* V, denoted $U \succcurlyeq V$, if $\forall \vec{v} \in V \; \exists \vec{u} \in U$ such that $\vec{u} \succcurlyeq \vec{v}$.

Reasoning with tradeoffs

Suppose we have learned some preferences of the decision maker (DM), i.e., a set Θ of pairs of the form $(\vec{u}, \vec{v})$ meaning that the decision maker prefers multi-objective vector $\vec{u}$ to $\vec{v}$. These can be viewed as a general kind of tradeoff between the objectives. We will use this input information to deduce further preferences, based on the assumption that the user's model is a weighted sum of the objectives (thus being a simple form of a Multi-attribute Utility Theory (MAUT) model [Figueira *et al.*, 2005]).

Let $\mathcal{W}$ be the set of vectors in $\mathbb{R}^p$ with all components non-negative and summing to 1. Elements of $\mathcal{W}$ are known as *weights vectors*. Each weights vector $\vec{w} \in \mathcal{W}$ induces a total pre-order $\succcurlyeq_{\vec{w}}$ on $\mathbb{R}^p$ by, for $\vec{u}, \vec{v} \in \mathbb{R}^p$, $\vec{u} \succcurlyeq_{\vec{w}} \vec{v}$ if and only if $\vec{w} \cdot \vec{u} \geq \vec{w} \cdot \vec{v}$, where e.g., $\vec{w} \cdot \vec{u} = \sum_{i=1}^{p} w_i u_i$.

We assume that the user's preference ordering over multi-objective vectors $\mathbb{R}^p$ is equal to $\succcurlyeq_{\vec{w}}$ for some weights vector $\vec{w} \in \mathcal{W}$. For $\vec{w} \in \mathcal{W}$ and preference pair $(\vec{u}, \vec{v})$, we say that $\vec{w}$ satisfies $(\vec{u}, \vec{v})$ if $\vec{u} \succcurlyeq_{\vec{w}} \vec{v}$, i.e., if $\vec{w} \cdot \vec{u} \geq \vec{w} \cdot \vec{v}$. We also say, for set of pairs Θ, that $\vec{w}$ satisfies Θ if $\vec{w}$ satisfies each pair in Θ; let $\mathcal{W}(\Theta)$, the set of *(consistent) scenarios*, be all such $\vec{w}$. If we knew the weights vector $\vec{w}$, the problem would reduce to a single-objective problem. However, all we know is that $\vec{w} \in \mathcal{W}(\Theta)$. This leads to the induced preference relation $\succcurlyeq_\Theta$ defined as follows: $\vec{u} \succcurlyeq_\Theta \vec{v}$ if and only if $\vec{u} \succcurlyeq_{\vec{w}} \vec{v}$ for all $\vec{w} \in \mathcal{W}(\Theta)$. Thus $\vec{u}$ is preferred to $\vec{v}$ if and only if the preference holds for every consistent scenario. The associated strict relation $\succ_\Theta$ is given by $\vec{u} \succ_\Theta \vec{v} \iff \vec{u} \succcurlyeq_\Theta \vec{v}$ and $\vec{v} \not\succcurlyeq_\Theta \vec{u}$. When Θ is empty, $\succcurlyeq_\Theta$ is just the Pareto ordering.

Undominated and Possibly Optimal solutions

In [Marinescu *et al.*, 2013], methods were developed for computing the set of solutions whose utility vectors are undominated with respect to $\succcurlyeq_\Theta$, i.e., the solutions $\vec{x}$ such that for all solutions $\vec{y}$, it is not the case that $\mathcal{F}(\vec{y}) \succ_\Theta \mathcal{F}(\vec{x})$. (If Θ is empty, the undominated solutions are therefore just the Pareto-optimal solutions.) However, there are other natural notions of optimality, in particular, being *possibly optimal*, i.e., optimal in at least one scenario. Thus $\vec{x}$ is possibly optimal if there exists some scenario $\vec{w} \in \mathcal{W}(\Theta)$ such that for all solutions $\vec{y}$, $\mathcal{F}(\vec{x}) \succcurlyeq_{\vec{w}} \mathcal{F}(\vec{y})$. Typically, this will be a smaller set of solutions, although some solutions are possibly optimal without being undominated; for example, if Θ is empty, a solution with maximal value of the first objective will be possibly optimal, since it is optimal in scenario $(1, 0, 0, \ldots)$, but may not be undominated, e.g., if it's Pareto dominated by a solution with equal value in the first objective and better values in other objectives. Pareto-optimal solutions which are also possibly optimal are sometimes called *supported* solutions.

Example 1 *Figure 1 shows a MOCOP instance with 5 bi-valued variables $\{X_0, X_1, X_2, X_3, X_4\}$ and 3 utility functions $\{f_1, f_2, f_3\}$. Its corresponding primal graph is depicted in Figure 1(b). The Pareto set of the problem contains 8 solutions with associated undominated utility vectors: $(3, 24)$, $(8, 21)$, $(9, 19)$, $(10, 16)$, $(11, 14)$, $(12, 12)$, $(13, 8)$ and $(14, 6)$. Of these all are possibly optimal except $(10, 16)$, $(11, 14)$, and $(13, 8)$, the latter not being supported solutions. For example, $(8, 21)$ is possibly optimal because it is optimal in scenario $(0.5, 0.5)$, with the weighted utility of $(0.5 \times 8) + (0.5 \times 21) = 14.5$. Vector $(10, 16)$ is not possibly optimal since in every scenario it is worse than either $(9, 19)$ or $(12, 12)$. Specifically, $(10, 16) \succcurlyeq_{\vec{w}} (9, 19)$ only if $10w_1 + 16w_2 \geq 9w_1 + 19w_2$, i.e., $w_1 \geq 3w_2$, which, since $w_2 = 1 - w_1$, is if and only if $w_1 \geq 3/4$. Similarly, $(10, 16) \succcurlyeq_{\vec{w}} (12, 12)$ only if $w_1 \leq 2/3$, so there is no $\vec{w}$ that satisfies both conditions. Suppose that the decision maker reveals that he prefers $(1, 0)$ to $(0, 1)$, which implies a unit of the first objective is more valuable than a unit of the second objective to the decision maker. Thus we have $\Theta = \{((1, 0), (0, 1))\}$, which implies that $w_1 \geq w_2$ in all scenarios $(w_1, w_2) \in \mathcal{W}(\Theta)$, and thus $w_1 \geq 0.5$. Vector $(3, 24)$ is now dominated by $(8, 21)$, since $(3, 24) \succcurlyeq_{\vec{w}} (8, 21)$ only if $5w_1 \leq 3w_2$, and so $(8, 21) \succ_{\vec{w}} (3, 24)$ for all $\vec{w} \in \mathcal{W}(\Theta)$. The $\succcurlyeq_\Theta$-undominated elements are then $\{(8, 21), (9, 19), (10, 16), (11, 14), (12, 12), (13, 8), (14, 6)\}$, and the possibly optimal ones are $\{(8, 21), (9, 19), (12, 12), (14, 6)\}$.*

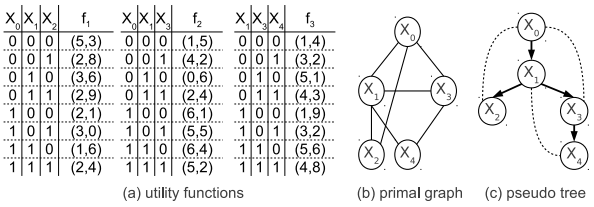

Figure 1: A MOCOP instance with 2 objectives.

2.2 AND/OR Search Spaces for MOCOPs

Significant recent improvements in search for solving MOCOPs have been achieved by using AND/OR search spaces, which often capture problem structure far better than standard OR search methods [Marinescu *et al.*, 2013; Marinescu, 2009; Dechter and Mateescu, 2007]. The AND/OR search space is defined using a *pseudo tree* of the primal graph which captures problem decomposition, as follows.

DEFINITION **2 (pseudo tree)** *A* pseudo tree *of an undirected graph $G = (V, E)$ is a directed rooted tree $\mathcal{T} = (V, E')$, such that every arc of G not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node in $\mathcal{T}$ to an ancestor in $\mathcal{T}$. The arcs in E' may not all be included in E.*

Given a MOCOP instance $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ with primal graph G and pseudo tree $\mathcal{T}$ of G, the *AND/OR search tree $S_\mathcal{T}$* based on $\mathcal{T}$ has alternating levels of OR nodes corresponding to the variables, and AND nodes corresponding to the values of the OR parent's variable, with edge weights extracted from the original functions $\mathbf{F}$ (for details see [Dechter and Mateescu, 2007; Marinescu, 2009]). Each node $n \in S_\mathcal{T}$ is associated with a *value* $v(n)$, defined as the set of utility vectors corresponding to the optimal solutions of the conditioned subproblem rooted at n. The node values can be computed recursively based on the values of their successors. The size of

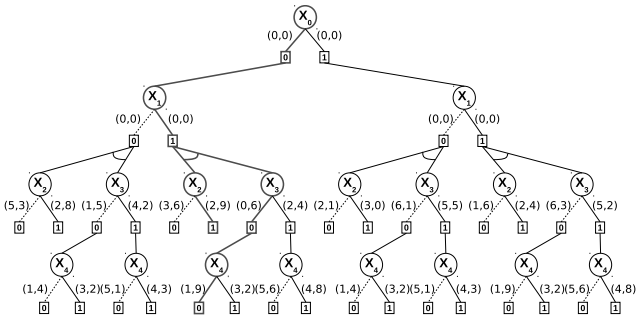

Figure 2: Weighted AND/OR search tree.

$S_{\mathcal{T}}$ is exponential in the depth of the pseudo tree [Dechter and Mateescu, 2007]. Figure 2 shows the AND/OR search tree of the MOCOP instance from Figure 1, relative to the pseudo tree given in Figure 1(c). The utility vectors displayed on the OR-to-AND arcs are the edge weights corresponding to the input utility functions. An optimal solution tree corresponding to the assignment $(X_0 = 0, X_1 = 1, X_2 = 1, X_3 = 0, X_4 = 0)$ with utility vector $(3, 24)$ is highlighted.

2.3 Multi-objective AND/OR Branch and Bound

One of the most effective methods for solving MOCOPs is the multi-objective AND/OR Branch-and-Bound (MOAOBB) algorithm [Marinescu, 2009; Marinescu *et al.*, 2013]. MOAOBB traverses the AND/OR search tree in a depth-first manner while maintaining at the root node s the set $v(s)$ of best solution vectors found so far. During node expansion, the algorithm uses a heuristic function $h(\cdot)$ to compute an upper bound set $f(T')$ on the set of optimal solutions extending the current partial solution tree T', and prunes the subproblem below the current tip node n' if $f(T')$ is dominated by $v(s)$ (i.e., $v(s) \succcurlyeq f(T')$). The node values are updated recursively in a bottom-up manner, starting from the terminal nodes in the search tree – AND nodes by summation, OR nodes by maximisation (union followed by removing dominated vectors).

The efficiency of MOAOBB greatly depends on the accuracy of the heuristic function. We used in our experiments the *multi-objective mini-bucket heuristic* [Marinescu, 2009] which was enhanced recently in [Marinescu *et al.*, 2013]. A control parameter, called i-bound, allows a tradeoff between accuracy of the heuristic and its time-space requirements.

3 Optimality Operators

We consider here an abstract notion of optimality. For set of alternatives or decisions A, $\mathrm{OPT}(A)$ is intended to be the set of optimal ones, with respect to some particular notion of optimality; we'll consider specific notions of optimality later, in Section 3.3.

3.1 Axioms and Properties

DEFINITION **3** *Let $\mathcal{D}$ be a finite set. We say that* OPT *is an Optimality Operator*[1] *over $\mathcal{D}$ if* OPT *is a function from $2^{\mathcal{D}}$ to $2^{\mathcal{D}}$ satisfying the following three conditions, for arbitrary $A, B \subseteq \mathcal{D}$.*

(1) $\mathrm{OPT}(A) \subseteq A$*;*

(2) If $A \subseteq B$ *then* $\mathrm{OPT}(B) \cap A \subseteq \mathrm{OPT}(A)$*;*

(3) If $\mathrm{OPT}(B) \subseteq A \subseteq B$ *then* $\mathrm{OPT}(A) = \mathrm{OPT}(B)$.

The axioms imply that OPT is idempotent. The following property, known as *Path Independence* in the social choice function literature [Moulin, 1985], is another important consequence of the axioms.

Union Decomposition: We say that function $\mathrm{OPT} : 2^{\mathcal{D}} \to 2^{\mathcal{D}}$ satisfies the *Union Decomposition property* if for any $A, B \subseteq \mathcal{D}$, $\mathrm{OPT}(A \cup B) = \mathrm{OPT}(\mathrm{OPT}(A) \cup B)$ (which is then equal to $\mathrm{OPT}(\mathrm{OPT}(A) \cup \mathrm{OPT}(B))$). This property will be important for the use of iterative algorithms, such as branch-and-bound algorithms, to compute $\mathrm{OPT}(T)$ for some set T. It shows that if we want to compute $\mathrm{OPT}(T)$, we can, if we wish, replace a subset A of T by $\mathrm{OPT}(A)$, without changing the result: $\mathrm{OPT}(T) = \mathrm{OPT}(\mathrm{OPT}(A) \cup (T - A))$.

Additive Decomposition: The branch-and-bound algorithms we will use in Section 5 make use of AND/OR search, where the AND nodes use an additive decomposition. Suppose that $\mathcal{D}$ is a subset of $\mathbb{R}^p$ for some $p = 1, 2, \ldots$. We say that $\mathrm{OPT} : 2^{\mathcal{D}} \to 2^{\mathcal{D}}$ satisfies the *Additive Decomposition property* if for any $A, B \subseteq \mathcal{D}$, $\mathrm{OPT}(A + B) = \mathrm{OPT}(\mathrm{OPT}(A) + B)$ (which is then equal to $\mathrm{OPT}(\mathrm{OPT}(A) + \mathrm{OPT}(B))$). This property is necessary for the correctness of our AND/OR search approach. Let us say that OPT is *translation invariant* if for any $A \subseteq \mathcal{D}$ and $b \in \mathcal{D}$, $\mathrm{OPT}(A + b) = \mathrm{OPT}(A) + b$. Proposition 1 shows that translation invariance is a sufficient condition for the Additive Decomposition property.

PROPOSITION **1** *Suppose that $\mathcal{D}$ is a subset of $\mathbb{R}^p$ for some $p = 1, 2, \ldots$, and let* OPT *be an Optimality Operator over $\mathcal{D}$ that is translation invariant. Then* OPT *satisfies the Additive Decomposition property.*

3.2 Incremental Computation of $\mathrm{OPT}(T)$

In this section we consider some Optimality Operator OPT (see Definition 3), and define algorithms for computing $\mathrm{OPT}(T)$, given an input set of decisions T. The set of decisions T may be very large, and in the combinatorial case might, for instance, be obtained using a backtracking search algorithm.

function INCREMENTALO(T)
$\Omega := \emptyset$
for $\alpha \in T$
  if $\alpha \in \mathrm{OPT}(\Omega \cup \{\alpha\})$ **then** $\Omega := \mathrm{OPT}(\Omega \cup \{\alpha\})$
end for
return Ω

The axioms in Definition 3 imply the correctness of the algorithm. In fact, given Property (1) from Definition 3, the

[1] These properties have been explored for social choice functions (which assume that $\mathrm{OPT}(A)$ is always non-empty), including, for instance, their relationship with path independence (union decomposition); see the survey article [Moulin, 1985], and also [Wilson *et al.*, 2015] for details.

algorithm is correct if and only if OPT is an Optimality Operator if and only if OPT satisfies union decomposition.

PROPOSITION **2** *Let* OPT *be an Optimality Operator based on universal set* $\mathcal{D}$. *Assume that* $T \subseteq \mathcal{D}$. *Then* INCREMENTALO$(T) = \mathrm{OPT}(T)$.

In INCREMENTALO, when we find that $\alpha \in \mathrm{OPT}(\Omega \cup \{\alpha\})$, this may mean that some elements of Ω are not in $\mathrm{OPT}(\Omega \cup \{\alpha\})$. Computing $\mathrm{OPT}(\Omega \cup \{\alpha\})$ can be expensive (and a good deal more expensive than just testing if $\alpha \in \mathrm{OPT}(\Omega \cup \{\alpha\})$). A variation of this algorithm is to delay the application of the OPT operator until the end, or only to apply it some of the time. The union decomposition property implies that these variations still correctly compute $\mathrm{OPT}(T)$.

3.3 A Framework for Different Notions of Optimality

In many decision making situations, there isn't a clear ordering on decisions. There can often be a set of different scenarios, each compatible with the preference information we have, with a different ordering on decisions in each scenario. In such a setup there are a number of different natural ways of defining the set of optimal solutions. We use a formalism for this setup from [Wilson and O'Mahony, 2011; O'Mahony and Wilson, 2013].

We define a *multiple-ordering decision structure (MODS)* $\mathcal{G}$ to be a tuple $\langle \mathcal{D}, S, \{\succcurlyeq_s : s \in S\}\rangle$, where $\mathcal{D}$ is a non-empty finite set, known as the set of decisions, S (the set of scenarios) is a non-empty set (that can be finite or infinite), and, for each $s \in S$, relation $\succcurlyeq_s$ is a total pre-order on $\mathcal{D}$. The strict part of $\succcurlyeq_s$ is written as $\succ_s$, and the corresponding equivalence relation is written $\equiv_s$. Thus, for $\alpha, \beta \in \mathcal{D}$, $\alpha \equiv_s \beta$ if and only if $\alpha \succcurlyeq_s \beta$ and $\beta \succcurlyeq_s \alpha$, and $\alpha \succ_s \beta$ if and only if $\alpha \succcurlyeq_s \beta$ and $\beta \not\succcurlyeq_s \alpha$.

Some basic notions associated with MODS $\mathcal{G}$: We make the following definitions, relative to some MODS $\mathcal{G} = \langle \mathcal{D}, S, \{\succcurlyeq_s : s \in S\}\rangle$, and where α and β are some arbitrary elements of $\mathcal{D}$.

- We say that α *necessarily dominates* β, written $\alpha \succcurlyeq_N \beta$, if for all $s \in S$, $\alpha \succcurlyeq_s \beta$. Thus $\succcurlyeq_N$ is the intersection of $\succcurlyeq_s$ over all $s \in S$.
- We define $\succ_N$ to be the strict part of $\succcurlyeq_N$.
- We define $\equiv$ to be the equivalence relation associated with $\succcurlyeq_N$. If $\alpha \equiv \beta$ then we say that α and β are *necessarily equivalent* (sometimes abbreviated to *equivalent*). $\equiv$ is the intersection of $\equiv_s$ over all $s \in S$.

Consider an arbitrary subset T of the set of decisions $\mathcal{D}$. There are quite a number of different ways of defining the set of optimal decisions in T.

$\mathrm{PO}_{\mathcal{G}}$(T) is the set of possibly optimal elements in T, i.e., elements that are optimal in some scenario. Thus $\mathrm{PO}_{\mathcal{G}}$ is the set of $\alpha \in T$ such that $\exists s \in S, \forall \beta \in T$: $\alpha \succcurlyeq_s \beta$.

$\mathrm{CSD}_{\mathcal{G}}(T)$: For $\alpha \in T$, $\alpha \in \mathrm{CSD}_{\mathcal{G}}(T)$ if and only if it *can strictly dominate* any non-equivalent decision in T, i.e., for all $\beta \in T$ such that $\beta \not\equiv \alpha$, there exists $s \in S$ such that $\alpha \succ_s \beta$. $\mathrm{CSD}_{\mathcal{G}}$ are the decisions that are undominated in T with respect to $\succ_N$.

$\mathrm{CD}_{\mathcal{G}}(T)$: For $\alpha \in T$, $\alpha \in \mathrm{CD}_{\mathcal{G}}(T)$ if and only if α *can dominate* any other decision in T, i.e., if and only if $\forall \beta \in T$, $\exists s \in S : \alpha \succcurlyeq_s \beta$.

This defines functions $\mathrm{PO}_{\mathcal{G}}$, $\mathrm{CD}_{\mathcal{G}}$ and $\mathrm{CSD}_{\mathcal{G}}$ from $2^{\mathcal{D}}$ to $2^{\mathcal{D}}$. When the intended choice of $\mathcal{G}$ is clear, we'll abbreviate $\mathrm{PO}_{\mathcal{G}}(T)$ to just $\mathrm{PO}(T)$, and similarly, for the other classes. We also define $\mathrm{POCSD}(T) = \mathrm{PO}(T) \cap \mathrm{CSD}(T)$. We always have: $\mathrm{POCSD}(T) \subseteq \mathrm{PO}(T), \mathrm{CSD}(T) \subseteq \mathrm{CD}(T)$.

PROPOSITION **3** *For any multiple-ordering decision structure (MODS)* $\mathcal{G}$, *the operators* $\mathrm{PO}_{\mathcal{G}}$, $\mathrm{CD}_{\mathcal{G}}$, $\mathrm{CSD}_{\mathcal{G}}$ *and* $\mathrm{POCSD}_{\mathcal{G}}$ *are all Optimality Operators.*

Multi-objective instance of MODS

We can generate a MODS $\mathcal{G}'$ for multi-objective reasoning with tradeoffs (see Section 2). Let $\mathcal{D}$ be the set of utility vectors generated by all solutions of a MOCOP. Given input preferences Θ, define $\mathcal{G}'$ to be $\langle \mathcal{D}, \mathcal{W}(\Theta), \{\succcurlyeq_{\vec{w}} : \vec{w} \in \mathcal{W}(\Theta)\}\rangle$. (For each $\vec{w} \in \mathcal{W}(\Theta)$, the total pre-order $\succcurlyeq_{\vec{w}}$ on $\mathbb{R}^p$ induces a relation on $\mathcal{D}$, which, for convenience we also call $\succcurlyeq_{\vec{w}}$.) The necessarily dominates relation $\succcurlyeq_N$ is then just $\succcurlyeq_\Theta$. Therefore, for a given set of solutions with associated set of multi-objective vectors $\mathcal{D}$, the set $\mathrm{CSD}_{\mathcal{G}'}(\mathcal{D})$ are the $\succcurlyeq_\Theta$-undominated vectors in $\mathcal{D}$. Also, $\mathrm{PO}_{\mathcal{G}'}(\mathcal{D})$ is the set of possibly optimal utility vectors.

In our combinatorial setting, each solution $\vec{x}$ is associated with a multi-objective vector $\vec{u}$. We extend the different notions of optimality to solutions. For example, we say that $\vec{x}$ is possibly optimal if and only if $\vec{u}$ is possibly optimal.

It follows easily that in this setting PO, CSD, POCSD and CD satisfy Translation Invariance, because we always have $\vec{u} \succcurlyeq_{\vec{w}} \vec{v} \iff \vec{u} + \vec{u}' \succcurlyeq_{\vec{w}} \vec{v} + \vec{u}'$. We thus have by Propositions 1 and 3 the following result, which is key for the correctness of the AND/OR branch-and-bound algorithm.

PROPOSITION **4** *Operators* $\mathrm{PO}_{\mathcal{G}'}$, $\mathrm{CSD}_{\mathcal{G}'}$, $\mathrm{POCSD}_{\mathcal{G}'}$ *and* $\mathrm{CD}_{\mathcal{G}'}$ *are Optimality Operators and satisfy Union Decomposition and Additive Decomposition.*

4 Computation of Possibly Optimal Solutions

We discuss how to adapt the AND/OR branch-and-bound algorithm for computing the set of possibly optimal solutions. The behavior when processing an OR-node is based on the INCREMENTALO algorithm, the correctness of which is implied by Propositions 2, 3 and 4. (If there were no AND-nodes, as in a more standard branch-and-bound algorithm, then INCREMENTALO gives the structure of the algorithm.) The correctness of the behavior at the AND nodes follows from the Additive Decomposition property (Proposition 4). Note that these results apply for the optimality operator CSD as well, thus justifying the correctness of the approach in [Marinescu *et al.*, 2013].

4.1 Checking PO Condition

A basic component of the algorithm INCREMENTALO(T) is repeated checking of a condition of the form $\vec{u} \in \mathrm{PO}(\Omega \cup \{\vec{u}\})$. We have that $\vec{u}$ is in $\mathrm{PO}(\Omega \cup \{\vec{u}\})$ if and only if there exists some $\vec{w} \in \mathcal{W}(\Theta)$ such that for all $\vec{v} \in \Omega$, $\vec{w} \cdot \vec{u} \geq \vec{w} \cdot \vec{v}$. The constraints defining $\mathcal{W}(\Theta)$ are linear inequalities, so checking if $\vec{u}$ is in $\mathrm{PO}(\Omega \cup \{\vec{u}\})$ amounts to checking if a

set of linear inequalities has a solution. Let $\mathcal{C}[\vec{u};\Omega]$ be this set of linear inequalities (on vector of variables $\vec{w}$).

For Algorithm INCREMENTALO(T), whenever we find that $\vec{u} \in \text{PO}(\Omega \cup \{\vec{u}\})$, it can be useful to store an associated scenario $\vec{w}^{\vec{u}}$ that makes $\vec{u}$ optimal in $\Omega \cup \{\vec{u}\}$, as this can help the efficiency of the next stage.

When we find that $\vec{u} \in \text{PO}(\Omega \cup \{\vec{u}\})$, we next need to compute $\text{PO}(\Omega \cup \{\vec{u}\})$. This involves eliminating elements of Ω that are no longer possibly optimal, because of adding $\vec{u}$. Consider any element $\vec{v}$ of Ω. When this was added we stored an associated scenario $\vec{w}^{\vec{v}}$ (see above) which then made $\vec{v}$ optimal w.r.t. the then-current set of decisions. We first check to see if $\vec{v} \cdot \vec{w}^{\vec{v}} \geq \vec{u} \cdot \vec{w}^{\vec{v}}$. If so, then $\vec{v}$ is still optimal within $\Omega \cup \{\vec{u}\}$ in scenario $\vec{w}^{\vec{v}}$. Otherwise, we test the consistency of the set of linear constraints $\mathcal{C}[\vec{v};\Omega \cup \{\vec{u}\}]$. If this has no solution then we delete $\vec{v}$ from Ω. If this has a solution $\vec{w}$ then we keep $\vec{v}$ in Ω, but reset $\vec{w}^{\vec{v}}$ to be $\vec{w}$.

4.2 Use of Bounds in Branch-and-Bound Algorithm

The algorithm from [Marinescu *et al.*, 2013] constructs an upper bound set at each node of the search tree. The upper bound set is a set $\mathcal{U}$ of utility vectors such that for any complete assignment $\bar{x}$ extending the current partial solution tree, there exists an element $\vec{u}$ in $\mathcal{U}$ with $\vec{u} \succcurlyeq_\Theta \vec{v}$, where $\vec{v}$ is the utility vector associated with $\bar{x}$.

For computing possibly optimal elements we can also use such an upper bound set, defined in exactly the same way. The results of [Marinescu *et al.*, 2013] suggest it is best to use a small upper bound set.

The idea behind the search tree pruning is that we can backtrack at this node if for each $\vec{u} \in \mathcal{U}$, $\vec{u} \notin \text{PO}(\Omega \cup \{\vec{u}\})$, where Ω is the current set of tentatively possibly optimal utility vectors. This is because this implies that for any complete assignment $\bar{x}$ extending the current search tree node, with associated utility vector $\vec{v}$, we have $\vec{v} \notin \text{PO}(\Omega \cup \{\vec{v}\})$, and thus $\bar{x}$ is not a possibly optimal solution. We consider different sufficient conditions for $\vec{u} \notin \text{PO}(\Omega \cup \{\vec{u}\})$.

Sufficient Condition (1): Checking $\vec{u} \notin \text{PO}(\Omega \cup \{\vec{u}\})$ directly, by checking the consistency of the associated set of linear constraints (see Section 4.1). A disadvantage of this is that it can involve a very large number of linear constraints; in particular, there is a constraint for each element of Ω.

Sufficient Condition (2): We check if $\vec{u} \notin \text{CD}(\Omega \cup \{\vec{u}\})$. It can be seen that this is equivalent to testing if there exists some $\vec{v} \in \Omega$, such that $\vec{u} \notin \text{PO}(\{\vec{v},\vec{u}\})$. That is, we prune if for all $\vec{u} \in \mathcal{U}$, there exists some $\vec{v} \in \Omega$ such that $\vec{u} \notin \text{PO}(\{\vec{v},\vec{u}\})$. The condition $\vec{u} \notin \text{PO}(\{\vec{v},\vec{u}\})$ amounts to checking the consistency of a smaller set of linear constraints.

Sufficient Condition (3): We choose some very small number $\epsilon > 0$, and let us write the vector $(\epsilon,\ldots,\epsilon)$ in $\mathbb{R}^p$ as $\bar{\epsilon}$. We check if there exists some $\vec{v} \in \Omega$, such that $\vec{v} \succcurlyeq_\Theta \vec{u} + \bar{\epsilon}$. This can be performed using the methods in [Marinescu *et al.*, 2013], in particular the method involving compilation of $\succcurlyeq_\Theta$-dominance using matrix multiplication.

Sufficient Condition (4): We check if there exists some $\vec{v} \in \Omega$, such that $\vec{v} \succ \vec{u}$, i.e., $\vec{v}$ Pareto-dominates $\vec{u}$.

The following result implies that the pruning given by (1) is at least as strong as that given by (2), which is at least as strong as that given by (3). (Pruning with Condition (4) will typically be weaker than using Condition (3), but this will not always hold.)

LEMMA **1** *If there exists some $\vec{v} \in \Omega$, such that $\vec{v} \succcurlyeq_\Theta \vec{u} + \bar{\epsilon}$ then there exists some $\vec{v} \in \Omega$ such that $\vec{u} \notin \text{PO}(\{\vec{v},\vec{u}\})$.*

If there exists some $\vec{v} \in \Omega$ such that $\vec{u} \notin \text{PO}(\{\vec{v},\vec{u}\})$ then $\vec{u} \notin \text{PO}(\Omega \cup \{\vec{u}\})$.

5 Experiments

In this section, we evaluate empirically the performance of the proposed branch-and-bound algorithm on three classes of MOCOP benchmarks: random networks, vertex coverings and combinatorial auctions [Marinescu *et al.*, 2013]. We consider the random generator from [Marinescu *et al.*, 2012] for generating consistent random tradeoffs, namely pairwise or binary tradeoffs (K) and 3-way tradeoffs (T) for 2, 3 and 5 objectives. We implemented all algorithms in C++ and the experiments were conducted on a 2.6GHz quad-core processor with 4 GB of RAM.

For all our experiments we use the similar upper bound bounding schemes presented in [Marinescu *et al.*, 2013]. Specifically, MOAOBB denotes the multi-objective AND/OR Branch-and-Bound from [Marinescu, 2009]. In addition, **B=b (PLUB)** is the extension of MOAOBB that uses the Pareto least upper bound to reduce the upper bound set to at most b (≥ 1) utility vectors, for both the Pareto and tradeoffs cases, while **B=b (LP)** is the extension of MOAOBB that uses an LP based method to compute the upper bound sets, for the tradeoffs case only (see also [Marinescu *et al.*, 2013]). For random networks we allow the upper bounds to have at most $B = 5$ elements, and for vertex covering and combinatorial auctions we use the upper bounds with at most $B = 2$ elements. For reference, we also ran a baseline OR branch-and-bound algorithm called MOBB [Marinescu, 2009]. All these algorithms can be used to solve the CSD, PO and POCSD tasks.

Comparison of AND/OR versus OR search: In Figure 3 we compare algorithms MOAOBB and MOBB for computing the PO sets on random networks with 5 objectives, $K = 6$ pairwise and $T = 3$ three-way tradeoffs. We plot both the number of problem instances solved (top) and the CPU time (bottom), as a function of problem size (number of variables). We see clearly that AND/OR search is superior to regular OR search, solving many more problems. The results obtained on vertex covering and combinatorial auctions had a similar pattern, and therefore are omitted for space reasons.

Comparison of sufficient conditions for pruning: Figure 4 shows the comparison between different sufficient conditions for pruning for vertex coverings with 5 objectives. The number of pairwise and 3-way tradeoffs are fixed to 5 and 2, respectively. Results are averaged over 50 problem instances. The time limit for solving each problem instance was 20 minutes. We can see that sufficient condition (3) is the fastest and is able to solve a larger number of problems. We noticed that using condition (1), the algorithm expands less nodes, which

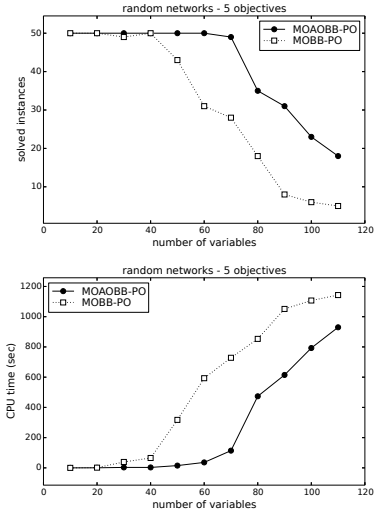

Figure 3: Number of problem instances solved (top) and CPU time in seconds (bottom) as a function of the number of variables for random networks with 5 objectives and $(K = 6, T = 3)$ tradeoffs. Time limit 20 minutes.

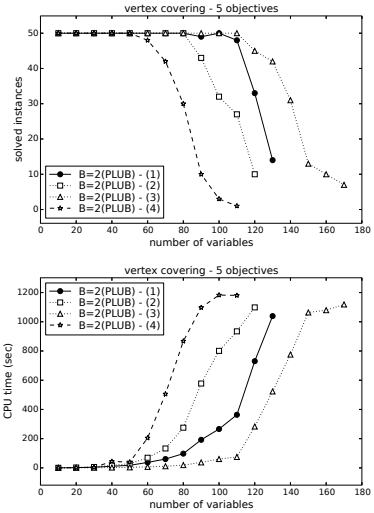

Figure 4: Number of problem instances solved (top) and CPU time in seconds (bottom) as a function of the number of variables for vertex coverings with 5 objectives and $(K = 5, T = 2)$ tradeoffs. Time limit 20 minutes.

implies that it prunes more effectively than the other sufficient conditions. However, because of the additional computational overhead, condition (1) is slower than (3).

Impact of the number of tradeoffs: Figure 5 shows the impact of the number of pairwise tradeoffs K for vertex covering problems with 130 variables and 5 objectives and for fixed number of 3-way tradeoffs ($T = 1$). Results were based on 100 problem instances and each instance was given a 20 minute time limit. As K increases, we can see that the algorithm solves more problems and consequently the running time of the algorithm decreases substantially. We observed similar results for the other two families of problems.

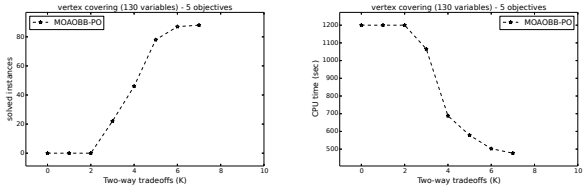

Figure 5: Number of problems solved (top) and CPU time in seconds (bottom) as a function of the number of pairwise tradeoffs (K) for vertex covering problems with n = 130 and 5 objectives ($T = 1$). Time limit 20 minutes.

Comparison between the CSD**s,** PO**s and** POCSD**s:** Figure 6 compares cardinalities of the sets CSD, PO and POCSD for combinatorial auctions with 3 objectives, vertex coverings with 5 objectives and random networks with 5 objectives, respectively. The number of pairwise and 3-way tradeoffs are fixed as follows: $(K = 2, T = 1)$ for auctions, $(K = 5, T = 2)$ for vertex coverings, and $(K = 6, T = 3)$ for random networks. Results were obtained on 50 randomly generated problem instances from each benchmark. However, we report the average cardinality and CPU time over those instances that were solved for all three optimality operators. For computing CSDs we take the approach from [Marinescu *et al.*, 2013]. We can notice that as the number of variables increase POs become much smaller than CSDs, for instance, for problems with 100 variables, POs are almost 4 times smaller than CSDs. We also observed that the average ratio $|\text{CSD}|/|\text{PO}|$ over the problem instances solved by both the algorithms is approximately linear in the number of variables (results were omitted for lack of space). On the other hand, computing POs is slower because of the repeated checking of the more expensive optimality condition. Finally, we see that the POCSD sets, which are almost the same size as the corresponding PO ones, can be computed almost as fast as the CSD ones. This is important because the decision maker can be presented with the POCSD sets first before making any decision or eliciting more tradeoffs.

6 Conclusion

We showed how an AND/OR search algorithm can be used for computing the possibly optimal solutions for multi-objective constraint optimisation problem involving tradeoffs. Our experimental results indicate that the approach scales to moderately-sized problems. The set of possibly optimal solutions is often considerably smaller than the set of undominated solutions, which can be helpful for a decision maker. Although the former set is somewhat slower to compute, computing the intersection set is almost as fast.

Our notion of possible optimality is related to the convex coverage sets introduced by [Roijers *et al.*, 2013; 2014] in the context of multi-objective coordination graphs. The algorithms proposed there are based on variable elimination and therefore limited to problems having relatively small induced

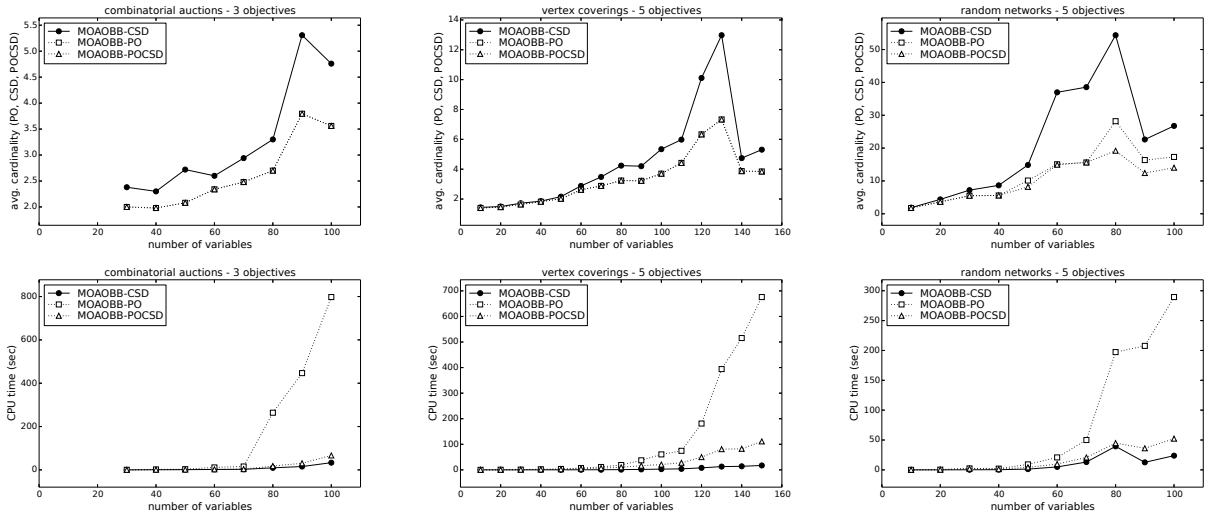

Figure 6: Average number of solutions (top) and CPU time in seconds for computing CSD, PO and POCSD (bottom), as a function of the number of variables, for combinatorial auctions with 3 objectives and ($K = 2$, $T = 1$) tradeoffs, vertex coverings with 5 objectives and ($K = 5$, $T = 2$) tradeoffs and random networks with 5 objectives and ($K = 6$, $T = 3$) tradeoffs, respectively. Time limit 20 minutes.

widths, unlike our branch-and-bound search scheme which scale up to much more difficult problem instances.

Having taken a very general approach to optimality operator, showing what properties are required for the correctness of iterative search algorithms, our work opens up the potential for AND/OR branch-and-bound (as well as more standard OR branch-and-bound) being used for other kinds of optimality operators, and other forms of preference input.

Acknowledgments

This work was supported in part by the Science Foundation Ireland grant SFI/12/RC/2289. Nic Wilson is also supported by the School of EEE&CS, Queen's University Belfast. We are grateful to the reviewers for their helpful comments.

References

[Dechter and Mateescu, 2007] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.

[Dechter, 2003] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[Figueira *et al.*, 2005] J. Figueira, S. Greco, and M. Ehrgott. *Multiple Criteria Decision Analysis—State of the Art Surveys*. Springer International Series in Operations Research and Management Science Volume 76, 2005.

[Marinescu *et al.*, 2012] R. Marinescu, A. Razak, and N. Wilson. Multi-objective influence diagrams. In *Uncertainty in Artificial Intelligence (UAI)*, pages 574–583, 2012.

[Marinescu *et al.*, 2013] R. Marinescu, A. Razak, and N. Wilson. Multi-objective constraint optimization with tradeoffs. In *Proc. CP-2013*, pages 497–512, 2013.

[Marinescu, 2009] R. Marinescu. Exploiting problem decomposition in multi-objective constraint optimization. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 592–607, 2009.

[Moulin, 1985] H. Moulin. Choice functions over a finite set: a summary. *Social Choice and Welfare*, 2(2):147–160, 1985.

[O'Mahony and Wilson, 2013] Conor O'Mahony and Nic Wilson. Sorted-pareto dominance and qualitative notions of optimality. In *Proc. ECSQARU'2013*, pages 449–460, 2013.

[Roijers *et al.*, 2013] D. Roijers, S. Whiteson, and F. Oliehoek. Computing convex coverage sets for multi-objective coordination graphs. In *International Conference on Algorithmic Decision Theory (ADT)*, pages 309–323, 2013.

[Roijers *et al.*, 2014] D. Roijers, S. Whiteson, and F. Oliehoek. Linear support for multi-objective coordination graphs. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1297–1304, 2014.

[Wilson and O'Mahony, 2011] Nic Wilson and Conor O'Mahony. The relationships between qualitative notions of optimality for decision making under logical uncertainty. In *Proc. AICS-2011*, 2011.

[Wilson *et al.*, 2015] N. Wilson, A. Razak, and R. Marinescu. *Computing Possibly Optimal Solutions for Multi-Objective Constraint Optimisation with Tradeoffs (extended version of current paper)*. http://ucc.insight-centre.org/nwilson/POIJCAI2015longer.pdf), 2015.