CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

| | |
|---|---|
| Title | Multi-objective constraint optimization with tradeoffs |
| Authors | Marinescu, Radu;Razak, Abdul;Wilson, Nic |
| Publication date | 2013-09 |
| Original Citation | MARINESCU, R., RAZAK, A. & WILSON, N. 2013. Multi-Objective Constraint Optimization with Tradeoffs. In: SCHULTE, C. (ed.) Principles and Practice of Constraint Programming. Uppsala, Sweden, 16-20 Sep. Berlin Heidelberg: Springer, pp. 497-512 |
| Type of publication | Conference item |
| Link to publisher's version | 10.1007/978-3-642-40627-0_38 |
| Rights | ©Springer-Verlag Berlin Heidelberg 2013. The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-642-40627-0_38 |
| Download date | 2026-11-15 18:07:08 |
| Item downloaded from | https://hdl.handle.net/10468/1402 |



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

# Multi-objective Constraint Optimization with Tradeoffs

Radu Marinescu[1], Abdul Razak[2], and Nic Wilson[2]

[1] IBM Research – Ireland
radu.marinescu@ie.ibm.com
[2] Cork Constraint Computation Centre
University College Cork, Ireland
a.razak,n.wilson@4c.ucc.ie

**Abstract.** In this paper, we consider the extension of multi-objective constraint optimization algorithms to the case where there are additional tradeoffs, reducing the number of optimal solutions. We focus especially on branch-and-bound algorithms which use a mini-buckets algorithm for generating the upper bound at each node (in the context of maximizing values of objectives). Since the main bottleneck of these algorithms is the very large size of the guiding upper bound sets we introduce efficient methods for reducing these sets, yet still maintaining the upper bound property. We also propose much faster dominance checks with respect to the preference relation induced by the tradeoffs. Furthermore, we show that our tradeoffs approach which is based on a preference inference technique can also be given an alternative semantics based on the well known Multi-Attribute Utility Theory. Our comprehensive experimental results on common multi-objective constraint optimization benchmarks demonstrate that the proposed enhancements allow the algorithms to scale up to much larger problems than before.

## 1 Introduction

Multi-objective Constraint Optimization (MOCOP) is a general framework that can be used to model many real-world problems involving multiple, conflicting and sometimes non-commensurate objectives that need to be optimized simultaneously. The solution space of these problems is typically only partially ordered and can contain many non-inferior or undominated solutions which must be considered equally good in the absence of information concerning the relevance of each objective relative to the others.

Solutions are compared on more than one (real-valued) objective, so that each complete assignment to the decision variables has an associated multi-objective utility value, represented by a vector in $\mathbb{R}^p$, where $p$ is the number of objectives. The utility vectors associated to solutions can be compared using the Pareto ordering. However, the Pareto ordering is rather weak, which can lead to the Pareto-undominated set becoming too large for the decision maker (DM) to handle. At the other extreme, it can be undesirable to force the decision maker to define precise tradeoffs between objectives, since they may have no clear idea about them, and it may lead to somewhat arbitrary decisions.

The approach we take, based on that of [1], is to allow as input a number of preferences between utility vectors, which may come e.g., from a brief elicitation procedure. These input preferences are used to strengthen the preference relation over $\mathbb{R}^p$; even a small number of such tradeoffs can greatly reduce the size of the undominated set.

In this paper, we extend MOCOP algorithms for the case with tradeoffs, including both branch-and-bound and variable elimination algorithms. The branch-and-bound algorithms perform a depth-first traversal of an AND/OR search tree that captures the underlying structure of the problem, and use a mini-buckets algorithm for generating an upper bound, which is a set of utility vectors, at each node of the search tree. The main contributions of the paper are as follows. First, the guiding upper bound sets can become quite large and therefore can have a dramatic impact on the performance of the search algorithms. To remedy this issue, we propose efficient methods for reducing these sets, yet still ensuring the upper bound property for both the Pareto and tradeoffs case. Second, the MOCOP algorithms need to make many dominance checks with respect to the preference relation induced by the tradeoffs. For computational efficiency we compile this dominance check by use of a matrix and show that in practice it can achieve almost an order of magnitude speed up over the current approach of [1] which is based on solving a linear program. Third, we show that our approach for handling tradeoffs can be given an alternate semantics based on the well known Multi-Attribute Utility Theory. Fourth, we show empirically on a variety of MOCOP benchmarks that our improved algorithms outperform the current state-of-the-art solvers by a significant margin and therefore they can scale up to much larger problems than before.

Following background on MOCOPs and on AND/OR search spaces for MOCOPs (Section 2), Section 3 defines the formalism for tradeoffs and shows how the induced notion of preference domination can be computed. Section 4 describes our proposed methods for reducing the size of the upper bound sets used by branch-and-bound search algorithms. Section 5 presents our empirical evaluation. Section 6 overviews related work, while Section 7 provides a summary and concluding remarks.

## 2 Background

### 2.1 Multi-objective Constraint Optimization

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

A *Multi-objective Constraint Optimization 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 decision variables having finite domains of values $\mathbf{D} = \{D_1, \ldots, D_n\}$ and $\mathbf{F} = \{f_1, \ldots, f_r\}$ is a set of utility functions.[3] 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 objective function is $\mathcal{F}(\mathbf{X}) = \sum_{i=1}^{r} f_i(Y_i)$. A *solution* is a complete assignment $\bar{x} = (x_1, \ldots, x_n)$ and is characterized by a utility vector $\boldsymbol{u} = \mathcal{F}(\bar{x})$. Therefore, the comparison of solutions reduces to that of their corresponding $p$-dimensional vectors.

We are interested in partial orders $\succcurlyeq$ on $\mathbb{R}^p$ satisfying the following two monotonicity properties, where $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^p$ are arbitrary vectors:

[3] Since we are expressing the optimization in terms of maximizing rather than minimizing, we use the terminology *utility function/vector* as opposed to *cost function/vector*.

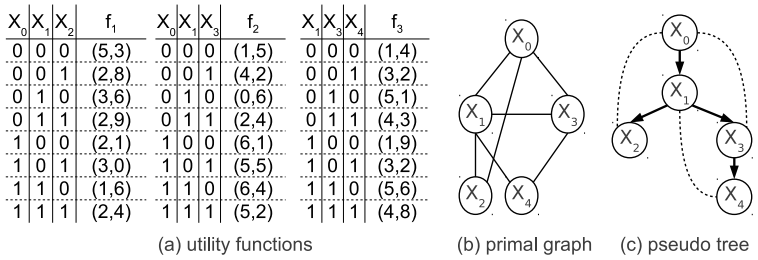



**Fig. 1.** A MOCOP instance with 2 objectives.

**Independence:** if $\boldsymbol{u} \succcurlyeq \boldsymbol{v}$ then $\boldsymbol{u} + \boldsymbol{w} \succcurlyeq \boldsymbol{v} + \boldsymbol{w}$;
**Scale-Invariance:** if $\boldsymbol{u} \succcurlyeq \boldsymbol{v}$ and $q \in \mathbb{R}$, $q \geq 0$ then $q \times \boldsymbol{u} \succcurlyeq q \times \boldsymbol{v}$.

An important example of such a partial order is the weak Pareto order, defined as follows.

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

Given $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^p$, if $\boldsymbol{u} \succcurlyeq \boldsymbol{v}$ then we say that $\boldsymbol{u}$ *dominates* $\boldsymbol{v}$. As usual, the symbol $\succ$ refers to the asymmetric part of $\succcurlyeq$, namely $\boldsymbol{u} \succ \boldsymbol{v}$ if and only if $\boldsymbol{u} \succcurlyeq \boldsymbol{v}$ and it is not the case that $\boldsymbol{v} \succcurlyeq \boldsymbol{u}$. Given finite sets $U, V \subseteq \mathbb{R}^p$, we say that $U$ *dominates* $V$, denoted $U \succcurlyeq V$, if $\forall \boldsymbol{v} \in V \; \exists \boldsymbol{u} \in U$ such that $\boldsymbol{u} \succcurlyeq \boldsymbol{v}$. In particular, relation $\geq$ (resp. $>$) is also called *weak Pareto dominance* (resp. *Pareto dominance*).

**Definition 2 (maximal/Pareto set).** *Given a partial order $\succcurlyeq$ and a finite set of vectors $U \subseteq \mathbb{R}^p$, we define the* maximal set, *denoted by $\max_{\succcurlyeq}(U)$, to be the set consisting of the undominated elements in $U$, i.e., $\max_{\succcurlyeq}(U) = \{\boldsymbol{u} \in U \mid \nexists \boldsymbol{v} \in U, \boldsymbol{v} \succ \boldsymbol{u}\}$. When $\succcurlyeq$ is the weak Pareto ordering $\geq$, we call $\max_{\succcurlyeq}(U)$ the* Pareto set *(or* Pareto frontier*).*

Solving a MOCOP instance means finding the set of optimal solutions that generate maximal utility vectors, namely values in the set $\max_{\succcurlyeq}\{\mathcal{F}(\bar{x}) \mid \text{solution } \bar{x}\}$.

Given a MOCOP instance $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, the scopes of the utility functions in $\mathbf{F}$ imply a *primal graph* $G$ (nodes correspond to the variables and edges connect any two nodes whose variables belong to the same function) with certain *induced width* [2].

*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 ternary utility functions $f_1(X_0, X_1, X_2)$, $f_2(X_0, X_1, X_3)$, and $f_3(X_1, X_3, X_4)$, respectively. Its corresponding primal graph is depicted in Figure 1(b). The Pareto set of the problem contains 8 solutions with undominated utility vectors: (3,24), (8,21), (9,19), (10,16), (11,14), (12,12), (13,8) and (14,6), respectively.

### 2.2 AND/OR Search Spaces for MOCOPs

The concept of AND/OR search spaces for graphical models [3] has been extended recently to multi-objective constraint optimization to better capture the problem structure

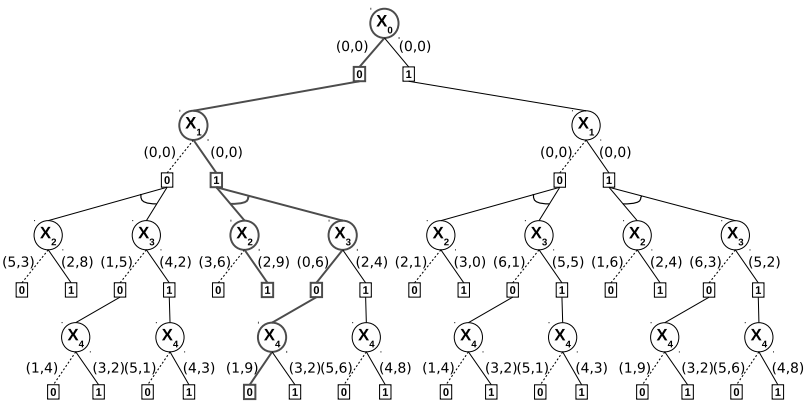



**Fig. 2.** Weighted AND/OR search tree.

during search [4]. The search space is defined using a *pseudo tree* of the primal graph which captures problem decomposition as follows.

**Definition 3 (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 $E$.*

*Weighted AND/OR Search Tree* Given a MOCOP instance $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, its primal graph $G$ and a pseudo tree $\mathcal{T}$ of $G$, the associated AND/OR search tree $S_{\mathcal{T}}$ has alternating levels of OR and AND nodes. Its structure is based on the underlying pseudo tree. The root node of $S_{\mathcal{T}}$ is an OR node labeled by the root of $\mathcal{T}$. The children of an OR node $\langle X_i \rangle$ are AND nodes labeled with value assignments $\langle X_i, x_j \rangle$; the children of an AND node $\langle X_i, x_j \rangle$ are OR nodes labeled with the children of $X_i$ in $\mathcal{T}$, representing conditionally independent subproblems. The OR-to-AND arcs in $S_{\mathcal{T}}$ are annotated by *weights* derived from the input utility functions, while 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, as shown in [4].

The size of the AND/OR search tree associated with a MOCOP instance with pseudo tree of depth $d$ is $O(n \cdot k^d)$, where $n$ is the number of variables and $k$ bounds the domain size [3]. 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 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.

*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) introduced recently by [4]. For completeness, we next review the algorithm. As usual, each node $n$ in the search tree is associated with a heuristic estimate $h(n)$ of $v(n)$.

**Algorithm 1:** MOAOBB

**Data**: $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, pseudo tree $\mathcal{T}$, heuristic $h$.
**Result**: Maximal set of $\mathcal{M}$, $\max_{\succcurlyeq}\{\mathcal{F}(\bar{x}) \mid \text{solution } \bar{x}\}$.

```
create an OR node s labeled by the root of T ;
OPEN ← {s}; CLOSED ← ∅;
while OPEN ≠ ∅ do
    move top node n from OPEN to CLOSED;
    expand n by creating its successors succ(n);
    foreach n' ∈ succ(n) do
        evaluate h(n') and add n' on top of OPEN;
        let T' be the current partial solution tree with tip n';
        if v(s) ≽ f(T') then
            remove n' from OPEN and succ(n);
    while ∃n ∈ CLOSED s.t. succ(n) = ∅ do
        remove n from CLOSED and let p be n's parent;
        if p is AND then v(p) ← v(p) + v(n);
        else v(p) ← max_≽{v(p) ∪ {w(p, n) + v(n)}};
        remove n from succ(p);
return v(s)
```

MOAOBB is described by Algorithm 1. Assuming maximization of the objective values, MOAOBB traverses the weighted AND/OR search tree in a depth-first manner while maintaining at the root node $s$ of the search tree the set $v(s)$ of best solution vectors found so far. During node expansion, the algorithm uses the $h(n)$ values 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., all utility vectors in $f(T')$ are dominated by at least one element in $v(s)$). 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 maximization (undominated closure wrt $\succcurlyeq$).

The time complexity of MOAOBB is bounded by $O(n \cdot k^d)$, the size of the weighted AND/OR search tree. Since the utility vectors are in $\mathbb{R}^p$, it is not easy to predict the size of the maximal set $\max_{\succcurlyeq}\{\mathcal{F}(\bar{x}) \mid \text{ solution } \bar{x}\}$, and therefore MOAOBB may use prohibitively large amounts of memory to store it.

*Mini-Bucket Heuristics* The heuristic function $h(n)$ that we used in our experiments is the *multi-objective mini-bucket heuristic* [4, 5]. It was shown that the intermediate functions generated by the mini-bucket approximation of multi-objective variable elimination can be used to derive a heuristic function that is *admissible*, namely in a maximization context it generates a set of utility vectors (upper bound set) such that the utility vectors of any optimal solution below node $n$ in the search tree is dominated by some element in $h(n)$. A control parameter, called $i$-bound, allows a tradeoff between accuracy of the heuristic and its time-space requirements.

## 3 Handling Imprecise Tradeoffs

In this section, we present our approach for handling tradeoffs between the objectives of a MOCOP. We first introduce some notation. For $W \subseteq \mathbb{R}^p$, define $\mathbf{C}(W)$, *the positive convex cone generated by* $W$, to be the set consisting of all vectors $\boldsymbol{u}$ such that there exists $k \geq 0$ and non-negative real scalars $q_1, \ldots, q_k$ and $\boldsymbol{w_j} \in W$ with $\boldsymbol{u} \geq \sum_{j=1}^{k} q_j \boldsymbol{w_j}$, where $\geq$ is the weak Pareto relation (and an empty summation is taken to be equal to $\mathbf{0}$, the zero vector $(0, \ldots, 0)$ in $\mathbb{R}^p$). $\mathbf{C}(W)$ is the set of vectors that weakly-Pareto dominate some (finite) positive linear combination of elements of $W$.

For $W \subseteq \mathbb{R}^p$, define $W^*$ to be the set of vectors $\boldsymbol{u} \in \mathbb{R}^p$ such that $\boldsymbol{u} \cdot \boldsymbol{v} \geq 0$ for all $\boldsymbol{v} \in W$, where $\boldsymbol{u} \cdot \boldsymbol{v}$ means $\sum_{i=1}^{p} u_i v_i$. A standard result for finitely generated convex cones (see e.g., [6]) states that for finite $W \subseteq \mathbb{R}^p$, $W^{**}$ (i.e., $(W^*)^*$) equals $\mathbf{C}(W)$.

### 3.1 Deducing Preferences from Additional Inputs

We assume that we have learned some preferences of the decision maker (DM), i.e., a set $\Theta$ of pairs of the form $(\boldsymbol{u}, \boldsymbol{v})$ meaning that the decision maker prefers $\boldsymbol{u}$ to $\boldsymbol{v}$. We will use this input information to deduce further preferences, in two different ways, the first taken from [1], the second based on a multi-attribute utility theory model.

**Partial Order-based Inference:** We will use the input preferences $\Theta$ to infer a preference relation $\succcurlyeq_\Theta$ extending $\Theta$. Here we assume that the DM has some unknown partial order $\succcurlyeq$ over $\mathbb{R}^p$, representing their preferences. We further assume that $\succcurlyeq$ extends Pareto (i.e., extends the weak Pareto order), and satisfies Independence and Scale-Invariance. We are given the information that the DM's preference relation includes pairs $\Theta$, i.e., $\succcurlyeq$ extends $\Theta$. This naturally leads to the following definitions:

- We say that $\Theta$ *is consistent* if there exists some partial order $\succcurlyeq$ (on $\mathbb{R}^p$) that extends $\Theta$, extends Pareto, and satisfies Scale-Invariance and Independence.
- For consistent $\Theta$, we define the induced preference relation $\succcurlyeq_\Theta$ on $\mathbb{R}^p$ by $\boldsymbol{u} \succcurlyeq_\Theta \boldsymbol{v} \iff \boldsymbol{u} \succcurlyeq \boldsymbol{v}$ for all partial orders $\succcurlyeq$ such that $\succcurlyeq$ extends Pareto and $\Theta$, and satisfies Independence and Scale-Invariance.

**MAUT-based Inference:** Here we take a different approach: we assume that the DM uses a weighted sum of the objectives to compare objective vectors, as in the additive form of the Multi-attribute Utility Theory (MAUT) model [7]. We say that pre-order $\succcurlyeq$ on $\mathbb{R}^p$ is MAUT-based if there exists some vector $\boldsymbol{w} \in \mathbb{R}^p$ with only non-negative values, such that, for all $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^p$, $\boldsymbol{u} \succcurlyeq \boldsymbol{v} \iff \sum_{i=1}^{p} u_i w_i \geq \sum_{i=1}^{p} v_i w_i$. If we knew the weights vector $\boldsymbol{w}$, the problem would reduce to a single-objective problem. However, all we know is that the induced preference relation satisfies the pairs $\Theta$. This leads to the induced preference relation $\succcurlyeq^2_\Theta$ defined as follows:

$\boldsymbol{u} \succcurlyeq^2_\Theta \boldsymbol{v}$ if and only if $\boldsymbol{u} \succcurlyeq \boldsymbol{v}$ holds for all MAUT-based $\succcurlyeq$ extending $\Theta$.

Thus $\boldsymbol{u}$ is preferred to $\boldsymbol{v}$ if and only if the preference holds for every compatible MAUT ordering.

Let $W_\Theta = \{\boldsymbol{u} - \boldsymbol{v} : (\boldsymbol{u}, \boldsymbol{v}) \in \Theta\}$. The following result from [1] gives a characterization of the first induced preference relation $\succcurlyeq_\Theta$.

**Proposition 1.** *Let $\Theta$ be a consistent set of pairs of vectors in $\mathbb{R}^p$. Then $\boldsymbol{u} \succeq_\Theta \boldsymbol{v}$ if and only if $\boldsymbol{u} - \boldsymbol{v} \in \mathbf{C}(W_\Theta)$.*

In fact, for finite consistent $\Theta$, the two induced preference relations are equal:

**Proposition 2.** *For finite $\Theta$, we have $\boldsymbol{u} \succcurlyeq^2_\Theta \boldsymbol{v} \iff \boldsymbol{u} - \boldsymbol{v} \in \mathbf{C}(W_\Theta)$. Therefore, if $\Theta$ is consistent then the relations $\succeq_\Theta$ and $\succcurlyeq^2_\Theta$ are equal.*

*Proof.* Any MAUT-based order $\succcurlyeq$ has an associated vector $\boldsymbol{w}$, so we write $\succcurlyeq$ as $\succcurlyeq_{\boldsymbol{w}}$. We then have $\boldsymbol{u} \succcurlyeq_{\boldsymbol{w}} \boldsymbol{v} \iff (\boldsymbol{u}-\boldsymbol{v})\cdot\boldsymbol{w} \geq 0$. Relation $\succcurlyeq_{\boldsymbol{w}}$ extends $\Theta$ if and only if, for all $\boldsymbol{u} \in W_\Theta$, $\boldsymbol{w}\cdot\boldsymbol{u} \geq 0$, i.e., iff $\boldsymbol{w} \in (W_\Theta)^*$. Thus $\boldsymbol{u} \succcurlyeq^2_\Theta \boldsymbol{v} \iff (\boldsymbol{u}-\boldsymbol{v})\cdot\boldsymbol{w} \geq 0$ for all $\boldsymbol{w} \in (W_\Theta)^*$, i.e., iff $\boldsymbol{u}-\boldsymbol{v} \in (W_\Theta)^{**}$. Since $(W_\Theta)^{**} = \mathbf{C}(W_\Theta)$, the result follows.

*Example 2.* Suppose that the decision maker has told us that she prefers $(0,1)$ to $(1,0)$, so that a unit of the second objective is considered more valuable than a unit of the first objective. $\Theta$ is then equal to $\{\big((0,1),(1,0)\big)\}$. The Independence property implies $(0,0) \succcurlyeq_\Theta (1,-1)$, and also, for example, $(8,21) \succcurlyeq_\Theta (9,20)$, and thus $(8,21) \succeq_\Theta (9,19)$ since $\succeq_\Theta$ extends Pareto. In Example 1 the additional preference implies that $(3,24)$ and $(8,21)$ are the only $\succcurlyeq_\Theta$-undominated solutions, illustrating that even a single tradeoff can greatly reduce the number of undominated solutions.

### 3.2 Implementing Dominance Tests

For multi-objective constraint optimization, we will need to make many dominance checks with respect to the induced preference relation $\succeq_\Theta$. In [1], a dominance check is performed using a linear programming (LP) solver, based on Proposition 1. For computational efficiency we compile this dominance check by use of a matrix. Specifically, we generate a matrix $\mathbf{A}$ that represents $\succeq_\Theta$ in the sense that $\boldsymbol{u} \succeq_\Theta \boldsymbol{v}$ if and only if $\mathbf{A}(\boldsymbol{u}-\boldsymbol{v}) \geq \mathbf{0}$ (which is if and only if $\mathbf{A}\boldsymbol{u} \geq \mathbf{A}\boldsymbol{v}$). Lemma 1 below shows that we can construct such a matrix $\mathbf{A}$ by finding a generating set of the dual cone $(\mathbf{C}(W_\Theta))^*$, where $(\mathbf{C}(W_\Theta))^*$ is the set of vectors $\boldsymbol{u}$ in $\mathbb{R}^p$ such that $\sum_{i=1}^p u_i v_i \geq 0$ for all $\boldsymbol{v} \in \mathbf{C}(W_\Theta)$. We use the approach from [8] for this task.

**Lemma 1.** *Matrix $\mathbf{A}$ represents $\succeq_\Theta$ if and only if the dual cone $(\mathbf{C}(W_\Theta))^*$ is equal to the cone generated by the rows of $A$, i.e., every row of $A$ is in $(\mathbf{C}(W_\Theta))^*$ and every element of $(\mathbf{C}(W_\Theta))^*$ is a positive convex combination of rows of $A$.*

*Proof.* Let $R$ be the set of rows of matrix $\mathbf{A}$. We have $\boldsymbol{u} \in R^*$ if and only if $\mathbf{A}\boldsymbol{u} \geq \mathbf{0}$. Abbreviate $\mathbf{C}(W_\Theta)$ to $\mathbf{C}$. It easily follows from Proposition 1, that matrix $\mathbf{A}$ represents $\succeq_\Theta$ if and only if the following equivalence holds for all vectors $\boldsymbol{u} \in \mathbb{R}^p$: $\boldsymbol{u} \in \mathbf{C} \iff \mathbf{A}\boldsymbol{u} \geq \mathbf{0}$. Thus $\mathbf{A}$ represents $\succeq_\Theta$ if and only $\mathbf{C} = R^*$. Now, $\mathbf{C} = R^*$ implies that $\mathbf{C}^* = R^{**} = \mathbf{C}(R)$. Conversely, if $\mathbf{C}^* = \mathbf{C}(R)$ then $\mathbf{C} = \mathbf{C}^{**} = (\mathbf{C}(R))^* = R^*$. Therefore, $\mathbf{A}$ represents $\succeq_\Theta$ if and only if $\mathbf{C}^* = \mathbf{C}(R)$.

Figure 3 compares the proposed matrix based dominance checks against the LP based ones on random multi-objective influence diagrams from [1]. These problems have 5 decisions, an increasing number of chance variables, and involve 3 and 5 objectives, respectively. Each data point represents an average over the number of solved instances by both methods out of 100 instances generated for the respective problem size. We can see that the former method clearly dominates the latter across all reported problem sizes, and in some cases it can achieve almost one order of magnitude speedup.

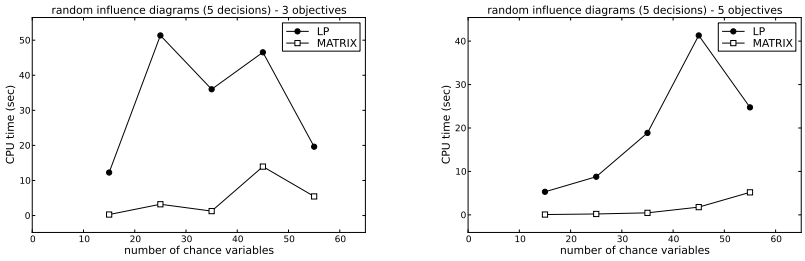



**Fig. 3.** Comparing matrix based versus LP based dominance checks on problems with 3 and 5 objectives. CPU time in seconds as a function of problem size. Time limit 20 minutes.

## 4 Reducing the Upper Bound Sets

Branch-and-bound algorithms such as MOAOBB involve use of an upper bound set during search, i.e., a set of utility vectors in $\mathbb{R}^p$ such that the utility vector for every assignment below the current node is weakly dominated by some element in the set. The upper bound sets can grow quite large and thus their manipulation during search may become computationally very expensive. In this section, we will require the upper bound set to have restricted cardinality, at most $B$ ($\geq 1$). We thus need a method for taking a larger upper bound set $\mathcal{U}$, with $|\mathcal{U}| > B$, and reducing it to have cardinality at most $B$, whilst maintaining its property of being an upper bound set at the node.

We do this by iteratively choosing a selection $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ of elements and producing an element $\boldsymbol{u}$ that is an upper bound of all of them. Elements $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ are then removed from the upper bound set, along with the elements that $\boldsymbol{u}$ dominates, and $\boldsymbol{u}$ is added. The new set is still an upper bound set. This gets repeated until we have $|\mathcal{U}| \leq B$. (For the last iteration we reduce the cluster size $k$ to being $|\mathcal{U}| - B + 1$ to avoid excessive "overshooting", so as to achieve a final upper bound set with cardinality closer to $B$.) For the case when $B = 1$, we use a single iteration with $k = |\mathcal{U}|$.

We go into more detail for the Pareto and tradeoffs cases below. Both make use of the Pareto least upper bound $\boldsymbol{v}$ of vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$, given by $\boldsymbol{v} = \max_{j=1}^{k} \boldsymbol{v}_j$, where the $\max$ is applied point-wise.

### 4.1 Pareto (no tradeoffs) Case

We remove only two elements from $\mathcal{U}$ in each iteration. We choose randomly $\boldsymbol{v} \in \mathcal{U}$, and find $\boldsymbol{w} \in \mathcal{U}$ that minimizes the Manhattan distance from $\boldsymbol{v}$, i.e., that minimizes $\sum_{i=1}^{p} |v_i - w_i|$, where $v_i$ and $w_i$ are the $i^{th}$ components of $\boldsymbol{v}$ and $\boldsymbol{w}$, respectively. We add $\boldsymbol{u}$, the Pareto least upper bound of $\boldsymbol{v}$ and $\boldsymbol{w}$, to $\mathcal{U}$, and remove $\boldsymbol{v}$ and $\boldsymbol{w}$ from $\mathcal{U}$ and the elements that are dominated by $\boldsymbol{u}$; this procedure is iterated until $|\mathcal{U}| \leq B$.

We also implemented a number of variations of this method, which seemed to perform slightly less well, including replacing every two elements of $\mathcal{U}$ with their Pareto least upper bound that are randomly selected, minimizing the Manhattan distance, maximizing the dot product value and maximizing the dot product value of the normalized vectors.

### 4.2 Tradeoffs Case

We assume a consistent set of inputs $\Theta$, leading to a preference relation $\succeq_\Theta$ which we abbreviate to $\succcurlyeq$. As described above, we make use of matrix $\mathbf{A}$ to represent $\succcurlyeq$. The key step is to choose a selection of $k$ elements in $\mathcal{U}$ and to generate an upper bound of them. We make use of the following result:

**Proposition 3.** *Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ be vectors in $\mathbb{R}^p$, and let $\boldsymbol{v} = \max_{j=1}^k \boldsymbol{v}_j$ be their Pareto least upper bound, and let $\boldsymbol{w} = \max_{j=1}^k \boldsymbol{A}\boldsymbol{v}_j$, (with $\max$ being applied pointwise in both cases). Then,*

*(i) $\boldsymbol{v} \succcurlyeq \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$, i.e., the Pareto least upper bound is an upper bound with respect to $\succcurlyeq$;*
*(ii) for $\boldsymbol{u} \in \mathbb{R}^p$, $\boldsymbol{u} \succcurlyeq \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \iff \boldsymbol{A}\boldsymbol{u} \geq \boldsymbol{w}$; and*
*(iii) $\boldsymbol{A}\boldsymbol{v} \geq \boldsymbol{w}$.*

*Proof.* (i): Let $j$ be an arbitrary element of $\{1, \ldots, k\}$. By definition, $\boldsymbol{v} \geq \boldsymbol{v}_j$. Since $\succcurlyeq$ extends Pareto, we have $\boldsymbol{v} \succcurlyeq \boldsymbol{v}_j$.
(ii): $\boldsymbol{u} \succcurlyeq \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \iff$ for all $j = 1, \ldots, k$, $\boldsymbol{u} \succcurlyeq \boldsymbol{v}_j \iff$ for all $j = 1, \ldots, k$, $\mathbf{A}\boldsymbol{u} \geq \mathbf{A}\boldsymbol{v}_j \iff \mathbf{A}\boldsymbol{u} \geq \max_{j=1}^k \mathbf{A}\boldsymbol{v}_j$.
(iii) follows immediately from (i) and (ii).

Our first approach for generating an upper bound $\boldsymbol{u}$ of vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ is to just use $\boldsymbol{u} = \boldsymbol{v}$, the Pareto least upper bound, which is an upper bound (w.r.t. $\succcurlyeq$) by Proposition 3(i). However, especially if $\succcurlyeq$ is much stronger than the Pareto ordering, we may be able to obtain a much tighter upper bound, leading to potentially much stronger pruning. To obtain an upper bound, we minimize objective function $\min \sum_i u_i$ (where $u_i$ is the $i^{th}$ value of vector $\boldsymbol{u}$), subject to $\mathbf{A}\boldsymbol{u} \geq \boldsymbol{w}$, plus the extra constraints that $\boldsymbol{u} \leq \max_{j=1}^k \boldsymbol{v}_j$, i.e., that $\boldsymbol{u}$ is weakly Pareto dominated by the Pareto least upper bound $\boldsymbol{v}$. Proposition 3(iii) shows that the constraints are satisfiable, since $\boldsymbol{v}$ is a solution, and Proposition 3(ii) shows that the solution will be an upper bound.

*Example 3.* Continuing Example 2, suppose that we are wanting to replace the pair of utility vectors $\{(21, 3), (3, 15)\}$ by an upper bound. We have $(9, 15) \succcurlyeq_\Theta (21, 3), (3, 15)$ so that $(9, 15)$ is a much tighter upper bound than the Pareto least upper bound $(21, 15)$.

The cluster $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ is chosen randomly. We again tried a number of variations of this approach. This included (1) replacing an element and its $k-1$ nearest neighbors (with respect to Manhattan distance) minimizing the sum of Manhattan distances (between the element and its neighbors) with their upper bound generated using the linear programming approach; (2) we set $k$ to 2 and iteratively replace pairs of elements with their upper bound generated using the linear programming approach.

## 5 Experiments

In this section, we evaluate empirically the performance of the proposed improvements to branch-and-bound algorithms on problem instances derived from three classes of

MOCOP benchmarks: random networks, combinatorial auctions and vertex covering problems. All algorithms were implemented in C++ (32 bit) and the experiments were run on a 2.6GHz quad-core processor with 4 GB of RAM.

For our purpose, we consider the following random problem generators:

- **Random Networks:** Our random networks are characterized by parameters $\langle n, c \rangle$, where $n$ is the number of variables and $c$ is the number of binary utility functions. For consistency, we used similar parameters to [4] and generated random instances with $n \in [10, 160]$ and $c = 1.6n$ having 2, 3, 4 and 5 objectives. The components of the utility vectors were uniformly distributed between 0 and 10. The induced width of these problems ranged between 5 and 14, respectively.
- **Vertex Coverings:** Given a graph $G = (V, E)$, the task is to find a *vertex covering* $S \subseteq V$ such that $\forall (u, v) \in E$, either $u \in S$ or $v \in S$, and $F(S) = \sum_{v \in S} \boldsymbol{w}(v)$ is maximized, where $\boldsymbol{w}(v) = (w_1, \ldots, w_p)$ is a $p$-dimensional utility vector corresponding to vertex $v \in V$. Following [4], we generated random graphs with $|V| \in [10, 180]$ vertices, $|E| = 1.6|V|$ edges and having 2, 3, 4 and 5 objectives. The components of the utility vectors were generated randomly between -10 and 0. The induced width of these problems ranged between 9 and 25, respectively.
- **Combinatorial Auctions:** In our multi-objective combinatorial auctions each bid is associated with the price, the probability of failing the payment upon acceptance, and the quality of service measure. The task is to determine the subset of winning bids that simultaneously maximize the profit, minimize the risk of not getting the full revenue and maximize the overall quality of the services represented by the selected bids. We generated auctions with 30 goods and increasing number of bids from the *paths* distribution of the CATS suite [9] and randomly added failure probabilities to the bids in the range (0,0.3) while the quality of service associated with each bid was set uniformly at random between 1 and 10. The induced width of these problems ranged between 6 and 61, respectively.
- **Tradeoffs:** Given a problem instance we generated consistent random tradeoffs between its objectives, using the generator from [1] with parameters $(K, T)$ where $K$ and $T$ are the number of pairwise and 3-way tradeoffs, respectively. For a pair $(i, j)$ of objectives picked randomly out of $p$ objectives we generate two tradeoffs $a\boldsymbol{e}_i - b\boldsymbol{e}_j$ and $b\boldsymbol{e}_j - ac\boldsymbol{e}_i$, where $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ are the $i$-th and $j$-th unit vectors. Intuitively, one of the tradeoffs indicates how much of objective $i$ one sacrifices to gain a unit of objective $j$, and the other is vice versa. We generate a 3-way tradeoff between three objectives $(i, j, k)$ picked randomly as well in the form of the tradeoff vector $a\boldsymbol{e}_i + b\boldsymbol{e}_j - c\boldsymbol{e}_k$, where $a, b, c \in [0.1, 1)$.

We consider the following solving alternatives:

- **MOAOBB($i$)** – the multi-objective AND/OR Branch-and-Bound from Section 2, where parameter $i$ is the mini-bucket $i$-bound and controls the accuracy of the heuristic. Larger values of $i$ typically yield more accurate estimates but they are more expensive to compute.
- **B=$b$ (PLUB)** – the extension of MOAOBB($i$) that uses the Pareto least upper bound based method described in Section 4.1 to reduce the upper bound set to at most $b$ ($\geq 1$) utility vectors, for both the Pareto and tradeoffs cases.

| | B=1 | B=2 | B=4 | B=10 | B=50 | B=100 |
|---|---|---|---|---|---|---|
| | vertex coverings (110 vars) - 3 obj - pareto | | | | | |
| PLUB | **59** | 67 | 70 | 95 | 188 | 238 |
| | vertex coverings (110 vars) - 5 obj - pareto | | | | | |
| PLUB | **270** | 581 | 598 | 646 | 894 | 983 |
| vertex coverings (160 vars) - 3 obj - $(K = 2, T = 1)$ tradeoffs | | | | | | |
| LP | 496 | 243 | **227** | 269 | 340 | 341 |
| PLUB | 552 | **94** | 150 | 268 | 340 | 340 |
| vertex coverings (160 vars) - 5 obj - $(K = 5, T = 2)$ tradeoffs | | | | | | |
| LP | 629 | 370 | **346** | 376 | 485 | 493 |
| PLUB | 972 | **166** | 211 | 345 | 487 | 495 |

**Table 1.** CPU time in seconds as a function of the upper bound set cardinality (B) for vertex covering problems with 3 and 5 objectives. Mini-bucket $i$-bound is 10. Time limit 20 minutes.

- **B=$b$ (LP)** – the extension of MOAOBB($i$) that uses the LP based method from Section 4.2 to compute the upper bound sets, for the tradeoffs case only. The cluster size was set to 30.
- **VE** – the variable elimination algorithm introduced recently by [1] for evaluating multi-objective influence diagrams, which we adapted here to solve MOCOPs. Unlike the branch-and-bound algorithms which can operate in linear space (ignoring the optimal solution sets), VE is time and space exponential in the induced width of the problem instance.

All competing algorithms were restricted to a static variable ordering obtained as a depth-first traversal of the guiding pseudo tree which was computed using a min-fill heuristic [3, 4]. The AND/OR search algorithms order the subproblems rooted at each node in the search tree in lexicographic order. In all our experiments we report the average CPU time in seconds and the number of problem instances solved (we omit the nodes expanded for space reasons). The best performance points are highlighted.

*Impact of the Upper Bound Set Size* Table 1 displays the average CPU time as a function of the upper bound set size (B) for vertex covering problems with 3 and 5 objectives, respectively. We consider both the Pareto (top two rows) and tradeoffs (bottom two rows) cases. For each problem size we generated 10 random instances and for each instance we generated 10 random sets of tradeoffs using the $(K, T)$ parameters shown in the table. The mini-bucket $i$-bound was set to 10. We can see clearly that using a singleton upper bound set which has a reduced computational overhead is best for the Pareto case. For example, on problems with 110 variables and 5 objectives, MOAOBB(10) manipulates very large upper bound sets with more than 5000 elements and consequently is able to solve only one instance within the 20 minute time limit. In contrast, algorithm B=1 (PLUB) solves all problem instances in less than 5 minutes on average. When looking at the tradeoffs case, we can see a different picture. Namely, the best option is to use an upper bound set with small cardinality (up to 5 elements) for both the Pareto and the LP based bounds. The singleton upper bound set, although less expensive to compute, is

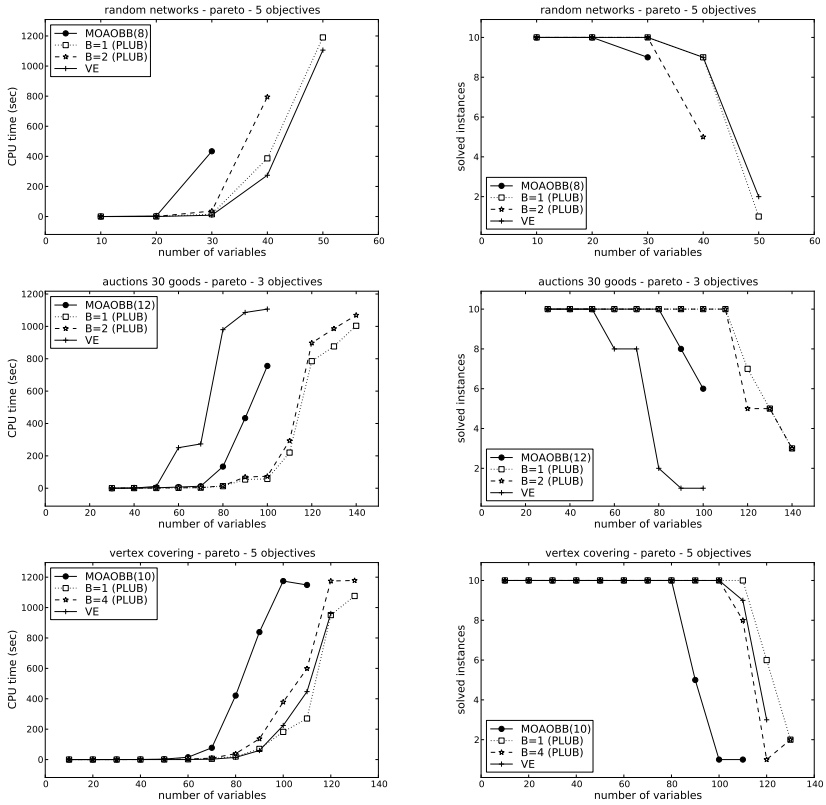



**Fig. 4.** CPU time in seconds (left) and number of problem instances solved (right) for random networks with 5 objectives, combinatorial auctions with 3 objectives and vertex covering problems with 5 objectives, respectively. Using the Pareto ordering. Time limit 20 minutes.

highly inaccurate and causes the algorithms to explore a much larger search space thus deteriorating their performance considerably. The results obtained on the other problem classes displayed a similar pattern and therefore were omitted for space reasons.

*Comparison with State-of-the-Art Approaches* Figure 4 shows the results obtained for random networks with 5 objectives, combinatorial auctions with 3 objectives and vertex covering problems with 5 objectives, respectively, using the Pareto ordering. Each data point represents an average over 10 random instances of the corresponding problem size (number of variables). The mini-bucket $i$-bounds were chosen as follows: 8 for random networks, 10 for vertex covering and 12 for combinatorial auctions, respectively. We can see that algorithm B=1 (PLUB) using a singleton upper bound set outperforms the state-of-the-art MOAOBB by a significant margin and thus offers the overall best performance. The second best algorithm across all reported problem sizes is B=2 (PLUB) which is slightly slower than B=1 (PLUB) due to computational overhead issues. For example, both B=1 (PLUB) and B=2 (PLUB) scale up to auctions with

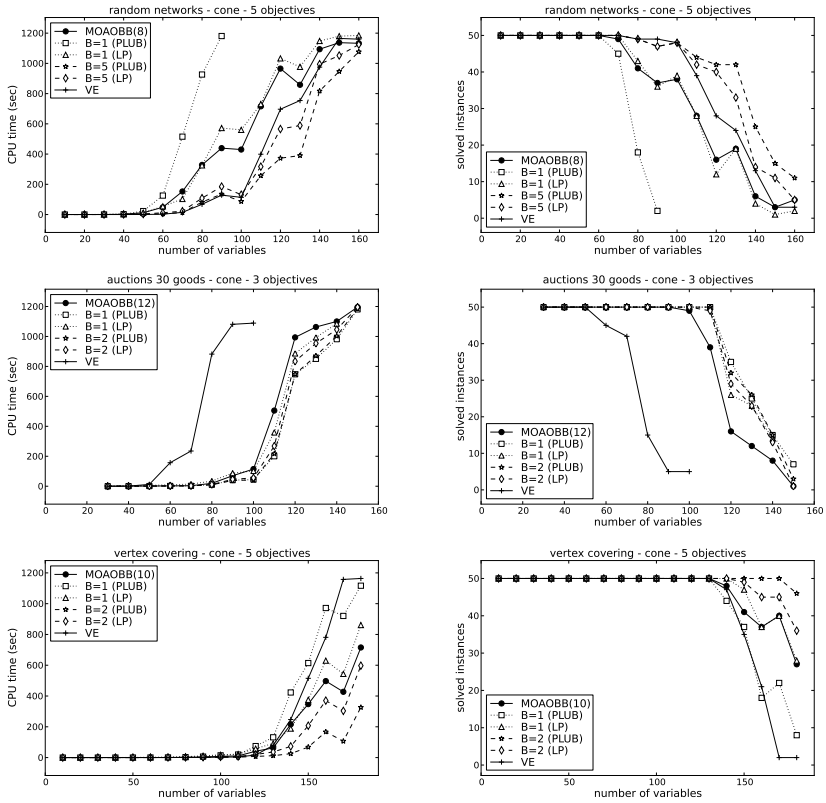



**Fig. 5.** CPU time in seconds (left) and number of problem instances solved (right) for random networks with 5 objectives, combinatorial auctions with 3 objectives and vertex covering problems with 5 objectives, respectively. Tradeoffs generated with parameters $(K = 6, T = 3)$ for random networks, $(K = 2, T = 1)$ for combinatorial auctions and $(K = 5, T = 2)$ for vertex covering. Time limit 20 minutes.

140 bids, whereas MOAOBB runs out of time beyond problems with 100 bids. VE is competitive only for medium size problems and quickly runs out of memory on larger problems (e.g., combinatorial auctions) because of larger induced widths.

In Figure 5, we summarize the results for the same problem classes using tradeoffs generated with parameters $(K, T)$ shown in the caption. The singleton upper bounds are very weak in this case and consequently algorithms B=1 (PLUB) and B=1 (LP) perform very poorly. The best performance is offered by the algorithms using upper bound sets with 5 (random networks) and 2 (auctions and vertex covering) elements, respectively. In summary, the algorithms using upper bound sets of relatively small cardinality, which are much less expensive to compute and manipulate during search, are superior to the current state-of-the-art solvers over a wide range of problem instances, and in many cases they scale up to much larger problems.

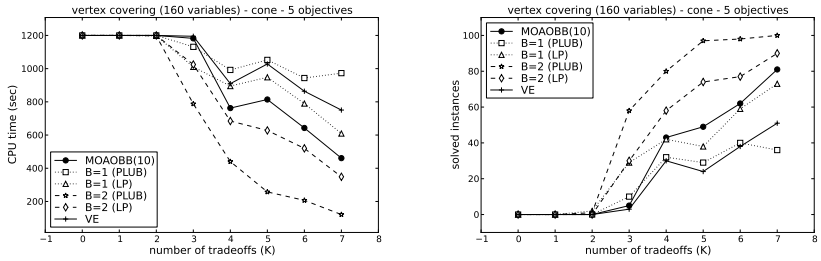



**Fig. 6.** CPU time in seconds (left) and number of problem instances solved (right) as a function of the number of pairwise tradeoffs ($K$) for vertex covering problems with $n = 160$ and 5 objectives ($T = 1$). The mini-bucket $i$-bound is 10. Time limit 20 minutes.

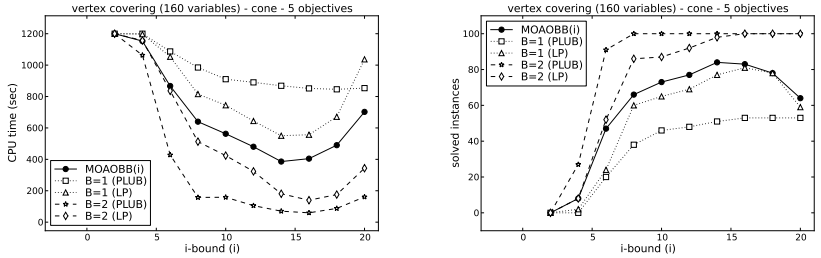



**Fig. 7.** CPU time in seconds (left) and number of problem instances solved (right) as function of the mini-bucket $i$-bound for vertex covering problems with $n = 160$, 5 objectives and $(K = 5, T = 2)$ tradeoffs. Time limit 20 minutes.

*Impact of the Number of Tradeoffs* Figure 6 plots the CPU time as a function of the number of pairwise tradeoffs $K$ for vertex covering problems with 160 variables and 5 objectives for fixed number of 3-way tradeoffs ($T = 1$). As more tradeoffs become available, the running time of the algorithms decreases substantially because the $\succcurlyeq_\Theta$-dominance gets stronger and therefore it can prune the search space more effectively. The singleton upper bounds are quite loose and therefore algorithms B=1 (PLUB) and B=1 (LP) have a relatively flat performance across the different values of $K$. We observed a similar behavior for fixed $K = 1$ and increasing number of 3-way tradeoffs $T$ (results omitted for lack of space).

*Impact of the Heuristic Information* Figure 7 plots the CPU time (left) and number of problem instances solved (right) as a function of the mini-bucket $i$-bound for vertex covering problems with 160 variables and 5 objectives. We notice the U-shaped curve characteristic of search algorithms using mini-bucket heuristics. As the $i$-bound increases, the total time decreases because the heuristics get stronger and prune the search space more effectively. But then as $i$ increases further, the heuristic strength does not outweigh its computational overhead and the time starts to increase again.

For the tradeoffs case, we observed that the LP-based upper bounds were typically tighter than the corresponding Pareto-based ones across all benchmark problems, but

they incurred a much higher computational overhead. Therefore, the pruning power of the former did not outweigh their overhead, except for the case when the upper bound set was restricted to a single element ($B = 1$).

## 6 Related Work

The optimization algorithms we use are built on the approach of [4]. The use of upper bound sets in the context of mini-buckets for branch-and-bound was also developed in [10, 11, 5, 12], and bound sets have been used in the approaches described in [13–16]. Constraint programming approaches for multi-criteria optimization include [17–19].

As mentioned in Section 3, the formalism for tradeoffs derives from that described in [1], and relates to convex cone-based approaches for multi-objective preferences such as [20, 8, 21, 22]. The variable elimination technique for the tradeoffs case derives from that in [1] and the correctness follows from the results in [23]; it can also be related with the general algorithmic approach described in [24].

## 7 Summary and Conclusion

We extended multi-objective constraint optimization algorithms – including a variable elimination algorithm and variants of a branch-and-bound algorithm – to the case where there are additional tradeoffs. The tradeoffs approach is based on a preference inference technique. We show that the inference technique from [1] can be given an alternative semantics based on Multi-Attribute Utility Theory, where it is assumed that the decision maker compares utility vectors by a weighted sum of the individual values.

The branch-and-bound algorithms use a mini-buckets procedure for generating the upper bound set at each node. Because the upper bound set can get large we consider different methods for reducing its size. This is achieved by incrementally replacing a selection of the elements by an upper bound of them. In almost all our experimental results for the Pareto (no tradeoffs) case, we found that using a singleton upper bound set is best, and this can considerably improve the current state-of-the-art. Although using a larger upper bound set pruned slightly more, it was not sufficient to make up for the additional overhead. For the tradeoffs case, our results suggest that it is usually best to use a non-singleton upper bound set, but which has quite small cardinality; even allowing a 2-element upper bound set can improve dramatically the efficiency of the algorithm because of the extra pruning power.

#### Acknowledgments

Abdul Razak is funded by IRCSET and IBM through the IRCSET Enterprise Partnership Scheme. This work was also supported in part by the Science Foundation Ireland under grant no. 08/PI/I1912.

## References


1. R. Marinescu, A. Razak, and N. Wilson. Multi-objective influence diagrams. In *Uncertainty in Artificial Intelligence (UAI)*, pages 574–583, 2012.
2. R. Dechter and I. Rish. Mini-buckets: A general scheme of approximating inference. *Journal of ACM*, 50(2):107–153, 2003.
3. R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.
4. 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.
5. E. Rollon. *Multi-Objective Optimization for Graphical Models*. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 2008.
6. D. E. Nering. *Linear Algebra and Matrix Theory (second edition)*. John Wiley and Sons, 1970.
7. 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.
8. K. Tamura. A method for constructing the polar cone of a polyhedral cone, with applications to linear multicriteria decision problems. *Journal of Optimization Theory and Applications*, 19(4):547–564, 1976.
9. K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Electronic Commerce*, pages 66–76, 2000.
10. E. Rollon and J. Larrosa. Bucket elimination for multi-objective optimization problems. *Journal of Heuristics*, 12:307–328, 2006.
11. E. Rollon and J. Larrosa. Constraint optimization techniques for exact multiobjective optimization. In *Proceedings of the Seventh International Conference on Multi-Objective Programming and Goal Programming*, 2006.
12. N. Wilson and H. Fargier. Branch-and-bound for soft constraints based on partially ordered degrees of preference. In *Proc. ECAI-08 Workshop on Inference methods based on graphical structures of knowledge (WIGSK08)*, 2008.
13. B. Villarreal and M.H. Karwan. Multicriteria integer programming: A (hybrid) dynamic programming recursive approach. *Mathematical Programming*, 21:204–223, 1981.
14. M. Ehrgott and X. Gandibleux. Bounds and bound sets for biobjective combinatorial optimization problems. *Notes in Economics and Mathematical Systems*, 507:241–253, 2001.
15. Francis Sourd and Olivier Spanjaard. A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing*, 20(3):472–484, 2008.
16. C. Delort and O. Spanjaard. Using bound sets in multiobjective optimization: application to the biobjective binary knapsack problem. In *Proceedings of the 9th international conference on Experimental Algorithms*, SEA-10, pages 253–265, Berlin, Heidelberg, 2010. Springer-Verlag.
17. U. Junker. Preference-based search and multi-criteria optimization. *Annals of Operations Research*, 130:75–115, 2004.
18. M. Gavanelli. Partially ordered constraint optimization problems. In *CP 2001*, 2001.
19. M. Gavanelli. An implementation of Pareto optimality in CLP(FD). In *Proc. CP-AI-OR'02*, pages 49–63, 2002.
20. P. Yu. Cone convexity, cone extreme points, and nondominated solutions in decision problems with multiobjectives. *Journal of Optimization Theory and Applications*, 14(3):319–377, 1974.

21. M. Wiecek. Advances in cone-based preference modeling for decision making with multiple criteria. *Decision Making in Manufacturing and Services*, 1(1-2):153–173, 2007.
22. Brian J. Hunt, Margaret M. Wiecek, and Colleen S. Hughes. Relative importance of criteria in multiobjective programming: A cone-based approach. *European Journal of Operational Research*, 207(2):936–945, 2010.
23. N. Wilson and R. Marinescu. An axiomatic framework for influence diagram computation with partially ordered utilities. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 210–220, 2012.
24. H. Fargier, E. Rollon, and N. Wilson. Enabling local computation for partially ordered preferences. *Constraints*, 15(4):516–539, 2010.