AND/OR Multi-Valued Decision Diagrams for Constraint Optimization

Robert Mateescu, Radu Marinescu and Rina Dechter

Donald Bren School of Information and Computer Science
University of California, Irvine, CA 92697-3425
{mateescu,radum,dechter}@ics.uci.edu

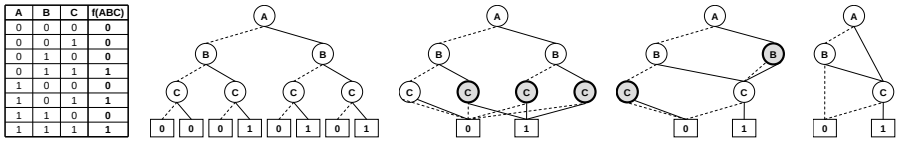
Abstract. We propose a new top down search-based algorithm for compiling AND/OR Multi-Valued Decision Diagrams (AOMDDs), as representations of the optimal set of solutions for constraint optimization problems. The approach is based on AND/OR search spaces for graphical models, state-of-the-art AND/OR Branch-and-Bound search, and on decision diagrams reduction techniques. We extend earlier work on AOMDDs by considering general weighted graphs based on cost functions rather than constraints. An extensive experimental evaluation proves the efficiency of the weighted AOMDD data structure.

1 Introduction

The compilation of graphical models, including constraint and probabilistic networks, has recently been under intense investigation. Compilation techniques are useful when an extended off-line computation can be traded for fast real-time answers. Typically, a tractable compiled representation of the problem is desired. Since the tasks of interest are in general NP-hard, this is not always possible in the worst case. In practice, however, it is often the case that the compiled representation is much smaller than the worst case bound, as was observed for Ordered Binary Decision Diagrams (BDDs) [1] which are extensively used in hardware and software verification.

In the context of constraint networks, compilation schemes are very useful for interactive solving or product configuration type problems [2, 3]. These are combinatorial problems where a compact representation of the feasible set of solutions is necessary. The system has to be *complete* (to represent all solutions), *backtrack-free* (to never encounter dead-ends) and *real-time* (to provide fast answers).

In this paper we present a compilation scheme for constraint optimization, which has been of interest recently in the context of post-optimality analysis [4]. Our goal is to obtain a compact representation of the set of optimal solutions, by employing techniques from search, optimization and decision diagrams. Our approach is based on three main ideas: (1) AND/OR search spaces for graphical models [5]. Their key feature is the exploitation of problem structure during search, sometimes yielding exponential improvement over structure-blind search methods. (2) Branch-and-Bound search for optimization, applied to AND/OR search spaces [6]. (3) Reduction rules similar to OBDDs, that lead to the compilation of the search algorithm trace into an AND/OR Multi-Valued Decision Diagram (AOMDD) [7]. The novelty over previous results consists in: (1) the



(a) Table (b) Ordered tree (c) Isomorphism (d) Redundancy (e) OBDD

Fig. 1. Boolean function representations

treatment of general weighted graphs based on cost functions, rather than constraints. (2) a top down search based approach for generating the AOMDD, rather than Variable Elimination based as in [7]. (3) Extensive experimental evaluation that proves the efficiency of the weighted AOMDD. We show that the compilation scheme can often be accomplished relatively efficiently and that we sometimes get a substantial reduction beyond the initial trace of state-of-the-art search algorithms.

2 Background

2.1 Constraint Optimization Problems

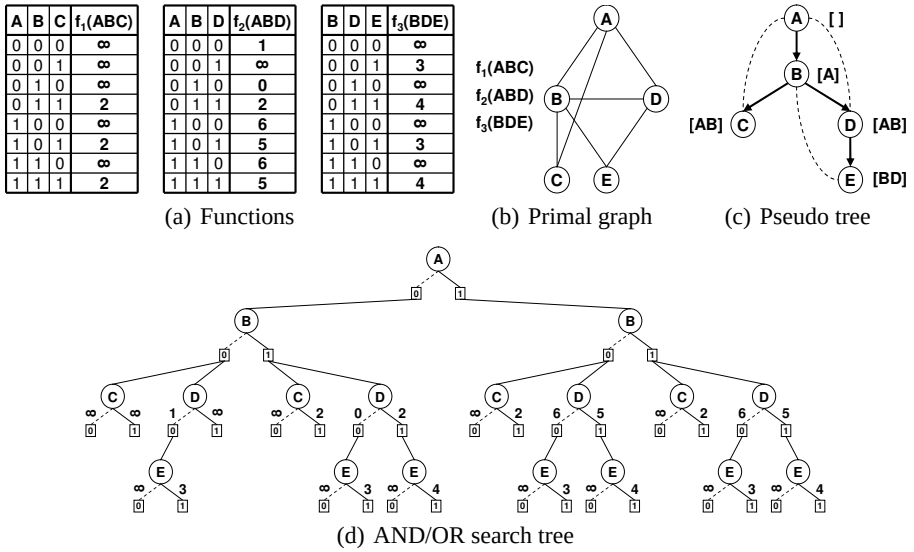
A finite *Constraint Optimization Problem* (COP) is a triple $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, where $\mathbf{X} = \{X_1, ..., X_n\}$ is a set of variables, $\mathbf{D} = \{D_1, ..., D_n\}$ is a set of finite domains and $\mathbf{F} = \{f_1, ..., f_r\}$ is a set of cost functions. Cost functions can be either *soft* or *hard* (constraints). Without loss of generality we assume that hard constraints are represented as (bi-valued) cost functions. Allowed and forbidden tuples have cost 0 and ∞, respectively. The scope of function f_i, denoted $scope(f_i) \subseteq \mathbf{X}$, is the set of arguments of f_i. The goal is to find a complete value assignment to the variables that minimizes the global cost function, namely to find $x = \arg\min_{\mathbf{X}} \sum_{i=1}^{r} f_i$.

Given a COP instance, its *primal graph* G has a node for each variable and connects any two nodes whose variables appear in the scope of the same function. The *induced graph* of G relative to an ordering d of its variables is obtained by processing the nodes in reverse order of d. For each node all its earlier neighbors are connected, including neighbors connected by previously added edges. The *width* of a node is the number of edges connecting it to nodes lower in the ordering. The *induced width* (also equal to the *treewidth*) of a graph along d, denoted $w^*(d)$, is the maximum width of nodes in the induced graph. The *pathwidth* of a graph along d, $pw^*(d)$, is equal to the induced width of the graph with extra edges added between any node and its successor in d.

2.2 Binary Decision Diagrams

Decision diagrams are widely used in many areas of research to represent decision processes. In particular, they can be used to represent functions. Due to the fundamental importance of Boolean functions, a lot of effort has been dedicated to the study of *Binary Decision Diagrams* (BDDs), which are extensively used in formal verification.

A BDD is a representation of a Boolean function. Given $\mathbf{B} = \{0, 1\}$, a Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}$, has n arguments, $X_1, \cdots, X_n$, which are Boolean variables, and takes Boolean values. A Boolean function can be represented by a table (see Figure 1(a)), but this is exponential in n, and so is the binary tree representation in Figure



(a) Functions (b) Primal graph (c) Pseudo tree

(d) AND/OR search tree

Fig. 2. AND/OR search tree for COP

1(b). OBDDs [1] provide a more compact representation, that also supports efficient operations, by imposing the same order to the variables along each path in the binary tree, and then applying the following two reduction rules exhaustively: (1) *isomorphism*: merge nodes that have the same label and the same respective children (Figure 1(c)); (2) *redundancy*: eliminate nodes whose low (zero) and high (one) edges point to the same node (see Figure 1(d)). The resulting OBDD is shown in Figure 1(e).

2.3 AND/OR Search Spaces for COP

The AND/OR search space [5] is a unifying framework for advanced algorithmic schemes for graphical models, including constraint networks and cost networks. Its main virtue consists in exploiting independencies between variables during search, which can provide exponential speedups over traditional structure-blind search methods. The search space is defined using a backbone *pseudo-tree* [8].

Definition 1 (pseudo-tree). *Given an undirected graph $G = (\mathbf{X}, E)$, a directed rooted tree $\mathcal{T} = (\mathbf{X}, E')$ defined on all its nodes is called* pseudo-tree *if any edge of G that is not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node to an ancestor in $\mathcal{T}$.*

AND/OR Search Trees Given a COP instance $\mathcal{P}$, 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. The OR nodes are labeled X_i and correspond to the variables. The AND nodes are labeled $\langle X_i, x_i \rangle$ (or just x_i) and correspond to value assignments of the variables. The structure of the AND/OR search tree is based on the underlying pseudo tree $\mathcal{T}$. The root of the AND/OR search tree is an OR node labeled with the root of $\mathcal{T}$. The children of an OR node X_i are AND nodes labeled with assignments $\langle X_i, x_i \rangle$ that are consistent with the assignments along the path from the root. The children of an AND

node $\langle X_i, x_i \rangle$ are OR nodes labeled with the children of variable X_i in $\mathcal{T}$. The AND/OR search tree can be traversed by a depth first search (DFS) algorithm, thus using linear space to compute the value of the root node. It was shown [8, 9]:

Theorem 1. *Given a COP instance $\mathcal{P}$ and a pseudo tree $\mathcal{T}$ of depth m, the size of the AND/OR search tree based on $\mathcal{T}$ is $O(n \cdot k^m)$, where k bounds the domains of variables. A COP having treewidth w^* has a pseudo tree of depth at most $w^* \log n$, therefore it has an AND/OR search tree of size $O(n \cdot k^{w^* \log n})$.*

Weighted AND/OR Search Trees The OR-to-AND arcs from nodes X_i to x_i in an AND/OR search tree are annotated by *weights* derived from the cost functions in $\mathbf{F}$.

Definition 2 (weight). *The* weight $w(X_i, x_i)$ *of the arc from the OR node X_i to the AND node x_i is the sum of all the cost functions whose scope includes X_i and is fully assigned along the path from the root to x_i, evaluated at the values along the path.*

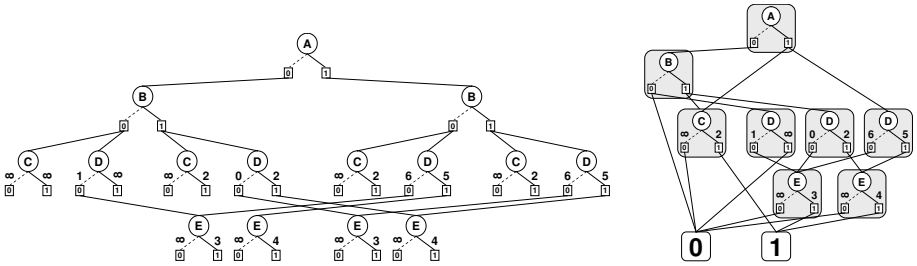
The AND/OR search tree in Figure 2(d) shows the weights on the OR-to-AND arcs. Given a weighted AND/OR search tree, each node can be associated with a *value*:

Definition 3 (value). *The* value $v(n)$ *of a node n in a weighted AND/OR tree is defined recursively as follows (where $succ(n)$ are the children of n):*
(i) $v(n) = 0$, if $n = x_i$ is a terminal AND node;
(ii) $v(n) = \sum_{n' \in succ(n)} v(n')$, if $n = x_i$ is an internal AND node;
(iii) $v(n) = min_{n' \in succ(n)}(w(n, n') + v(n'))$, if $n = X_i$ is an OR node.

It is easy to see that the value $v(n)$ of a node in the AND/OR search tree $S_{\mathcal{T}}$ is the minimal cost solution to the subproblem rooted at n, subject to the current variable instantiation along the path from the root to n. If n is the root of $S_{\mathcal{T}}$, then $v(n)$ is the minimal cost solution to the initial problem [6, 5].

Example 1. Figure 2 shows an example of AND/OR search tree for a COP with binary variables. The cost functions are given in Figure 2(a). The value ∞ indicates a hard constraint. The primal graph is given in Figure 2(b), and the pseudo tree in Figure 2(c). The square brackets indicate the context of the variables, a notion explained below. The AND/OR search tree is given in Figure 2(d). The numbers on the OR-to-AND arcs are the weights corresponding to the function values. Note that the tree is pruned whenever a weight shows inconsistency (e.g., for $A = 0, B = 0, D = 1$).

AND/OR Search Graphs The AND/OR search tree may contain nodes that root identical conditioned subproblems. These nodes are said to be *unifiable*. When unifiable nodes are merged, the search space becomes a graph. Its size becomes smaller at the expense of using additional memory by the search algorithm. The depth first search algorithm can therefore be modified to cache previously computed results, and retrieve them when the same nodes are encountered again. Some unifiable nodes can be identified based on their *contexts*. We can define graph based contexts for both OR nodes and AND nodes, just by expressing the set of ancestor variables in $\mathcal{T}$ that completely determine a conditioned subproblem. It can be shown that using caching based on OR contexts makes caching based on AND contexts redundant, so we only use *OR caching*.



(a) Context minimal AND/OR graph (b) AOMDD

Fig. 3. AND/OR graphs for COP

Given a pseudo tree $\mathcal{T}$ of an AND/OR search space, the *context* of an OR node X, denoted by $context(X) = [X_1 \dots X_p]$, is the set of ancestors of X in $\mathcal{T}$ ordered descendingly, that are connected in the primal graph to X or to descendants of X. The context of X separates the subproblem below X from the rest of the network.

Definition 4 (context minimal AND/OR graph). *Given a COP instance $\mathcal{P}$ and a pseudo-tree $\mathcal{T}$, the* context minimal *AND/OR graph, denoted by $C_{\mathcal{T}}$ is obtained from the AND/OR search tree of $\mathcal{P}$ along $\mathcal{T}$ by merging all the context unifiable OR nodes.*

Theorem 2 ([9, 5]). *Given a COP $\mathcal{P}$, its primal graph G and a pseudo tree $\mathcal{T}$, the size of the context minimal AND/OR search graph $C_{\mathcal{T}}$ is $O(n \cdot k^{w^*_{\mathcal{T}}(G)})$, where $w^*_{\mathcal{T}}(G)$ is the induced width of G over the DFS traversal of $\mathcal{T}$, and k bounds the domain size.*

Example 2. Figure 2(c) shows a pseudo tree and the context of each variable in square brackets. The context minimal AND/OR graph is given in Figure 3(a). Note that only the cache table of E will get cache hits during the depth first search traversal (E is the only level of OR nodes that has more than one incoming arc). It can be determined from the pseudo tree inspection that all variables except for E generate *dead-caches* [5, 10] (also explained in following section), and their cache tables need not be stored.

3 Weighted AND/OR Multi-valued Decision Diagrams

The context minimal AND/OR graph offers an effective way of identifying some unifiable nodes. However, merging based on context is not complete, i.e. there may still be unifiable nodes in the search graph that do not have identical contexts. The context-based merging uses only information available from the ancestors in the pseudo tree. If all the information from the descendants would also be available, it could lead to the identification of more unifiable nodes. This comes at a higher cost, however, since information from descendants in the pseudo tree means that the entire associated subproblem has to be solved. Orthogonal to the problem of unification, some of the nodes in an AND/OR search graph may be redundant, for example when the set of solutions rooted at variable X_i is not dependent on the specific value assigned to X_i.

The above criteria suggest that once an AND/OR search graph is available (e.g., after search terminates, and its trace is saved) reduction rules based on *isomorphism*

and *redundancy* (similar to OBDDs) can be applied further, reducing the size of the AND/OR search graph that was explicated by search. In order to apply the reduction rules, it is convenient to group each OR node and its children into a *meta-node*:

Definition 5 (meta-node). *A meta-node v in a weighted AND/OR search graph consists of an OR node labeled $var(v) = X_i$ and its k_i AND children labeled $x_{i_1}, ..., x_{i_{k_i}}$ that correspond to its value assignments. Each AND node labeled x_{i_j} points to a list of child meta-nodes, $u.children_j$, and also stores the weight $w(X_i, x_{i_j})$.*

The reduction rules are straightforward. Two meta-nodes are *isomorphic* if they have the same variable label and the same respective lists of children and weights. A meta-node is *redundant* if all its lists of children and weights are respectively identical.

When reduction rules are applied exhaustively to an AND/OR search graph, the result is an AND/OR Multi-Valued Decision Diagram (AOMDD). The AOMDD data structure for constraint networks (where weights are all 1) was introduced in [7], along with a Variable Elimination type algorithm to generate it, based on the *apply* operator, similar to the OBDD case.

An example of a AOMDD appears in Figure 3(b), representing the exhaustive reduction of the context minimal AND/OR graph in Figure 3(a). The terminal nodes labeled with **0** and **1** denote inconsistent and consistent assignments, respectively. Note that when $A = 1$, B is redundant and its common list of children becomes the list of children for $A = 1$, namely the problem already splits into two independent components after $A = 1$, even though this can not be read from the pseudo tree in Figure 2(c).

4 Compiling COPs into AOMDDs

We next define the AOMDD describing the set of optimal solutions to a COP and present a general scheme for generating these compiled data structures.

Definition 6. *Given a set of tuples S over variables $\mathbf{X}$ and a tree $\mathcal{T}$ over $\mathbf{X}$, $\mathcal{T}$ expresses S iff there exists an AND/OR tree guided by $\mathcal{T}$ that expresses all and only tuples in S.*

Proposition 1. *If $\mathcal{T}$ is a pseudo-tree of a COP $\mathcal{P}$, then $\mathcal{T}$ can be used to express S^{opt}, the set of optimal solutions of $\mathcal{P}$.*

Definition 7. *Given a COP $\mathcal{P}$, its set of optimal solutions S^{opt} and a pseudo tree $\mathcal{T}$ of $\mathcal{P}$, its AOMDD$^{opt}_{\mathcal{T}}$ is the AOMDD that expresses all and only S^{opt} relative to $\mathcal{T}$.*

The target is to generate AOMDD$^{opt}_{\mathcal{T}}$ of a COP. The idea is to use a pseudo tree $\mathcal{T}$ that can express all solutions and explore a subset of its context minimal AND/OR graph, $C_{\mathcal{T}}$ that contains all the optimal solutions and then process it so that it will represent only optimal solutions and be completely reduced relative to isomorphism and redundancy. Therefore, any search algorithm for optimal solutions that explores the context minimal graph can be used to generate the initial trace. The better the algorithm we use, the more efficient the procedure would be because the initial trace will be tight around the context minimal graph that is restricted to the optimal solutions.

In recent years several Branch-and-Bound and Best-First search algorithms were developed to search the context minimal AND/OR graph for solving COPs [6, 10, 11]. In this paper we use a depth-first AND/OR Branch-and-Bound (AOBB) algorithm.

4.1 AND/OR Branch-and-Bound Search

AOBB traverses the context minimal AND/OR search graph in a depth-first manner via full caching. It interleaves forward expansion of the current partial solution subtree with a backward cost revision step that updates node values, until search terminates. The efficiency of the algorithm also depends on the strength of its heuristic evaluation function (i.e., lower bound). Specifically, each node n along the path from the root has an associated *static* heuristic function $h(n)$ underestimating $v(n)$ that can be computed efficiently when the node n is first expanded. The algorithm then improves the heuristic function dynamically during search. The *dynamic heuristic function* $f_h(n)$ is computed based on the search space below n that has already been explored [6], and is used to prune irrelevant portions of the search space, based on the Branch-and-Bound principle.

In the forward step the algorithm expands alternating levels of OR and AND nodes. Since we are using OR caching, before expanding an OR node, its cache table is checked. If the same context was encountered before, it is retrieved from the cache, and its successors set is set to empty which will trigger the cost revision step. If an OR node is not found in the cache, it is expanded in the usual way. Before expanding an AND node n, the algorithm updates the heuristic function $f_h(a)$ for every ancestor a of n along the current path, and discontinues search below n if, for some a, $f_h(a)$ is greater or equal than the best cost solution found at a (the upper bound).

The backward cost revision step is triggered when a closed node has an empty set of successors. This means that all its children have been evaluated, and its final value can now be computed. If the current node is the root, then the search terminates with its value. OR nodes update their values by minimization, while AND nodes combine their children values by summation.

4.2 The Compilation Algorithm

The compilation algorithm, called AOBB-COMPILE, is described in Algorithm 1. It extends the AND/OR Branch-and-Bound algorithm described above by compiling the trace of the search into an AND/OR Multi-Valued Decision Diagram representing all optimal solutions to the input COP instance.

The algorithm is based on two mutually recursive steps, similar to AOBB: EXPAND and REVISE which call each other until the search terminates. The fringe of the search is maintained on a stack called OPEN. The current node is n, its parent p, and the current path π_n. The children of the current node in the AND/OR search graph are denoted by $succ(n)$. The AND/OR decision diagram being constructed is denoted by AOMDD. Each node u in the AND/OR search graph has a pointer, denoted by $u.metanode$, to the corresponding meta-node in AOMDD.

In the EXPAND step, when the current OR node n is expanded, AOBB-COMPILE creates a new meta-node corresponding to n and adds it to AOMDD. If n is already present in cache, then AOBB-COMPILE ensures that the meta-node corresponding to n's parent in the context minimal search graph points to the meta-node that was created when n was first expanded.

In the REVISE step when node values are propagated backwards, the algorithm also attempts to reduce the diagram by removing isomorphic meta-nodes. Specifically, if n is the current OR node being evaluated and if there exists a meta-node m which

Algorithm 1: AOBB-COMPILE

Data: A COP instance $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, pseudo-tree $\mathcal{T}$, root s, heuristic function f_h.
Result: AOMDD containing the optimal solutions to $\mathcal{P}$.

```
1  v(X_1) ← ∞; OPEN ← {X_1}; AOMDD ← ∅;                                   // Initialize
2  while OPEN ≠ ∅ do
3      n ← top(OPEN); remove n from OPEN
4      let π_n be the assignment along the path from the root to n
5      if n is an OR node, labeled X_i then                                // EXPAND
6          if Cache(n, context(X_i)) ≠ ∅ then
7              v(n) ← Cache(n, context(X_i))
8              succ(n) ← ∅
9              let p = ⟨X_j, x_j⟩ be the AND parent of n in the AND/OR search graph
10             p.metanode.children_{x_j} ← p.metanode.children_{x_j} ∪ {n.metanode}
11         else
12             succ(n) ← {⟨X_i, x_i⟩ | ⟨X_i, x_i⟩ is consistent with π_n}
13             for ⟨X_i, x_i⟩ ∈ succ(n) do
14                 v(⟨X_i, x_i⟩) ← 0; h(⟨X_i, x_i⟩) ← heuristic(X_i, x_i)
15                 w(X_i, x_i) ← Σ_{f∈F, X_i∈scope(f)} f(π_n)
16             create a new meta-node m for X_i and add it to AOMDD
17             let p = ⟨X_j, x_j⟩ be the AND parent of n in the AND/OR search graph
18             p.metanode.children_{x_j} ← p.metanode.children_{x_j} ∪ {m}
19         Add succ(n) on top of OPEN
20     else if n is an AND node, labeled ⟨X_i, x_i⟩ then
21         for a ∈ ancestors(X_i, x_i) do
22             if (a is OR) and (f_h(a) > v(a)) then
23                 n.deadend ← true
24                 n.metanode.children_{x_i} ← UNSOLVED
25                 break
26         if n.deadend == false then
27             succ(n) ← {X_j | X_j ∈ children_T(X_i)}
28             v(X_j) ← ∞; h(X_j) ← heuristic(X_j)
29             Add succ(n) on top of OPEN
30             if succ(n) == ∅ then
31                 n.metanode.children_{x_i} ← SOLVED
32     while succ(n) == ∅ do                                               // REVISE
33         let p be the parent of n
34         if n is an OR node, labeled X_i then
35             if X_i == X_1 then                                          // Search is complete
36                 return AOMDD
37             Cache(n, context(X_i)) ← v(n)
38             v(p) ← v(p) + v(n)
39             n.metanode.value ← v(n)
40             if findIsomorphism(n.metanode) == true then
41                 let m be the meta-node isomorphic with n.metanode
42                 redirect the links of n.metanode's parents in AOMDD to point to m
43                 AOMDD ← AOMDD − {n.metanode}
44         if n is an AND node, labeled ⟨X_i, x_i⟩ then
45             v(p) ← min(v(p), w(X_i, x_i) + v(n))
46         remove n from succ(p)
47         n ← p
```

is isomorphic with $n.metanode$, then the parents of $n.metanode$ in the AOMDD are updated to point to m instead of $n.metanode$, which is then removed from the diagram.

The compiled AOMDD may contain sub-optimal solutions that were visited during the Branch-and-Bound search but were not pruned. Therefore, a second pass over the

decision diagram is necessary to remove any path which does not appear in any optimal solution (omitted from the algorithm due to space limitations). Specifically, in the second pass AOBB-COMPILE traverses the AOMDD in a depth-first manner for every meta-node u along the current path from the root, it prunes $u.children_j$ from the diagram if $(\sum_{u' \in u.children_j} v(u') + w(X_i, x_{i_j})) > v(u)$, namely the optimal cost solution to the problem below the j child of u is not better than the optimal cost solution at u.

Theorem 3. *Given a COP instance $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ and a pseudo tree $\mathcal{T}$ of $\mathcal{P}$, the AOMDD generated by AOBB-COMPILE along $\mathcal{T}$ is AOMDD$^{opt}_{\mathcal{T}}$.*

The complexity of AOBB-COMPILE is bounded time and space by the trace generated, which is $O(n \cdot exp(w^*))$. However, the heuristic evaluation function used by AOBB typically restricts the trace far below this complexity bound.

5 Experiments

In this section we evaluate empirically the compilation scheme on two common classes of optimization problems: Weighted CSPs (WCSP) [12] and 0/1 Integer Linear Programs (0/1 ILP) [13]. In our experiments we compiled the search trace relative to isomorphic meta-nodes only, without removing redundant nodes. Also we did not perform the second top-down pass over the diagram to remove additional sub-optimal solutions.

5.1 Weighted CSPs

Weighted CSP [12] extends the classic CSP formalism with *soft constraints* which assign positive integer costs to forbidden tuples (allowed tuples have cost 0). The goal is to find a complete assignment with minimum aggregated cost. The model has numerous applications in domains such as resource allocation, scheduling or planning.

We consider the compilation algorithm based on the AND/OR Branch-and-Bound algorithm with pre-compiled mini-bucket heuristics and full caching introduced by [10] and denoted by AOBB(i). The parameter i represents the mini-bucket i-bound and controls the accuracy of the heuristic.

For each test instance we report the number of OR nodes in the context minimal AND/OR search graph (#cm) visited by AOBB(i), and the number of meta-nodes in the resulting AND/OR decision diagram (#aomdd), as well as their ratio defined as $ratio = \frac{\#cm}{\#aomdd}$. In some cases we also report the compilation time. We record the number of variables (n), the number of constraints (c), the depth of the pseudo-trees (h) and the induced width of the graphs (w^*) obtained for the test instances. The pseudo-trees were generated using the min-fill heuristic, as described in [6].

Earth Observing Satellites The problem of scheduling an Earth observing satellite is to select from a set of candidate photographs, the best subset such that a set of imperative constraints are satisfied and the total importance of the selected photographs is maximized. We experimented with problem instances from the SPOT5 benchmark [14] which can be formulated as non-binary WCSPs. For our purpose we considered a simplified MAX-CSP version of the problem where the goal is to minimize the number of imperative constraints violations (i.e., we ignored the importance of the photographs).

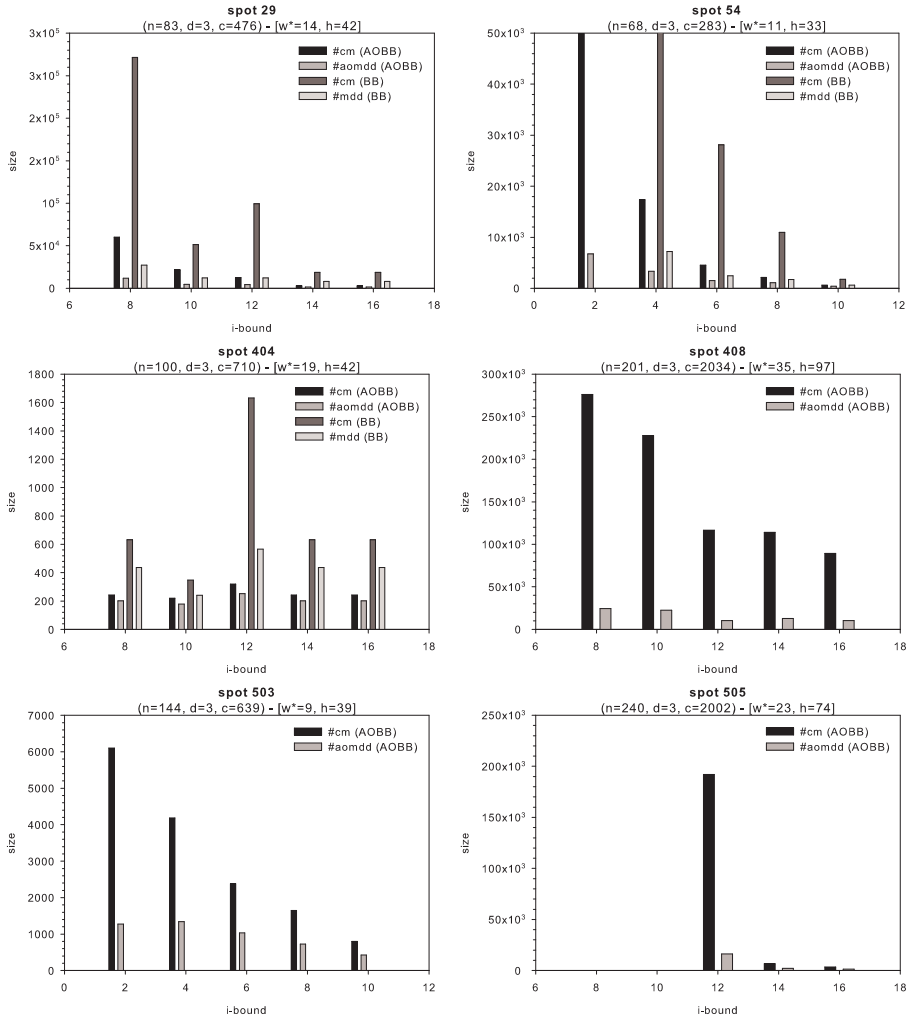


Fig. 4. The trace of AND/OR Branch-and-Bound search (nodes) versus the AOMDD size (aomdd) for the SPOT5 networks. Compilation time limit 1 hour.

Figure 4 displays the results for experiments with 6 SPOT5 networks. Each subgraph depicts the trace of `AOBB(`i`)` and the size of the resulting AND/OR decision diagram as a function of the i-bound of the mini-bucket heuristic. For comparison, we also include the results obtained with the OR version of the compilation scheme that explores the traditional OR search space.

We observe that the resulting AOMDD is substantially smaller than the context minimal AND/OR graph traversed by `AOBB(`i`)`, especially for relatively small i-bounds that generate relatively weak heuristic estimates. For instance, on the `408` network, we were able to compile an AOMDD 11 times smaller than the AND/OR search graph explored by `AOBB(8)`. As the i-bound increases, the heuristic estimates become stronger

AOBB+SMB(i)															
iscas	n	w*	i=10			i=12			i=14			i=16		
	c	h	time	#cm	#aomdd	time	#cm	#aomdd	time	#cm	#aomdd	time	#cm	#aomdd
s386	172	19	0.50	2,420	811	0.17	1,132	558	0.21	527	360	0.38	527	360
	172	44		ratio =	**2.98**		ratio =	**2.03**		ratio =	**1.46**		ratio =	**1.46**
s953	440	66		-			-		981.20	186,658	37,084	22.46	22,053	9,847
	464	101								ratio =	**5.03**		ratio =	**2.24**
s1423	748	24	21.12	21,863	9,389	7.47	13,393	6,515	5.09	10,523	6,043	2.01	5,754	4,316
	751	54		ratio =	**2.33**		ratio =	**2.06**		ratio =	**1.74**		ratio =	**1.33**
s1488	667	47	250.18	83,927	20,774	4.48	15,008	3,929	10.72	23,872	5,375	5.54	5,830	3,246
	667	67		ratio =	**4.04**		ratio =	**3.82**		ratio =	**4.44**		ratio =	**1.80**
s1494	661	48	138.61	63,856	18,501	387.73	125,030	22,393	37.78	31,355	11,546	39.75	30,610	12,467
	661	69		ratio =	**3.45**		ratio =	**5.58**		ratio =	**2.72**		ratio =	**2.46**
c432	432	27	1867.49	395,766	41,964	1.29	7,551	4,024	1.30	7,112	3,693	0.74	1,120	881
	432	45		ratio =	**9.43**		ratio =	**1.88**		ratio =	**1.93**		ratio =	**1.27**
c499	499	23	363.78	93,936	33,157	6.66	12,582	7,051	271.26	88,131	23,502	16.75	17,714	9,536
	499	74		ratio =	**2.83**		ratio =	**1.78**		ratio =	**3.75**		ratio =	**1.86**

Table 1. CPU time in seconds, the trace of AND/OR Branch-and-Bound search (#cm) and the AOMDD size (#aomdd) for the ISCAS'89 circuits. Compilation time limit 1 hour.

AOBB+SMB(i)														
planning	n	w*	i=6			i=8			i=10			i=12		
	c	h	time	#cm	#aomdd	time	#cm	#aomdd	time	#cm	#aomdd	time	#cm	#aomdd
bwt3ac	45	16	77.45	28,558	12,152	45.76	22,475	11,106	8.92	3,878	2,537	99.00	1,775	1,252
d=11	301	34		ratio =	**2.35**		ratio =	**2.02**		ratio =	**1.53**		ratio =	**1.42**
bwt3bc	45	11	54.22	23,560	10,544	29.62	18,734	9,422	8.61	3,455	2,243	85.73	1,599	1,141
d=11	301	33		ratio =	**2.23**		ratio =	**1.99**		ratio =	**1.54**		ratio =	**1.40**
bwt3cc	45	19	32.55	19,643	9,122	20.03	15,696	8,149	8.51	3,113	2,046	85.57	935	731
d=11	301	42		ratio =	**2.15**		ratio =	**1.93**		ratio =	**1.52**		ratio =	**1.28**
depot01ac	66	14	1.45	7,420	2,504	0.73	4,056	1,995	0.42	1,214	830	1.48	506	432
d=5	298	33		ratio =	**2.96**		ratio =	**2.03**		ratio =	**1.46**		ratio =	**1.17**
depot01bc	66	14	1.31	7,068	2,358	0.55	3,333	1,641	0.39	1,316	886	1.47	514	432
d=5	298	33		ratio =	**3.00**		ratio =	**2.03**		ratio =	**1.49**		ratio =	**1.19**
depot01cc	66	14	1.36	7,156	2,411	0.82	4,333	2,196	0.38	1,262	841	1.47	269	219
d=5	298	33		ratio =	**2.97**		ratio =	**1.97**		ratio =	**1.50**		ratio =	**1.23**
			i=2			i=4			i=6			i=8		
driverlog01ac	71	9	1.37	7,490	2,134	0.41	3,143	1,412	0.05	279	237	0.10	451	331
d=4	271	38		ratio =	**3.51**		ratio =	**2.23**		ratio =	**1.18**		ratio =	**1.36**
driverlog01bc	71	9	1.36	7,447	2,128	0.42	3,098	1,389	0.04	231	210	0.07	247	212
d=4	271	38		ratio =	**3.50**		ratio =	**2.23**		ratio =	**1.10**		ratio =	**1.17**
driverlog01cc	71	9	1.61	7,741	2,185	0.10	883	622	0.04	279	237	0.07	295	239
d=4	271	38		ratio =	**3.54**		ratio =	**1.42**		ratio =	**1.18**		ratio =	**1.23**
mprime03ac	49	9	2.12	7,172	1,562	0.66	3,343	863	0.11	595	386	0.16	111	94
d=10	185	23		ratio =	**4.59**		ratio =	**3.87**		ratio =	**1.54**		ratio =	**1.18**
mprime03bc	49	9	2.07	7,266	1,573	0.68	3,486	849	0.12	641	396	0.10	111	94
d=10	185	23		ratio =	**4.62**		ratio =	**4.11**		ratio =	**1.62**		ratio =	**1.18**
mprime03cc	49	9	1.47	5,469	1,391	0.45	2,336	721	0.12	534	366	0.10	111	94
d=10	185	23		ratio =	**3.93**		ratio =	**3.24**		ratio =	**1.46**		ratio =	**1.18**

Table 2. CPU time in seconds, the trace of AND/OR Branch-and-Bound search (#cm) and the AOMDD size (#aomdd) for the planning instances. Compilation time limit 1 hour.

and they are able to prune the search space significantly. In consequence, the difference in size between the AOMDD and the AND/OR graph explored decreases. When looking at the OR versus the AND/OR compilation schemes, we notice that AOMDD is smaller than the OR MDD, for all reported i-bounds. On some of the harder instances, the OR compilation scheme did not finish within the 1 hour time limit (e.g., `408`, `505`).

ISCAS'89 Benchmark Circuits ISCAS'89 circuits are a common benchmark used in formal verification and diagnosis (*http://www.fm.vslib.cz/ kes/asic/iscas/*). For our purpose, we converted each of these circuits into a non-binary WCSP instance by removing

flip-flops and buffers in a standard way, creating hard constraints for gates and uniform unary cost functions for the input signals. The penalty costs were distributed uniformly randomly between 1 and 10.

Table 1 shows the results for experiments with 7 circuits. The columns are indexed by the mini-bucket i-bound. We observe again that the difference in size between the resulting AOMDD and the AND/OR search graph explored by `AOBB(`i`)` is more prominent for relatively small i-bounds. For example, on the `c432` circuit and at $i = 10$, the AOMDD is about 9 times smaller than the corresponding AND/OR graph.

Planning We also experimented with problems from planning in temporal and metric domains (*http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/BenchmarkS*). These instances were converted into binary WCSPs as follows: each fluent of the planning graph is represented by a variable with domain values representing possible actions to produce this fluent. Hard binary constraints represent mutual exclusions between fluents and actions, and activity edges in the graph. Soft unary constraints represent action costs. The goal is to find a valid plan which minimizes the sum of the action costs.

Table 2 shows the results for experiments with 12 planning networks. On this domain we only observe minor differences between the size of the compiled AOMDD and the corresponding AND/OR search graph. This is due to very accurate mini-bucket heuristics which cause the AND/OR Branch-and-Bound to avoid expanding nodes that correspond to solutions whose cost is above the optimal one.

5.2 0/1 Integer Linear Programs

A *0/1 Integer Linear Programming* (0/1 ILP) [13] consists of a set of integer decision variables (restricted to values 0 or 1) and a set of linear constraints (equalities or inequalities). The goal is to minimize a global linear cost function subject to the constraints. 0/1 ILPs can formulate many practical problems such as capital budgeting, cargo loading, combinatorial auctions or maximum satisfiability problems.

We consider the AND/OR Branch-and-Bound algorithm developed in [15] and denoted by `AOBB`, as the basis for our AND/OR compilation scheme. The heuristic evaluation function used by `AOBB` is computed by solving the linear relaxation of the current subproblem with the SIMPLEX method [16] (our code used the implementation from the open source library `lp_solve 5.5` available at http://lpsolve.sourceforge.net/5.5/).

MIPLIB Instances MIPLIB is a library of Mixed Integer Linear Programming instances that is commonly used for benchmarking integer programming algorithms. For our purpose we selected four 0/1 ILP instances of increasing difficulty. Table 3 reports a summary of the experiment. We observe that the AOMDD is smaller than the corresponding AND/OR search graph, especially for harder problems where the heuristic function generates relatively weak estimates. Results on these instances have been reported in [4], but we note that they are only for optimal solutions, while we also include suboptimal ones here. Also, in [4] they use of-the-shelf optimized BDD packages, while we use our own far-from-optimized implementation, and in particular we do not use redundancy reduction in these experiments. Moreover, the order of variables plays an enormous part in compilation and it may be the case that the ordering selected by the BDD tools is far superior (albeit OR ordering).

miplib	(n, c)	(w*, h)	time	#cm	#aomdd	ratio
p0033	(33, 15)	(19, 21)	0.52	441	164	**2.69**
p0040	(40, 23)	(19, 23)	0.36	129	77	**1.66**
p0201	(201, 133)	(120, 142)	89.44	12,683	5,499	**2.31**
lseu	(89, 28)	(57, 68)	454.79	109,126	21,491	**5.08**

Table 3. The trace of AND/OR Branch-and-Bound search (#cm) versus the AOMDD size (#aomdd) for the MIPLIB instances. Compilation time limit 1 hour.

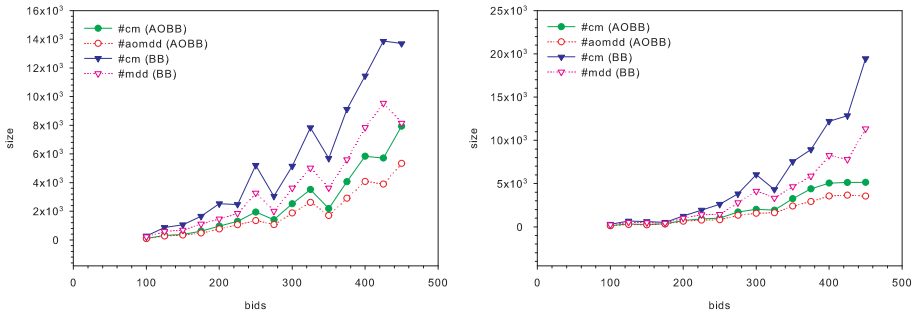


Fig. 5. The trace of AND/OR Branch-and-Bound search versus the AOMDD size for the `regions-upv` and `regions-npv` combinatorial auctions.

Combinatorial Auctions In combinatorial auctions, an auctioneer has a set of goods, $M = \{1, 2, ..., m\}$ to sell and the buyers submit a set of bids, $\mathcal{B} = \{B_1, B_2, ..., B_n\}$. A bid is a tuple $B_j = \langle S_j, p_j \rangle$, where $S_j \subseteq M$ is a set of goods and $p_j \geq 0$ is a price. The winner determination problem is to label the bids as winning or loosing so as to maximize the sum of the accepted bid prices under the constraint that each good is allocated to at most one bid. We used the 0/1 ILP formulation described in [17].

Figure 5 shows results for experiments with combinatorial auctions drawn from the `regions` distribution of the CATS 2.0 test suite [17]. The suffixes `npv` and `upv` indicate that the bid prices were drawn from either a normal or uniform distribution. These problem instances simulate the auction of radio spectrum in which a government sells the right to use specific segments of spectrum in different geographical areas. We looked at auctions with 100 goods and increasing number of bids. Each data point represents an average over 10 random instances. For comparison, we also included results obtained with the OR compilation scheme. On this domain, we observe that the compiled AOMDD improves only slightly over the size of the AND/OR search graph. This is because the context minimal AND/OR graph is already compact enough due to very accurate heuristic estimates.

Maximum Satisfiability (MAX-SAT) Given a set of Boolean variables the goal of MAX-SAT is to find a truth assignment to the variables that violates the least number of clauses. The MAX-SAT problem can be formulated as a 0/1 ILP [18]. We experimented with problem classes `pret` and `dubois` from the SATLIB (*http://www.satlib.org/*) library, which were previously shown to be difficult for 0/1 ILP solvers (CPLEX) [19].

Table 4 shows the results for experiments with 8 `pret` instances. These are unsatisfiable instances of graph 2-coloring with parity constraints. The size of these problems

pret	(w*, h)	time	#cm	#aomdd	ratio
pret60-25	(6, 13)	2.74	593	255	**2.33**
pret60-40	(6, 13)	3.39	698	256	**2.73**
pret60-60	(6, 13)	3.31	603	222	**2.72**
pret60-75	(6, 13)	2.70	565	253	**2.23**
pret150-25	(6, 15)	18.19	1,544	851	**1.81**
pret150-40	(6, 15)	29.09	2,042	922	**2.21**
pret150-60	(6, 15)	30.09	2,051	877	**2.34**
pret150-75	(6, 15)	29.08	2,033	890	**2.28**

Table 4. The trace of AND/OR Branch-and-Bound search (#cm) versus the AOMDD size (#aomdd) for MAX-SAT `pret` instances.

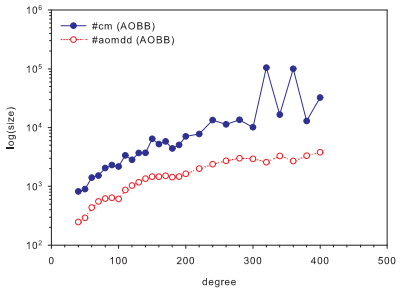


Fig. 6. The trace of AND/OR Branch-and-Bound search (#cm) versus the AOMDD size (#aomdd) for MAX-SAT `dubois` instances.

is relatively small (60 variables with 160 clauses for `pret60` and 150 variables with 400 clauses for `pret150`, respectively). However, they have a very small context with size 6 and a shallow pseudo-tree with depths between 13 and 15. For this problem class we observe that the AND/OR decision diagrams have about 2 times fewer nodes than the AND/OR search graphs explored by `AOBB`. This is because the respective search spaces are already small enough, and this does not leave much room for additional merging of isomorphic nodes in the diagram.

Figure 6 displays the results for experiments with random `dubois` instances with increasing number of variables. These are 3-SAT instances with $3 \times degree$ variables and $8 \times degree$ clauses, each of them having 3 literals. As in the previous test case, the `dubois` instances have very small contexts of size 6 and shallow pseudo-trees with depths ranging from 10 to 20. The AND/OR decision diagrams compiled for these problem instances are far smaller than the corresponding AND/OR search graphs, especially for some of the larger instances. For example, at degree 320, the corresponding AOMDD is 40 times smaller than the trace of `AOBB`.

6 Conclusion and Discussion

We presented a new search based algorithm for compiling the optimal solutions of a constraint optimization problem into a weighted AND/OR Multi-Valued Decision Diagram (AOMDD). Our approach draws its efficiency from: (1) AND/OR search spaces for graphical models [5] that exploit problem structure, yielding memory intensive search algorithms exponential in the problem's *treewidth* rather than *pathwidth*. (2) Heuristic search algorithms exploring tight portions of the AND/OR search space. In particular, we use here a state-of-the-art AND/OR Branch-and-Bound search algorithm [6, 10], with very efficient heuristics (mini-bucket, or simplex-based), that in practice traverses only a small portion of the context minimal graph by exploiting the pruning power of the cost function. (3) Reduction techniques similar to OBDDs further reduce the trace of the search algorithm.

The paper extends earlier work on AOMDDs [7] by considering weighted AOMDDs based on cost functions, rather than constraints. This can now easily be extended to

any weighted graphical model, for example to probabilistic networks. Finally, using an extensive experimental evaluation we show the efficiency and compactness of the weighted AOMDD data structure.

For future work, we mention the possibility of using best-first AND/OR search for the compilation task, whose virtue is that it does not expand nodes whose heuristic evaluation function is larger than the optimal cost.

References

1. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transaction on Computers **35** (1986) 677–691
2. Fargier, H., Vilarem, M.: Compiling CSPs into tree-driven automata for interactive solving. Constraints **9** (2004) 263–287
3. Hadzic, T., Andersen, H.R.: A BDD-based polytime algorithm for cost-bounded interactive configuration. In: National Conference on Artificial Intelligence (AAAI-2006). (2006)
4. Hadzic, T., Hooker, J.: Cost-bounded binary decision diagrams for 0-1 programming. In: International Conference on Integration of AI and OR Techniques (CPAIOR-2007). (2007)
5. Dechter, R., Mateescu, R.: AND/OR search spaces for graphical models. Artificial Intelligence **171** (2007) 73–106
6. Marinescu, R., Dechter, R.: AND/OR branch-and-bound for graphical models. In: International Joint Conferences on Artificial Intelligence (IJCAI-2005). (2005) 224–229
7. Mateescu, R., Dechter, R.: Compiling constraint networks into AND/OR multi-valued decision diagrams (AOMDDs). In: International Conference on Principles and Practice of Constraint Programming (CP-2006). (2006) 329–343
8. Freuder, E.C., Quinn, M.J.: Taking advantage of stable sets of variables in constraint satisfaction problems. In: International Joint Conferences on Artificial Intelligence (IJCAI-1985). (1985) 1076–1078
9. Bayardo, R., Miranker, D.: A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. In: National Conference on Artificial Intelligence (AAAI-1996). (1996) 298–304
10. Marinescu, R., Dechter, R.: Memory intensive branch-and-bound search for graphical models. In: National Conference on Artificial Intelligence (AAAI-2006). (2006)
11. Marinescu, R., Dechter, R.: Best-first AND/OR search for graphical models. In: National Conference on Artificial Intelligence (AAAI-2007). (2007)
12. Bistarelli, S., Montanari, U., Rossi, F.: Semiring based constraint solving and optimization. Journal of ACM **44** (1997) 309–315
13. Nemhauser, G., Wolsey, L.: Integer and combinatorial optimization. Wiley (1988)
14. Bensana, E., Lemaitre, M., Verfaillie, G.: Earth observation satellite management. Constraints **4** (1999) 293–299
15. Marinescu, R., Dechter, R.: AND/OR branch-and-bound search for pure 0/1 integer linear programming problems. In: International Conference on Integration of AI and OR Techniques (CPAIOR-2006). (2006) 152–166
16. Dantzig, G.: Maximization of a linear function of variables subject to linear inequalities. Activity Analysis of Production and Allocation (1951)
17. Leyton-Brown, K., Pearson, M., Shoham, Y.: Towards a universal test suite for combinatorial auction algorithms. In ACM Electronic Commerce (2000) 66–76
18. Joy, S., Mitchell, J., Borchers, B.: A branch and cut algorithm for max-SAT and weighted max-SAT. Satisfiability Problem: Theory and Applications (1997) 519–536
19. de Givry, S., Larrosa, J., Schiex, T.: Solving max-SAT as weighted CSP. In: International Conference on Principles and Practice of Constraint Programming (CP-2003). (2003)