Dynamic Orderings for AND/OR Branch-and-Bound Search in Graphical Models

Radu Marinescu and **Rina Dechter** [1]

Abstract. *AND/OR search spaces* have recently been introduced as a unifying paradigm for advanced algorithmic schemes for graphical models. The main virtue of this representation is its sensitivity to the structure of the model, which can translate into exponential time savings for search algorithms. Since the variable selection can have a dramatic impact on search performance when solving optimization tasks, we introduce in this paper a new *dynamic* AND/OR Branch-and-Bound algorithmic framework which accommodates variable ordering heuristics. The efficiency of the dynamic AND/OR approach is demonstrated empirically in a variety of domains.

1 INTRODUCTION

Graphical models (e.g. constraint and belief networks) are a powerful representation framework for various automated reasoning tasks. These models use graphs to capture conditional independencies between variables, allowing a concise representation of the knowledge, as well as efficient graph-based query processing algorithms. *Constraint Optimization Problems* such as finding a solution that violates the least number of constraints or finding the most likely state of a belief network can be defined within this framework and they are typically tackled with either *search* or *inference* algorithms [9].

The AND/OR search space for graphical models [8] is a newly introduced framework for search that is sensitive to the independencies in the model, often resulting in exponentially reduced complexities. It is based on a pseudo-tree that captures independencies in the graphical model, resulting in a search tree exponential in the depth of the pseudo-tree, rather than in the number of variables.

The AND/OR Branch-and-Bound algorithm (AOBB) is a new search method that explores the AND/OR search tree for solving optimization tasks in graphical models [16]. If restricted to a static variable ordering, the algorithm was shown to outperform a static version of the traditional OR Branch-and-Bound. In practice however, variable selection can have a dramatic influence on search performance for solving constraint satisfaction and optimization tasks [9]. In the context of OR search spaces there exists a whole line of research that mitigates this problem and focuses on the practical use of variable ordering heuristics during search. The most powerful OR Branch-and-Bound solvers developed in the past years, such as those maintaining a form directional local consistency [6] or those guided by bounded tree-clustering inference [7] rely heavily on variable ordering heuristics in order to improve efficiency.

In this paper we introduce a collection of *dynamic* AND/OR Branch-and-Bound algorithms that extend AOBB by combining the AND/OR decomposition principle with variable ordering heuristics. There are three approaches to incorporating dynamic orderings into AOBB. The first one improves AOBB by applying an independent semantic variable ordering heuristic whenever the partial order dictated by the static decomposition principle allows. The second, orthogonal approach gives priority to the semantic variable ordering heuristic and applies problem decomposition as a secondary principle. Since the structure of the problem may change dramatically during search we introduce a third approach that uses a dynamic decomposition method coupled with semantic variable ordering heuristics.

We apply the dynamic AND/OR Branch-and-Bound algorithms on two optimization tasks: solving Weighted CSPs [6] and pure 0-1 Integer Linear Programming problems [17]. We experiment with various random models and real-world scheduling and resource allocation problems. Our results show conclusively that the new dynamic AND/OR approach outperforms significantly the classic OR as well as the existing static AND/OR tree search algorithms.

[1] University of California, Irvine, USA, email: {radum,dechter}@ics.uci.edu

2 BACKGROUND

A finite *Constraint Optimization Problem* (COP) is a six-tuple $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \otimes, \Downarrow, Z \rangle$, where $\mathcal{X} = \{X_1, ..., X_n\}$ is a set of variables, $\mathcal{D} = \{D_1, ..., D_n\}$ is a set of finite domains and $\mathcal{F} = \{f_1, ..., f_m\}$ is a set of constraints. Constraints can be either *soft* (cost functions) or *hard* (sets of allowed tuples). 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 $scope(f_i) \subseteq \mathcal{X}$, is the set of arguments of f_i. The operators $\otimes$ and $\Downarrow$ are defined as follows: $\otimes_i f_i$ is a *combination* operator, $\otimes_i f_i \in \{\prod_i f_i, \sum_i f_i\}$ and $\Downarrow_Y f$ is an *elimination* operator, $\Downarrow_Y f \in \{max_{S-Y} f, min_{S-Y} f\}$, where S is the scope of function f and $Y \subseteq \mathcal{X}$. The scope of $\Downarrow_Y f$ is Y.

An optimization task is defined by $g(Z) = \Downarrow_Z \otimes_{i=1}^{m} f_i$, where $Z \subseteq \mathcal{X}$. A *global optimization* is the task of finding the best global cost, namely $Z = \emptyset$. For simplicity we will develop our work assuming a COP instance with *summation* and *minimization* as combination and elimination operators, and a global cost function defined by $f(\mathcal{X}) = min_{\mathcal{X}} \sum_{i=1}^{m} f_i$.

The *constraint graph* of a COP instance has the variables $\mathcal{X}$ as its nodes and an arc connects any two variables that appear in the scope of the same function.

3 AND/OR SEARCH TREES

The classical way to do search is to instantiate variables one at a time, following a static/dynamic variable ordering. In the simplest case, this process defines a search tree (called here OR search tree), whose nodes represent states in the space of partial assignments. The traditional search space does not capture independencies that appear in

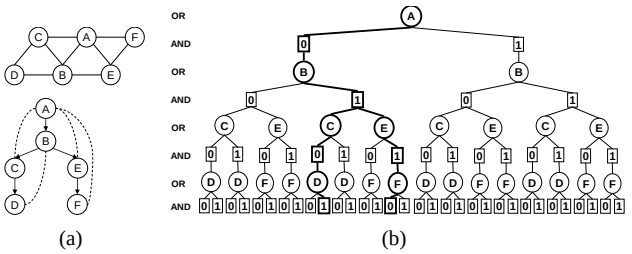

Figure 1. The AND/OR search space.

the structure of the constraint graph. Introducing AND states into the search space can capture the structure, decomposing the problem into independent subproblems by conditioning on values. The AND/OR search space is defined using a backbone *pseudo-tree* [10, 1, 8].

DEFINITION **1 (pseudo-tree)** *Given an undirected graph $G = (V, E)$, a directed rooted tree $T = (V, E')$ defined on all its nodes is called* pseudo-tree *if any arc of G which is not included in E' is a back-arc, namely it connects a node to an ancestor in T.*

Given a COP instance $\langle \mathcal{X}, \mathcal{D}, \mathcal{F} \rangle$, its constraint graph G and a pseudo-tree T of G, the associated AND/OR search tree S_T has alternating levels of OR nodes and AND nodes. The OR nodes are labeled by X_i and correspond to the variables. The AND nodes are labeled by $\langle X_i, x_i \rangle$ and correspond to value assignments in the domains of the variables. The structure of the AND/OR tree is based on the underlying pseudo-tree T of G. The root of the AND/OR search tree is an OR node, labeled with the root of T.

The children of an OR node X_i are AND nodes labeled with value assignments $\langle X_i, x_i \rangle$, consistent along the path from the root, $path(X_i, x_i) = (\langle X_1, x_1 \rangle, ..., \langle X_{i-1}, x_{i-1} \rangle)$. The children of an AND node $\langle X_i, x_i \rangle$ are OR nodes labeled with the children of variable X_i in T. In other words, the OR states represent alternative ways of solving the problem, whereas the AND states represent problem decomposition into independent subproblems, all of which need be solved. When the pseudo-tree is a chain, the AND/OR search tree coincides with the regular OR search tree.

A *solution subtree* Sol_{S_T} of S_T is an AND/OR subtree such that: (i) it contains the root of S_T; (ii) if a nonterminal AND node $n \in S_T$ is in Sol_{S_T} then all its children are in Sol_{S_T}; (iii) if a nonterminal OR node $n \in S_T$ is in Sol_T then exactly one of its children is in Sol_{S_T}.

Example 1 *Figure 1(a) shows the constraint graph of a COP instance and a pseudo-tree together with the back-arcs (dotted lines). Figure 1(b) shows the AND/OR search tree based on the pseudo-tree, for bi-valued variables. A solution subtree is highlighted.*

The AND/OR search tree can be traversed by a depth-first search algorithm that is guaranteed to have a time complexity exponential in the depth of the pseudo-tree and can use linear space [16]. The arcs from X_i to $\langle X_i, x_i \rangle$ are annotated by appropriate *labels* of the cost functions. The nodes in S_T can be associated with *values*, defined over the subtrees they root.

DEFINITION **2 (label)** *The* label $l(X_i, x_i)$ *of the arc from the OR node X_i to the AND node $\langle X_i, x_i \rangle$ is defined as the sum of all the cost functions values for which variable X_i is contained in their scope and whose scope is fully assigned along $path(X_i, x_i)$.*

```
function: AOBB(st, X, D, F)
1  if X = ∅ then return 0;
2  else
3     X_i ← SelectVar(X);
4     v(X_i) ← ∞;
5     foreach x_i ∈ D_i do
6        st' ← st ∪ (X_i, x_i);
7        h(X_i, x_i) ← LB(X, D, F);
8        foreach k = 1..q do
9           h(X_k) ← LB(X_k,D_k,F_k);
10          UpdateContext(out, X_k, lb_k);
11       end
12       if ¬FindCut(X_i,x_i,in,out,h(X_i, x_i)) then
13          v(X_i, x_i) ← 0;
14          foreach k = 1..q do
15             D'_k ← LookAhead(X_k,D_k,F_k);
16             if ¬EmptyDomain(D'_k) then
17                val ←AOBB(st',X_k,D'_k,F_k);
18                v(X_i, x_i) ← v(X_i, x_i) + val;
19             else
20                v(X_i, x_i) ← ∞;
21                break;
22          end
23          v(X_i, x_i) ← v(X_i, x_i)+label(X_i,x_i);
24          UpdateContext(in, v(X_i, x_i));
25          v(X_i) ← min(v(X_i), v(X_i, x_i));
26       end
27    end
28    return v(X_i);
29 end
```

Figure 2. AND/OR Branch-and-Bound search.

DEFINITION **3 (value)** *The* value $v(n)$ *of a node $n \in S_T$ is defined recursively as follows: (i) if $n = \langle X_i, x_i \rangle$ is a terminal AND node then $v(n) = l(X_i, x_i)$; (ii) if $n = \langle X_i, x_i \rangle$ is an internal AND node then $v(n) = l(X_i, x_i) + \sum_{n' \in succ(n)} v(n')$; (iii) if $n = X_i$ is an internal OR node then $v(n) = min_{n' \in succ(n)} v(n')$, where $succ(n)$ are the children of n in S_T.*

Clearly, the value of each node can be computed recursively, from leaves to root.

PROPOSITION **1** *Given an AND/OR search tree S_T of a COP instance $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, the value $v(n)$ of a node $n \in S_T$ is the minimal cost solution to the subproblem rooted at n, subject to the current variable instantiation along the path from root to n. If n is the root of S_T, then $v(n)$ is the minimal cost solution to $\mathcal{P}$.*

4 AND/OR BRANCH-AND-BOUND SEARCH

AND/OR Branch-and-Bound (AOBB) was recently proposed by [16] as a depth-first Branch-and-Bound that explores an AND/OR search tree for solving optimization tasks in graphical models. In this section we overview the static version of the algorithm.

4.1 Lower Bounds on Partial Trees

At any stage during search, a node n along the current path roots a current *partial solution subtree*, denoted by $G_{sol}(n)$, which must be connected, must contain its root n and will have a *frontier* containing all those nodes that were generated and not yet expanded. Furthermore, there exists a *static* heuristic function $h(n)$ underestimating $v(n)$ that can be computed efficiently when node n is first generated.

Given the current partially explored AND/OR search tree S_T, the *active path* $\mathcal{AP}(t)$ is the path of assignments from the root of S_T

to the current tip node t. The *inside context* $in(\mathcal{AP})$ of $\mathcal{AP}(t)$ contains all nodes that were fully evaluated and are children of nodes on $\mathcal{AP}(t)$. The *outside context* $out(\mathcal{AP})$ of $\mathcal{AP}(t)$, contains all the frontier nodes that are children of the nodes on $\mathcal{AP}(t)$. The *active partial subtree* $\mathcal{APT}(n)$ rooted at a node $n \in \mathcal{AP}(t)$ is the subtree of $G_{sol}(n)$ containing the nodes on $\mathcal{AP}(t)$ between n and t together with their OR children. A *dynamic heuristic evaluation function* of a node n relative to $\mathcal{APT}(n)$ which underestimates $v(n)$ is defined as follows (see [16] for more details).

DEFINITION **4 (dynamic heuristic evaluation function)** *Given an active partial tree $\mathcal{APT}(n)$, the* dynamic heuristic evaluation function *of n, $f_h(n)$, is defined recursively as follows: (i) if $\mathcal{APT}(n)$ consists only of a single node n, and if $n \in in(\mathcal{AP})$ then $f_h(n) = v(n)$ else $f_h(n) = h(n)$; (ii) if $n = \langle X_i, x_i \rangle$ is an AND node, having OR children $m_1, ..., m_k$ then $f_h(n) = max(h(n), l(X_i, x_i) + \sum_{i=1}^{k} f_h(m_i))$; (iii) if $n = X_i$ is an OR node, having an AND child m, then $f_h(n) = max(h(n), f_h(m))$.*

4.2 Static AND/OR Branch-and-Bound

AOBB can calculate a *lower bound* on $v(n)$ for any node n on the active path, by using $f_h(n)$. It also maintains an *upper bound* on $v(n)$ which is the current minimal cost solution subtree rooted at n. If $f_h(n) \geq ub(n)$ then the search can be safely terminated below the tip node of the active path.

Figure 2 shows AOBB. The following notation is used: $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ is the problem with which the procedure is called, st is the current partial solution subtree being explored, in (resp. out) represents the inside (resp. outside) context of the active path. These contexts are constantly updated during search. The algorithm assumes that variables are selected statically according to a pseudo-tree.

If the set $\mathcal{X}$ is empty, then the result is trivially computed (line 1). Else, AOBB selects a variable X_i (i.e. expands the OR node X_i) and iterates over its values (line 5) to compute the OR value $v(X_i)$. For each value x_i the problem rooted by the AND node $\langle X_i, x_i \rangle$ is decomposed into a set of q independent subproblems $P_k = (\mathcal{X}_k, \mathcal{D}_k, \mathcal{F}_k)$, one for each child X_k of X_i in the pseudo-tree. Procedure LB computes the static heuristic function $h(n)$ for every node n in the search tree.

When expanding the AND node $\langle X_i, x_i \rangle$, AOBB successively computes the *dynamic heuristic evaluation function* $f_h(m)$ for every ancestor node m along the active path and terminates the current search path if, for some m, $f_h(m) \geq ub(m)$. Else, the independent subproblems are solved recursively (lines 14-22) and the results accumulated by the AND value $v(X_i, x_i)$ (line 18). A lookahead procedure is also executed in which unfeasible values are removed from future domains (line 15). After trying all feasible values of variable X_i, the minimal cost solution to the problem rooted by X_i remains in $v(X_i)$, which is returned (line 28).

5 DYNAMIC AND/OR BRANCH-AND-BOUND

In this section we go beyond static orderings and introduce a collection of *dynamic* AND/OR Branch-and-Bound algorithms that incorporate variable ordering heuristics used in the classic OR space. Similar structure-aware variable ordering heuristics have been previously investigated in the context of SAT and model counting [2, 11, 15].

We distinguish two classes of variable ordering heuristics. *Graph*-based heuristics (e.g. pseudo-tree arrangements) that try to maximize problem decomposition and *semantic*-based heuristics (e.g. min-domain, max-degree, min-dom/deg) that aim at shrinking the search space. These two forces are orthogonal, namely we can use one as the primary goal and break ties based on the other. Moreover, we can use each class statically or dynamically. We present next three ways of combining efficiently these two classes of heuristics.

5.1 Partial Variable Ordering (PVO)

The first approach, called *AND/OR Branch-and-Bound with Partial Variable Ordering* (AOBB+PVO) combines the static graph-based decomposition given by a pseudo-tree with a dynamic semantic ordering heuristic. It is an adaptation of the ordering heuristics developed by [11] and [15] for solving large-scale SAT problem instances.

Let us illustrate the idea with an example. Consider the pseudo-tree from Figure 1(a) inducing the following variable group ordering: {A,B}, {C,D}, {E,F}; which dictates that variables {A,B} should be considered before {C,D} and {E,F}. Variables in each group can be dynamically ordered based on a second, independent semantic-based heuristic. Notice that after variables {A,B} are instantiated, the problem decomposes into independent components that can be solved separately.

AOBB+PVO is similar to the algorithm from Figure 2 traversing an AND/OR search tree guided by a pre-computed pseudo-tree. Procedure SelectVar in this case implements the dynamic semantic ordering heuristic which selects next the best scoring variable from the current variable group of the pseudo-tree.

5.2 Dynamic Variable Ordering (DVO)

The second, orthogonal approach to PVO called *AND/OR Branch-and-Bound with Dynamic Variable Ordering* (DVO+AOBB), gives priority to the dynamic semantic ordering heuristic and applies static problem decomposition as a secondary principle during search. This idea was also explored by [2] in the context of SAT model counting.

DVO+AOBB is also based on the algorithm from Figure 2. It instantiates variables dynamically using a semantic ordering heuristic while constantly updating the problem graph structure. Specifically, after variable X_i is selected by procedure SelectVar, DVO+AOBB tentatively removes X_i from the graph and, if disconnected components are detected their corresponding subproblems are then solved separately and the results combined in an AND/OR manner (lines 14-22). It easy to see that in this case a variable may have the best semantic heuristic to tighten the search space, yet, it may not yield a good decomposition for the remaining of the problem, in which case the algorithm would explore primarily an OR space.

5.3 Dynamic Separator Ordering (DSO)

The third approach, called *AND/OR Branch-and-Bound with Dynamic Separator Ordering* (AOBB+DSO), combines a dynamic graph-based decomposition heuristic with a dynamic semantic ordering heuristic giving priority to the first. The idea is supported by the constraint propagation (e.g. lookahead) procedure which may detect singleton variables (i.e. with only one feasible value left in their domains). When the value of a variable is known, we can remove it from the corresponding subproblem. Therefore, we may expect a better decomposition based on a simplified constraint graph.

AOBB+DSO is also based on the algorithm from Figure 2. It creates and maintains a separator S of the current constraint graph. A graph separator can be computed using the hypergraph partitioning method presented in [15]. The next variable is chosen dynamically

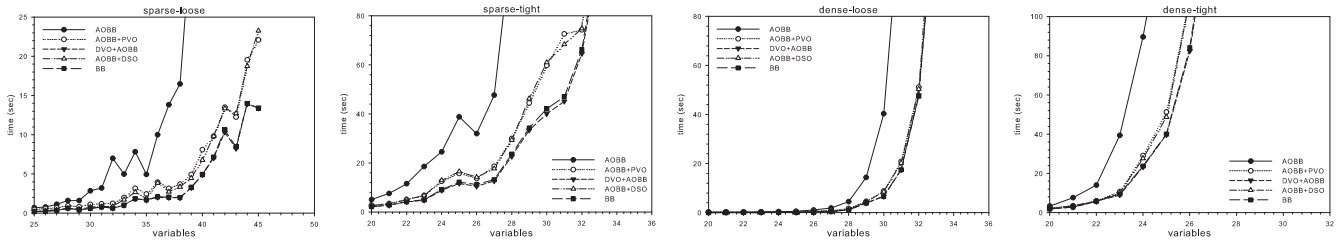

Figure 3. Time in seconds to prove optimality for random binary WCSPs.

from S by the semantic ordering heuristic until all variables are instantiated and the current problem decomposes into several subproblems. These independent subproblems are then solved separately and their solutions combined in an AND/OR fashion. The separator of each component is created from a simplified subgraph resulted from previous constraint propagation steps and it may differ for different value assignments. Clearly, if no singleton variables are discovered by the lookahead steps this approach is computationally identical to AOBB+PVO, although it may have a higher overhead due to the dynamic generation of the separators.

6 EXPERIMENTS

In this section we evaluate empirically the performance of the dynamic AND/OR Branch-and-Bound algorithms on two optimization tasks: solving Weighted CSPs and pure 0-1 Integer Linear Programming problems[2]. For comparison, we include results obtained with the classic OR Branch-and-Bound (BB) algorithm.

BB explores an OR search tree in a depth-first manner maintaining an upper bound, the best solution cost found so far, and a heuristic evaluation function (i.e. lower bound) of the optimal extension of the current partial assignment. The subtree below the current path is pruned as soon as the lower bound exceeds the upper bound. The algorithm uses a dynamic semantic-based variable ordering heuristic.

Weighted CSP (WCSP) [6] extends the classic CSP formalism with so-called *soft constraints* which assign a positive integer penalty cost to each forbidden tuple (allowed tuples have cost 0). The goal is to find a complete assignment with minimum aggregated cost.

A *pure 0-1 Integer Linear Programming* (0-1 ILP) [17] 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.

The semantic-based variable ordering and heuristic evaluation function used in each domain by the OR and AND/OR Branch-and-Bound algorithms are as follows. For WCSPs we use the *min-dom/deg* heuristic (which selects the variable with the smallest ratio of the domain size divided by the future degree) and the heuristic evaluation function is computed by maintaining Existential Directional Arc Consistency (EDAC) [6]. For 0-1 ILPs we use the *fraction* heuristic (which selects the fractional variable closest to 0.5) and the heuristic evaluation function is computed by solving the linear relaxation of the current subproblem with the SIMPLEX[3] method [5, 17]. Ties were broken lexicographically. The pseudo-tree that guides AOBB and AOBB+PVO as well as the graph separator used by AOBB+DSO were generated based on the recursive decomposition of a hypergraph representation of the problem [11, 15].

[2] All experiments were done on a 2.4GHz Pentium IV with 2GB of RAM.

[3] Our code used the SIMPLEX implementation from the open source library lp_solve 5.5 available at http://groups.yahoo.com/group/lp_solve/

Table 1. Time in seconds to prove optimality for SPOT5 benchmarks.

spot	(w*,h)	BB time	AOBB time	AOBB+ PVO time	DVO+ AOBB time	AOBB+ DSO time
29	(15, 22)	0.41	1.91	1.93	**0.33**	0.68
42	(38, 46)	-	-	-	**4709**	-
54	(12, 16)	0.06	1.13	1.07	**0.03**	0.08
404	(20, 25)	0.03	2.50	2.50	**0.02**	0.03
503	(11, 21)	11.70	2.78	2.78	**0.03**	0.13
505	(27, 42)	4010	75.43	75.37	**17.28**	82.74

We report the average CPU time required for proving optimality of the solution (the nodes visited are not shown for space reasons), the induced width (w*) and depth of the pseudo-tree (h) obtained for the test instances. The best performance points are highlighted.

6.1 Random Weighted CSPs

We have generated 4 classes of binary random WCSPs with domain size and penalty weight set to 10 (Sparse-Loose, Sparse-Tight, Dense-Loose, Dense-Tight) as proposed in [6]. Figure 3 gives the time results in seconds (each data point represents an average over 20 samples). When comparing AOBB+PVO with static AOBB we notice a considerable improvement in terms of both running time and size of search space explored. AOBB+DSO has a similar performance as AOBB+PVO indicating that both algorithms use decompositions of similar quality. The best performance of all 4 problem classes is offered by DVO+AOBB and BB with no clear winner between the two. This implies that the semantic ordering heuristic is powerful and it does not leave much room for additional problem decompositions.

6.2 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 [3] which can be formulated as non-binary WCSPs [3]. 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.

Table 1 shows the results for 6 scheduling problems. We observe that in this domain DVO+AOBB is the best performing algorithm. In spot-42, the hardest instance, DVO+AOBB proves optimality in about one and a half hours, thus demonstrating the power of augmenting a semantic-based ordering heuristic with a static decomposition principle. The other algorithms exceeded the 10 hour time limit. AOBB+DSO is the second best algorithm, always exploring

Table 2. Time in seconds to prove optimality for CELAR6 benchmarks.

celar6	(w*,h)	BB time	AOBB time	AOBB+ PVO time	DVO+ AOBB time	AOBB+ DSO time
sub0	(7, 8)	0.88	0.73	0.81	0.92	**0.71**
sub1	(9, 9)	2260	3101	2200	**2182**	2246
sub1-24	(9, 9)	136	171	128	**127**	132
sub2	(10, 12)	4696	10963	7047	**4024**	6696
sub3	(10, 13)	14687	32439	28252	**11131**	28407
sub4-20	(11, 15)	681	137	157	**70**	179

fewer nodes than the other AND/OR algorithms. Its computational overhead of computing the separators dynamically did not pay off in some test cases (e.g. `spot-505`).

6.3 Radio Link Frequency Assignment Problem

RLFAP is a communication problem where the goal is to assign frequencies to a set of radio links in such a way that all links may operate together without noticeable interferences [4]. It can be naturally casted as a binary WCSP where each forbidden tuple has an associated penalty cost. Table 2 compares the OR and AND/OR algorithms for solving 6 publicly available RLFAP subinstances called CELAR6-SUB$_i$ $(i = 1, ..., 4)$. We observe that in this domain also DVO+AOBB offers the best performance. On average, the speedups caused by DVO+AOBB over the other algorithms are as follows: 1.9x over AOBB, 1.6x over AOBB+PVO and 2.5x over BB. AOBB+DSO performs similarly to AOBB+PVO indicating that the quality of the dynamic problem decomposition is comparable to the static one.

6.4 Uncapacitated Warehouse Location Problem

In UWLP a company considers opening m warehouses at some candidate locations in order to supply its n existing stores. The objective is to determine which warehouse to open, and which of these warehouses should supply the various stores, such that the sum of the maintenance and supply costs is minimized. Each store must be supplied by exactly one warehouse. UWLP is typically formulated as a pure 0-1 integer linear program [17].

Table 3. Time in seconds to prove optimality for UWLP benchmarks.

uwlp	BB time	AOBB time	AOBB+ PVO time	DVO+ AOBB time
1	**6.27**	15.72	6.28	7.23
2	11.34	17.22	**5.78**	11.75
3	73.66	15.78	**5.83**	77.94
4	836.52	27.94	**11.97**	904.15
5	2501.75	32.69	**16.98**	2990.19
6	43.36	18.70	**8.03**	45.99
7	1328.40	27.89	**8.53**	1515.48
8	76.88	25.20	**13.70**	88.38
9	224.33	46.06	**17.17**	367.14
10	7737.65	28.03	**9.13**	9276.98

Table 3 displays the results obtained for 10 randomly generated UWLP problem instances[4] with 50 warehouses and 200 stores. The warehouse opening and store supply costs were chosen uniformly randomly between 0 and 1000. These are large problems with 10,050 variables and 10,500 constraints. We can see that AOBB+PVO dominates in all test cases, outperforming BB (resp. DVO+AOBB) with several orders of magnitude in terms of running time and size of the search space explored. This is due to the problem's structure partially captured by a shallow pseudo-tree with depth 123 and induced width 50. DVO+AOBB has a similar performance as BB (it is slower than BB due to its computational overhead), indicating that these problems do not break into disconnected components when semantic variable ordering has higher priority than problem decomposition.

In summary, for WCSP instances with relatively low connectivity (e.g. random WCSPs, SPOT5 networks) as well as for instances with higher graph density (e.g. CELAR6 networks) DVO+AOBB outperforms significantly both its OR and AND/OR competitors. For sparse 0-1 integer programs (e.g. UWLP) which yield extremely shallow pseudo-trees AOBB+PVO appears to be the preferred choice.

[4] We used the random problem generator from http://www.mpi-sb.mpg.de/units/ag1/projects/benchmarks/UflLib/

7 CONCLUSION

In this paper we extended the AND/OR Branch-and-Bound algorithmic framework with dynamic orderings for combining efficiently variable ordering heuristics with problem decomposition principles. The effectiveness of the dynamic AND/OR approach is demonstrated empirically in a variety of domains including random models as well as hard real-world problem instances.

Related Work: AOBB is related to the Branch-and-Bound method proposed by [13] for acyclic AND/OR graphs and game trees, as well as to the pseudo-tree search algorithm proposed in [14] for boosting Russian Doll search. BTD algorithm developed in [12] for semi-ring CSPs can also be interpreted as an AND/OR graph search algorithm.

ACKNOWLEDGEMENTS

This work was supported by the NSF grant IIS-0412854.

REFERENCES

[1] R. Bayardo and D. Miranker, 'On the space-time trade-off in solving constraint satisfaction problems.', in *IJCAI*, pp. 558–562, (1995).
[2] R. Bayardo and J. D. Pehoushek, 'Counting models using connected components', in *AAAI/IAAI*, pp. 157–162, (2000).
[3] E. Bensana, M. Lemaitre, and G. Verfaillie, 'Earth observation satellite management.', *Constraints*, **4**(3), 293–299, (1999).
[4] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. Warners, 'Radio link frequency assignment.', *Constraints*, **1**(4), 79–89, (1999).
[5] G.B. Dantzig, 'Maximization of a linear function of variables subject to linear inequalities.', *Activity Analysis of Production and Allocation*, (1951).
[6] S. de Givry, F. Heras, J. Larrosa, and M. Zytnicki, 'Existential arc consistency: getting closer to full arc consistency in weighted csps.', in *IJCAI*, pp. 84–89, (2005).
[7] R. Dechter, K. Kask, and J. Larrosa, 'A general scheme for multiple lower bound computation in constraint optimization', in *CP*, pp. 346–360, (2001).
[8] R. Dechter and R. Mateescu, 'Mixtures of deterministic-probabilistic networks.', in *UAI*, pp. 120–129, (2004).
[9] Rina Dechter, *Constraint Processing*, MIT Press, 2003.
[10] E. Freuder and M. Quinn, 'Taking advantage of stable sets of variables in constraint satisfaction problems.', in *IJCAI*, pp. 1076–1078, (1985).
[11] J. Huang and A. Darwiche, 'A structure-based variable ordering heuristic.', in *IJCAI*, pp. 1167–1172, (2003).
[12] P. Jegou and C. Terrioux, 'Decomposition and good recording for solving max-csps.', in *ECAI*, (2004).
[13] L. Kanal and V. Kumar, *Search in artificial intelligence.*, Springer-Verlag., 1988.
[14] J. Larrosa, P. Meseguer, and M. Sanchez, 'Pseudo-tree search with soft constraints.', in *ECAI*, pp. 131–135, (2002).
[15] W. Li and P. van Beek, 'Guiding real-world sat solving with dynamic hypergraph separator decomposition.', in *ICTAI*, pp. 542–548, (2004).
[16] R. Marinescu and R. Dechter, 'And/or branch-and-bound for graphical models.', in *IJCAI*, pp. 224–229, (2005).
[17] G. Nemhauser and L. Wolsey, *Integer and combinatorial optimization.*, Wiley, 1988.