Best-First AND/OR Search for 0/1 Integer Programming

Radu Marinescu and Rina Dechter

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

Abstract. AND/OR search spaces are 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. In this paper we introduce an AND/OR search algorithm that explores a context-minimal AND/OR search graph in a *best-first* manner for solving 0/1 Integer Linear Programs (0/1 ILP). We also extend to the 0/1 ILP domain the *depth-first* AND/OR Branch-and-Bound search with caching algorithm which was recently proposed by [1] for solving optimization tasks in graphical models. The effectiveness of the best-first AND/OR search approach compared to the depth-first AND/OR Branch-and-Bound search is demonstrated on a variety of benchmarks for 0/1 ILPs, including instances from the MIPLIB library, real-world combinatorial auctions, random uncapacitated warehouse location problems and MAX-SAT instances.

1 Introduction

In *constraint optimization* the goal is to minimize (or maximize) an objective function, subject to a set of constraints on the possible values of a set of independent decision variables. An important class of constraint optimization problems are the 0/1 Integer Linear Programming problems (0/1 ILP) [2] where the objective is to optimize a linear function of binary integer variables, subject to a set of linear equality or inequality constraints defined on subsets of variables. The classical approach to solving 0/1 ILPs is the *Branch-and-Bound* method [3] which maintains the best solution found so far, while discarding partial solutions which cannot improve on the best.

The AND/OR search space for graphical models [4] is a 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 space exponential in the depth of the pseudo-tree, rather than in the number of variables.

The AND/OR Branch-and-Bound search ($\texttt{AOBB}_t$) was first introduced by [5] as a Branch-and-Bound algorithm that explores an AND/OR search tree in a depth-first manner for solving optimization tasks in graphical models. The AND/OR Branch-and-Bound search with caching algorithm ($\texttt{AOBB}_g$) due to [1] improves $\texttt{AOBB}_t$ by allowing the algorithm to save previously computed results and retrieve them when the same subproblems are encountered again. These algorithms are restricted to a static variable ordering determined by the underlying pseudo-tree. More recently, [6, 7] proposed

several extensions of $\texttt{AOBB}_t$ that incorporate dynamic variable ordering heuristics and explore dynamic AND/OR search trees. Two such extensions, *AND/OR Branch-and-Bound with Partial Variable Ordering* ($\texttt{AOBB}_t$+PVO) and *AND/OR Branch-and-Bound with Full Dynamic Variable Ordering* ($\texttt{AOBB}_t$+DVO) were shown to outperform significantly the static $\texttt{AOBB}_t$ algorithm as well as state-of-the-art classic OR Branch-and-Bound algorithms on various domains, including 0/1 ILPs.

In this paper we present and evalute an new AND/OR search algorithm, that explores an AND/OR search graph in a *best-first* manner for solving 0/1 ILPs. Under conditions of admissibility and monotonicity of the guiding heuristic function, best-first search is known to expand the minimal number of nodes, at the expense of using additional memory [8]. In practice, these savings in number of nodes may often translate into impressive time savings as well. Since variable selection can have a dramatic impact on search performance, we also introduce a best-first AND/OR search algorithm that explores an AND/OR search tree, rather than a graph, and combines the AND/OR decomposition principle with dynamic variable selection heuristics, in a similar fashion as the dynamic AND/OR Branch-and-Bound algorithms described in [6, 7]. We also adapt the static $\texttt{AOBB}_g$ algorithm for solving 0/1 ILPs.

We demonstrate empirically the efficiency of our best-first AND/OR search approach compared to the depth-first AND/OR Branch-and-Bound search on several benchmarks for 0/1 ILP, including test instances from the MIPLIB library, combinatorial auctions simulating radio spectrum allocation, random uncapacitated warehouse location problems and MAX-SAT instances from the SATLIB library.

The paper is organized as follows. In Section 2 we present background on 0/1 Integer Linear Programming and AND/OR search spaces. In Section 3 we introduce the best-first AND/OR search algorithm as well as the extension to 0/1 ILPs of the depth-first AND/OR Branch-and-Bound search with caching. In Section 4 we present a best-first AND/OR search algorithm that incorporates dynamic variable orderings. Section 5 shows our empirical evaluation and Section 6 concludes.

2 Background

2.1 Integer Linear Programming

A *Linear Program* (LP) consists of a set of continuous variables and a set of linear constraints (equalities or inequalities). The goal is to optimize a global linear cost function subject to the constraints. One of the standard forms of a linear program is:

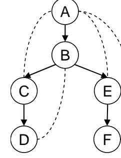
$$min\{c^\top x \mid Ax \leq b, x \geq 0\} \tag{1}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$. Here c represents the cost vector and x is the vector of decision variables. The vector b and the matrix A define the m linear constraints. Linear programs are usually solved by Dantzig's SIMPLEX method [9].

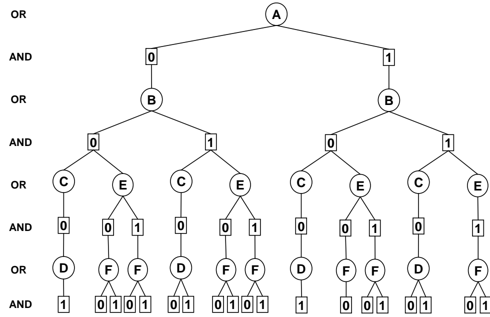
An *Integer Linear Programming* (ILP) problem is a linear program where all the decision variables are constrained to have integer values at the optimal solution. An important special case is a decision variable x_i that is integer with $0 \leq x_i \leq 1$. This forces x_i to be either 0 or 1 at the solution. Variables like x_i are called *0/1* or *binary*

$$\text{minimize} : z = 7A + 3B - 2C + 5D - 6E + 8F$$
subject to :
$$3A - 12B + C \leq 3$$
$$-2B + 5C - 3D \leq -2$$
$$2A + B - 4E \leq 2$$
$$A - 3E + F \leq 1$$
$$A, B, C, D, E, F \in \{0,1\}$$

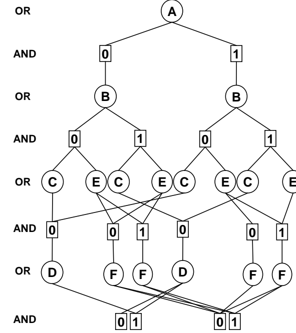
(a)



(b)



(c)



(d)

Fig. 1. The AND/OR search space.

integer variables. A *0/1 Integer Linear Programming* problem is an ILP where all the decision variables are binary. 0/1 ILPs can formulate many practical problems such as capital budgeting [10], cargo loading [11], processor allocation in distributed systems [12], combinatorial auctions [13, 14] or maximum satisfiability problems [15, 16].

With every 0/1 ILP instance we can associate an *interaction graph* G which has a node for each variable and connects any two nodes whose variables appear in the scope of the same constraint. The *induced graph* of G relative to an ordering d of its variables, denoted $G^*(d)$, 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. Given a graph and an ordering of its nodes, the *width* of a node is the number of edges connecting it to nodes lower in the ordering. The *induced width* of a graph, denoted $w^*(d)$, is the maximum width of nodes in the induced graph.

In the remainder, we will consider the *minimization* of a 0/1 ILP instance defined by a linear objective function $z = \sum_{i=1}^{n} c_i X_i$ subject to m linear constraints $\mathcal{F} = \{F_1, ..., F_m\}$, over n decision variables $\mathcal{X} = \{X_1, ..., X_n\}$ with binary domains $\mathcal{D} = \{D_1, ..., D_n\}$. We use the notation $\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, z \rangle$ to refer to any 0/1 ILP instance.

2.2 AND/OR Search Spaces for 0/1 Integer Linear Programs

The common way of solving 0/1 Integer Linear Programs is by search, namely to instantiate variables one at a time following a static or dynamic variable ordering. In the simplest case, this process defines an OR search tree, whose nodes represent states in the

space of partial assignments. This search space does not capture independencies that appear in the structure of the problem. To remedy this problem an AND/OR search space was recently introduced in the context of general graphical models [4]. The AND/OR search space is defined using a backbone *pseudo-tree* [17].

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.*

We will next specialize the AND/OR search space for a 0/1 ILP which is a special type of a graphical model.

AND/OR Search Trees Given a 0/1 ILP instance $\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, z\rangle$, its interaction 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 assignments $\langle X_i, x_i\rangle$, consistent along the path from the root, $path(x_i) = (\langle X_1, x_1\rangle, ..., \langle X_{i-1}, x_{i-1}\rangle)$. 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. Semantically, 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 tree* 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 of its children are in SOL_{S_T}; (iii) if a nonterminal OR node $n \in S_T$ is in SOL_{S_T} then exactly one of its children is in SOL_{S_T}.

Example 1. For illustration consider the 0/1 ILP with 6 decision variables A, B, C, D, E, F and 4 linear constraints $F_1(A, B, C)$, $F_2(B, C, D)$, $F_3(A, B, E)$, $F_4(A, E, F)$ from Figure 1(a). The objective function to be minimized is $z = 7A + B - 2C + 5D - 6E + 8F$. The pseudo-tree arrangement of the interaction graph, together with the back-arcs (dotted lines) are given in Figure 1(b). Figure 1(c) shows the corresponding AND/OR search tree.

Arc Labels and Node Values The arcs from OR nodes X_i to AND nodes $\langle X_i, x_i\rangle$ in the AND/OR search tree S_T are annotated by *labels* derived from the objective function.

Definition 2 (label). *Given a 0/1 ILP instance with objective function $z = \sum_{i=1}^{n} c_i X_i$ and a corresponding AND/OR search tree S_T, the* label $l(n, m)$ *of the arc from the OR node $n = X_i$ to the AND node $m = \langle X_i, x_i\rangle$ is defined as $l(n, m) = c_i \cdot x_i$.*

Given a labeled AND/OR search tree, each node can be associated with a *value* [4].

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* $v(n) = 0$*; (ii) if* $n = \langle X_i, x_i \rangle$ *is an internal AND node then* $v(n) = \sum_{m \in succ(n)} v(m)$*; (iii) if* $n = X_i$ *is an internal OR node then* $v(n) = min_{m \in succ(n)}(l(n,m) + v(m))$*, where* $succ(n)$ *are the children of* n *in* S_T*.*

It is easy to see that the value $v(n)$ of a node in the AND/OR search tree S_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_T, then $v(n)$ is the minimal cost solution to the initial problem [6].

Clearly, the AND/OR search tree can be traversed to compute each node's value either by a depth-first or best-first search algorithm.

AND/OR Search Graphs The AND/OR search tree may contain nodes that root identical subtrees (in particular, subproblems with identical optimal solutions). These are called *unifiable*. When unifiable nodes are merged, the search tree becomes a graph and its size becomes smaller. Some unifiable nodes can be identified based on their *contexts*.

Definition 4 (context). *Given a 0/1 ILP instance and the corresponding AND/OR search tree* S_T *relative to a pseudo-tree* T*, the* context *of any AND node* $\langle X_i, x_i \rangle \in S_T$*, denoted by* $context(X_i)$*, is defined as the set of ancestors of* X_i *in* T*, including* X_i*, that are connected to descendants of* X_i*.*

It is easy to verify that any two nodes having the same context represent the same subproblem. Therefore, we can solve P_{X_i}, the subproblem rooted at X_i, once and use its optimal solution whenever the same subproblem is encountered again.

The *context-minimal* AND/OR search graph, denoted by G_T, is obtained by merging all the AND nodes that have the same context. It can be shown [4] that the size of the largest context is bounded by the induced width w^* of the interaction graph, extended with the pseudo-tree extra arcs, over the ordering given by the depth-first traversal of T (i.e. induced width of the pseudo-tree). Therefore,

Theorem 1 (complexity). *The complexity of any search algorithm traversing a context-minimal AND/OR search graph is time and space* $O(exp(w^*))$*, where* w^* *is the induced width of the underlying pseudo-tree [4].*

Example 2. Consider the context-minimal AND/OR search graph in Figure 1(d) of the pseudo-tree from Figure 1(b). Its size is far smaller that that of the AND/OR tree from Figure 1(c) (16 nodes vs. 36 nodes). The contexts of the nodes can be read from the pseudo-tree, as follows: $context(A)$ = {A}, $context(B)$ = {B,A}, $context(C)$ = {C,B}, $context(D)$ = {D}, $context(E)$ = {E,A} and $context(F)$ = {F}.

3 Algorithms Exploring the Context-Minimal AND/OR Graph

In this section we introduce two algorithms that explore a context-minimal AND/OR search graph in either a *depth-first* or *best-first* manner for solving optimization problems from the class of 0/1 ILP. First, we present the depth-first AND/OR Branch-and-Bound search algorithm ($\texttt{AOBB}_g$) which extends the 0/1 ILP algorithm presented in [6]

Algorithm 1: $\texttt{AOBB}_g$

Data: A 0/1 ILP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F}, z)$, pseudo-tree T, root s.
Result: Minimal cost solution to $\mathcal{P}$.
1. Create a list OPEN, consisting solely of the start node s. Set $v(s) = \infty$.
2. **until** s is labeled SOLVED, **do**:
   (a) Remove the first node n from OPEN and add it to CLOSED.
   (b) If n is an AND node, then set $v(n) = cache(n)$.
   (c) Try to prune the subtree below n, as follows: if for some ancestor m of n in CLOSED, $f_h(m) \geq v(m)$, then set $v(n) = \infty$ and continue from step (e).
   (d) Expand node n generating all its successor nodes n_i. For each new node n_i compute $h(n_i)$; if n_i is an AND node then set $v(n_i) = 0$, else if n_i is an OR node then set $v(n_i) = \infty$; add n_i on top of OPEN.
   (e) Create a set S. If n has no successors then label n SOLVED and add it to S.
   (f) **until** S is empty, **do**:
      i. Remove the first node m from S.
      ii. Update the value $v(p)$ of the parent p of m as follows:
         A. **if** p is an AND node **then** $v(p) = v(p) + v(m)$.
         B. **if** p is an OR node **then** $v(p) = min(v(p), l(p, m) + v(m))$. Save the AND value $v(m)$ in cache by setting $cache(m) = v(m)$, if $v(m) \neq \infty$.
      iii. Remove m from the successors of p. If p has no successors left, label p SOLVED and add it to S. Remove m from CLOSED.
3. **return** $v(s)$.

for searching AND/OR trees to searching AND/OR graphs. The algorithm specializes recent AND/OR graph search algorithms for general constraint optimization problems described in [1] to the 0/1 ILP case.

3.1 Depth-First AND/OR Branch-and-Bound Search

The AND/OR Branch-and-Bound search algorithm, denoted by $\texttt{AOBB}_g$, that explores the context-minimal AND/OR search graph in a depth-first manner is described in Algorithm 1. Its pruning strategy is similar to that of the Branch-and-Bound algorithm searching AND/OR trees developed in [6]. Specifically, each node n along the path from the root has associated a *static* heuristic function $h(n)$ underestimating $v(n)$ that can be computed efficiently by solving the linear relaxation (i.e. relaxing the integrality restrictions) of the subproblem rooted at n. The algorithm also 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, as described in [6], and is used to prune unpromising portions of the search space that cannot improve the best solution found so far.

$\texttt{AOBB}_g$ is restricted to a static variable ordering determined by the underlying pseudo-tree and explores the context-minimal AND/OR search graph via *full caching*. The algorithm saves previously computed results and retrieves them when the same nodes are encountered again, during search. A simple way of implementing the caching mechanism is to have a *cache table* for each variable X_k recording its context. Specifically, let us assume that the context of X_k is $context(X_k) = \{X_i, ..., X_k\}$. A cache table entry corresponds to a particular instantiation $\{x_i, ..., x_k\}$ of the variables in $context(X_k)$ and records the optimal cost solution to the subproblem P_{X_k}.

However, some tables might never get cache hits. These are called *dead-caches* [18, 1]. In the context-minimal AND/OR search graph, dead-caches appear at nodes that have only one incoming arc. $\texttt{AOBB}_g$ needs to record only nodes that are likely to have

Algorithm 2: $\texttt{AOBF}_g$

Data: A 0/1 ILP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F}, z)$, pseudo-tree T, root s.
Result: Minimal cost solution to $\mathcal{P}$.
1. Create explicit graph G'_T, consisting solely of the start node s. Set $v(s) = h(s)$.
2. **until** s is labeled SOLVED, **do**:
   (a) Compute a *partial solution tree* by tracing down the *marked* arcs in G'_T from s and select any nonterminal tip node n.
   (b) Expand node n and add any new successor node n_i to G'_T. For each new node n_i set $v(n_i) = h(n_i)$. Label SOLVED any of these successors that are terminal nodes.
   (c) Create a set S containing node n.
   (d) **until** S is empty, **do**:
      i. Remove from S a node m such that m has no descendants in G'_T still in S.
      ii. Revise the value $v(m)$ as follows:
         A. **if** m is an AND node **then** $v(m) = \sum_{m_j \in succ(m)} v(m_j)$. If all the successor nodes are labeled SOLVED, then label node m SOLVED.
         B. **if** m is an OR node **then** $v(m) = min_{m_j \in succ(m)}(l(m, m_j) + v(m_j))$ and mark the arc through which this minimum is achieved. If the marked successor is labeled SOLVED, then label m SOLVED.
      iii. If m has been marked SOLVED or if the revised value $v(m)$ is different than the previous one, then add to S all those parents of m such that m is one of their successors through a marked arc.
3. **return** $v(s)$.

additional incoming arcs, and some of these nodes can be determined by inspecting the pseudo-tree. Namely, if the context of a node includes that of its parent, then there is no need to store anything for that node, because it would be a dead-cache. For example, node B in the AND/OR search graph from Figure 1(d) is a dead-cache because its context includes the context of its parent A in the pseudo-tree from Figure 1(b).

If the memory requirements are prohibitive, rather than using full caching, $\texttt{AOBB}_g$ can be modified to use a memory bounded caching scheme that saves only those nodes whose context size can fit in the available memory, as suggested by [1].

3.2 Best-First AND/OR Search

The context-minimal AND/OR search graph can be traversed in a best-first rather than depth-first manner to compute the optimal cost solution to a 0/1 ILP. It is known that under conditions of admissibility and monotonicity of the guiding heuristic function, best-first search algorithms are guaranteed to expand the minimal number of nodes, at the expense of using additional memory [8].

Our best-first AND/OR graph search algorithm, denoted by $\texttt{AOBF}_g$, that traverses the context-minimal AND/OR search graph is described in Algorithm 1. It specializes Nillson's $\texttt{AO}^*$ algorithm [19] to solving 0/1 ILPs and interleaves forward expansion of the best partial solution tree with a cost revision step that updates estimated node values. First, a top-down, graph-growing operation (`step 2.a`) finds the best partial solution tree by tracing down through the marked arcs of the explicit AND/OR search graph G'_T. These previously computed marks indicate the current best partial solution tree from each node in G'_T. One of the nonterminal leaf nodes n of this best partial solution tree is then expanded, and a static heuristic estimate $h(n_i)$ is assigned to its successors (`step 2.b`). The successors of an AND node $n = \langle X_j, x_j \rangle$ are X_j's children in the pseudo-tree, while the successors of an OR node $n = X_j$ correspond to X_j's domain values. Notice that when expanding an OR node, the algorithm does not generate AND

children that are already present in the explicit search graph G'_T. All these identical AND nodes in G'_T are easily recognized based on their contexts.

The second operation in $\texttt{AOBF}_g$ is a bottom-up, cost revision, arc marking, SOLVE-labeling procedure (`step 2.c`). Starting with the node just expanded n, the procedure revises its value $v(n)$ (using the newly computed values of its successors) and marks the outgoing arcs on the estimated best path to terminal nodes. This revised value is then propagated upwards in the graph. The revised cost $v(n)$ is an updated estimate of the cost of an optimal solution to the subproblem rooted at n. If we assume the monotone restriction on h, the algorithm considers only those ancestors that root best partial solution subtrees containing descendants with revised values (`step 2.d.iii`). The optimal cost solution to the initial problem is obtained when the root node s is solved.

The static heuristic function $h(n)$ is obtained by solving the linear relaxation of the subproblem P_n rooted at node n in the search graph, subject to the current variable instantiation of the best partial solution tree. If P_n is infeasible then we assume $h(n) = \infty$. The bottom-up operation of $\texttt{AOBF}_g$ will then propagate this high cost upward, which eliminates any chances that a subtree containing this node might be selected as an estimated best solution subtree.

4 Dynamic Variable Orderings

It is well known that variable selection may influence dramatically search performance. Recent work by [6, 7] showed how several dynamic variable orderings affect depth-first Branch-and-Bound search on AND/OR trees. One extension, called AND/OR Branch-and-Bound with Partial Variable Ordering ($\texttt{AOBB}_t$+PVO) that orders dynamically the variables forming chains in the pseudo-tree, was shown to outperform significantly static AND/OR as well as state-of-the-art OR Branch-and-Bound solvers for general COPs and in particular for 0/1 ILPs [6, 7]. Next, we extend the idea of partial variable ordering to best-first search on AND/OR trees.

Partial Variable Orderings $\texttt{AOBF}_g$ described in the previous section is restricted to a static variable ordering determined by the pseudo-tree arrangement. The mechanism of identifying unifiable AND nodes based solely on their contexts is hard to extend when variables are instantiated in a different order than that dictated by the pseudo-tree, and therefore it cannot be used to accommodate dynamic variable orderings. If we explore the AND/OR search tree we can use dynamic variable orderings while exploring the AND/OR search tree in a best-first manner.

Best-first AND/OR search with Partial Variable Ordering ($\texttt{AOBF}_t$+PVO) traverses an AND/OR search tree by combining the static graph-based problem decomposition given by a pseudo-tree with a dynamic semantic variable selection heuristic. We 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 heuristic (e.g., min reduced cost, min pseudo cost, etc.). Notice that after variables {A,B} are instantiated, the problem decomposes into two independent components that can be solved separately.

5 Experiments

In this section we evaluate empirically the performance of the best-first AND/OR search algorithms on several benchmarks for 0/1 ILPs including problem instances from the MIPLIB library[1], combinatorial auctions, uncapacitated warehouse location problems and MAX-SAT problems. All our experiments were done on a 2.4GHz Pentium IV with 2GB of RAM, running Windows XP.

We consider two classes of best-first search algorithms exploring an AND/OR search tree and using either a static variable ordering (SVO) or a partial variable ordering (PVO). The algorithms are denoted by $\texttt{AOBF}_t$+SVO and $\texttt{AOBF}_t$+PVO, respectively. We also consider two classes of depth-first and best-first search algorithms traversing context-minimal AND/OR search graphs, both restricted to a static variable ordering and denoted by $\texttt{AOBB}_g$+SVO and $\texttt{AOBF}_g$+SVO, respectively. For comparison we include results obtained with two depth-first AND/OR Branch-and-Bound algorithms without caching developed recently in [6] and denoted by $\texttt{AOBB}_t$+SVO and $\texttt{AOBB}_t$+PVO, respectively. The guiding heuristic of the AND/OR search algorithms is computed by solving the linear relaxation of the current subproblem. We used the SIMPLEX implementation from the open-source lp_solve[2] library. The guiding pseudo-tree used by the AND/OR algorithms was constructed using the hypergraph partitioning heuristic described in [6].

For reference, we also report results obtained with the classic depth-first OR Branch-and-Bound algorithm, denoted by BB. BB traverses an OR search tree using linear relaxations to guide the search and is available from the lp_solve library.

The algorithms BB, $\texttt{AOBB}_t$+PVO and $\texttt{AOBF}_t$+PVO used a dynamic semantic variable selection heuristic based on *reduced costs* (i.e. dual values) [2]. Specifically, the next fractional variable to instantiate has the smallest reduced cost. Ties are broken lexicographically.

We report the average effort, as CPU time (in seconds) and number of nodes visited (which is equivalent to the number of times the SIMPLEX routine was called to solve the linear relaxation of the current subproblem), required for proving optimality of the solution. We also 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 best performance points are highlighted.

5.1 MIPLIB

MIPLIB is a library of Mixed Integer Linear Programming instances that is commonly used for benchmarking integer programming algorithms. For our purpose we selected 4 0/1 ILP instances of increasing difficulty. Table 1 reports a summary of the experiment. We see that, overall the best-first AND/OR search algorithms explore the smallest search space, which sometimes translates into significant time savings. For example, on lseu, one of the hardest instances, $\texttt{AOBF}_t$+SVO causes a speedup of 2.5 over $\texttt{AOBB}_t$+SVO, while exploring a search space 3 times smaller. Similarly, $\texttt{AOBF}_g$+SVO

[1] available at http://miplib.zib.de/miplib2003.php

[2] lp_solve 5.5.0.9 is available at http://lpsolve.sourceforge.net/5.5/

Table 1. Results for MIPLIB problem instances.

miplib	n	w^*		BB	AOBB$_t$	AOBF$_t$	AOBB$_g$	AOBF$_g$	AOBB$_t$	AOBF$_t$
	c	h		(lp_solve)	SVO	SVO	SVO	SVO	PVO	PVO
p0033	33	19	time	5.34	0.31	0.27	**0.19**	0.39	0.28	0.33
	15	21	nodes	15,832	438	403	339	281	428	374
p0040	40	19	time	**0.08**	0.11	0.11	0.11	0.09	0.27	0.18
	23	23	nodes	134	113	100	113	100	142	121
p0201	201	120	time	98.21	91.36	**71.62**	90.52	76.05	84.36	91.45
	133	142	nodes	23,742	15,187	10,387	15,130	10,387	9,653	8,261
lseu	89	57	time	282.27	89.04	**35.44**	86.88	36.50	44.85	36.45
	28	68	nodes	386,122	70,322	21,396	63,906	19,692	30,202	18,383

Table 2. Results for combinatorial auction problem instances.

auction	n	w^*		BB	AOBB$_t$	AOBF$_t$	AOBB$_g$	AOBF$_g$	AOBB$_t$	AOBF$_t$
	c	h		(lp_solve)	SVO	SVO	SVO	SVO	PVO	PVO
reg-upv	203	145	time	5.95	7.83	6.82	8.08	6.79	7.02	**3.66**
b200g50	87	162	nodes	658	500	310	500	310	533	189
reg-upv	251	166	time	45.42	19.24	14.92	19.37	15.07	16.59	**8.31**
b250g75	124	190	nodes	3,321	663	333	663	333	620	170
reg-upv	304	173	time	198.07	155.76	90.61	148.34	91.67	125.95	**48.67**
b300g100	157	204	nodes	7,756	2,561	1,084	2,561	1,084	2,617	569
reg-npv	202	140	time	4.41	4.58	3.52	4.56	3.64	4.75	**1.66**
b200g50	88	161	nodes	491	280	158	280	158	367	64
reg-npv	251	160	time	18.04	15.52	10.06	15.39	10.21	15.35	**4.55**
b250g75	120	187	nodes	1,177	593	250	593	250	659	95
reg-npv	302	172	time	185.65	69.81	50.55	69.27	51.24	62.17	**24.14**
b300g100	156	206	nodes	7,131	1,195	537	1,195	537	1,335	237

is 2.4 times faster than AOBB$_g$+SVO, while AOBF$_t$+PVO is only slightly better than AOBB$_t$+PVO. This observation verifies the theory because best-first search is likely to expand the smallest number of nodes at the search frontier having relatively weak heuristic estimates.

5.2 Combinatorial Auctions

In **combinatorial auctions** (CA), 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 [6].

Table 2 shows the results for experiments with 6 classes of moderate size combinatorial auctions from [6]. These auctions were drawn from the `regions` distribution

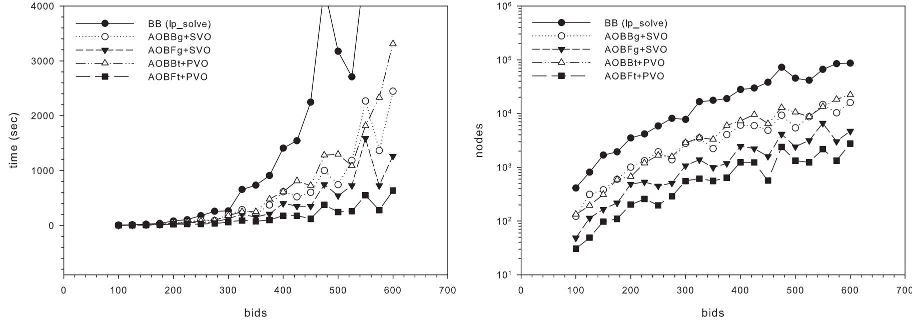


Fig. 2. Results for `regions-upv` auctions with 100 goods and increasing number of bids.

of the CATS 2.0 test suite [14] and simulate the auction of radio spectrum in which a government sells the right to use specific segments of spectrum in different geographical areas. We observe that $\texttt{AOBF}_t\texttt{+PVO}$ is the best performing algorithm, exploring the smallest search space. If we look for example at the 300 bid problem instances from the `reg-npv` distribution, $\texttt{AOBF}_t\texttt{+PVO}$ is on average about 2.5 times faster than the other AND/OR algorithms and the search space explored is about 4 times smaller. When compared with the classic OR Branch-and-Bound algorithm, $\texttt{AOBF}_t\texttt{+PVO}$ causes an even higher speedup, exploring a search space 30 times smaller. Notice that the AND/OR graph search algorithms $\texttt{AOBB}_g\texttt{+SVO}$ and $\texttt{AOBF}_g\texttt{+SVO}$ expanded the same number of nodes as the AND/OR tree search algorithms $\texttt{AOBB}_t\texttt{+SVO}$ and $\texttt{AOBF}_t\texttt{+SVO}$, respectively. This indicates that, for these problem classes, the context-minimal AND/OR search graph is a tree and all cache entries are actually dead.

Figure 2 displays the results for experiments with `regions-upv` auctions having 100 goods and increasing number of bids. Each data point represents an average over 10 random samples. We observe that $\texttt{AOBF}_t\texttt{+PVO}$ is the best performing algorithm and, on some of the hardest instances, it outperforms its competitors with up to one order of magnitude in terms of both CPU time and size of the search space explored. When comparing the best-first versus the depth-first search algorithm traversing the context minimal AND/OR search graph, the savings in the number of nodes caused by $\texttt{AOBF}_g\texttt{+SVO}$ over $\texttt{AOBB}_g\texttt{+SVO}$ do translate into time savings as well, especially when the number of bids increases.

5.3 Uncapacitated Warehouse Location Problems

In the **uncapacitated warehouse location problem** (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. We used the 0/1 ILP formulation from [6].

Table 3. Results for uncapacitated warehouse location problem instances.

uwlp 50x200	n c	w^* h		BB (lp_solve)	AOBB$_t$ SVO	AOBF$_t$ SVO	AOBB$_g$ SVO	AOBF$_g$ SVO	AOBB$_t$ PVO	AOBF$_t$ PVO
uwlp001	10,050	50	time	48.61	69.55	44.39	69.53	42.70	25.63	**20.22**
	10,500	123	nodes	86	62	20	62	20	20	7
uwlp004	10,050	50	time	61.08	46.39	37.58	46.42	36.27	17.47	**15.49**
	10,500	123	nodes	142	46	24	46	24	10	3
uwlp013	10,050	50	time	13693.76	116.19	111.28	116.25	105.72	78.86	**74.53**
	10,500	123	nodes	14,846	44	26	44	26	24	13
uwlp018	10,050	50	time	1477.74	161.03	54.58	161.05	52.41	59.52	**32.33**
	10,500	123	nodes	2,666	146	21	146	21	37	8
uwlp020	10,050	50	time	2179.39	190.77	87.58	190.81	83.70	68.91	**48.33**
	10,500	123	nodes	3,668	138	33	138	33	36	10
uwlp024	10,050	50	time	2177.67	125.85	86.64	125.86	82.27	28.19	**25.89**
	10,500	123	nodes	3,288	71	31	71	31	16	4

Table 3 displays the results obtained on 6 randomly generated UWLP problem instances[3] 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, but having relatively shallow pseudo-trees with depths of 123. We can see that AOBF$_t$+PVO dominates in all test cases, outperforming the classic BB with several orders of magnitude in terms of both running time and size of the search space explored. In uwlp013 for example, one of the hardest instances, AOBF$_t$+PVO causes a speed-up of 186 over the classic OR Branch-and-Bound algorithm, exploring a search tree 1,142 times smaller. When comparing the best-first AND/OR search algorithms with the depth-first AND/OR Branch-and-Bound algorithms we observe only minor time savings. This is because the corresponding AND/OR search spaces are already small enough and the savings in number of nodes caused by the best-first AND/OR search algorithms do not translate into time savings as well. Notice that for this problem class the context minimal AND/OR search graph explored by the AOBB$_g$+SVO and AOBF$_g$+SVO algorithms is in fact a tree and therefore all cache entries are dead.

5.4 MAX-SAT Problems

Given a set of Boolean variables the goal of **maximum satisfiability** (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 as described in [15]. We experimented with problem classes pret and dubois from the SATLIB[4] library, which were previously shown to be difficult for 0/1 ILP solvers [20, 16]. For comparison, we also ran three specialized MAX-SAT solvers: MaxSolver [16], a DPLL-based algorithm that uses a 0/1 non-linear integer formulation of the MAX-SAT problem,

[3] Problem generator from http://www.mpi-sb.mpg.de/units/ag1/projects/benchmarks/UflLib/
[4] http://www.satlib.org/

Table 4. Results for `pret` MAX-SAT problem instances.

pret	n	w^*		MaxSolver	toolbar3	PBS	BB	AOBB$_t$	AOBF$_t$	AOBB$_g$	AOBF$_g$	AOBB$_t$	AOBF$_t$
	c	h					(lp_solve)	SVO	SVO	SVO	SVO	PVO	PVO
pret60-40	60	6	time	9.47	53.89	**0.00**	27208.09	7.88	7.56	7.38	3.58	8.41	8.70
	160	13	nodes		7,297,773	565	4,194,302	1,255	1,202	1,216	568	1,216	1,326
pret60-60	60	6	time	9.48	53.66	**0.00**	27628.52	8.56	8.08	7.30	3.56	8.70	8.31
	160	13	nodes		7,297,773	495	4,194,302	1,259	1,184	1,140	538	1,247	1,206
pret60-75	60	6	time	9.37	53.52	**0.00**	26990.70	6.97	7.38	6.34	3.08	6.80	8.42
	160	13	nodes		7,297,773	543	4,194,302	1,124	1,145	1,067	506	1,089	1,149
pret150-40	150	6	time	-	-	**0.02**	-	95.11	101.78	75.19	19.70	108.84	101.97
	400	15	nodes			2,592		6,625	6,535	5,625	1,379	7,152	6,246
pret150-60	150	6	time	-	-	**0.01**	-	98.88	106.36	78.25	19.75	112.64	102.28
	400	15	nodes			2,873		6,851	6,723	5,813	1,393	7,347	6,375
pret150-75	150	6	time	-	-	**0.02**	-	108.14	98.95	84.97	20.95	115.16	103.03
	400	15	nodes			2,898		7,311	6,282	6,114	1,430	7,452	6,394

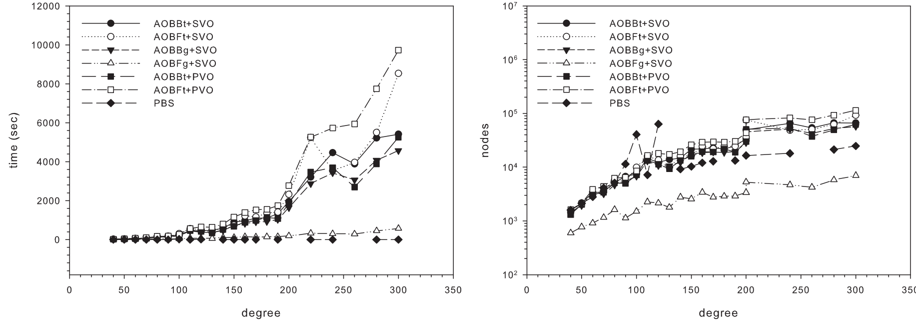


Fig. 3. Results for `dubois` MAX-SAT problem instances.

`toolbar3` [20], a classic OR Branch-and-Bound algorithm that solves MAX-SAT as a Weighted CSP problem, and `PBS` [21], a specialized pseudo-Boolean optimizer.

Table 4 shows the results for experiments with 6 `pret` instances. These are unsatisfiable instances of graph 2-coloring with parity constraints. The size of these problems is relatively small (60 variables with 160 clauses for `pret60` and 150 variables with 400 clauses for `pret150`, respectively). We observe that, for this problem class, AOBF$_g$+SVO is the best performing algorithm amongst the 0/1 ILP solvers. For example, on `pret150-75`, the hardest instance, AOBF$_g$+SVO is 4 times faster than AOBB$_g$+SVO and the search space explored is 6 times smaller. This is due to the problem structure which is partially captured by a very small context with size 6 and a shallow pseudo-tree with depth 13. Overall, PBS offers the best performance on this dataset. However, the search space explored by AOBF$_g$+SVO appears to be the smallest. This indicates that the computational overhead of AOBF$_g$+SVO is due to evaluating its guiding lower bound (i.e., solving the linear relaxation of the current subproblem via SIMPLEX). Notice that BB, `MaxSolver` and `toolbar3` solvers were not able to solve any of the `pret150` instances within a 10 hour time limit.

Figure 3 displays the results for experiments with random `dubois` instances with increasing number of variables. These are hard 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. We can see that $\texttt{AOBF}_g$+SVO takes full advantage of the relatively small context-minimal AND/OR search graph and, on some of the larger instances, it outperforms its 0/1 ILP competitors with up to one order of magnitude in terms of both running time and number of nodes expanded. PBS is again the overall best-performing algorithm, however it fails to solve 4 test instances (e.g., `dubois130`, `dubois180`, `dubois200` and `dubois260`) due to exceeding the memory limit. We observe that in this domain also $\texttt{AOBF}_g$+SVO explores the smallest search space as compared to PBS, but its computational overhead does not pay off in terms of running time. BB, `MaxSolver` and `toolbar3` performed very poorly on this dataset and they were not able to solve any of test instances within a 10 hour time limit. Notice that in some test cases the best-first search algorithms traversing the AND/OR search tree (i.e., $\texttt{AOBF}_t$+SVO, $\texttt{AOBF}_t$+PVO) expand more nodes than their Branch-and-Bound counterparts. We suspect that this is because the guiding LP lower-bound is not monotone which causes the best-first search to expand a non-minimal number of nodes.

6 Conclusion

The contribution of this paper is three-fold. First, we introduced an AND/OR search algorithm that explores a context-minimal AND/OR search graph in a *best-first* manner for solving 0/1 ILPs. Second, we extended the algorithm to incorporate dynamic variable orderings. $\texttt{AOBF}_t$+PVO augments a static pseudo-tree based problem decomposition with a dynamic semantic variable selection heuristic, while exploring an AND/OR search tree in a best-first manner. Third, we adapted the depth-first AND/OR Branch-and-Bound algorithm with full caching to the 0/1 ILP domain. Our empirical evaluation demonstrated on a variety of 0/1 ILP benchmark problems that the best-first AND/OR search algorithms are promising candidate solvers, outperforming the depth-first OR and AND/OR Branch-and-Bound algorithms with several of magnitude in terms of both running time and size of the search space explored.

Our best-first AND/OR search approach leaves room for future improvements, which are likely to make it more efficient in practice. For instance, it can be modified to incorporate *cutting planes* to tighten the linear relaxation of the current subproblem. The space required by the best-first AND/OR search can be enormous, due to the fact that all the nodes generated by the algorithm have to be saved prior to termination. Therefore, the algorithm can be extended to incorporate a memory bounding scheme similar to the one suggested in [22].

Acknowledgments

We would like to thank the anonymous reviewers for commenting on an earlier version of the paper. This work has been partially supported by the NSF grant IIS-0412854.

References

1. R. Marinescu and R. Dechter. Memory intensive branch-and-bound search for graphical models. *In National Conference on Artificial Intelligence (AAAI'06)*, 2006.
2. G. Nemhauser and L. Wolsey. *Integer and combinatorial optimization.* Wiley, 1988.
3. E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.
4. R. Dechter and R. Mateescu. And/or search spaces for graphical models. *Artificial Intelligence*, 2006.
5. R. Marinescu and R. Dechter. And/or branch-and-bound for graphical models. *In International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 224–229, 2005.
6. R. Marinescu and R. Dechter. And/or branch-and-bound search for pure 0/1 integer linear programming problems. *In International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization (CPAIOR'06)*, pages 152–166, 2006.
7. R. Marinescu and R. Dechter. Dynamic orderings for and/or branch-and-bound search in graphical models. *In European Conference on Artificial Intelligence (ECAI'06)*, pages 138–142, 2006.
8. R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of a*. *In Journal of ACM*, 32(3):505–536, 1985.
9. G.B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, 1951.
10. M. Vasquez and J. Hao. A hybrid approach for the 0/1 multidimensional knapsack approach. *In International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 328–333, 2001.
11. W. Shih. A branch-and-bound method for the multiconstraint 0/1 knapsack problem. *Journal of the Operational Research Society*, 30:369–378, 1979.
12. B. Gavish and H. Pirkul. Allocation of data bases and processors in a distributed computing system. *Management of Distributed Data Processing*, 31:215–231, 1982.
13. T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. *In International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 542–547, 1999.
14. 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.
15. S. Joy, J. Mitchell, and B. Borchers. A branch and cut algorithm for max-sat and weighted max-sat. *In Satisfiability Problem: Theory and Applications*, pages 519–536, 1997.
16. Z. Xing and W. Zhang. Efficient strategies for (weighted) maximum satisfiability. *In Constraint Programming (CP'04)*, pages 660–705, 2004.
17. E. Freuder and M. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. *In International Joint Conference on Artificial Intelligence (IJCAI'85)*, pages 1076–1078, 1985.
18. A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.
19. K. Nillson. *Principles of Artificial Intelligence.* Tioga, 1980.
20. S. de Givry, J. Larrosa, and T. Schiex. Solving max-sat as weighted csp. *In Constraint Programming (CP'03)*, 2003.
21. F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Pbs: A backtrack search pseudo-boolean solver. *In Symposium on the Theory and Applications of Satisfiability Testing (SAT'02)*, 2002.
22. P. Chakrabati, S. Ghose, A. Acharya, and S. de Sarkar. Heuristic search in restricted memory. *In Artificial Intelligence*, 3(41):197–221, 1989.