Memory Intensive Branch-and-Bound Search for Graphical Models

Radu Marinescu and **Rina Dechter**
School of Information and Computer Science
University of California, Irvine, CA 92627
{radum,dechter}@ics.uci.edu

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. AND/OR Branch-and-Bound (AOBB) is a new algorithm that explores the AND/OR search tree for solving optimization tasks in graphical models. In this paper we extend the algorithm to explore an AND/OR search *graph* by equipping it with a context-based adaptive caching scheme similar to good and no-good recording. The efficiency of the new graph search algorithm is demonstrated empirically on various benchmarks, including the very challenging ones that arise in genetic linkage analysis.

Introduction

Graphical models such as belief networks or constraint networks are a widely used representation framework for reasoning with probabilistic and deterministic information. 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 the most likely state of a belief network or finding a solution that violates the least number of constraints can be defined within this framework and they are typically tackled with either *search* or *inference* algorithms (Dechter 2003).

The AND/OR search space for graphical models (Dechter & Mateescu 2004) is a new 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.

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 (Marinescu & Dechter 2005). In this paper we improve the AOBB scheme significantly by using *caching* schemes. Namely, we extend the algorithm to explore the AND/OR search *graph* rather than the AND/OR search tree, using a flexible caching mechanism that can adapt to memory limitations.

The caching scheme is based on *contexts* and is similar to good and no-good recording and recent schemes appearing in Recursive Conditioning (Darwiche 2001) and Valued Backtracking (Bacchus, Dalmao, & Pittasi 2003). The efficiency of the proposed search methods also depends on the accuracy of the guiding heuristic function, which is based on the mini-bucket approximation (Dechter & Rish 2003). We focus our empirical evaluation on two common optimization tasks such as solving Weighted CSPs (de Givry *et al.* 2005) and finding the Most Probable Explanation in belief networks (Pearl 1988), and illustrate our results over various benchmarks, including the very challenging ones that arise in the field of genetic linkage analysis.

Section 2 provides background on constraint optimization problems, AND/OR search trees and the AOBB algorithm. In Section 3 we introduce the AND/OR search *graph* and AOBB with caching. In Section 4 we describe two context-based caching schemes. Section 5 gives experimental results and Section 6 concludes.

Preliminaries

Constraint Optimization Problems

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 elimina-

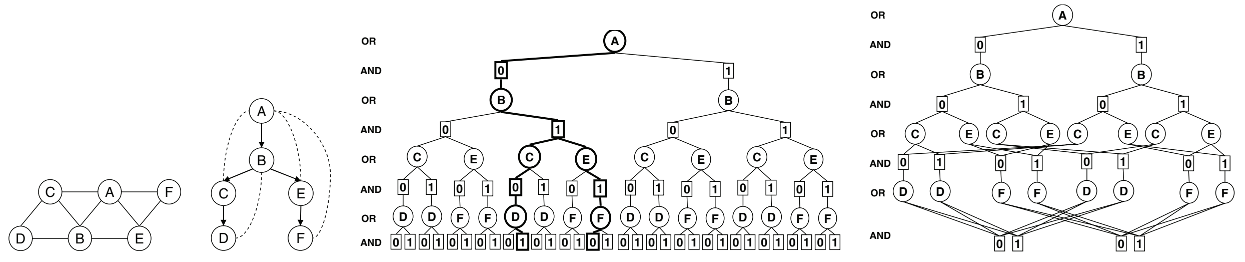

Figure 1: AND/OR Search Spaces

tion operators, yielding a global cost function defined by $f(\mathcal{X}) = min_{\mathcal{X}} \sum_{i=1}^{m} f_i$.

Given a COP instance, its *primal graph* G associates each variable with a node and connects any two nodes whose variables appear in the scope of the same function.

AND/OR Search Trees

The usual way to do search is to instantiate variables in turn (we only consider a static 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 the structure of the underlying graphical model. 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* (Freuder & Quinn 1985; Bayardo & Miranker 1995).

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 $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$, its primal 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 X_i and correspond to the variables. The AND nodes are labeled $\langle X_i, x_i \rangle$ 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 arrangement 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, $path(X_i, x_i) = (\langle X_1, x_1 \rangle, ..., \langle X_{i-1}, x_{i-1} \rangle)$. 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 *Figures 1(a) and 1(b) show a COP instance and its pseudo-tree together with the back-arcs (dotted lines). Figure 1(c) 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 (Dechter & Mateescu 2004). The arcs from X_i to $\langle X_i, x_i \rangle$ are annotated by appropriate *labels* of the functions in $\mathcal{F}$. 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 whose scope includes X_i and is fully assigned along $path(X_i, x_i)$.*

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** ***(Marinescu & Dechter 2005)*** *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 $\mathcal{P}$.*

AND/OR Branch-and-Bound Tree Search

AND/OR Branch-and-Bound (AOBB) was introduced in (Marinescu & Dechter 2005) as a depth-first Branch-and-Bound that explores an AND/OR search tree for solving optimization tasks in graphical models. In the following we review briefly the algorithm.

At any stage during search, a node n along the current path roots a current *partial solution subtree* 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.

AOBB traverses the AND/OR search tree in a depth-first manner. For any node n on the current search path the algorithm calculates a *dynamic heuristic evaluation function* $f_h(n)$, which underestimates $v(n)$. The definition of $f_h(n)$ is based on the portion of the search space below n that has already been explored, as described in (Marinescu & Dechter 2005). The algorithm 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 is terminated below the tip node of the current search path (for more details see (Marinescu & Dechter 2005)).

AND/OR Search Graphs

The AND/OR search tree may contain nodes that root identical subtrees (in particular, their root nodes values are identical). These are called *unifiable*. When unifiable nodes are merged, the search tree becomes a graph and its size becomes smaller. A depth-first search algorithm can explore the AND/OR graph using additional memory. The algorithm can 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*.

DEFINITION **4 (context)** *Given a COP 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 the context of X_i d-separates (Pearl 1988) the subproblem P_{X_i} below X_i from the rest of the network. Namely, it is possible to solve P_{X_i} for any assignment of $context(X_i)$ and record its optimal value, thus avoiding to solve P_{X_i} again for the same assignment. The *context-minimal* AND/OR graph is obtained by merging all the context unifiable AND nodes. The size of the largest context is bounded by the induced width w^* of the primal 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, the time and space complexity of a search algorithm traversing the context-minimal AND/OR graph is $O(exp(w^*))$ (Dechter & Mateescu 2004).

For illustration, consider the context-minimal 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. 54 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}.

AND/OR Branch-and-Bound Graph Search

In this section we extend AOBB to traverse an AND/OR search graph by equipping it with a caching mechanism.

```
function: AOBB_g(st, $\mathcal{X}$, $\mathcal{D}$, $\mathcal{F}$)
1  if $\mathcal{X} = \emptyset$ then return 0;
2  else
3      $X_i \leftarrow$ SelectVar($\mathcal{X}$);
4      $v(X_i) \leftarrow \infty$;
5      foreach $x_i \in D_i$ do
6          $st' \leftarrow st \cup (X_i, x_i)$;
7          $v \leftarrow$ ReadCache($X_i$,$x_i$);
8          if $v \neq NULL$ then
9              $tmp \leftarrow v$+label($X_i$,$x_i$);
10             if $\neg$FindCut($X_i$,$x_i$,$tmp$) then
11                 $v(X_i) \leftarrow min(v(X_i), tmp)$;
12                 continue;
13         end
14         $h(X_i, x_i) \leftarrow$ LB($\mathcal{X}$,$\mathcal{D}$,$\mathcal{F}$);
15         foreach $k = 1..q$ do
16             $h(X_k) \leftarrow$ LB($\mathcal{X}_k$,$\mathcal{D}_k$,$\mathcal{F}_k$);
17         end
18         if $\neg$FindCut($X_i$,$x_i$,$h(X_i, x_i)$) then
19             $v(X_i, x_i) \leftarrow 0$;
20             foreach $k = 1..q$ do
21                 $val \leftarrow$AOBB_g($st'$,$\mathcal{X}_k$,$\mathcal{D}_k$,$\mathcal{F}_k$);
22                 $v(X_i, x_i) \leftarrow v(X_i, x_i) + val$;
23             end
24             WriteCache($X_i$,$v(X_i, x_i)$);
25             $v(X_i, x_i) \leftarrow v(X_i, x_i)$+label($X_i$,$x_i$);
26             $v(X_i) \leftarrow min(v(X_i), v(X_i, x_i))$;
27         end
28     end
29     return $v(X_i)$;
30 end
```

Figure 2: Graph AND/OR Branch-and-Bound.

Figure 2 shows the graph AOBB$_g$ algorithm. The following notation is used: $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ is the problem with which the procedure is called and st is the current partial solution subtree being explored. The algorithm assumes that variables are selected according to a pseudo-tree arrangement.

If the set $\mathcal{X}$ is empty, then the result is trivially computed (line 1). Else, AOBB$_g$ 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 $v(X_i)$. The algorithm attempts to retrieve the results cached at the AND nodes (line 7). If a valid cache entry v is found for the current AND node $\langle X_i, x_i\rangle$ then the OR value $v(X_i)$ is updated (line 11) and the search continues with the next value in X_i's domain. Otherwise, the problem is decomposed into a set of q independent subproblems, 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 in the search graph.

When expanding the AND node $\langle X_i, x_i\rangle$, AOBB$_g$ successively updates the *dynamic heuristic 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 sequentially solved (line 21) and the solutions are accumulated by the AND value

$v(X_i, x_i)$ (line 23). 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 31).

Caching Schemes

In this section we present two caching schemes that can adapt to the current memory limitations. They are based on *contexts*, which are pre-computed from the pseudo-tree and use a parameter called *cache bound* (or j-bound) to control the amount of memory used for storing unifiable nodes.

Naive Caching

The first scheme, called *naive caching* and denoted by AOBB+C(j), stores nodes at the variables whose context size is smaller than or equal to the cache bound j. It is easy to see that when j equals the induced width of the pseudo-tree the algorithm explores the context-minimal AND/OR graph.

A straightforward way of implementing the caching scheme is to have a *cache table* for each variable X_k recording the context. Specifically, lets assume that the context of X_k is $context(X_k) = \{X_i, ..., X_k\}$ and $|context(X_k)| \leq j$. 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* (Darwiche 2001). In the AND/OR search graph, dead-caches appear at nodes that have only one incoming arc. AOBB+C(j) needs to record only nodes that are likely to have additional incoming arcs, and 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).

Adaptive Caching

The second scheme, called *adaptive caching* and denoted by AOBB+AC(j), is inspired by the AND/OR cutset conditioning scheme and was first explored in (Mateescu & Dechter 2005). It extends the naive scheme by allowing caching even at nodes with contexts larger than the given cache bound, based on *adjusted contexts*.

We will illustrate the idea with an example. Consider the node X_k with $context(X_k) = \{X_i, ..., X_k\}$, where $|context(X_k)| > j$. During search, when variables $\{X_i, ..., X_{k-j}\}$ are assigned, they can be viewed as part of a w-*cutset* (Pearl 1988). The w-cutset method consists of enumerating all the possible instantiations of a subset of variables (i.e. cutset), and for each one solving the remaining easier subproblem within w-bounded space restrictions.

Therefore, once variables $\{X_i, ..., X_{k-j}\}$ are instantiated, the problem rooted at X_{k-j+1} can be solved as a simplified subproblem from the cutset method. In the subproblem, conditioned on the values $\{x_i, ..., x_{k-j}\}$, $context(X_k)$ is $\{X_{k-j+1}, ..., X_k\}$ (we call this the *adjusted context* of X_k), so it can be stored within the j-bounded space restrictions. However, when AOBB+AC(j) retracts to X_{k-j} or above, all the nodes cached at variable X_k need to be discarded.

This caching scheme requires only a linear increase in additional memory, compared to AOBB+C(j), but it has the potential of exponential time savings. Specifically, for solving the subproblem rooted by X_k in the pseudo-tree, AOBB+AC(j) requires $O(exp(m))$ time and $O(exp(j))$ space, whereas AOBB+C(j) needs $O(exp(h_k))$ time and linear space, where h_k is the depth of the subtree rooted at X_k in the pseudo-tree, $m = |context(X_k)|$ and $m \leq h_k$.

Additional dead-caches in the adaptive scheme can also be identified by inspecting the pseudo-tree. Consider the node X_k from the previous example and let $anc(X_k)$ be the ancestors of X_k in the pseudo-tree between X_k and X_{k-j}, including X_k. If $anc(X_k)$ contains only the variables in the adjusted context of X_k then X_k is a dead-cache.

Experiments

In this section we evaluate empirically the performance of the AND/OR Branch-and-Bound graph search algorithm on two optimization tasks: solving Weighted CSPs and finding the Most Probable Explanation (MPE) in belief networks[1].

Weighted CSP (de Givry *et al.* 2005) extends the classic CSP formalism with so-called *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.

A *Belief Network* (Pearl 1988) provides a formalism for reasoning under conditions of uncertainty by representing a joint probability distribution over the variables of interest via a directed acyclic graph. A function of the model encodes the *conditional probability distribution* (CPT) of a variable given its parents in the graph. The MPE problem is the task of finding a complete assignment with maximum probability that is consistent with the evidence. It is equivalent to solving a COP instance with *multiplication* and *maximization* as the combination and elimination operators.

We consider two classes of AND/OR Branch-and-Bound graph search algorithms guided by the pre-compiled mini-bucket heuristics (Marinescu & Dechter 2005) and using either *naive* or *adaptive* caching schemes. They are denoted by AOMB+C(i,j) and AOMB+AC(i,j), respectively. The parameters i and j denote the mini-bucket i-bound (which controls the accuracy of the heuristic) and the cache bound. The pseudo-trees were generated using the min-fill heuristic, as described in (Marinescu & Dechter 2005).

We report the average effort as CPU time (in seconds) and number of nodes visited, required for proving optimality of the solution, the induced width (w*) and depth of the pseudo-tree (h) obtained for the test instances. The best performance points are highlighted. For comparison, we also report results obtained with the tree version of the algorithms denoted by AOMB(i). The latter was shown to outperform significantly the OR Branch-and-Bound version (BBMB) in various domains (Marinescu & Dechter 2005).

[1]All our experiments were done on a 2.4GHz Pentium IV with 2GB of RAM, running Windows XP.

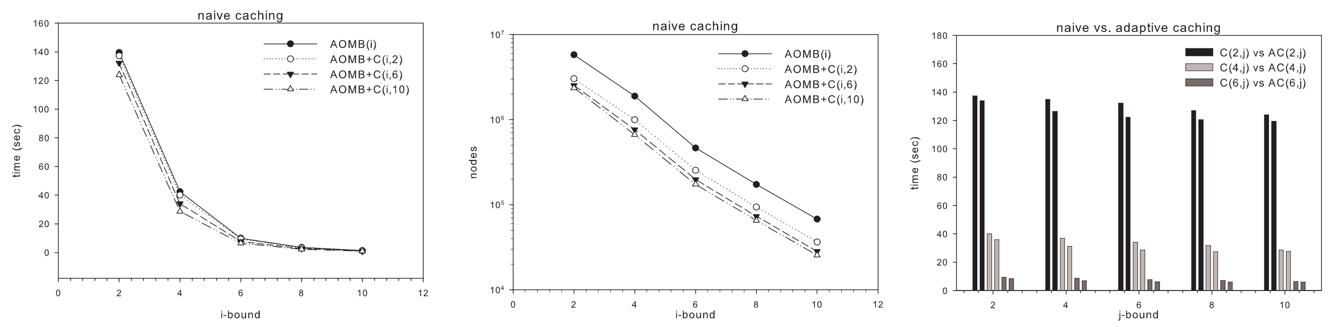

Figure 3: Time in seconds and nodes visited to prove optimality for random belief networks with 120 variables.

spot	(w*, h)	MEDAC	AOEDAC	(i, j)	AOMB(i,j)		
					j=0	C(j)	AC(j)
29	(14, 42)	0.41	1.91	(8, 6)	1.03	**1.00**	1.06
54	(11, 33)	0.06	1.13	(8, 8)	0.12	**0.06**	**0.06**
404	(19, 42)	0.03	2.50	(8, 6)	0.03	**0.02**	**0.02**
408	(12, 27)	0.39	8.50	(6, 6)	0.05	**0.03**	**0.03**
503	(9, 39)	11.70	2.78	(8, 8)	0.10	0.05	**0.03**
505	(23, 74)	4,010	75.43	(16, 16)	896.8	23.2	**23.1**

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

ped	(w*, h)	VEC	SUPER LINK	(i, j)	AOMB(i,j)		
					j=0	C(j)	AC(j)
1	(15, 61)	24.62	131.3	(10, 10)	0.609	0.249	**0.218**
20	(24, 69)	1,304	**12.44**	(16, 16)	480.2	182.0	192.0
23	(23, 38)	1,144	6,809	(16, 18)	16.60	11.33	**11.29**
30	(26, 51)	26,719	28,740	(20, 22)	61.57	38.85	**38.81**
38	(17, 59)	15,860	**62.18**	(12, 12)	1,212	104.4	124.7
50	(18, 58)	85,637	716.6	(10, 12)	83.52	**29.72**	36.41

Table 2: Time in seconds to prove optimality for pedigrees.

Random Belief Networks

We have generated a class of random belief networks using the parametric model (n, d, c, p) proposed in (Kask & Dechter 2001). Figure 3 reports the average time results in seconds and number of nodes visited for 20 random instances of a network with n=120 variables, domain size d=2, c=110 CPTs and p=2 parents per CPT. The average induced width and pseudo-tree depth were 20 and 32, respectively. The i-bound of the mini-bucket heuristic ranged between 2 and 10, and we chose three caching levels as follows: *low* (j=2), *medium* (j=6) and *high* (j=10).

We observe that naive caching improves when the heuristic is relatively weak (corresponding to small i-bounds). As the i-bound increases and the heuristics become strong enough to cut the search space substantially, the added savings in the number of nodes caused by caching do not translate into time savings as well. In Figure 3(c) we compare the two caching schemes, in terms of CPU time, for different values of the i-bound ($i \in${2,4,6}). We observe only a minor improvement of the adaptive scheme over the naive one, more noticeable for small i and j-bounds.

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 (Bensana, Lemaitre, & Verfaillie 1999) 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.

Table 1 shows a summary of the results obtained for 6 scheduling problems. In addition, we consider MEDAC and AOEDAC which are the OR and AND/OR Branch-and-Bound algorithms maintaining Existential Directional Arc Consistency (EDAC) (de Givry *et al.* 2005) and are not restricted to a static variable ordering. We experimented with a wide range of values for the i and j bounds, but we report only the (i,j) combination for which we obtained the best results. Both `AOMB+C`(i,j) and `AOMB+AC`(i,j) are the best performing algorithms in this domain. The impact of the caching schemes is minor for most of the test instances. This is due the accuracy of the heuristic estimates which prune the search space very effectively. In `505` however, the hardest instance, `AOMB+AC`(16,16) improves dramatically the performance causing a speedup of 39 over `AOMB`(16).

Genetic Linkage Analysis

The *maximum likelihood haplotype* problem in genetic linkage analysis is the task of finding a joint haplotype configuration for all members of the pedigree which maximizes the probability of data. It has been shown to be equivalent to finding the MPE of a belief network which represents the pedigree data (Fishelson & Geiger 2002).

Table 2 displays a summary of the results obtained for 6 hard linkage analysis networks[2]. For comparison, we include results obtained with VEC and SUPERLINK `v1.5`. SUPERLINK is currently the most efficient solver for genetic linkage analysis, is dedicated to this domain, uses a combination of variable elimination and conditioning, and takes advantage of the determinism in the network. VEC is our implementation of the elimination/conditioning hybrid and is not sensitive to determinism. As both algorithms use non-deterministic algorithms for computing the elimination order, their running time may vary significantly from one run

[2]http://bioinfo.cs.technion.ac.il/superlink/

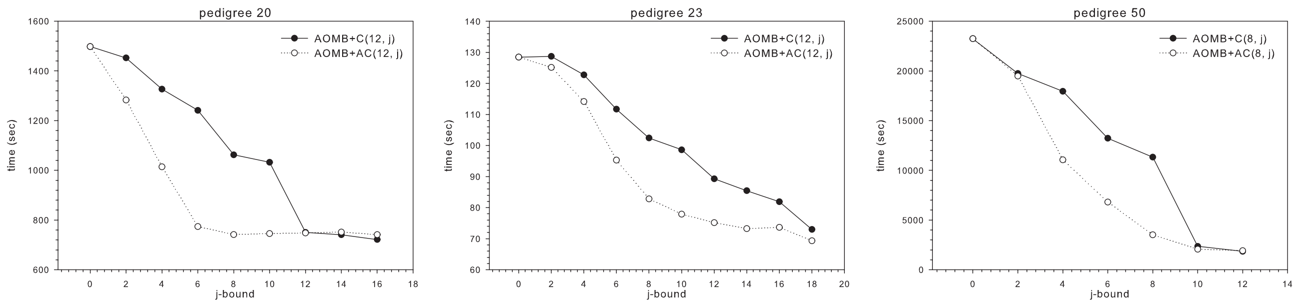

Figure 4: Detailed time results in seconds comparing the naive vs. adaptive caching for genetic linkage analysis.

to the next. We therefore report an average over 5 runs.

We observe that AOMB+C(i,j) and AOMB+AC(i,j) are the best performing algorithms in this domain. The time savings caused by both naive and adaptive caching schemes are significant and in some cases the differences add up to several orders of magnitude over both VEC and SUPERLINK (e.g. `ped-23`, `ped-50`). Figure 4 provides an alternative view comparing the two caching schemes, in terms of CPU time, for a smaller i-bound of the mini-bucket heuristic. We notice that adaptive caching improves significantly over the naive scheme especially for relatively small j-bounds. This may be important because small j-bounds mean restricted space. At large j-bounds the two schemes are identical.

In summary, the effect of caching (either naive or adaptive) is more prominent for relatively weak guiding heuristics estimates. The merit of adaptive caching over naive one is evident when the j-bound is much smaller than the induced width and there is a relatively small number of dead-caches. This translates sometimes into impressive time savings for the Branch-and-Bound algorithms.

Conclusion

In this paper we extended the AND/OR Branch-and-Bound algorithm to traversing an AND/OR search graph rather than an AND/OR search tree by equipping it with an efficient caching mechanism. We investigated two flexible context-based caching schemes that can adapt to the current memory restrictions. The efficiency of the new AND/OR Branch-and-Bound graph search algorithms is demonstrated empirically on various benchmarks including the very challenging ones from the field of genetic linkage analysis.

Related Work: AOBB graph search is related to the Branch-and-Bound method proposed by (Kanal & Kumar 1988) for acyclic AND/OR graphs and game trees. BTD developed in (Jegou & Terrioux 2004) can also be interpreted as an AND/OR graph search algorithm with a caching mechanism based on the separators of the guiding tree-decomposition. When compared with BTD, AOBB$_g$ with naive/adaptive caching uses parameterized cache bounds and can adapt to various memory needs, unlike BTD which must use space exponential in the separator size. Only under full caching BTD and AOBB$_g$ are computationally identical because the context of a node in the AND/OR search space and the separator in the tree-decomposition used by BTD are equivalent.

Acknowledgments

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

References

Bacchus, F.; Dalmao, S.; and Pittasi, T. 2003. Value elimination: Bayesian inference via backtracking search. In *UAI*, 20–28.

Bayardo, R., and Miranker, D. 1995. On the space-time trade-off in solving constraint satisfaction problems. In *IJCAI*, 558–562.

Bensana, E.; Lemaitre, M.; and Verfaillie, G. 1999. Earth observation satellite management. *Constraints* 4(3):293–299.

Darwiche, A. 2001. Recursive conditioning. *Artificial Intelligence* 126(1-2):5–41.

de Givry, S.; Heras, F.; Larrosa, J.; and Zytnicki, M. 2005. Existential arc consistency: getting closer to full arc consistency in weighted csps. In *IJCAI*, 84–89.

Dechter, R., and Mateescu, R. 2004. Mixtures of deterministic-probabilistic networks. In *UAI*, 120–129.

Dechter, R., and Rish, I. 2003. Mini-buckets: A general scheme for approximating inference. *J. ACM*.

Dechter, R. 2003. *Constraint Processing*. MIT Press.

Fishelson, M., and Geiger, D. 2002. Exact genetic linkage computations for general pedigrees. *Bioinformatics*.

Freuder, E., and Quinn, M. 1985. Taking advantage of stable sets of variables in constraint satisfaction problems. In *IJCAI*, 1076–1078.

Jegou, P., and Terrioux, C. 2004. Decomposition and good recording for solving max-csps. In *ECAI*, 196–200.

Kanal, L., and Kumar, V. 1988. *Search in artificial intelligence*. Springer-Verlag.

Kask, K., and Dechter, R. 2001. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence* 129:91–131.

Marinescu, R., and Dechter, R. 2005. And/or branch-and-bound for graphical models. In *IJCAI*, 224–229.

Mateescu, R., and Dechter, R. 2005. And/or cutset conditioning. In *IJCAI*, 230–235.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan-Kaufmann.