Advances in AND/OR Branch-and-Bound Search for Constraint Optimization

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* 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. In [1] we introduced a linear space AND/OR Branch-and-Bound (AOBB) search scheme that explores the AND/OR search tree for solving optimization tasks. In this paper we extend the algorithm by equipping it with a context-based adaptive caching scheme similar to good and nogood recording, thus it explores an AND/OR graph rather than the AND/OR tree. We also improve the algorithm by using a new heuristic for generating close to optimal height pseudo-trees, based on a well known recursive decomposition of the hypergraph representation. We illustrate our results using a number of benchmark networks, including the very challenging ones that arise in genetic linkage analysis.

1 Introduction

Graphical models such as Bayesian 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. Optimization tasks such as finding the most likely state of a Bayesian network or finding a solution that violates the least number of constraints in a constraint network, are typically tackled with either *search* or *inference* algorithms. Search methods (e.g. depth-first Branch-and-Bound, best-first search) are time exponential in the number of variables and can operate in polynomial space. Inference algorithms (e.g. variable elimination, tree-clustering) are time and space exponential in a topological parameter called *tree width*. If the tree width is large, the high space complexity makes the latter methods impractical in many cases.

The AND/OR search space for graphical models [2] 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.

In [1] we presented a linear space Branch-and-Bound scheme that explores the AND/OR search tree for solving optimization tasks in graphical models, called AOBB.

In this paper we improve the AOBB scheme significantly by using caching schemes. Namely, we extend the algorithm to explore the AND/OR graph rather than the AND/OR 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 nogood recording and recent schemes appearing in Recursive Conditioning and Valued Backtracking [3–5]. We also introduce a new heuristic for generating close to optimal height pseudo-trees based on the recursive decomposition of the problem's hypergraph representation. A similar idea was already exploited in [4] for constructing low-width decomposition trees. 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 or maintaining soft arc-consistency. We focus our empirical evaluation on two common optimization tasks such as solving Weighted CSPs [6] and finding the Most Probable Explanation in Bayesian networks [7], and illustrate our results over a variety of benchmark networks, including the very challenging ones that arise in the field of genetic linkage analysis.

2 Background

2.1 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 $\Downarrow$ can be defined using the semi-ring framework [6], but in this paper we assume that: $\otimes_i f_i$ 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, 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 (hard or soft) constraint.

2.2 AND/OR Search Spaces

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, 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

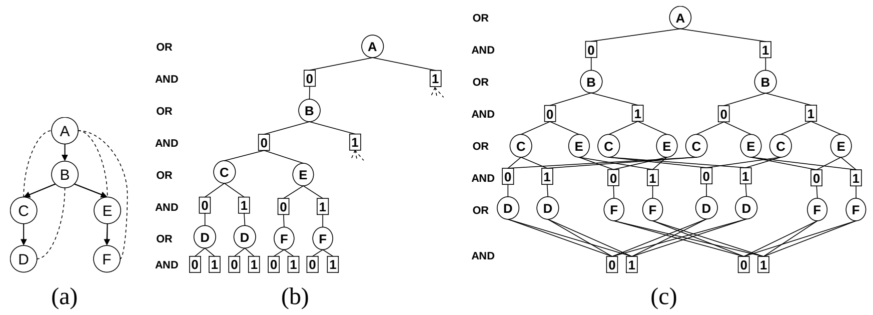


Fig. 1. The AND/OR search space.

AND states into the search space can capture the structure decomposing the problem into independent subproblems by conditioning on values [8, 2]. The AND/OR search space is defined using a backbone *pseudo-tree*.

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.*

AND/OR Search Trees 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, $path(x_i) = (\langle X_1, x_1 \rangle, ..., \langle X_{i-1}, x_{i-1} \rangle)$. 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.

Example 1. Figure 1(a) shows the pseudo-tree arrangement of a primal graph of a COP instance, together with the back-arcs (dotted lines). Figure 1(b) shows a partial AND/OR search tree based on the pseudo-tree, for bi-valued variables.

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 operate in linear time. The arcs from X_i to $\langle X_i, x_i \rangle$ are annotated by appropriate *labels* of the cost functions in $\mathcal{F}$. The nodes in S_T can be associated with *values*, accumulating the result of the computation resulted from the subtree below.

Definition 2 (label). *The* label $l(X_i, \langle X_i, x_i \rangle)$ *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 $path(x_i)$.*

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 $v(n) = l(X_i, \langle X_i, x_i \rangle)$; (ii) if $n = \langle X_i, x_i \rangle$ is an internal AND node then $v(n) = l(X_i, \langle X_i, x_i \rangle) + \sum_{n' \in succ(n)} v(n')$; (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 $\mathcal{P}$.*

AND/OR Search Graphs The AND/OR search tree may contain nodes that root identical subtrees. 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 $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ 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 the induced pseudo-tree, including X_i, that are connected to descendants of X_i.*

It is easy to verify that the context of X_i d-separates [7] the subproblem below X_i from the rest of the network. The *context-minimal* AND/OR graph is obtained by merging all the context unifiable AND nodes. For illustration, consider the context-minimal graph in Figure 1(c) of the pseudo-tree from Figure 1(a). The contexts of the nodes can be read from the pseudo-tree, as follows: $context(A) = \{A\}$, $context(B) = \{B, A\}$, $context(C) = \{C, B, A\}$, $context(D) = \{D\}$, $context(E) = \{E, B, A\}$ and $context(F) = \{F\}$ (for more information see [2]).

3 AND/OR Branch-and-Bound Search

AND/OR Branch-and-Bound (AOBB) was recently introduced in [1] as a depth-first Branch-and-Bound that explores an AND/OR search tree for solving optimization tasks in graphical models. Our empirical evaluation demonstrated clearly the improved performance of the AND/OR tree search over the traditional OR tree search. In this section we move from searching the AND/OR tree to searching AND/OR graphs. The new algorithm, denoted here by AOBB(j), augments AOBB with a flexible context-based caching scheme that stores the results in a cache after the first computation and retrieves them when the same nodes are encountered again.

3.1 Caching Schemes

Traversing an AND/OR search graph requires caching some nodes during search and the ability to recognize unifiable nodes. The caching scheme is based on *contexts*, which are precomputed from the pseudo-tree. As it was mentioned earlier, the context of an AND node $\langle X_i, x_i \rangle$ is the set of ancestors of X_i in the induced pseudo-tree, including X_i, that are connected to descendants of X_i. Algorithm AOBB(j) stores nodes at variables whose context size is smaller than or equal to j (called cache bound or j-bound). It is easy to see that when j equals the induced width of the pseudo-tree the algorithm explores the minimal context AND/OR graph.

This rather straightforward scheme can be further improved. The second caching scheme is inspired by the cutset conditioning ideas from [9]. Lets assume the context of a node X_k is $context(X_k) = \{X_1, ..., X_k\}$, where $|context(X_k)| > j$. During the search, when variables $\{X_1, ..., X_{k-j}\}$ are assigned, they can be viewed as a cutset. Therefore, the problem rooted at X_{k-j+1} can be solved in isolation, once variables $\{X_1, ..., X_{k-j}\}$ are assigned. In the subproblem, conditioned on the values $\{x_1, ..., x_{k-j}\}$, $context(X_k)$ is $\{X_{k-j+1}, ..., X_k\}$, so it can be stored within the j-bounded space restrictions. However, when AOBB(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.

The usual way of caching is to have a table for each variable, called *cache table*, which records the context. However, some tables might never get cache hits. We call these *dead-caches*. In the AND/OR search graph, dead-caches appear at nodes that have only one incoming arc. AOBB(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 illustration, consider the AND/OR search graph from Figure 1(c). Node B is a dead-cache because its context includes the context of node A, which is its parent in the pseudo-tree.

3.2 Lower Bounds on Partial Trees

At any stage during search, any node n along the current path roots a current *partial solution subtree*, denoted by $G_{sol}(n)$, to the corresponding subproblem. By the nature of the search process, $G_{sol}(n)$ must be connected, must contain its root n and will have a *frontier* containing all those nodes that were generated but not yet expanded. The leaves of $G_{sol}(n)$ are called *tip* nodes. Furthermore, we assume that there exists a *static* heuristic evaluation 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 graph G_T, the *active path* $\mathcal{AP}(t)$ is the path of assignments from the root of G_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. We can define now a *dynamic heuristic function* of a node n relative to $\mathcal{APT}(n)$, as follows.

```
ALGORITHM: AOBB($j, \mathcal{P}, T$)
Input: A COP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F}, +, min)$, pseudo-tree $T$, root $X_0$, cache bound $j$.
Output: Minimal cost solution to $\mathcal{P}$.
(1)  Initialize OPEN by adding OR node $X_0$ to it; PATH $\leftarrow \phi$;
     Initialize cache tables for every variable $X_i$ such that $|context(X_i)| \leq j$;
(2)  if (OPEN == $\phi$)
         return $v(X_0)$;
     Remove the first node $n$ in OPEN; Add $n$ to PATH;
(3)  Retrieve cached values as follows:
     if ($n$ is AND node, denote $n = \langle X_i, x_i \rangle$)
         if ($|context(X_i)| \leq j$)
             $A \leftarrow \{f_j \mid f_j \in \mathcal{F} \wedge (X_i \in var(f_j)) \wedge (var(f_j) \subseteq$ PATH $)\}$;
             $l(X_i, \langle X_i, x_i \rangle) \leftarrow \sum_A f_j$;
             $v(n) \leftarrow cache(X_i, x_i)$;
             goto step(5);
(4)  Try to prune the subtree below $n$ as follows:
     foreach $m \in PATH$, where $m$ is an ancestor of $n$
         if ($f_h(m) \geq ub(m)$)
             $v(n) \leftarrow \infty$; (dead-end)
             goto step (4);
     Expand $n$ generating all its successors as follows:
     $succ(n) \leftarrow \phi$;
     if ($n$ is OR node, denote $n = X_i$)
         $v(n) \leftarrow \infty$;
         foreach value $x_i \in D_i$
             $h(\langle X_i, x_i \rangle) \leftarrow$ LB($X_i, x_i$);
             $succ(n) \leftarrow succ(n) \cup \{\langle X_i, x_i \rangle\}$;
     else ($n$ is AND node, denote $n = \langle X_i, x_i \rangle$)
         $A \leftarrow \{f_j \mid f_j \in \mathcal{F} \wedge (X_i \in var(f_j)) \wedge (var(f_j) \subseteq$ PATH $)\}$;
         $v(n) \leftarrow 0$; $l(X_i, \langle X_i, x_i \rangle) \leftarrow \sum_A f_j$;
         foreach variable $Y \in ch_T(X_i)$
             $h(Y) \leftarrow$ LB($Y$);
             $succ(n) \leftarrow \{Y\}$;
     Add $succ(n)$ on top of OPEN;
(5)  while $succ(n) == \phi$
         if ($n$ is OR node)
             $v(Parent(n)) \leftarrow v(Parent(n)) + v(n)$;
         else ($n$ is AND node)
             $cache(X_i, x_i) \leftarrow v(n)$;
             $v(n) \leftarrow v(n) + l(X_i, \langle X_i, a \rangle)$;
             $v(Parent(n)) \leftarrow min(v(Parent(n)), v(n))$;
         $succ(Parent(n)) \leftarrow succ(Parent(n)) - \{n\}$;
         PATH $\leftarrow$ PATH – $\{n\}$;
         $n \leftarrow Last$(PATH);
(6)  goto step (2);
```

Fig. 2. AOBB(j): AND/OR Branch-and-Bound graph search.

Definition 5 (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 $f_h(n) = max(h(n), l(X_i, \langle X_i, x_i \rangle) + \sum_{i=1}^{k} f_h(m_i))$; (iii) if $n = X_i$ is an OR node, having an AND child m, then $f_h(n) = max(h(n), f_h(m))$.*

We can show that:

Theorem 1. *(1) $f_h(n)$ is a* lower bound *on the minimal cost solution to the subproblem rooted at n, namely $f_h(n) \leq v(n)$; (2) $f_h(n) \geq h(n)$, namely the dynamic heuristic function is tighter than the static one.*

3.3 AND/OR Branch-and-Bound with Caching

A search algorithm traversing the AND/OR search space can calculate a *lower bound* on $v(n)$ of a node n on the active path, by using $f_h(n)$. It can also compute an *upper bound* on $v(n)$, based on the portion of the search space below n that has already been explored. The upper bound $ub(n)$ on $v(n)$ is the current minimal cost solution subtree rooted at n.

The depth-first *AND/OR Branch-and-Bound* graph search algorithm with j-bounded caching is described in Figure 2. A list called OPEN simulates the recursion stack. The list PATH maintains the current assignment on the active path. $Parent(n)$ refers to the predecessor of n in the AND/OR search graph, $succ$ denotes the set of successors of a node in the AND/OR search graph and $ch_T(X_i)$ denotes the children of variable X_i in the pseudo-tree T. Procedure $\mathrm{LB}(n)$ computes the static heuristic estimate $h(n)$ of $v(n)$ for any node n.

In the initialization step, AOBB(j) computes the context of every variable. A cache table is created for every context whose size is less than or equal to the cache bound j. In Step (3), the algorithm attempts to retrieve the results cached at the AND nodes. If a valid cache entry α is found for node $n = \langle X_i, x_i \rangle$, namely the subproblem rooted at n has already been solved for the current instantiation of the variables in $context(X_i)$, then $v(n)$ is set to α and the search continues with Step (4), thus avoiding n's expansion.

Step (4) is where the search goes forward and expands alternating levels of OR and AND nodes. Upon the expansion of n, the algorithm successively updates the *lower bound function* $f_h(m)$ for every ancestor m of n along the active path, and prunes the subgraph below n if, for some m, $f_h(m) \geq ub(m)$.

Step (5) is where the value functions are propagated backward. This is triggered when a node has an empty set of successors and it typically happens when the node's descendants are all evaluated.

Theorem 2. *AOBB(j) is sound and complete for COP.*

4 Heuristics

In this section we describe briefly several schemes for generating static heuristic estimates $h(n)$, based on bounded inference and soft arc-consistency.

4.1 Mini-Bucket Heuristics

In this section we briefly describe two general schemes for generating heuristic estimates that can guide Branch-and-Bound search, and which are based on the Mini-Bucket approximation. Mini-Bucket Elimination (MBE) [10] is an approximation algorithm designed to avoid the high time and space complexity of Bucket Elimination (BE) [11], by partitioning large buckets into smaller subsets, called *mini buckets*, each containing at most i (called i-bound) distinct variables, and which are processed independently. The heuristics generators are therefore parameterized by the Mini-Bucket i-bound, thus allowing for a controllable trade-off between heuristic strength and its overhead.

Static Mini-Bucket Heuristics (sMB) In the past, [12] showed that the intermediate functions generated by the Mini-Bucket algorithm MBE(i) can be used to compute a heuristic function, that underestimates the minimal cost extension of the current partial assignment in a regular OR search tree. In [1] we extended this idea to AND/OR search spaces.

Dynamic Mini-Bucket Heuristics (dMB) The dynamic version of the mini-bucket heuristics has been recently proposed in [1] for both OR and AND/OR search spaces. The heuristic lower-bound estimate is computed by the Mini-Bucket algorithm MBE(i), at each node n in the search space, restricted to the subproblem rooted at n and subject to the current partial instantiation (for more details see [1]).

4.2 Directional Arc-Consistency Heuristics

Maintaining full directional arc-consistency (FDAC) [13] and the more recent existential directional arc-consistency (EDAC) [14] provide a powerful mechanism for generating high quality lower bound heuristic estimates of the minimal cost extension of any partial assignment in a regular OR search tree. In the context of AND/OR search spaces we showed in [1] that it is possible to maintain arc-consistency separately, on independent components rooted at AND nodes, thus computing local lower-bounds on the minimal cost solutions to the respective subproblems.

5 Finding a Pseudo-Tree

The performance of AND/OR tree/graph search algorithms is influenced by the quality of the pseudo-tree. Finding the minimal depth/context pseudo-tree is a hard problem [8, 15]. In the following we describe two heuristics for generating pseudo-trees with relatively small heights/contexts.

5.1 Min-Fill Heuristic

Min-Fill [16] is one of the best and most widely used heuristics for creating small induced width elimination orders. An ordering is generated by placing the variable with the smallest *fill set* (i.e. number of induced edges that need be added to fully connect the neighbors of a node) at the end of the ordering, connecting all of its neighbors and then removing the variable from the graph. The process continues until all variables have been eliminated. Once an elimination order is given, the pseudo-tree can be extracted as a depth-first traversal of the min-fill induced graph, starting with the variable that initiated the ordering, always preferring as successor of a node the earliest adjacent node in the induced graph. An ordering uniquely determines a pseudo-tree. This approach was first used by [15].

5.2 Hypergraph Separator Decomposition

An alternative heuristic for generating a low height balanced pseudo-tree arrangement is based on recursive decomposition. Given a COP instance $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F})$ we convert it into a *hypergraph* $\mathcal{H} = (V, E)$ where each constraint in $\mathcal{F}$ is a vertex $v_i \in V$ and each variable in $\mathcal{X}$ is an edge $e_j \in E$ connecting all the constraints in which it appears.

Definition 6 (separators). *Given a hypergraph $\mathcal{H} = (V, E)$, a* hypergraph separator decomposition *is a triple $(\mathcal{H}, \mathcal{S}, \mathcal{R})$ where: (i) $\mathcal{S} \subset E$, and the removal of $\mathcal{S}$ separates $\mathcal{H}$ into k disconnected components (subgraphs) $\mathcal{H}_1, ..., \mathcal{H}_k$; (ii) $\mathcal{R}$ is a relation over the size of the disjoint subgraphs (i.e. balance factor).*

It is well known that the problem of generating optimal hypergraph partitions is hard. However heuristic approaches were developed over the years. A good approach is packaged in `hMeTiS`[1]. We will use this software as a basis for our pseudo-tree generation. This idea and software were also used by [4] to generate low width decomposition trees. Generating a pseudo-tree T for $\mathcal{P}$ using `hMeTiS` is fairly straightforward. The vertices of the hypergraph are partitioned into two balanced (roughly equal-sized) parts, denoted by $\mathcal{H}_{left}$ and $\mathcal{H}_{right}$ respectively, while minimizing the number of hyperedges across. A small number of crossing edges translates into a small number of variables shared between the two sets of functions. $\mathcal{H}_{left}$ and $\mathcal{H}_{right}$ are then each recursively partitioned in the same fashion, until they contain a single vertex. The result of this process is a tree of hypergraph separators which is also a pseudo-tree of the original model since each separator corresponds to a subset of variables chained together.

In Table1 we computed the height of the pseudo-tree obtained with the hypergraph and minfill heuristics for 10 belief networks from the UAI Repository[2] and 10 constraint networks derived from the SPOT5 benchmark [17]. For each pseudo-tree we also computed the induced width of the elimination order obtained from the depth-first traversal of the tree. We observe that the minfill heuristic generates lower-width elimination orders, while the hypergraph heuristic produces much smaller height pseudo-trees. The hypergraph pseudo-trees appear to be favorable for tree search algorithms, while the minfill pseudo-trees, which minimize the context size, are more appropriate for graph search algorithms.

[1] http://www-users.cs.umn.edu/ karypis/metis/hmetis
[2] http://www.cs.huji.ac.il/labs/compbio/Repository

Network	hypergraph		min-fill		Network	hypergraph		min-fill	
	width	height	width	height		width	height	width	height
barley	7	13	7	23	spot_5	47	152	39	204
diabetes	7	16	4	77	spot_28	108	138	79	199
link	21	40	15	53	spot_29	16	23	14	42
mildew	5	9	4	13	spot_42	36	48	33	87
munin1	12	17	12	29	spot_54	12	16	11	33
munin2	9	16	9	32	spot_404	19	26	19	42
munin3	9	15	9	30	spot_408	47	52	35	97
munin4	9	18	9	30	spot_503	11	20	9	39
water	11	16	10	15	spot_505	29	42	23	74
pigs	11	20	11	26	spot_507	70	122	59	160

Table 1. Bayesian Networks Repository (left); SPOT5 benchmarks (right).

6 Experiments

In this section we evaluate the performance of the new AND/OR Branch-and-Bound graph search schemes on two common optimization problems: solving Weighted CSPs (WCSP) and finding the Most Probable Explanation (MPE) in Bayesian networks[3].

Weighted CSP [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.

Bayesian Networks provide a formalism for reasoning about partial beliefs under conditions of uncertainty [7]. They are defined by a directed acyclic graph over nodes representing variables of interest. The arcs indicate the existence of direct causal influences between linked variables quantified by conditional probability tables (CPTs) that are attached to each family of parents-child nodes in the network. The MPE problem is the task of finding a complete assignment with maximum probability that is consistent with the evidence. It easy to see that MPE can be trivially expressed as a WCSP by replacing the probability tables by their negative logarithm.

We consider three classes of AND/OR Branch-and-Bound tree search algorithms, each one of them using a specific heuristics generator as follows. Classes s-AOMB(i) and d-AOMB(i) are guided by static/dynamic mini-bucket heuristics, while AOMFDAC maintains full directional arc-consistency (FDAC). We also consider the graph versions of these algorithms, denoted by s-AOMB(i,j), d-AOMB(i,j) and AOMFDAC(j), respectively, which perform caching only at the variables for which the context size is smaller than or equal to the cache bound j.

In all our experiments, the competing algorithms were restricted to a static variable ordering resulted from a depth-first traversal of the pseudo-tree. We report the average effort, as CPU time (in seconds) and number of visited nodes required for proving optimality of the solution. For all test instances we record the number of variables (n), domain size (d), number of functions (c), induced width (w*) and height of the pseudo-

[3] Experiments were done on a 2.4GHz Pentium IV with 1GB of RAM, running Windows XP.

Network	Algorithm	hypergraph					minfill				
		(w*,h)	no cache		cache		(w*,h)	no cache		cache	
			time	nodes	time	nodes		time	nodes	time	nodes
29b	AOMFDAC	(16,22)	5.938	170,823	1.492	40,428	(14,42)	5.036	79,866	3.237	34,123
(83,394)	sAOMB(12)		1.002	8,458	1.012	1,033		**0.381**	997	0.411	940
42b	AOMFDAC	(31,43)	1,043	6,071,390	884.1	3,942,942	(18,62)	-	22,102,050	-	17,911,719
(191,1151)	sAOMB(16)		132.0	2,871,804	127.4	2,815,503		3.254	11,636	**3.164**	9,030
54b	AOMFDAC	(12,14)	0.401	6,581	0.29	3,377	(9,19)	1.793	28,492	0.121	2,087
(68,197)	sAOMB(10)		0.03	74	0.03	74		**0.02**	567	**0.02**	381
404b	AOMFDAC	(19,23)	0.02	148	**0.01**	138	(19,57)	2.043	21,406	0.08	1,222
(101,595)	sAOMB(12)		**0.01**	101	**0.01**	101		0.02	357	**0.01**	208
503b	AOMFDAC	(9,14)	0.02	405	**0.01**	307	(8,46)	1077.1	19,041,552	0.05	701
(144,414)	sAOMB(8)		**0.01**	150	**0.01**	150		0.03	1,918	**0.01**	172
505b	AOMFDAC	(19,32)	17.8	368,247	5.20	69,045	(16,98)	-	9,872,078	15.43	135,641
(241,1481)	sAOMB(14)		5.618	683	6.208	683		**4.997**	1912	5.096	831

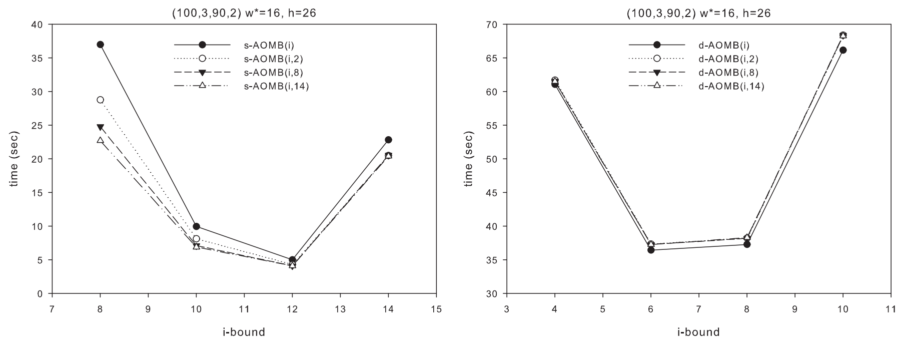
Table 2. Results for SPOT5 benchmarks.

tree (h). A ”-” indicates that a time limit was exceeded by the respective algorithm. The best results are highlighted.

6.1 Weighted CSPs

For our first experiment, we consider the scheduling of an Earth observing satellite. The original formulation of the problem states that given a set of candidate photographs, select the best subset that the satellite will actually take. The selected subset of photographs must satisfy a set of imperative constraints and, at the same time, maximize the importance of the selected photographs. We experimented with problem instances from the SPOT5 benchmark [17] that can be trivially translated into the WCSP formalism. These instances have binary and ternary constraints and domains of size 1 and 3. For our purpose we consider a simplified binary MAX-CSP version of the problem (i.e. 0/1 binary cost functions) and search for a complete value assignment to all variables that violates the least number of constraints.

Table 2 reports the results obtained for 6 SPOT5 networks. The first column identifies the instance, the number of variables (n) and the number of binary constraints (c). For each instance we ran two algorithms (given by the second column): AOMFDAC and s-AOMB(i). For the latter we report only the i-bound for which we obtained the best results. The remaining columns are divided into two vertical blocks, each corresponding to a specific heuristic used for constructing the pseudo-tree (e.g. hypergraph, min-fill). Each block reports the induced width (w^*), the height of the pseudo-tree (h), the running time and number of nodes explored by the tree (no cache) as well as the graph (cache) version of each algorithm. The cache bound j was set to 16. It can be observed that caching improves considerably the performance of both algorithms, especially for AOMFDAC. On instance 505b for example, the graph version of AOMFDAC is as much as 3.4 times faster than the tree version when running with a hypergraph based pseudo-tree. The same instance could not be solved within a 1 hour limit by the tree AOMFDAC using a min-fill based pseudo-tree, but it was solved in about 15 seconds by the graph version of the algorithm. The effect of caching is not too prominent for s-AOMB(i). This is most likely due to the very good quality of the heuristic estimates which able to prune the search space very effectively. Regarding the quality of the pseudo-trees we observe that the hypergraph heuristic generates lower height trees



(a) (b)

Fig. 3. Results for random Bayesian networks.

which appear to favor AOMFDAC. Alternatively, min-fill based trees produce lower width orderings which can in turn generate more accurate mini-bucket heuristic estimates.

6.2 Bayesian Networks

Our second experiment consists of uniform random Bayesian networks. The networks were generated using parameters (n, d, c, p), where n is the number of variables, d is the domain size, c is the number of conditional probability tables (CPTs) and p is the number of parents in each CPT. The structure of the network is created by randomly picking c variables out of n and, for each, randomly picking p parents from their preceding variables, relative to some ordering. The entries of each probability table are generated uniformly randomly, and the table is then normalized.

Figure 3 displays the results for a class of random Bayesian networks with parameters (n=100,d=3,c=90,p=2). The pseudo-tree was constructed by the min-fill heuristic. We consider two classes of algorithms s-AOMB(i,j) and d-AOMB(i,j), respectively. The i-bound of the mini-bucket heuristic ranged between 2 and 14, and we chose three caching levels as follows: *low* ($j = 2$), *medium* ($j = 8$) and *high* ($j = 14$). It can be observed that caching improves s-AOMB(i) (see Figure 3(a)) especially for smaller i-bounds of the static mini-bucket heuristic (e.g. $i = 8$). When using the dynamic mini-bucket heuristic (see Figure 3(b)) caching does not outweigh its overhead for all reported i-bounds. This is due primarily to the accuracy of the heuristic which is able to prune a substantial portion of the search space.

6.3 Genetic Linkage Analysis

For our third experiment we consider the problem of computing the *maximum likelihood haplotype configuration* of a general pedigree. In human genetic linkage analysis [18], the *haplotype* is the sequence of alleles at different loci inherited by an individual from one parent, and the two haplotypes (maternal and paternal) of an individual constitute this individual's *genotype*. When genotypes are measured by standard procedures, the

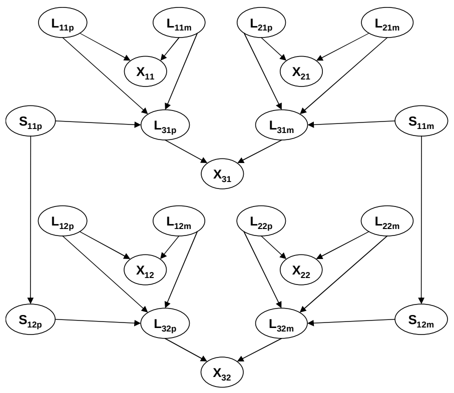


Fig. 4. A fragment of Bayesian network used in genetic linkage analysis.

		hypergraph					minfill				
Pedigree	Algorithm	(w*,h)	no cache		cache		(w*,h)	no cache		cache	
(n,d)			time	nodes	time	nodes		time	nodes	time	nodes
bn2_7	sAOMB(14)	(20,36)	2.273	42,276	**0.83**	11,358	(18,43)	5.998	8,364	5.979	8,077
(460,5)	VE+C		n/a								
	Superlink		1.140								
bn2_9	sAOMB(14)	(22,39)	8.222	169,983	1.823	20,201	(21,52)	8.532	80,007	7.741	69,140
(566,5)	VE+C		n/a								
	Superlink		**1.571**								
bn11_3	sAOMB(12)	(17,27)	0.771	11,899	0.551	3,706	(15,41)	0.721	9,147	0.681	8,294
(186,4)	VE+C		11.98								
	Superlink		**0.030**								
bn11_4	sAOMB(12)	(22,33)	14.79	462,701	6.660	167,333	(20,55)	20.50	498,305	22.32	490,003
(234,5)	VE+C		17.41								
	Superlink		**0.430**								
bnLB_3	sAOMB(18)	(25,42)	26.72	467,642	3.944	21,783	(24,74)	33.47	357,316	9.083	40,310
(642,4)	VE+C		0.881								
	Superlink		**0.110**								
bnLB_4	sAOMB(18)	(26,45)	1,390	24,961,269	23.79	289,914	(21,90)	131.8	1,562,510	22.34	215,793
(799,4)	VE+C		1.011								
	Superlink		**0.130**								
bnGB_27_1	sAOMB(14)	(19,29)	28.28	863,073	**10.47**	168,540	(21,40)	67.75	1,726,232	74.23	1,716,848
(178,4)	VE+C		172.5								
	Superlink		32.88								
bnGB_67_1	sAOMB(18)	(24,39)	9.744	47,869	**9.564**	36,715	(25,50)	170.3	95,504	225.2	94,587
(212,4)	VE+C		597.5								
	Superlink		11.72								

Table 3. Results for genetic linkage analysis networks.

result is a list of unordered pairs of alleles, one pair for each locus. The maximum likelihood haplotype problem consists of finding a joint haplotype configuration for all members of the pedigree which maximizes the probability of data.

The pedigree data can be represented as a Bayesian network with three types of random variables: *genetic loci* variables which represent the genotypes of the individuals in the pedigree (two genetic loci variables per individual per locus, one for the paternal allele and one for the maternal allele), *phenotype* variables, and *selector* variables which are auxiliary variables used to represent the gene flow in the pedigree. Figure 4 represents a fragment of a network that describes parents-child interactions in a simple

2-loci analysis. The genetic loci variables of individual i at locus j are denoted by $L_{i,jp}$ and $L_{i,jm}$. Variables $X_{i,j}$, $S_{i,jp}$ and $S_{i,jm}$ denote the phenotype variable, the paternal selector variable and the maternal selector variable of individual i at locus j, respectively. The conditional probability tables that correspond to the selector variables are parameterized by the *recombination ratio* θ [19]. The remaining tables contain only deterministic information. It can be shown that given the pedigree data, the haplotyping problem is equivalent to computing the Most Probable Explanation of the corresponding Bayesian network (for more details consult [19, 20]).

In Table 3 we show results for several hard genetic linkage problem instances[4]. We experimented with three algorithms: s-AOMB(i) (tree and graph versions), VE+C and Superlink. Superlink v1.5 is currently the most efficient solver for genetic linkage analysis, is dedicated to this domain and uses a combination of variable elimination and conditioning, as well as a proprietary matrix multiplication scheme. VE+C is our implementation of the elimination and conditioning hybrid, without the special multiplication scheme, and it uses the elimination order output by Superlink. For s-AOMB(i) we report only the best i-bound of the mini-bucket heuristic. For the graph version of s-AOMB(i) the cache bound was equal to the i-bound. We observe that on this domain, the hypergraph based pseudo-trees produced the best results for both the tree and graph versions of s-AOMB(i). In several instances, the hypergraph heuristic was also able to produce orderings with widths smaller than those obtained with the min-fill heuristic (e.g. bnGB_27_1, bnGB_67_1).

Caching improves dramatically the performance of s-AOMB(i) in all test cases. On the bnLB_4 pedigree, the graph version of s-AOMB(18) is about 58 times faster than the tree version, reducing the size of the search space explored from 25M to about 290K nodes. The graph s-AOMB(i) is consistently better than VE+C, except on instances bnLB_3 and bnLB_4. In that case, the elimination order produced by Superlink had a width of 13, which was much smaller than that obtained by both the hypergraph and min-fill heuristics. When comparing the graph s-AOMB(i) with Superlink we observe that the graph s-AOMB(i) is better than Superlink in 3 out of the 8 instances (e.g. bn2_7, bnGB_27_1, bnGB_67_1) and they are about the same order of magnitude on the remaining instances.

7 Conclusion

This paper rests on two contributions. First, we extended the AND/OR Branch-and-Bound tree search algorithm with a flexible context-based caching scheme allowing the algorithm to explore an AND/OR search graph rather than a tree. The new graph search algorithm was then specialized with heuristics based on either the mini-bucket approximation or soft arc-consistency. Second, we introduced a new heuristic for generating pseudo-trees based on the recursive decomposition of the problem's hypergraph. Both contributions are supported by experimental results for solving WCSPs and computing the MPE configuration in belief networks on a variety of synthetic and real-world networks, including some very challenging networks from the field of genetic linkage analysis. Finally, some new directions of research include combining the AND/OR

[4] All networks are available at http://bioinfo.cs.technion.ac.il/superlink/

search algorithms with constraint propagation for efficiently handling the determinism in Bayesian networks, as well as improving the heuristics that guide the search process.

Related Work: AOBB is related to the Branch-and-Bound method proposed by [21] for acyclic AND/OR graphs and game trees, as well as the pseudo-tree search algorithm proposed in [22] for boosting Russian Doll search. The optimization method developed in [23] for semi-ring CSPs can also be interpreted as an AND/OR graph search algorithm.

References

1. R. Marinescu and R. Dechter. And/or branch-and-bound for graphical models. *In IJCAI'05*.
2. R. Dechter and R. Mateescu. Mixtures of deterministic-probabilistic networks. *In UAI'04*.
3. R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.
4. A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.
5. F. Bacchus, S. Dalmao, and T. Pittasi. Value elimination: Bayesian inference via backtracking search. *Proc. of UAI'03*, pages 20–28, 2003.
6. S. Bistarelli, U. Montanari, and F. Rossi. Semiring based constraint solving and optimization. *Journal of ACM*, 44(2):309–315, 1997.
7. J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan-Kaufmann, 1988.
8. E. Freuder and M. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. *Proc. of IJCAI'85*, 1985.
9. R. Mateescu and R. Dechter. And/or cutset conditioning. *In IJCAI'05*.
10. R. Dechter and I. Rish. Mini-buckets: A general scheme for approximating inference. *ACM*, 2003.
11. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 1999.
12. K. Kask and R. Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence*, 2001.
13. J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted csp. *Proc. of IJCAI'03*, pages 631–637, 2003.
14. S. de Givry, F. Heras, J. Larrosa, and M. Zytnicki. Existential arc consistency: getting closer to full arc consistency in weighted csps. *Proc. of IJCAI'05*, 2005.
15. R. Bayardo and D. Miranker. On the space-time trade-off in solving constraint satisfaction problems. *Proc. of IJCAI'95*, 1995.
16. U. Kjæaerulff. Triangulation of graph-based algorithms giving small total space. *Technical Report, University of Aalborg, Denmark*, 1990.
17. E. Bensana, M. Lemaitre, and G. Verfaillie. Earth observation satellite management. *Constraints*, 4(3):293–299, 1999.
18. Jurg Ott. *Analysis of Human Genetic Linkage*. The Johns Hopkins University Press, 1999.
19. M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. *Bioinformatics*, 2002.
20. M. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*, 2005.
21. L. Kanal and V. Kumar. *Search in artificial intelligence.* Springer-Verlag., 1988.
22. J. Larrosa, P. Meseguer, and M. Sanchez. Pseudo-tree search with soft constraints. *Proc. of ECAI'02*, 2002.
23. P. Jegou and C. Terrioux. Decomposition and good recording for solving max-csps. *In ECAI'04*.