Anytime Depth-First Search with Problem Decomposition for Optimization in Graphical Models

Lars Otten and **Rina Dechter**
Department of Computer Science
University of California, Irvine, U.S.A.
{lotten,dechter}@ics.uci.edu

Abstract

One popular and efficient scheme for solving *exactly* MPE/MAP and related problems over graphical models is depth-first Branch and Bound. However, when the algorithm exploits problem decomposition using AND/OR search spaces, its anytime behavior breaks down. This paper 1) analyzes and demonstrates this inherent conflict between effective exploitation of problem decomposition (through AND/OR search spaces) and the anytime behavior of depth-first search (DFS), 2) presents a first scheme to address this issue while maintaining desirable DFS memory properties, 3) analyzes and demonstrates its effectiveness. Our work is applicable to *any* problem that can be cast as search over an AND/OR search space.

1 Introduction

Max-product problems over graphical models, generally known as MPE or MAP, have many applications with practical significance, ranging from computational biology and genetics to scheduling tasks and coding networks. One established and efficient class of algorithms for solving these problems exactly is depth-first Branch and Bound over AND/OR search spaces. Developed in the past decade within the probabilistic reasoning and constraint communities, these methods are effective because they use sophisticated lower bound schemes such as soft arc-consistency [8] or the Mini Bucket heuristic [4; 9], because they avoid redundant computation using caching schemes, and most significantly, because they take advantage of problem decomposition by exploring an AND/OR search space [11] or an equivalent representation. The efficiency of these algorithms was established in several evaluations, including recent UAI competitions [5], and their properties (for exact computation) are well documented [6; 9; 10].

A principled alternative is presented by best-first schemes, but while provably superior in terms of number of node expansions, these often fail when a problem has large induced width; moreover, they can only provide a solution at termination [10]. Depth-first search is therefore often preferred because of its flexibility in working with bounded memory and because of its *anytime behavior*. Namely, when finding a feasible solution is easy but an optimal one is hard, depth-first Branch and Bound generates solutions that get better and better over time, until it eventually discovers an optimal one. Thus it can function also as an approximation scheme for otherwise infeasible problems or when time is limited [12].

Indeed, in the 2010 UAI challenge participating Branch and Bound solvers performed competitively wrt. approximation (placing 1st and 3rd in some categories). But we also observed an inability to produce even a single solution on some instances, especially when the time bound was small. Thus motivated, this paper will demonstrate that the issue is rooted in the underlying AND/OR search space.

Originally introduced to graphical models to facilitate problem decomposition during search (e.g. [3]), these search spaces can be explored by any search strategy. When traversed depth-first, however, all but one decomposed subproblem will be *fully solved* before a single overall solution can be composed, voiding the algorithm's anytime characteristics.

This paper's main contribution is a new Branch and Bound scheme over AND/OR search spaces, called *Breadth-Rotating AND/OR Branch and Bound (BRAOBB)* that addresses the anytime issue in a principled way, while maintaining the favorable complexity guarantees of depth-first search. The algorithm combines depth-first and breadth-first exploration by periodically rotating over the different subproblems, each of which is processed depth-first.

Experimental evaluation on a variety of benchmark domains compares BRAOBB against one of the best variants of AND/OR branch and Bound search, AOBB [9], and against an "ad hoc" fix that we suggest, which relies on a heuristic to quickly find a solution to each subproblem before reverting to depth-first search. The empirical results demonstrate superior anytime behavior of BRAOBB, especially over problematic cases where standard AOBB and its ad hoc fix fail. In particular, we obtained good performance on two very hard instances from the 2010 UAI challenge (Figure 5).

The work presented in this paper is focused on optimization problems defined over graphical models. As such our results are also relevant for related schemes like recursive conditioning [2] and value elimination [1] in the area of probabilistic reasoning, or BTD (Backtracking Tree Decomposition [6]) in constraint optimization. We note, however, that the presented concepts carry over to combinatorial AND/OR search spaces in general.

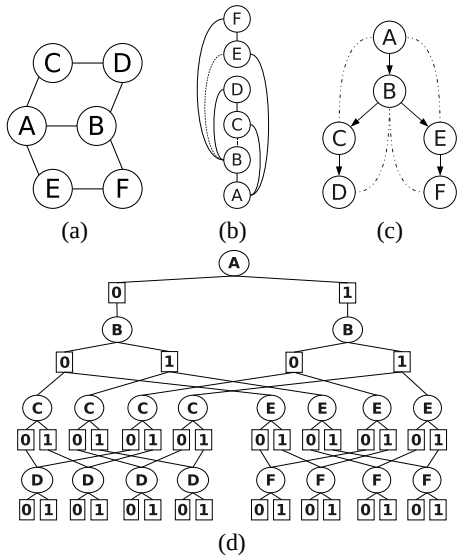


Figure 1: (a) Example primal graph of a graphical model with six variables, (b) its induced graph along ordering $d = A, B, C, D, E, F$, (c) a corresponding pseudo tree, and (d) the resulting context-minimal AND/OR search graph.

The remainder of this paper is structured as follows: Section 2 introduces the underlying concepts while Section 3 identifies the central issue and provides empirical evidence. The new algorithm BRAOBB is proposed in Section 4 and its properties analyzed. Section 5 presents experimental results and analysis before Section 6 concludes.

2 Background

We consider a MPE (Most Probable Explanation, sometimes also MAP, Maximum A Posteriori assignment) problem over a graphical model, $(X, F, D, \max, \prod)$. $F = \{f_1, \ldots, f_r\}$ is a set of functions over variables $X = \{X_1, \ldots, X_n\}$ with discrete domains $D = \{D_1, \ldots, D_n\}$, we aim to compute $\max_X \prod_i f_i$, the probability of the most likely assignment. The set of function scopes implies a primal graph and, given an ordering of the variables, an *induced graph* (where, from last to first, each node's earlier neighbors are connected) with a certain *induced width*. Another closely related combinatorial optimization problem is the *weighted constraint problem*, where we aim to minimize the sum of all costs, i.e. compute $\min_X \sum_i f_i$. These tasks have many practical applications but are known to be NP-hard.

The concept of **AND/OR search spaces** has recently been introduced to graphical models to better capture the structure of the underlying graph during search [3]. The search space is defined using a *pseudo tree* of the graph, which captures problem decomposition:

DEFINITION **1.** *A* pseudo tree *of an undirected graph $G = (X, E)$ is a directed, rooted tree $\mathcal{T} = (X, E')$, such that every arc of G not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node in $\mathcal{T}$ to an ancestor in $\mathcal{T}$. The arcs in E' may not all be included in E.*

AND/OR Search Trees and Graphs : Given a graphical model instance with variables X and functions F, its primal graph (X, E), and a pseudo tree $\mathcal{T}$, the associated *AND/OR search tree* consists of alternating levels of OR and AND nodes. Its structure is based on the underlying pseudo tree $\mathcal{T}$: the root of the AND/OR search tree is an *OR node* labeled with the root of $\mathcal{T}$. The children of an OR node $\langle X_i \rangle$ are *AND nodes* labeled with assignments $\langle X_i, x_j \rangle$ that are consistent with the assignments along the path from the root; the children of an AND node $\langle X_i, x_j \rangle$ are OR nodes labeled with the children of X_i in $\mathcal{T}$, representing conditionally independent subproblems.

Identical subproblems, identified by their context (the partial instantiation that separates the subproblem from the rest of the network), can be merged, yielding the *context-minimal AND/OR search graph* [3]. It was shown that, given a pseudo tree $\mathcal{T}$ of height h, the size of the AND/OR search tree based on $\mathcal{T}$ is $O(n \cdot k^h)$, where k bounds the variables' domain size. The context-minimal AND/OR search graph has size $O(n \cdot k^{w^*})$, where w^* is the induced width of the problem graph along a depth-first traversal of $\mathcal{T}$ [3]. Note that in Figure 1(a) the AND nodes for B have two children each, representing independent subproblems and thus demonstrating problem decomposition.

Given an AND/OR search space $S_{\mathcal{T}}$, a *solution subtree* $Sol_{S_{\mathcal{T}}}$ is a tree such that (1) it contains the root of $S_{\mathcal{T}}$; (2) if a nonterminal AND node $n \in S_{\mathcal{T}}$ is in $Sol_{S_{\mathcal{T}}}$ then all its children are in $Sol_{S_{\mathcal{T}}}$; (3) if a nonterminal OR node $n \in S_{\mathcal{T}}$ is in $Sol_{S_{\mathcal{T}}}$ then exactly one of its children is in $Sol_{S_{\mathcal{T}}}$.

AND/OR Branch and Bound (AOBB) is a state-of-the-art algorithm for solving optimization problems such as max-product over graphical models. The edges of the AND/OR search graph can be be annotated by weights derived from the set of cost functions F in the graphical model; finding the optimal-cost solution subtree solves the stated optimization task. Assuming a maximization query, AOBB traverses the weighted context-minimal AND/OR graph in a depth-first manner while keeping track of the current lower bound on the maximal solution cost. A node n will be pruned if this lower bound exceeds a heuristic upper bound on the solution to the subproblem below n, often obtained by solving a relaxed problem (e.g. through Mini Buckets [7]). The algorithm interleaves forward node expansion with a backward cost revision or propagation step that updates node values (capturing the current best solution to the subproblem rooted at each node), until search terminates and the optimal solution has been found. [9].

3 Anytime versus AND/OR

We will use AOBB to denote the algorithm above in its specific graphical models context as well as a generic name for any depth-first Branch and Bound scheme over an AND/OR search space. As a depth-first scheme one would expect AOBB to quickly produce a non-optimal solution and then gradually improve upon it, maintaining the current best one throughout the search. However this ability is compromised in the context of AND/OR search.

Specifically, in AND/OR search spaces depth-first traversal

of a set of independent subproblems will solve to completion all but one subproblem before the last one is even considered. As a consequence, the first generated overall non-optimal solution contains conditionally optimal solutions to all subproblems but the last one. Furthermore, depending on the problem structure and the complexity of the independent subproblems, the time to return even this first non-optimal overall solution can be significant, practically negating the anytime behavior of depth-first search (DFS).

3.1 Subproblem Ordering

In certain cases, the above suggests a simple remedy: if decomposition yields only one large subproblem and several smaller ones, the latter can be solved depth-first in relatively little time, to be then combined with the incrementally improving solutions of the larger subproblem. Thus for anytime behavior an AOBB algorithm would need to process independent subproblems from "easy" to "hard".

To demonstrate the practical impact of subproblem orderings, we use a simple heuristic that takes the induced width as a measure of subproblem hardness (motivated by its exponential role in the asymptotic complexity), i.e. we modify AOBB such that subproblems with smaller induced width will be processed first (in the general description of AOBB the subproblem ordering is left unspecified).

Figure 2 contrasts the anytime performance of AOBB using this "increasing" subproblem order against the inverse one ("decreasing") by plotting the probability of the best assignment found over time on a set of example problems; all other aspects of the algorithm remain constant. Pedigree30x1 in particular features exactly one single complex subproblem and a number of relatively simple ones; in this case processing subproblems by increasing induced width right away produces a non-optimal solution that improves rapidly. The inverse order yields the first solution only after about 90 minutes – the one complex subproblem has been fully solved and the overall solution is already optimal. Pedigree41x1 has a similarly advantageous structure and thus yields similar results – with the distinction that the inverse subproblem order does not produce any solution at all within 24 hours.

In case of pedigree34x2, however, decomposition yields two complex subproblems: the increasing subproblem order still outperforms its inverse, yet it returns the initial solution only after about 1,000 seconds. In fact, no possible subproblem ordering can lead to acceptable anytime behavior in this case due to the structure of subproblems, clearly highlighting the limits of this approach.

Independent of anytime behavior, we point out that incorporating different subproblem orderings impacts the algorithm's overall efficiency (i.e., the time to finding and proving an optimal solution): knowing the solution to one subproblem can aid the pruning of Branch and Bound in the next one to varying degrees. However, this issue has not been treated systematically in the literature for graphical models, with sporadic experiments also suggesting an easy-to-hard order, using some heuristic to determine subproblem complexity [9]. This general problem is outside the scope of the present paper, however.

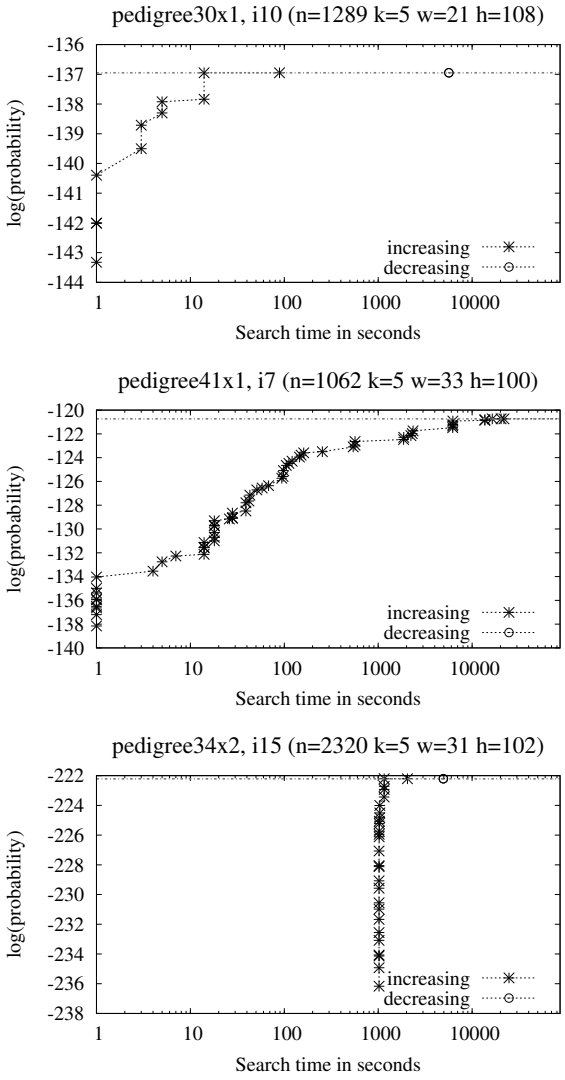


Figure 2: Impact of subproblem ordering on AOBB. The number of variables n, the max. domain size k, the induced width w along the chosen ordering, as well as the height of the corresponding pseudo tree h are specified for each network.

3.2 Greedy Subproblem Dive

Another relatively straightforward remedy that can be viewed as an "ad hoc" fix is the following: Every time decomposition is encountered within the search space, we will try to greedily find a single initial solution to each independent subproblem before successively solving each of them to completion depth-first, through normal AOBB. To obtain this initial solution the algorithm can perform a greedy "dive" into each subproblem by only considering one value for each variable along the path (in case of the Mini Bucket heuristic, it is easy to see that this is equivalent to a forward pass over the bucket structure [7]).

Clearly, the choice of the dive path is crucial for the algorithm's performance. Namely, if the chosen path leads to a dead end (zero probability), the dive will be futile and

not yield a subproblem solution. And in fact experiments in Section 5 will demonstrate that the resulting performance depends heavily on the quality of the heuristic, which often prevents satisfactory anytime behavior. In the next section we will therefore propose a new search strategy that addresses the anytime issue over AND/OR search spaces in a principled manner.

4 Breadth-Rotating AOBB

In the following we develop a new search scheme called *Breadth-Rotating AND/OR Branch and Bound (BRAOBB)* that addresses the issue of anytime performance over AND/OR search spaces. It combines depth-first exploration with the notion of "rotating" through different subproblems in a breadth-first manner. Namely, node expansion still occurs depth-first as in standard AOBB, but the algorithm takes turns in processing subproblems, each up to a given number of operations at a time, round-robin style.

To motivate this approach, consider again that a solution is represented by a *solution tree* over an AND/OR search space, guided by a pseudo tree. A pure DFS scheme will construct the different branches of a solution tree one by one, ensuring optimality for each branch before moving to the next. To restore anytime behavior, we instead aim to develop all branches of the solution tree "simultaneously", which we emulate by rotating through them.

More systematically, the algorithm repeats the following high-level steps until completion:

1. Move breadth-first to next open subproblem P.
2. Process P depth-first, until either:
   - P is solved optimally,
   - P decomposes into child subproblems, or
   - a predefined threshold of operations is reached.

The threshold is needed to ensure the algorithm does not get stuck in one large subproblem (where the other two conditions do not occur for a long time). Furthermore, in order to focus on a single solution tree at a time, a subproblem is only considered "open" if it does not currently have any open child subproblems, as illustrated below.

Algorithm 1 gives more detailed pseudo code for the scheme (with some details from standard AOBB omitted, cf. [9]). The key element lies in rotating over the different subproblems of the search space; by organizing these in a global first-in-first-out queue (*GLOBAL*), we emulate breadth-first exploration across the different branches of the solution tree. The input parameter Z determines how many nodes should be processed before moving on to the next subproblem (if none of the other conditions are satisfied first). Each subproblem is itself explored depth-first (via a local last-in-first-out stack of nodes, *LOCAL*); whenever a new level of decomposition is encountered, as captured by the pseudo tree, the resulting child subproblems are pushed to the end of the global queue. Finally, subproblems are only considered in the rotation if they don't currently have any open child subproblems. The following example illustrates:

Example : Figure 3 demonstrates the scheme's application ($Z = 2$) to the AND/OR search graph in Figure 1 (assuming

Algorithm 1 Breadth-Rotating AOBB

Given: Graphical model $(X, F, D, \max, \prod)$ and pseudo tree $\mathcal{T}$ with root X_o, rotation threshold Z
Output: cost of optimal solution

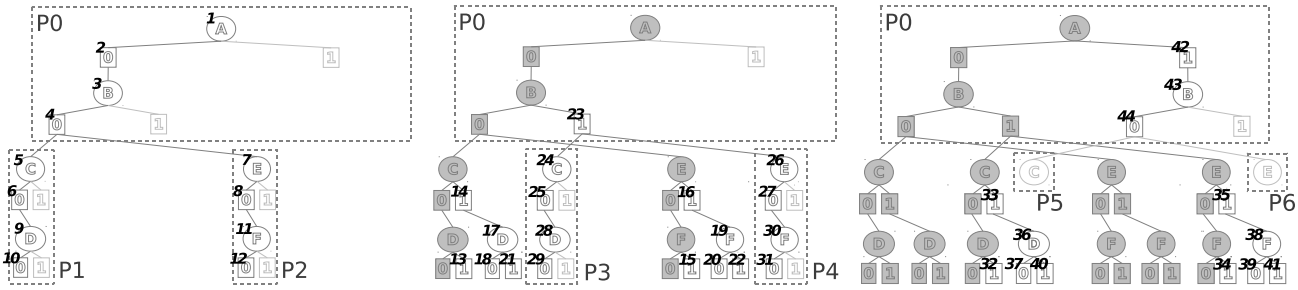
```
1: ROOT ← {⟨X_0⟩}  // generate root subproblem
2: push ROOT to end of GLOBAL
3: while GLOBAL≠ ∅
4:    LOCAL ← front(GLOBAL)  // next subproblem in queue
5:    for z ← 1 to Z or until LOCAL = ∅
        or until childSubprob(LOCAL) ≠ ∅
6:       n ← top(LOCAL)  // top node from current subproblem
7:       ...  // caching and pruning as in AOBB
8:       if n = ⟨X_i⟩ is OR node
9:          for x_j ∈ D_i
10:            create AND child ⟨X_i, x_j⟩
11:            add ⟨X_i, x_j⟩ to top of LOCAL
12:      else if n = ⟨X_i, x_j⟩ is AND node
13:         Y_1, ..., Y_m ← children_T(X_i)
14:         generate OR children ⟨Y_1⟩, ..., ⟨Y_r⟩
15:         if m=1  // no decomposition
16:            push ⟨Y_1⟩ to top of LOCAL
17:         else if m > 1  // problem decomposition
18:            for r ← 1 to m
19:               NEW ← {⟨Y_r⟩}  // new child subproblem
20:               push NEW to back of GLOBAL
21:      if children(n)= ∅  // n is leaf
22:         propagate(n)  // upwards in search space
23:   if LOCAL ≠ ∅  // subproblem not yet solved
24:      push LOCAL to end of GLOBAL
25: return value(⟨X_0⟩)  // root node contains optimal solution
```

no pruning). Part (a) shows the first 12 nodes expanded during the first seven iterations of the outer while loop as follows: *(1)* Taking the overall problem as subproblem P0, expand $\langle A\rangle$ and $\langle A, 0\rangle$ before reaching the threshold $Z = 2$. *(2)* With no decomposition so far rotation returns to subproblem P0. Expand $\langle B\rangle$ and $\langle B, 0\rangle$, yielding subproblems P1 and P2 rooted at $\langle C\rangle$ and $\langle E\rangle$, respectively, which are added to the queue. *(3)* Rotate to subproblem P1 and expand $\langle C\rangle$ and $\langle C, 0\rangle$. *(4)* Rotate to subproblem P2. Expand $\langle E\rangle$ and $\langle E, 0\rangle$. *(5)* Rotate to subproblem P0 but skip it at this point, since its child subproblems P1 and P2 are still open. *(6)* Rotation moves to subproblem P1. Expand $\langle D\rangle$ and $\langle D, 0\rangle$, discover a leaf and propagate. *(7)* Rotate to subproblem P2, expand $\langle F\rangle$ and $\langle F, 0\rangle$ – which, as a leaf, is propagated to yield the first overall solution.

Figure 3(b) illustrates how the search then proceeds to take turns solving subproblems P1 and P2 to completion (nodes 13–22) before reopening subproblem P0. Expansion 23 yields two new independent subproblems P3 and P4; their solution is depicted by nodes 24–41. After that subproblem P0 gets reopened, where expanding nodes 42–44 again yields two new subproblems P5 and P6, and so forth.

4.1 Analysis of Breadth-Rotating AOBB

Complexity : We assume a graphical model with n variables whose domain size is bounded by k. Let w^* be the induced width of the problem along a given ordering and h the height of the corresponding pseudo tree $\mathcal{T}$. Despite the breadth-first component the algorithm maintains the asymptotic complex-



(a) Expansion of nodes 1–12 (b) Expansion of nodes 13–31 (c) Expansion of nodes 32–44

Figure 3: BRAOBB exploration ($Z = 2$) at different stages. Nodes are numbered in order of their expansion.

ity of standard AOBB:

THEOREM **2.** *When searching an AND/OR search tree (i.e., without caching of redundant subproblems), BRAOBB has time complexity $O(n \cdot k^h)$ and space complexity linear in n. When searching an AND/OR search graph (with full caching), time and space complexity are $O(n \cdot k^{w^*})$.*

Proof. BRAOBB explores the same underlying AND/OR search space as standard AOBB, hence its asymptotic time complexity remains unchanged, i.e. exponential in h for tree and exponential in w^* for graph search. Space complexity for AND/OR graph search is dominated by the caching and thus also remains unchanged exponential in w^*. In case of tree search, recall that subproblems with open child subproblems are not processed further. Therefore every variable will appear in at most one subproblem at any given time. And since each subproblem is processed depth-first, i.e. in linear space, the space across all subproblems is also linear in n. □

It is worth pointing out that these are worst-case bounds that are often not met in practice, because the Branch and Bound scheme is typically very efficient and prunes large parts of the search space. In particular, the exponential memory bound for AND/OR graph search is usually not an issue since only relatively few cache entries will be written.

Comparison with standard AOBB : First, we expect that the anytime performance of BRAOBB will be robust with respect to different subproblem orderings, since the algorithm is not forced to "commit" to a single subproblem – which we identified as the main reason for the poor anytime behavior of plain AOBB in Section 3.1. We will confirm this experimentally in Section 5.

Second, the actual number of nodes explored by BRAOBB might differ from plain AOBB (for both graph and tree search), since the pruning behavior of the algorithm can be impacted by the order in which nodes are explored and subproblem solutions produced: On the one hand, solving a subproblem to completion before processing the next (in AOBB) might allow the algorithm to calculate a tighter upper bound using this optimal solution, resulting in better pruning. On the other hand, exploring subproblems concurrently in BRAOBB might lead to a tighter overall lower bound through combining solutions across subproblems as they are discovered (in an anytime fashion).

Significance of Z : The rotation threshold Z keeps the scheme from getting stuck in large subproblems, where the other two "natural" rotation conditions would not occur for a long time. As we will see in the next section, however, in practical problems we typically encounter frequent subproblem branching. The Z threshold is thus practically never reached and its value has little effect.

5 Empirical Evaluation

To validate and compare the performance of the various schemes we recorded their anytime behavior on a variety of problem instances using a common variable ordering and Mini Bucket heuristic for each instance (24 hour timeout), subproblems were ordered by increasing width (cf. Section 3.1). We ran "plain" AOBB, AOBB with the dive extension, and BRAOBB; as a baseline we also included OR Branch and Bound (without problem decomposition).

Our initial test set was comprised of 19 pedigree instances, 50 randomly generated grid networks, and 8 mastermind game instances, all part of the UAI 2008 evaluation. To ensure the presence of more than one complex subproblem, we created additional versions of each network with two and three identical copies connected at the root (signified by the "x2" and "x3" suffix, respectively), yielding a total of 57 pedigree (each run with three different heuristics), 150 grid, and 24 grid instances and resulting in over 60,000 CPU hours worth of experiments (enabled by a 320-core cluster).

Figure 4 presents results on eight instances with more than one complex subproblem, on which plain AOBB does poorly wrt. anytime. OR Branch and Bound finds an early lower bound in three cases, but provides very little improvement over time and never gets close to the optimum. The dive extension shows acceptable anytime behavior only on three instances, confirming our conjecture that its performance depends solely on the success of the initial dive – if misguided by the heuristic, the anytime behavior is predictably as bad as, or even slightly worse than the plain scheme.

The proposed BRAOBB, on the other hand, exhibits impressive anytime performance on almost all instances, often by a large margin; in seven cases the first solution is produced more or less instantly, even on pedigree51x3, where "plain" and "dive" do not return anything within 24 hours.

Table 1 summarizes the entire set of experiments by showing, at different points of time, the number of instances for

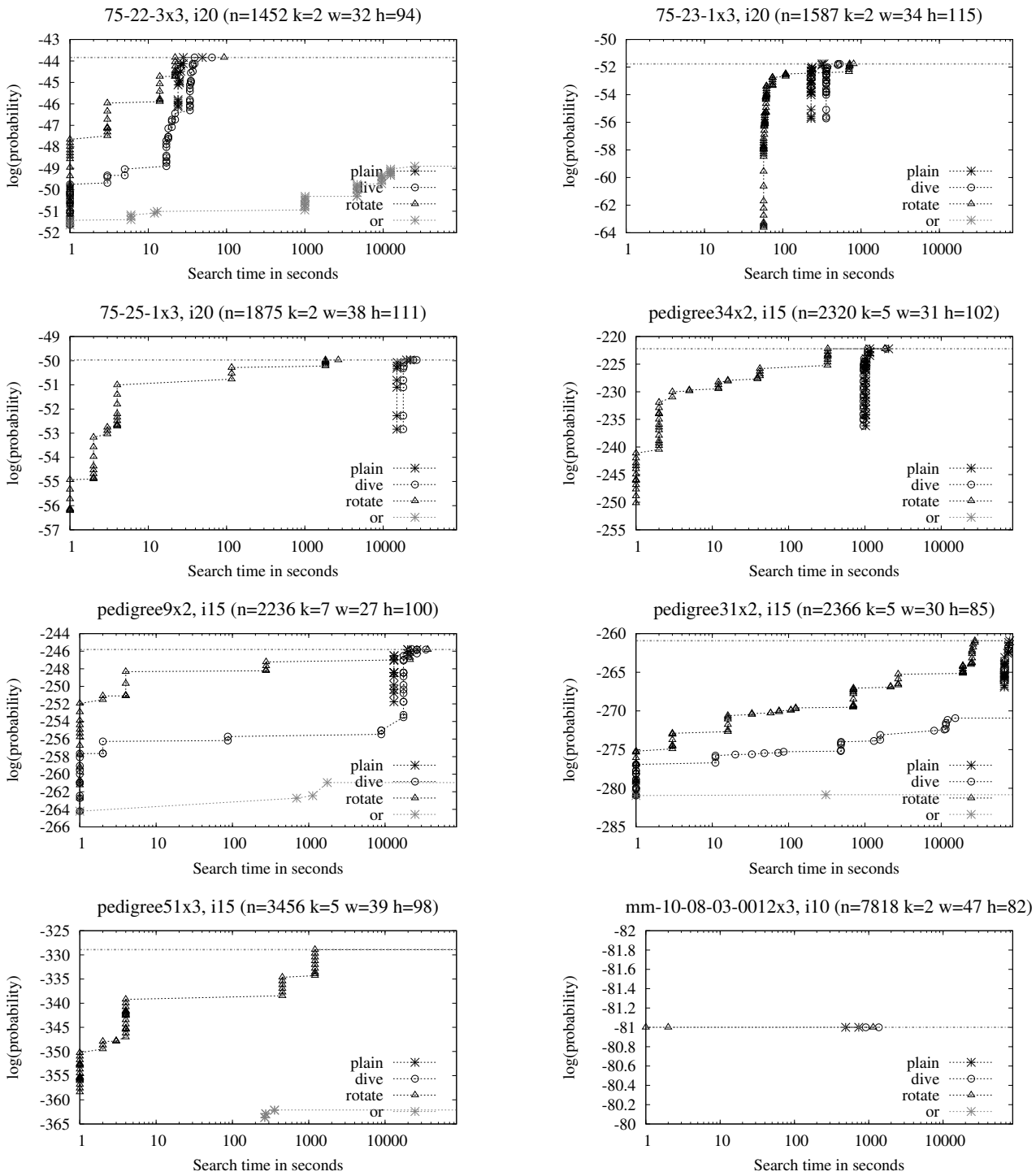


Figure 4: Anytime profiles of plain AOBB, AOBB with subproblem dive, BRAOBB, and OR Branch and Bound on selected instances with more than one hard subproblem (3 grids, 4 pedigree, 1 mastermind network).

which any solution was found, for which the optimal solution was found, and for which optimality was proven (i.e. the algorithm terminated). The results confirm that BRAOBB yields superior anytime performance: for example, within 1 second it provides an initial solution on 293 instances (out of 345), compared to just 98 for plain AOBB and 131 for the dive extension; this lead is maintained for higher time bounds. BRAOBB also finds the optimal solution quicker

Table 1: Summary statistics over 345 instances for each scheme: given are the number of cases for which, within the respective time bound, (1) any solution was found, (2) the optimal solution was found, (3) optimality was proven.

	Time bound						
	1 sec	5 sec	10 sec	1 min	5 min	1 hour	24 hours
Pedigree networks (171 total)							
plain	52 / 19 / 6	70 / 36 / 17	75 / 42 / 24	87 / 56 / 48	101 / 76 / 68	111 / 90 / 86	129 / 117 / 108
dive	76 / 16 / 5	86 / 29 / 13	94 / 38 / 20	105 / 53 / 48	116 / 69 / 64	127 / 89 / 86	135 / 114 / 105
rotate	153 / 26 / 2	157 / 40 / 15	160 / 47 / 24	162 / 59 / 48	164 / 74 / 60	165 / 98 / 84	167 / 127 / 102
or	73 / 6 / 1	76 / 7 / 5	77 / 10 / 5	79 / 10 / 9	82 / 12 / 11	87 / 16 / 15	90 / 22 / 21
Grid networks (150 total)							
plain	38 / 10 / 0	48 / 19 / 0	58 / 32 / 4	84 / 62 / 52	101 / 82 / 76	128 / 120 / 113	149 / 148 / 147
dive	47 / 6 / 0	52 / 12 / 0	55 / 24 / 1	82 / 54 / 37	97 / 78 / 71	121 / 111 / 104	147 / 147 / 146
rotate	122 / 16 / 0	128 / 27 / 0	129 / 35 / 1	136 / 69 / 38	143 / 86 / 73	146 / 126 / 110	149 / 149 / 147
or	45 / 0 / 0	45 / 1 / 0	46 / 1 / 0	53 / 2 / 0	57 / 4 / 2	64 / 10 / 9	74 / 21 / 21
Mastermind networks (24 total)							
plain	8 / 8 / 1	8 / 8 / 3	8 / 8 / 3	10 / 10 / 4	13 / 13 / 7	17 / 17 / 12	24 / 24 / 24
dive	8 / 8 / 1	8 / 8 / 3	8 / 8 / 3	11 / 11 / 5	12 / 12 / 6	21 / 21 / 19	24 / 24 / 24
rotate	18 / 18 / 1	18 / 18 / 3	18 / 18 / 3	18 / 18 / 3	21 / 21 / 4	24 / 24 / 19	24 / 24 / 24
or	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0

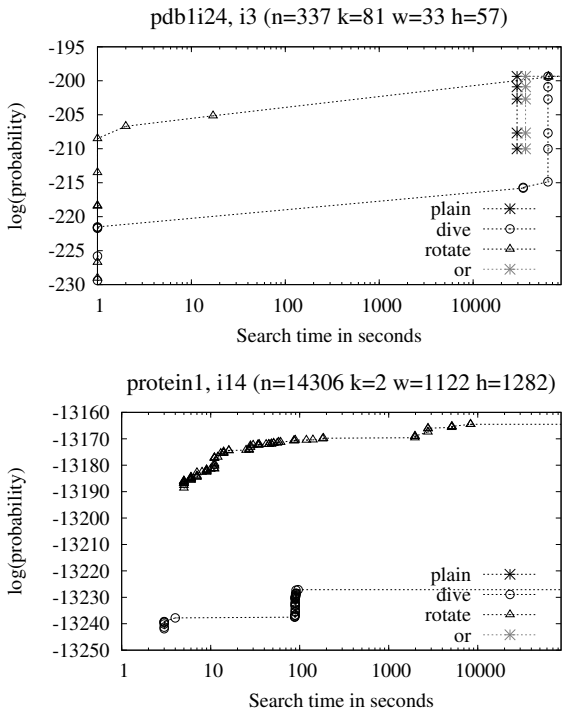


Figure 5: Anytime profiles on two example instances from the UAI'10 challenge (top: protein folding, bottom: protein-protein interaction).

than the other schemes, e.g., for 100 instances after 10 seconds (versus 80 instances for plain). Finally, we see that plain AOBB has a slight edge in terms of proving optimality, confirming that exploring subproblems concurrently can slightly impair the pruning (cf. Section 4.1).

Notably, BRAOBB did indeed restore anytime performance on example problems from the UAI 2010 inference challenge where plain AOBB failed [5]. Out of the ten instances made available, eight were solved in seconds by standard AOBB and hence not further considered. Anytime results on the remaining two, modeling protein folding and protein-protein interaction, are shown in Figure 5, once more demonstrating superiority of BRAOBB (note the large domain size of pdb1i24, the massive induced width of protein1, and that optimality for either problem could not be proved within 24 hours).

To investigate to what extent the different schemes depend on the heuristic's accuracy, Figure 6 (top) contrast plain AOBB, dive, and BRAOBB each with two different heuristics, parametrized by the Mini Bucket i-bound (where higher is better) [7]. Plain AOBB fails or does very poorly due to problem decomposition; AOBB with dive depends very much on the heuristic and fails with the weaker one. BRAOBB, however, exhibits acceptable anytime behavior even with the weaker heuristic.

Going back to Section 3.1, Figure 6 (middle) compares the performance of BRAOBB with subproblems ordered by increasing and decreasing width. In contrast to AOBB (included for reference) our new scheme is very robust and delivers nearly the same performance in both cases. Finally, experiments with different values for the rotation threshold $Z \in 10, 1000, 100000, 10000000$ in Algorithm 1 showed no significant difference in performance as exemplified by Figure 6 bottom, confirming the analysis in Section 4.1.

6 Summary

Exploiting problem decomposition in search methods has been proven to yield significantly better overall complexity in many cases. Yet this paper has demonstrated how it can be in direct conflict with the depth-first nature of Branch and Bound, thus impairing the important anytime properties of this class of algorithms.

We devised a "quick fix" that employs an initial greedy subproblem dive, but whose performance was lacking due to

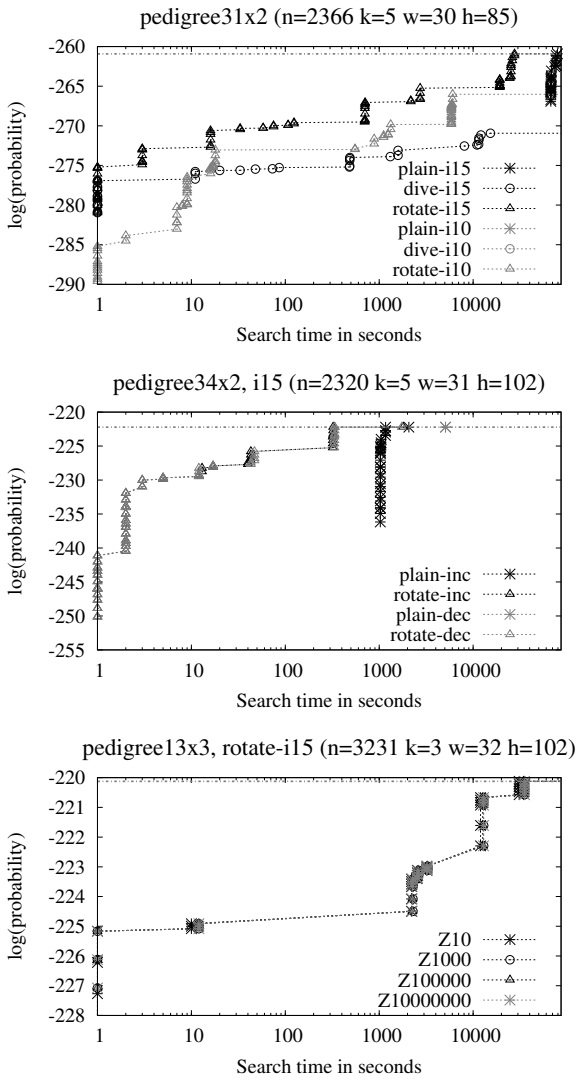


Figure 6: *Top:* impact of heuristic accuracy on anytime performance. *Middle:* impact of subproblem ordering. *Bottom:* impact of rotation threshold Z.

heavy dependence on the underlying heuristic.

The main contribution of this work is the new scheme *Breadth-Rotating AND/OR Branch and Bound (BRAOBB)*, which periodically iterates over the different subproblems in a "breadth-first" manner but was shown to retain many desirable properties of the depth-first strategy. In particular, its memory complexity remains linear in the number of variables (not accounting for caching).

We presented a large set of successful experiments that confirmed vastly improved anytime performance, especially in cases where standard depth-first Branch and Bound and its "ad hoc" extensions fail, including two hard instances from the UAI 2010 challenge. Through analysis and experiments we also showed our new scheme to be robust with respect to the order of subproblems as well as the accuracy of the guiding heuristic.

Acknowledgements

This work was partially supported by NSF grants IIS-0713118, IIS-1065618 and NIH grant 5R01HG004175-03.

References

[1] F. Bacchus, S. Dalmao, and T. Pitassi. Value elimination: Bayesian interence via backtracking search. In *UAI*, pages 20–28, 2003.

[2] A. Darwiche. Recursive conditioning. *Artif. Intell.*, 126(1-2):5–41, 2001.

[3] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artif. Intell.*, 171(2-3):73–106, 2007.

[4] R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM*, 50(2):107–153, 2003.

[5] G. Elidan and A. Globerson. UAI 2010 approximate inference challenge. `http://www.cs.huji.ac.il/project/UAI10/`.

[6] P. Jégou and C. Terrioux. Decomposition and good recording for solving max-CSPs. In *ECAI*, pages 196–200, 2004.

[7] K. Kask and R. Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. *Artif. Intell.*, 129(1-2):91–131, 2001.

[8] J. Larrosa and T. Schiex. Solving weighted CSP by maintaining arc consistency. *Artif. Intell.*, 159(1-2):1–26, 2004.

[9] R. Marinescu and R. Dechter. AND/OR Branch-and-Bound search for combinatorial optimization in graphical models. *Artif. Intell.*, 173(16-17):1457–1491, 2009.

[10] R. Marinescu and R. Dechter. Memory intensive AND/OR search for combinatorial optimization in graphical models. *Artif. Intell.*, 173(16-17):1492–1524, 2009.

[11] N. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, 1998.

[12] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.