Anytime AND/OR Depth-first Search for Combinatorial Optimization

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

One popular and efficient scheme for solving combinatorial optimization problems over graphical models *exactly* is depth-first Branch and Bound. However, when the algorithm exploits problem decomposition using AND/OR search spaces, its anytime behavior breaks down. This article 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 new search scheme to address this issue while maintaining desirable DFS memory properties, and 3) analyzes and demonstrates its effectiveness through comprehensive empirical evaluation. Our work is applicable to *any* problem that can be cast as search over an AND/OR search space.

Keywords: combinatorial optimization, graphical models, Bayesian and constraint networks, anytime performance, AND/OR search, problem decomposition.

1. Introduction

Max-product problems over graphical models, generally known as MPE (most probable explanation) or MAP (maximum a posteriori) inference, 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 [15] or the mini-bucket heuristic [6,16], 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 [19] or an equivalent representation. The efficiency of these algorithms was established in several evaluations, including recent UAI competitions [7], and their properties when used for exact computation are well documented [10,16,17].

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 due to the generally exponential size of the algorithm's OPEN list; moreover, they can only provide a solution at termination [17]. Depth-first search is therefore often preferred because of its flexibility in working with bounded memory – the OPEN list of nodes grows linearly – 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 [24].

Indeed, in the 2010 UAI Approximate Inference Challenge participating Branch and Bound solvers performed competitively with respect to 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 article will demonstrate that the issue is rooted in the underlying AND/OR search space.

These search spaces were originally introduced to graphical models to facilitate problem decomposition during search (e.g. [5]) and 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.

We observe that under certain conditions, namely if only one of the decomposed subproblems is "hard",

*Corresponding author: 4099 Donald Bren Hall, Dept. of Computer Science, University of California, Irvine, CA 92697, U.S.A. E-mail: lotten@ics.uci.edu, Fax: +1-949-824-4056.

AI Communications
ISSN 0921-7126, IOS Press.

this adverse effect can be mitigated by processing subproblems in a suitable order, which we demonstrate empirically.

This article'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 is conducted on a variety of benchmark domains, including haplotype computation problems in genetic pedigrees, random grid networks, and protein side-chain prediction instances. We compare BRAOBB against one of the best variants of AND/OR branch and Bound search, AOBB [16], and against an "ad hoc" fix that we suggest – the latter algorithm relies on a heuristic to quickly find a solution to each subproblem before reverting to depth-first search. We furthermore compare against a state-of-the-art stochastic local search solver, which is specifically targeted at anytime performance but cannot provide any proof of optimality [9].

The empirical results demonstrate superior anytime behavior of BRAOBB, especially over problematic cases where standard AOBB and its ad hoc fix fail, including several very hard instances from the 2010 UAI Approximate Inference Challenge that were made available and three weighted constraint satisfaction problem instances that are known to be very complex. We also show how combining local search and exhaustive AND/OR search lets us enjoy the benefits of both approaches. Notably, a solver based on this concept recently won all three categories (20 seconds, 20 minutes, and 1 hour) in the MPE track of the PASCAL 2011 Inference Challenge [8], the successor to the 2010 UAI Challenge.

Related Work

The work presented here is focused on optimization problems defined over graphical models. As such our results are also relevant for related schemes like recursive conditioning [4] and value elimination [2] in the area of probabilistic reasoning, or BTD (Backtracking Tree Decomposition [10]) in constraint optimization. In fact, the presented concepts carry over to combinatorial AND/OR search spaces in general.

Second, we note Interleaved Depth-First Search [18], which uses a similar idea of interleaved processing of different branches to mitigate mistakes with respect to successor ordering. However, it was only presented in a general, non-optimization OR search context for constraint satisfaction problems.

Connections can also be made to recent contributions in the area of distributed, multi-agent constraint optimization, where algorithms like NCBB [3] and BnB-ADOPT [23] organize agents along a pseudo tree-like structure, thereby obtaining solutions to independent subproblems in parallel.

Finally, most directly related to the objective of this work is the concept of local search, which can be seen as specifically targeting anytime performance. However, it differs from AOBB and the proposed BRAOBB in that it cannot prove optimality of the solutions it returns. We include the state-of-the-art stochastic local search solver GLS+ in our empirical evaluation [9].

Paper Outline

The remainder of this article is structured as follows: Section 2 introduces the underlying concepts of AND/OR Branch and Bound. Section 3 identifies the central conflict between problem decomposition and anytime performance and provides empirical results where the latter is compromised. The new algorithm Breadth-Rotating AOBB is proposed in Section 4 and its theoretical properties are analyzed. Section 5 presents exhaustive experimental results and analysis using a wide range of example problems as well as summary statistics across more than 500 instances. Section 6 concludes.

2. Background

We consider a MPE (most probable explanation, sometimes also called MAP, maximum a posteriori assignment) problem over a graphical model, defined by

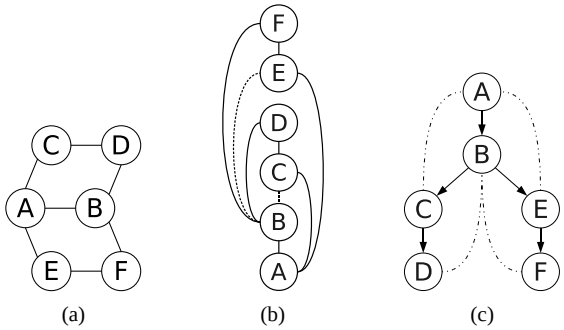
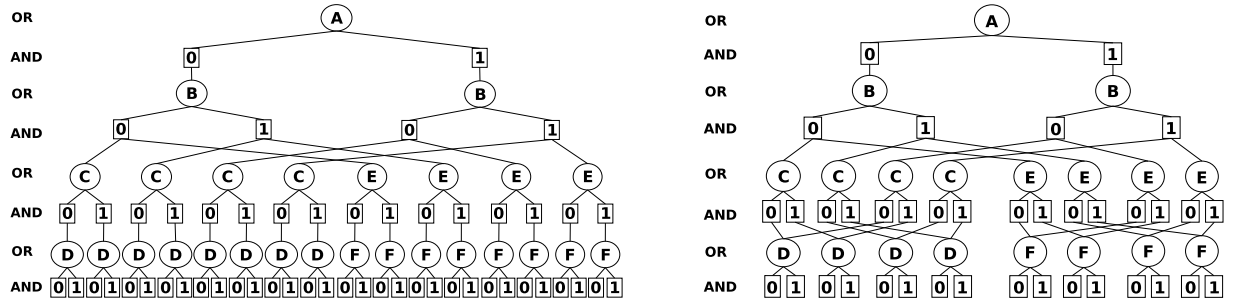


Fig. 1. (a) Example primal graph of a graphical model with six variables, (b) its induced graph along ordering $d = A, B, C, D, E, F$, and (c) a corresponding pseudo tree.



(a) (b)

Fig. 2. (a) AND/OR search tree and (b) context-minimal AND/OR search graph corresponding to the pseudo tree in Figure 1(c).

the tuple $(X, F, D, \max, \prod)$. $F = \{f_1, \ldots, f_r\}$ 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. Another closely related combinatorial optimization problem is the *weighted constraint problem*, where we aim to minimize the sum of all costs, i.e. compute $\min_X \sum_i f_i$ [21]. These tasks have many practical applications but are known to be NP-hard.

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*, the maximum number of earlier neighbors over all nodes [20,12].

EXAMPLE **1.** *Figure 1(a) depicts the (primal) graph of an example graphical model with six variables, A through F. The induced graph for the example problem along ordering $d = A, B, C, D, E, F$ is depicted in Figure 1(b), with two new induced edges, (B, C) and (B, E). Its induced width is 2.*

Different orderings will vary in their induced width; finding an ordering of minimal induced width is known to be equally NP-hard. In practice heuristics like *min-fill* or *mindegree* have proven to produce reasonable approximations [14,13].

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 [5]. The search space is defined using a *pseudo tree* of the graph, which captures problem decomposition as follows:

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.*

EXAMPLE **2.** *A pseudo tree for the example in Figure 1(a) is shown in Figure 1(c), corresponding to the induced graph in Figure 1(b) along ordering $d = A, B, C, D, E, F$. Note how B has two children, capturing the fact that the two subproblems over C, D and E, F, respectively, are independent once A and B have been instantiated.*

2.1. AND/OR Search Trees

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 [5]. 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.

EXAMPLE **3.** *Figure 2(a) shows the AND/OR search tree resulting from the primal graph in Figure 1(a) when guided by the pseudo tree in Figure 1(c). Note that the AND nodes for B have two children each, representing independent subtrees rooted at C and E, respectively, thereby capturing problem decomposition.*

In general, 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 domain size of variables [5].

2.2. AND/OR Search Graphs

Identical subproblems, identified by their context (the partial instantiation that separates the subproblem from the rest of the network), can be merged, yield-

ing an *AND/OR search graph* [5]. Merging all context-mergeable nodes yields the *context-minimal* AND/OR search graph. It was shown that 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 $\mathcal{T}$ [5].

EXAMPLE **4.** *Figure 2(b) displays the context-minimal AND/OR graph obtained when applying full caching to the AND/OR search tree in Figure 2(a). In particular, the OR nodes for D (with context $\{B, C\}$) and F (context $\{B, E\}$) have two edges converging from the AND level above them, signifying caching (namely, the assignment of A does not matter).*

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 $Sol_{S_{\mathcal{T}}}$.

2.3. Weighted AND/OR Search Spaces

Given an AND/OR search graph, each edge from an OR node X_i to an AND node x_i can be annotated by *weights* derived from the set of cost functions F in the graphical model: the weight $l(X_i, x_i)$ is the combination of all cost functions whose scope includes X_i and is fully assigned along the path from the root to x_i, evaluated at the values along this path. Furthermore, each node n in the AND/OR search graph can be associated with a *value* $v(n)$, capturing the optimal solution cost to the subproblem rooted at n, subject to the current variable instantiation along the path from the root to n. $v(n)$ can be computed recursively using the values of n's successors [5].

2.4. AND/OR Branch and Bound

AND/OR Branch and Bound (AOBB) is a state-of-the-art algorithm for solving optimization problems such as max-product over graphical models [16,17]. 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 (cf. Section 2.5). 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 [16].

Algorithm 1 AND/OR Branch and Bound (AOBB)

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

```
1:  OPEN ← {⟨X_0⟩}
2:  while OPEN ≠ ∅
3:      n ← top(OPEN) // top node from stack, depth-first
4:      if checkpruning(n) = success
5:          prune(n) // perform pruning
6:      else if cachelookup(n) = success
7:          readcache(n) // retrieve cached value
8:      else 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 OPEN
12:     else if n = ⟨X_i, x_j⟩ is AND node
13:         for Y_r ∈ children_T(X_i)
14:             generate OR node ⟨Y_r⟩
15:             add ⟨Y_r⟩ to top of OPEN
16:     if children(n)= ∅  // n is leaf
17:         propagate(n) // upwards in search space
18: return value(⟨X_0⟩) // root node has optimal solution
```

Algorithm 1 shows pseudo code for AOBB: Starting with just the root node $\langle X_0 \rangle$ on the stack, the algorithm iteratively takes the top node n from the stack (line 3). Lines 4–7 try to prune the subproblem below n (by comparing a heuristic estimate of n against the current lower bound) and check the cache to see if the subproblem below n has previously been solved (details in [16]). If neither of these is successful, the algorithm generates the children of n (if any) and pushes them back onto the stack (8–15). If n is a terminal node in the search space (it was pruned, its solution retrieved from cache, or the corresponding X_i is a leaf in $\mathcal{T}$) its value is propagated upwards in the search space, towards the root node (16–17). When the stack eventually becomes empty, the value of the root node $\langle X_0 \rangle$ is returned as the solution to the problem (18).

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.

2.5. Mini-Bucket Heuristics

The heuristic $h(n)$ that we use in our experiments is the mini-bucket heuristic. It is based on mini-bucket elimination, which is an approximate variant of vari-

able elimination and computes approximations to reasoning problems over graphical models [6]. A control parameter i allows a trade-off between accuracy of the heuristic and its time and space requirements – higher values of i yield a more accurate heuristic but take more time and space to compute. It was shown that the intermediate functions generated by the mini-bucket algorithm MBE(i) can be used to derive a heuristic function that is *admissible*, namely in a maximization context it overestimates the optimal cost solution to a subproblem in the AND/OR search graph [11].

3. Anytime Behavior versus Problem Decomposition in AND/OR Search

As a depth-first branch and bound 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

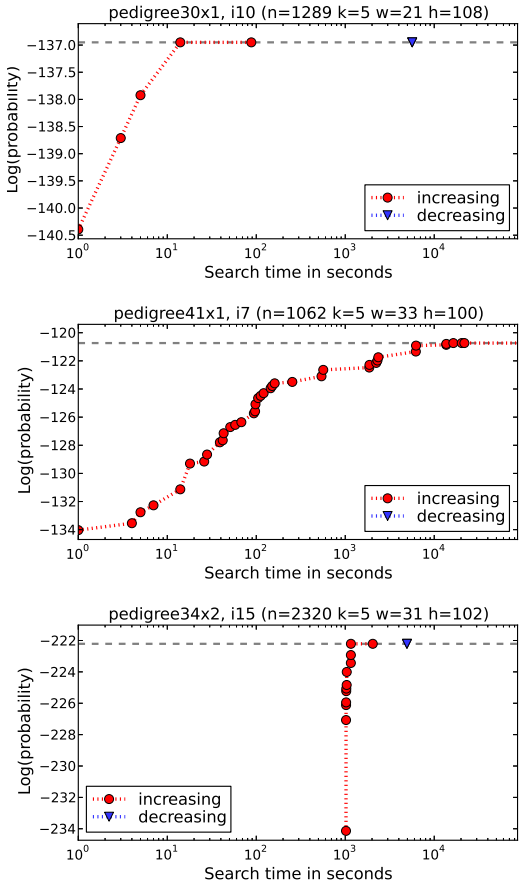


Fig. 3. Impact of subproblem ordering on AOBB. Specified for each network: number of variables n, max. domain size k, induced width w along the chosen ordering, height of the corresponding pseudo tree h. The dashed gray line indicates the optimal solution value.

(in the general description of AOBB the subproblem ordering is left unspecified).

Figure 3 contrasts the anytime behavior of AOBB using this "increasing" subproblem order against the inverse one ("decreasing") by plotting the solution cost generated as a function of time on two example problems (the dashed horizontal line is the optimum cost); 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 find and prove 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 [16]. This general problem is outside the scope of the present paper, however.

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 [11]).

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. This again negates the desired anytime behavior, since the subproblem for which the dive failed will not be reconsidered until the normal depth-first AOBB phase. 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 achieve by rotating through them.

4.1. Subproblem Rotation

More systematically, the algorithm maintains a list of currently open subproblems and repeats the following high-level steps until completion:

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

The threshold in (c) is needed to ensure the algorithm does not get stuck in one large subproblem where the other two conditions, (a) and (b), 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 child subproblems, as illustrated below.

4.2. Algorithm Pseudo Code

Algorithm 2 gives more detailed pseudo code for the scheme (with some details from standard AOBB omitted, cf. Algorithm 1 and [16]). 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 explo-

Algorithm 2 Breadth-Rotating AOBB

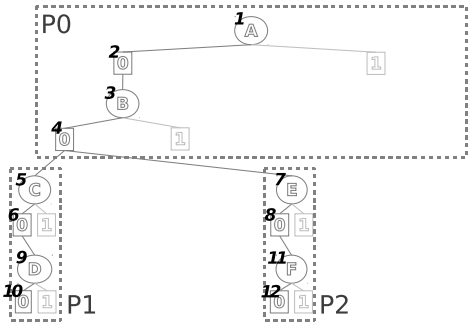
Given: Graphical model $(X, F, D, \max, \prod)$ and pseudo tree $\mathcal{T}$ with root X_o, rotation threshold Z
Output: cost of optimal solution

```
1: ROOT ← {⟨X_0⟩} // generate root subproblem
2: push ROOT to end of GLOBAL
3: while GLOBAL ≠ ∅
4:    LOCAL ← front(GLOBAL) // next subproblem
5:    for z ← 1 to Z or until LOCAL = ∅
       or until childSubprob(LOCAL) ≠ ∅
6:       n ← top(LOCAL) // next node in 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 has optimal solution
```

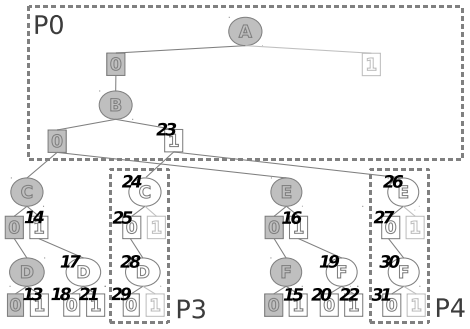
ration across the different branches of the solution tree. The input parameter Z gives the rotation threshold. Each subproblem is itself explored depth-first (via a local last-in-first-out stack of nodes, *LOCAL*); whenever a new level of decomposition is encountered, as captured by the pseudo tree, 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 child subproblems.

4.3. Example Execution

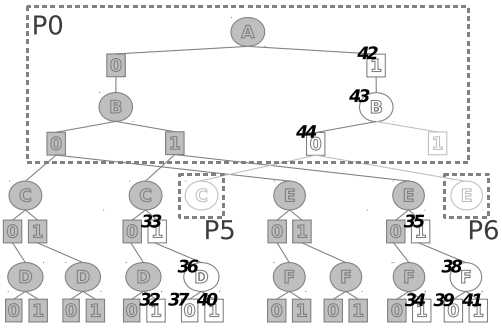
Figure 4 demonstrates the scheme's application ($Z = 2$) to the AND/OR search graph in Figure 2(b) (assuming no pruning). Part (a) shows the first 12 nodes expanded during the first seven iterations of the outer while loop as follows: *(1)* Taking the overall problem as subproblem P0, expand $\langle A\rangle$ 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



(a) Expansion of nodes 1–12



(b) Expansion of nodes 13–31



(c) Expansion of nodes 32–44

Fig. 4. BRAOBB exploration ($Z = 2$) at different stages. Nodes are numbered in order of their expansion.

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.

Figures 4(b) and (c) illustrate 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.

4.4. Analysis of Breadth-Rotating AOBB

In this Section we analyze the Breadth-Rotating AOBB algorithm and its properties and contrast it with standard AOBB.

4.4.1. Correctness, Completeness, and Complexity

Recall that a heuristic function is said to be admissible if it never underestimates (in a maximization scenario) the cost of the optimal solution to a given subproblem. The mini-bucket heuristic satisfies this requirement [11]. Further recall that n denotes the number of problem variables, k the maximum domain size, h the height of the guiding pseudo tree $\mathcal{T}$ with induced width and w^*.

THEOREM **1.** *Breadth-Rotating AOBB is complete and correct assuming an admissible heuristic. Furthermore, 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 $O(n)$. When searching the context-minimal AND/OR search graph (with full caching), time and space complexity are $O(n \cdot k^{w^*})$.*

Proof. Because of the heuristic's admissibility, a subspace is pruned only if it provably cannot yield a better solution than what is already known at this point. Just like standard AOBB the search also remains systematic and all solution trees are considered; the algorithm is guaranteed to eventually terminate and return the optimal solution to the problem.

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 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 worst-case bounds are often very loose, because the Branch and Bound scheme is typically very efficient and prunes large parts of the search space. In particular, we observe that in practice the pruning keeps the cache tables from reaching their worst-case exponential size.

4.4.2. Significance of Z

The rotation threshold Z acts as a safeguard against overly large subproblems, that take a long time to solve optimally (condition (a), Section 4.1) or where recursive decomposition does not occur for a long time (condition (b)). Being "stuck" in this way could again impair anytime performance, which is why we limit the number of node expansions before enforcing a rotation. As we see in Section 5, however, practical problems typically exhibit frequent subproblem branching, so a relatively large threshold of $Z = 1000$ or similar is sufficient, if rarely reached.

4.4.3. Maximum Queue Size

It is easy to see that the maximum number of entries in the GLOBAL queue is dependent on the number of branchings in the solution tree, corresponding to pseudo tree nodes with more than one successor, since that is where the algorithm generates new child subproblems (lines 17–20, Algorithm 2). In particular, decomposition does not occur along the chains in the pseudo tree, i.e., paths where no node (besides the end points) has outdegree greater than 1. The number of queue entries is thus bounded by the number of maximal chains in the pseudo tree. We can thus state the following:

THEOREM **2.** *When exploring an AND/OR search space using a guiding pseudo tree $\mathcal{P}$ with l leaves, the number of subproblems in the GLOBAL queue of BRAOBB is bounded by $2l - 1$.*

Proof. Since $\mathcal{T}$ is a tree with l leaves, there can be at most $l - 1$ branchings (nodes with outdegree greater than 1) in $\mathcal{T}$ to yield these leaves. Each such branching sits at the end of one maximal chain. Together with the leaf chains, we obtain an upper bound of $2l - 1$ maximal chains. □

4.4.4. Comparison with Standard AOBB

We expect the anytime performance of BRAOBB to 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.

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).

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 time limit); unless noted otherwise subproblems were ordered by increasing width (cf. Section 3.1). We ran "plain" AOBB, AOBB with the dive extension (cf. Section 3.2), and Breadth-Rotating AOBB as presented in Section 4; we also included OR Branch and Bound (without problem decomposition) as a baseline. In addition, we ran an advanced stochastic local search (SLS) algorithm [9], both on its own and as a initialization step for our own exhaustive search; in particular we consider the GLS+ implementation from [9], for which source code is publicly available. Note that as an incomplete search scheme, it does not provide a proof of optimality and always runs for the full 24 hours in our experiments. All algorithms are implemented in C++ and were run on 2.67 GHz Intel Xeon CPUs with 2GB of RAM per core.

Our initial test set (instance name suffix "x1") is comprised of 19 genetic linkage pedigree problems, 50 randomly generated grid networks, 8 mastermind game instances (all part of the UAI 2008 evaluation[1]) as well as 66 protein side-chain prediction problems (taken from [22]). However, several of these instances are relatively simple or have only one complex subproblem, which renders them less interesting for the purpose of this work. Namely, plain AOBB (with subproblems ordered by increasing width) already yields good anytime performance and neither the dive extension nor BRAOBB can provide significant improvements. Hence we also created additional versions of each network with two or three identical copies connected at the root (thus ensuring the presence of more than one complex subproblem), signified by the "x2" and "x3" suffix, respectively. This yields a total of 57 pedigree (each run with three different heuristic strengths), 150 grid, 24 mastermind, and 198 protein prediction instances and resulting in over 90,000 CPU hours worth of experiments.

We present detailed performance results for a representative subset of the problem instances in Section 5.1 before Section 5.2 compiles summary statistics across all 543 problem instances. Section 5.3 evaluates how our proposed exhaustive search method can benefit from limited local search. Section 5.4 presents results on two additional very challenging problem domains, protein-protein interaction from the UAI 2010 Challenge and CELAR radio link frequency assignments. Finally, Section 5.5 analyzes a number of algorithm parameters empirically.

5.1. Detailed Performance Analysis

Figure 5 shows anytime profiles for some of the more interesting initial problem instances ("x1" suffix), while Figure 6 presents results on a subset of problem instances with more than one complex subproblem ("x2" and "x3" suffix). For every problem instance, the plot title specifies number of variables n , max. domain size k , induced width w along the chosen ordering, and height of the corresponding pseudo tree h . If known, the optimal solution value is indicated by a gray dashed horizontal line. The title of each plot also notes the mini-bucket i-bound; this was typically chosen to fit a 1GB memory limit, except for pedigree instances, where three different heuristic strengths ($i = 7, 10, 15$) were applied for each instance.

In Figure 5 as well as 6 we note that OR Branch and Bound finds an early lower bound in some cases, but generally provides little improvement over time and never gets close to the optimum. The dive extension shows acceptable anytime behavior only on half the instances shown, 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 in both Figures 5 and

[1] http://graphmod.ics.uci.edu/uai08/

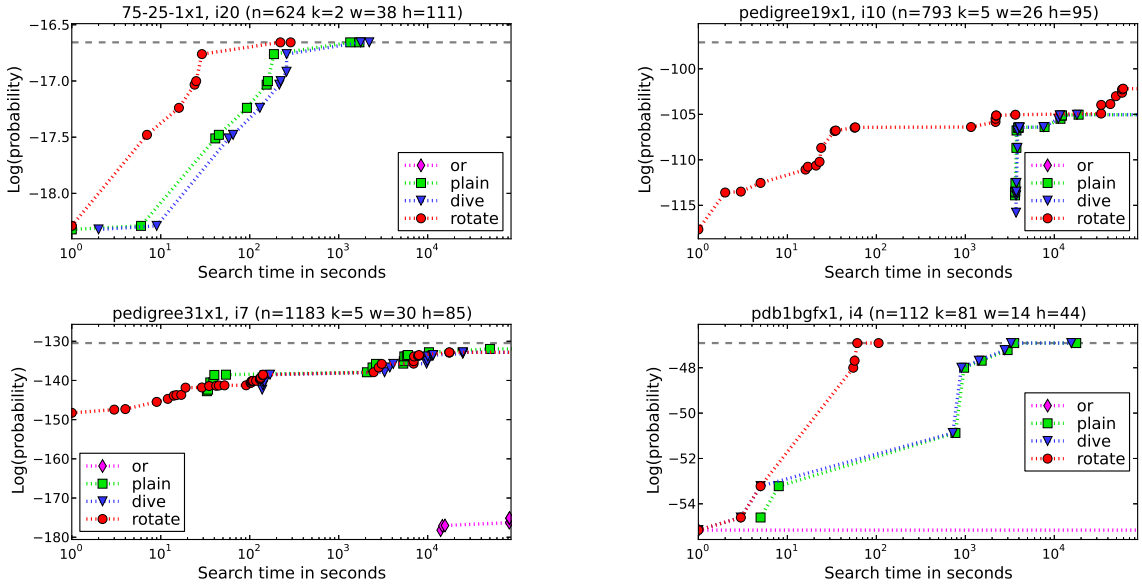


Fig. 5. Anytime profiles of plain AOBB ("plain"), AOBB with subproblem dive ("dive"), Breadth-Rotating AOBB ("rotate"), and OR Branch and Bound ("or") on selected instances from the initial set of problems ("x1" suffix; 1 grid, 2 pedigree, 1 side-chain prediction).

6, often by a large margin; in all but one case the first solution is produced more or less instantly, even on pedigree51x3 (Fig. 6), where "plain" and "dive" do not return anything within 24 hours. Note that mastermind instances are highly deterministic and the initial solution is already optimal – but again BRAOBB returns it first.

5.2. Summary Statistics

Table 1 summarizes the entire set of experiments by showing, at different points of time, the number of instances for 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 502 instances (out of 543), compared to just 232 for plain AOBB, 339 for the dive extension, and 424 for local search; performance remains superior to the other AOBB versions and very competitive with local search for higher time bounds.

We also note that BRAOBB finds the optimal solution quicker than the other schemes (with the exception of local search on side-chain prediction), e.g. for overall 266 instances after 10 seconds (versus 220 for plain). Similarly, in the full 24 hours, BRAOBB found the optimal solution to 495 instances, versus 486 for plain and just 316 for local search.

Furthermore, we see that plain AOBB sometimes has a slight edge in terms of proving optimality, e.g. 172 proved optimal at 10 seconds versus 153 for BRAOBB, confirming that exploring subproblems concurrently can slightly impair the pruning (cf. Section 4.4). Again, as an incomplete solver local search proves no optimality at all.

We observe that SLS does very well on the side-chain prediction networks – these problems have only a few hundred variables but a large max. domain size of 81. On the other problem classes, however, with thousands of variables and smaller max. domains, BRAOBB shows better performance, in particular with respect to finding the optimal solution. The reason for this lies in the heuristic used by AOBB: mini-bucket space complexity is $O(nk^i)$ – large domain size bounds k thus necessitate a significantly lower i-bound ($i = 3$ in case of the side-chain prediction problems), which leads to far less accurate heuristics.

5.3. Combining Local and Exhaustive Search

Looking again at Table 1, we notice that SLS can sometimes find solutions more quickly than any kind of exhaustive search – in particular it has found a solution for all 543 problems after 10 seconds (versus 514 instances for BRAOBB and just 293 for plain AOBB). As outlined above, however, SLS is often quickly outperformed by AOBB and BRAOBB in terms of find-

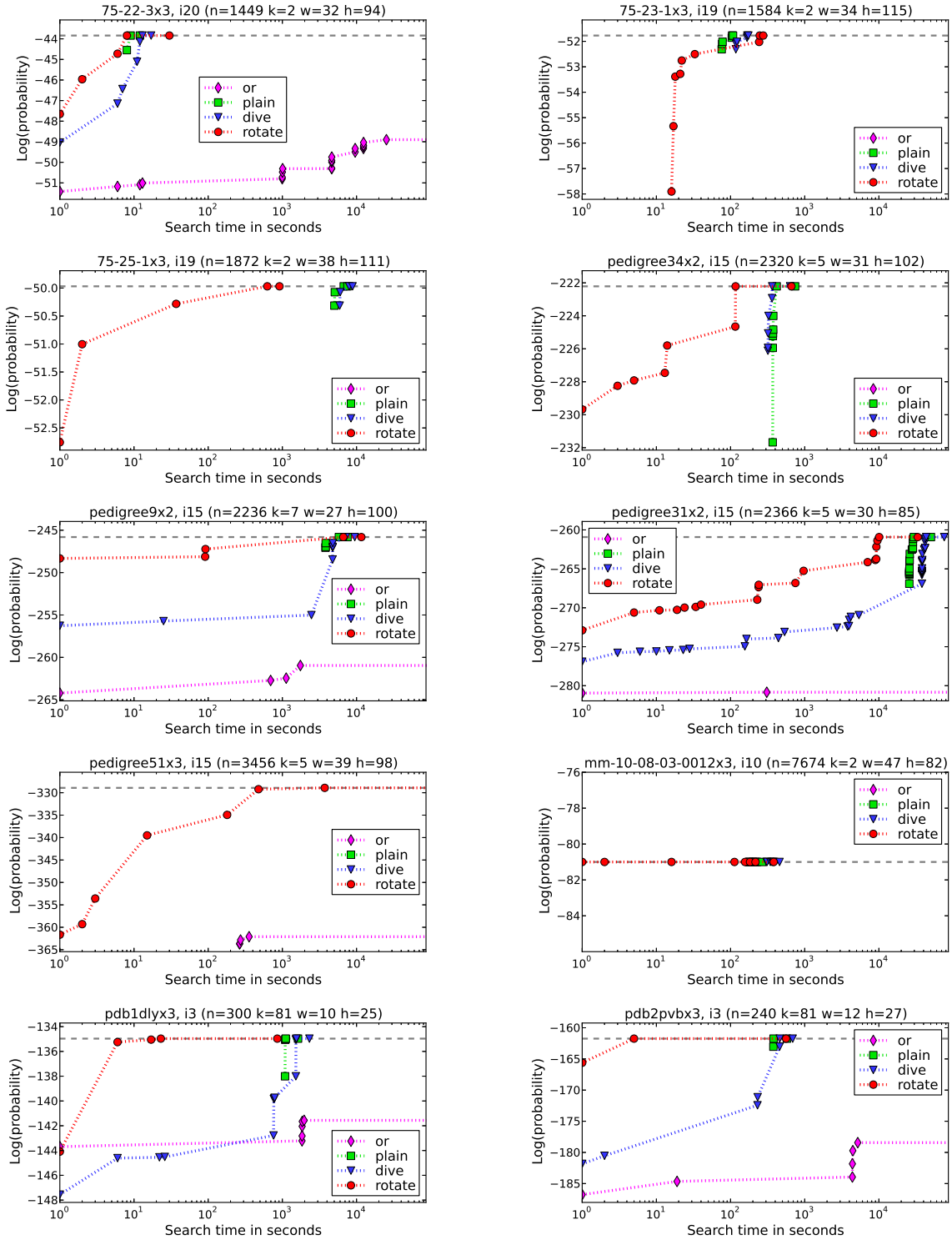


Fig. 6. Anytime profiles of plain AOBB (“plain”), AOBB with subproblem dive (“dive”), Breadth-Rotating AOBB (“rotate”), and OR Branch and Bound (“or”) on selected instances with more than one complex subproblem (“x2” or “x3” suffix; 3 grids, 4 pedigree, 1 mastermind, 2 side-chain prediction).

Table 1

Summary statistics over 543 instances for OR Branch and Bound, plain AOBB, AOBB with dive extension, breadth-rotating AOBB, stochastic local search, as well as plain and breadth-rotating AOBB with 10 seconds of initial local search. In each case we list 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)							
or	77 / 6 / 6	78 / 10 / 8	82 / 11 / 9	84 / 14 / 12	87 / 15 / 13	91 / 18 / 18	94 / 23 / 22
plain	65 / 31 / **24**	77 / 45 / **41**	85 / **55** / **45**	99 / 72 / **64**	105 / 80 / **75**	113 / 94 / 87	136 / 125 / **119**
dive	83 / 24 / 17	95 / 43 / 36	102 / 51 / 43	113 / 67 / 61	119 / 77 / 72	127 / 92 / 87	136 / 120 / 113
rotate	**157** / **38** / 22	161 / **50** / 37	162 / **55** / 44	162 / 69 / 59	163 / 80 / 71	165 / 97 / 87	168 / 132 / 112
sls	144 / 9 / 0	**171** / 21 / 0	**171** / 24 / 0	**171** / 39 / 0	**171** / 66 / 0	**171** / 78 / 0	**171** / 78 / 0
plain+sls	146 / 9 / 0	**171** / 13 / 0	**171** / 30 / 12	**171** / 74 / 62	**171** / 84 / 74	**171** / 100 / **88**	**171** / 129 / 118
rotate+sls	148 / 10 / 0	**171** / 13 / 0	**171** / 45 / 15	**171** / **80** / 57	**171** / **93** / 72	**171** / **106** / 86	**171** / **136** / 111
Grid networks (150 total)							
or	45 / 1 / 0	47 / 1 / 0	51 / 2 / 0	55 / 3 / 2	62 / 7 / 4	67 / 15 / 10	78 / 25 / 24
plain	47 / 15 / **3**	65 / 39 / **23**	77 / 52 / **40**	94 / 76 / **69**	109 / 97 / 89	138 / 135 / **133**	149 / **149** / **149**
dive	52 / 11 / 1	65 / 32 / 13	72 / 44 / 27	94 / 70 / 64	106 / 91 / 84	134 / 129 / 123	149 / **149** / **149**
rotate	**129** / **24** / 1	133 / **44** / 9	136 / **59** / 23	140 / 82 / 65	143 / **107** / **90**	147 / **139** / 132	149 / **149** / **149**
sls	81 / 0 / 0	**150** / 0 / 0	**150** / 0 / 0	**150** / 2 / 0	**150** / 6 / 0	**150** / 21 / 0	**150** / 21 / 0
plain+sls	76 / 0 / 0	**150** / 0 / 0	**150** / 10 / 1	**150** / 79 / 67	**150** / 97 / 88	**150** / 135 / 132	**150** / **149** / **149**
rotate+sls	83 / 0 / 0	**150** / 0 / 0	**150** / 14 / 0	**150** / **83** / 64	**150** / 106 / **90**	**150** / **139** / 131	**150** / **149** / **149**
Protein side-chain prediction networks (198 total)							
or	**198** / 78 / 49	**198** / 79 / 52	**198** / 80 / 53	**198** / 82 / 57	**198** / 89 / 61	**198** / 90 / 70	**198** / 99 / 82
plain	114 / 95 / 78	120 / 102 / **85**	124 / 106 / **87**	133 / 117 / 102	145 / 132 / 116	168 / 163 / 148	191 / 188 / 186
dive	**198** / 102 / 76	**198** / 108 / 84	**198** / 110 / 86	**198** / 122 / 100	**198** / 133 / 112	**198** / 161 / 141	**198** / 185 / 181
rotate	**198** / 128 / **79**	**198** / 133 / **85**	**198** / 136 / 86	**198** / 151 / **104**	**198** / 165 / **120**	**198** / 180 / **157**	**198** / 190 / **190**
sls	**198** / **193** / 0	**198** / **198** / 0	**198** / **198** / 0	**198** / **198** / 0	**198** / **198** / 0	**198** / **198** / 0	**198** / **198** / 0
plain+sls	**198** / **193** / 0	**198** / **198** / 0	**198** / **198** / 51	**198** / **198** / 81	**198** / **198** / 98	**198** / **198** / 132	**198** / **198** / 169
rotate+sls	**198** / 191 / 0	**198** / **198** / 0	**198** / **198** / 47	**198** / **198** / 83	**198** / **198** / 104	**198** / **198** / 140	**198** / **198** / 172
Mastermind networks (24 total)							
or	0 / 0 / **0**	0 / 0 / **0**	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
plain	6 / 6 / **0**	7 / 7 / **0**	7 / 7 / 0	9 / 9 / **3**	12 / 12 / 6	20 / 20 / 18	**24** / **24** / **24**
dive	6 / 6 / **0**	7 / 7 / **0**	8 / 8 / **1**	10 / 10 / **3**	12 / 12 / 6	23 / 23 / **22**	**24** / **24** / **24**
rotate	**18** / **16** / **0**	**18** / **16** / **0**	18 / 16 / 0	19 / **19** / **3**	**24** / **24** / **10**	**24** / **24** / 21	**24** / **24** / **24**
sls	1 / 1 / **0**	**18** / 14 / **0**	**24** / **18** / 0	**24** / 18 / 0	**24** / 19 / 0	**24** / 19 / 0	**24** / 19 / 0
plain+sls	1 / 1 / **0**	**18** / 13 / **0**	**24** / 13 / 0	**24** / 14 / **3**	**24** / 16 / 6	**24** / 22 / 18	**24** / **24** / **24**
rotate+sls	2 / 2 / **0**	**18** / 13 / **0**	**24** / 13 / 0	**24** / 17 / **3**	**24** / **24** / 9	**24** / **24** / 21	**24** / **24** / **24**
Overall (543 total)							
or	320 / 85 / 55	323 / 90 / 60	331 / 93 / 62	337 / 99 / 71	347 / 111 / 78	356 / 123 / 98	370 / 147 / 128
plain	232 / 147 / **105**	269 / 193 / **149**	293 / 220 / **172**	335 / 274 / **238**	371 / 321 / 286	439 / 412 / 386	500 / 486 / **478**
dive	339 / 143 / 94	365 / 190 / 133	380 / 213 / 157	415 / 269 / 228	435 / 313 / 274	482 / 405 / 373	507 / 478 / 467
rotate	**502** / **206** / 102	510 / **243** / 131	514 / 266 / 153	519 / 321 / 231	528 / 376 / **291**	534 / 440 / **397**	539 / 495 / 475
sls	424 / 203 / 0	**537** / 233 / 0	**543** / 240 / 0	**543** / 257 / 0	**543** / 289 / 0	**543** / 316 / 0	**543** / 316 / 0
plain+sls	421 / 203 / 0	**537** / 224 / 0	**543** / 251 / 64	**543** / 365 / 213	**543** / 395 / 266	**543** / 455 / 370	**543** / 500 / 460
rotate+sls	431 / 203 / 0	**537** / 224 / 0	**543** / **270** / 62	**543** / **378** / 207	**543** / **421** / 275	**543** / **467** / 378	**543** / **507** / 456

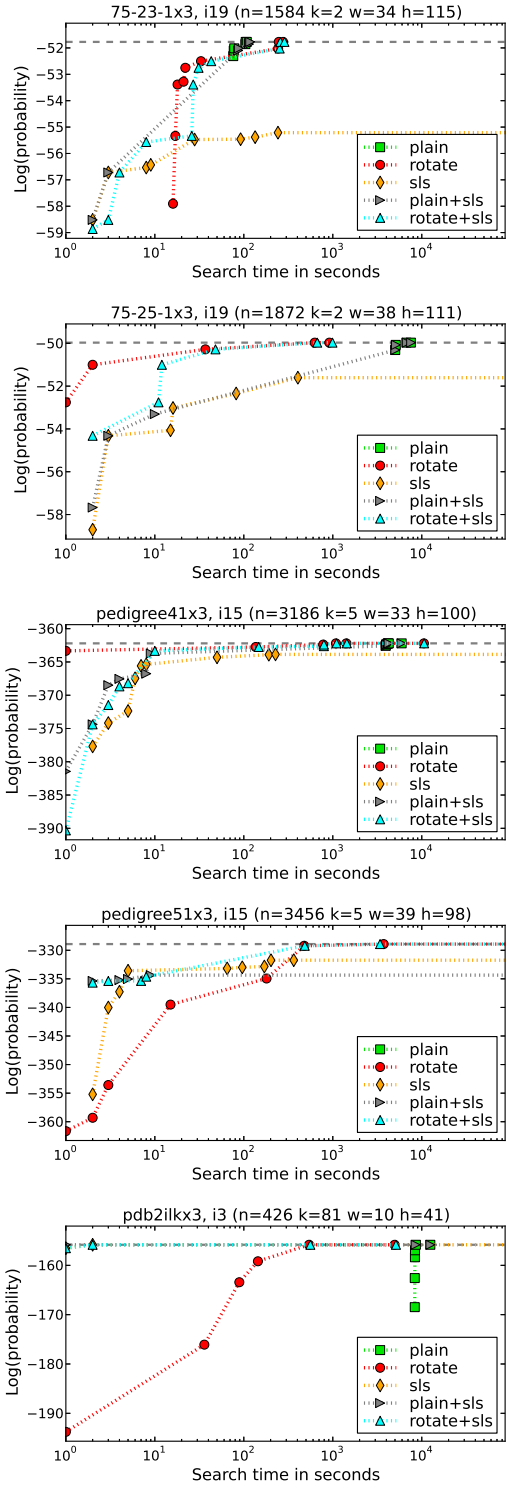


Fig. 7. Anytime profiles comparing exhaustive AOBB against SLS and combinations of the two on select problem instances.

ing the optimal solution, let alone proving optimality (which is impossible with local search). For instance, local search has found only 257 optimal solutions after 1 minute (versus 321 for BRAOBB and 274 for plain AOBB) or 316 at the 24 hour timeout (compared to 495 for BRAOBB, 486 for plain AOBB).

We have thus devised simple, combined schemes that run local search for 10 seconds as a preprocessing step; the resulting solution is then used as an initial lower bound for the exhaustive search. Results for plain AOBB and BRAOBB augmented in this way, denoted "plain+sls" and "rotate+sls", respectively, are included in Table 1. Figure 7 also shows detailed anytime profiles on five problem instances, comparing local and exhaustive search, as well as their combinations.

Indeed we see "plain+sls" and "rotate+sls" match local search in terms of initial performance and quickly returning a solution (deviations here are due to randomization). Just as for SLS, however, solution quality can be inferior to BRAOBB (see instances 75-25-1x3 and pedigree41x3, Fig. 7), but after 10 seconds the combined schemes quickly catch up to plain AOBB and BRAOBB, respectively, as local search preprocessing finishes and exhaustive search takes over. Here "rotate+sls" has the edge over "plain+sls" in terms of getting to and proving optimality. Overall we therefore believe that "rotate+sls" best combines the benefits of the two search paradigms.

5.4. Additional Problem Domains

In addition to the benchmark set used in the previous sections, we consulted two additional, very challenging problem domains: protein-protein interaction instances (seven problems were made available from the UAI 2010 Challenge) and three CELAR radio link frequency assignment instances converted from weighted constraint satisfaction problems (see e.g. [1]), for all of which optimal solutions are unavailable.

Figure 8 shows anytime profiles for three of the protein-protein interaction networks while Figure 9 presents the results for the three CELAR problems; plain AOBB fails to produce any solution within the 24 hour time limit and is thus omitted. In all cases BRAOBB and even the initial subproblem dive successfully restore the anytime performance – as before, BRAOBB is typically far superior.

For protein-protein interaction BRAOBB also outperforms SLS, which seems to struggle with the large number of problem variables and never improves upon its initial solution. "rotate+sls" suffers from this as

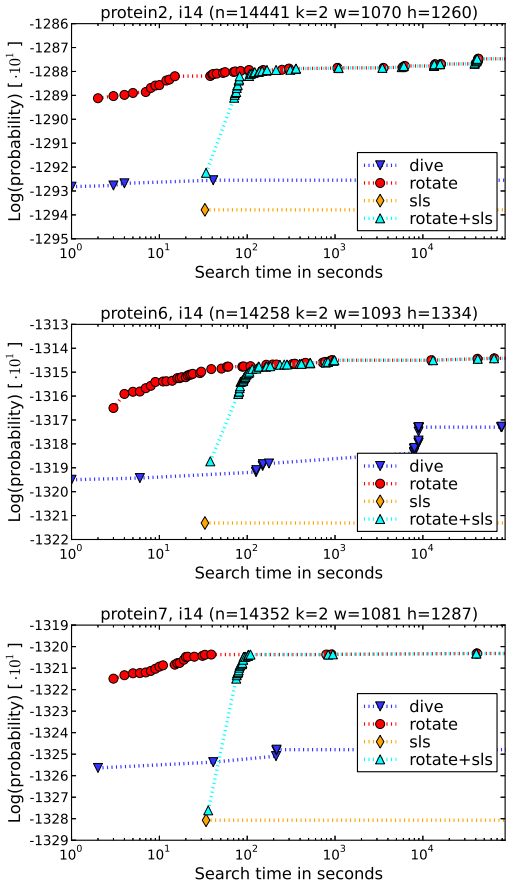


Fig. 8. Anytime profiles on very hard protein-protein interaction instances from the UAI'10 and PASCAL'11 Inference Challenges. "plain" and "or" did not produce any solutions within 24 hours.

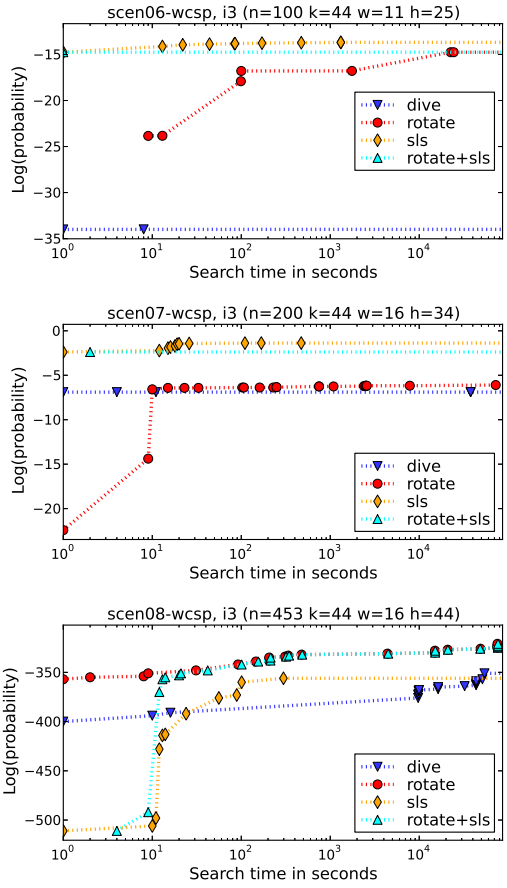


Fig. 9. Anytime profiles on three very hard WCSPs (encoded as MPE problems) from the CELAR radio link frequency assignment domain. "plain" and "or" did not produce any solution within 24 hours.

well during its local search preprocessing, but quickly catches up to BRAOBB, as seen in previous sections.

CELAR networks have fewer variables and large domain sizes ($k = 44$), which as before results in a weaker mini-bucket heuristic (with lower i-bound) for AOBB. This gives an advantage to SLS, which does not rely on the heuristic and is able to outperform BRAOBB in two of the tree plots shown in Figure 9. Yet in neither of these cases does SLS improve significantly over time, so the combined "rotate+sls" again presents the best compromise.

5.5. *BRAOBB Analysis*

In the following we investigate several aspects of BRAOBB more closely and compare some of its properties to plain AOBB empirically.

5.5.1. *Heuristic Accuracy*

We've demonstrated above how the performance of BRAOBB can suffer if the heuristic is very inaccurate (i.e., the i-bound of the mini-buckets is low) when comparing against local search. Here we compare the different AOBB schemes with respect to their sensitivity for the heuristic's accuracy. Figure 10 contrasts plain AOBB, dive, and BRAOBB each with two different heuristics, parametrized by the mini-bucket i-bound. In both cases plain AOBB fails or does very poorly due to problem decomposition; AOBB with dive depends very much on the heuristic and fails or does poorly with the weaker one. BRAOBB, however, exhibits acceptable anytime behavior even with the weaker heuristic and is more robust.

5.5.2. *Subproblem Ordering*

Going back to Section 3.1, Figure 11 compares the performance of BRAOBB with subproblems ordered

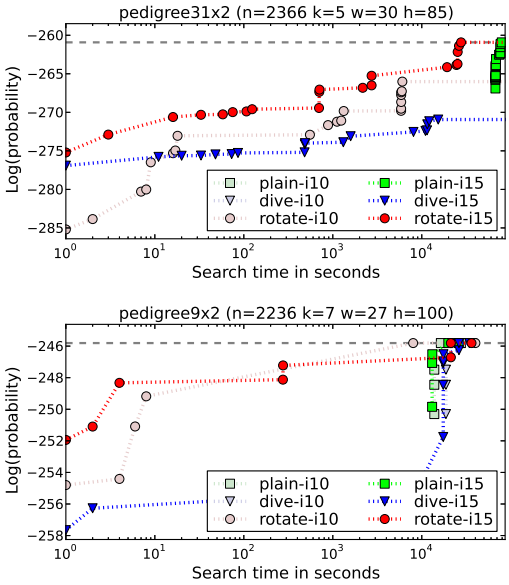


Fig. 10. Impact of heuristic accuracy on anytime performance: comparing i-bound 10 and 15 on two pedigree instances.

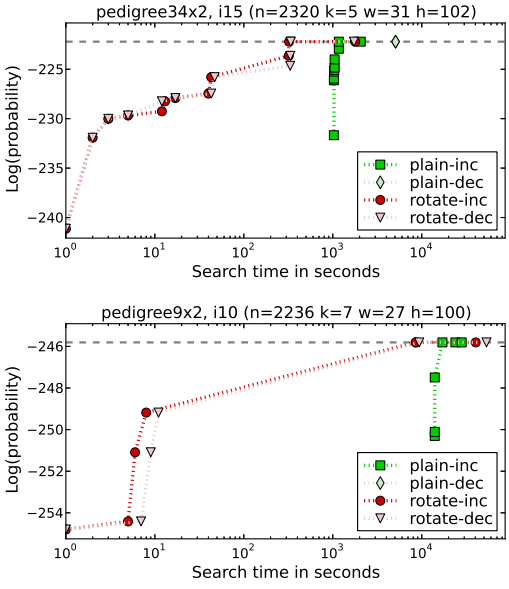


Fig. 11. Impact of subproblem ordering on anytime performance: subproblems ordered by increasing and decreasing induced width for plain AOBB and BRAOBB.

by increasing and decreasing width. In contrast to plain AOBB (included in Figure 11 for reference) our new scheme is very robust and delivers nearly the same performance in both cases.

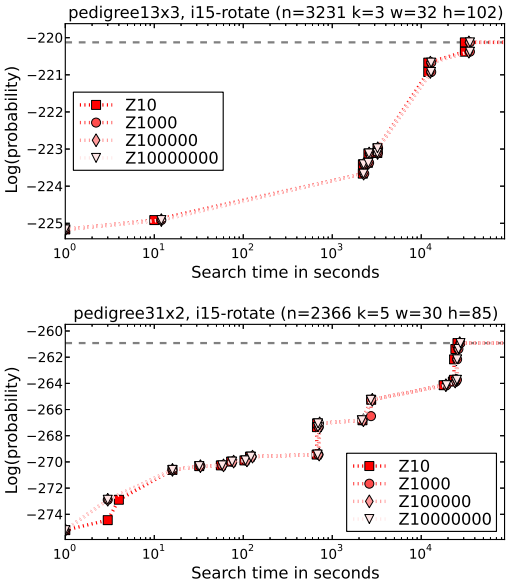


Fig. 12. Impact of rotation threshold Z: running BRAOBB with $Z \in \{10, 1000, 100000, 10000000\}$.

5.5.3. Rotation Threshold

Finally, we conducted experiments with different values for the rotation threshold Z in Algorithm 2, ranging from 10 to 10 million node expansions. As shown in Figure 12, no significant difference in practical performance is visible. To investigate further, we conducted a number of BRAOBB runs where we recorded the number of node expansions between stack rotations. Representative results are shown in Figure 13 in the form of histograms: noting the vertical log scale, we observe that the majority of stack rotations happens after only very few node expansions, further confirming the analysis in Section 4.4.

6. Summary

Exploiting problem decomposition in search methods has been proven to yield significantly better overall complexity in many cases. Yet this article 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. Specifically, to obtain an overall result, a partial solution is required from every independent subproblem, which we have shown to be in direct contradiction to the depth-first, consecutive processing of subproblems.

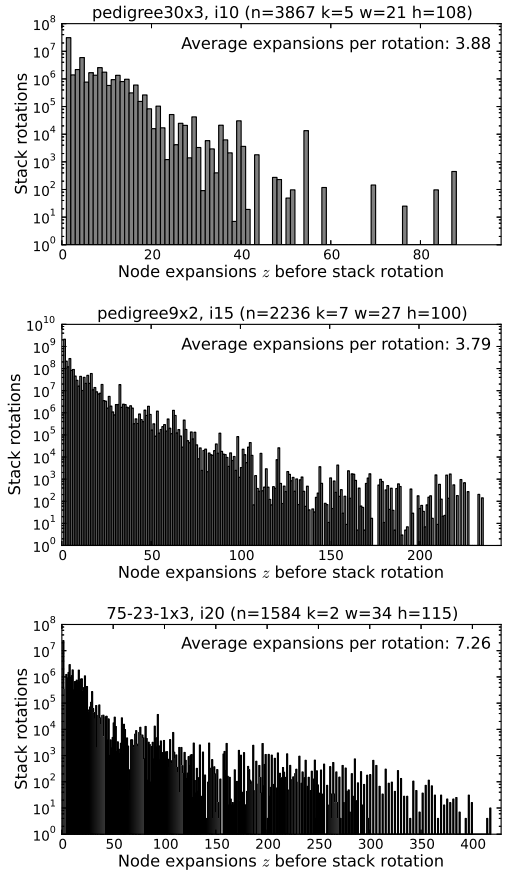


Fig. 13. Histograms showing number of node expansions between stack rotations for three different runs of BRAOBB (note the vertical log scale). The rotation threshold was set to $Z = 1000$ in each case, but evidently never reached in practice.

We devised a "quick fix" that employs an initial greedy subproblem dive, but whose performance we found to be lacking due to 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. Yet we have shown that it retains 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 an exhaustive set of successful experiments on problems from several different domains. The results confirmed the vastly improved anytime performance of BRAOBB, especially in cases where standard depth-first Branch and Bound and its "ad hoc" extensions fail. We also showed BRAOBB to be very competitive with a state-of-the-art stochastic local search algorithm, in many cases even surpassing it (unless the mini-bucket heuristic is very inaccurate). In addition, we demonstrated how the two paradigms can be combined to get the best of both worlds. The power of this enhanced algorithm was recently further exemplified by placing first in all three categories of the MPE track of the PASCAL 2011 Inference Challenge [8].

Possible future directions include more elaborate rotation schemes, for instance assigning the rotation threshold dynamically based on subproblem-specific heuristic estimates. Given the observations in Section 5.5.3, however, it is unclear whether these would have much a major impact in practice.

Acknowledgments

We thank the reviewers of this article and of an earlier conference version for their feedback and constructive suggestions. This work was partially supported by NSF grants IIS-0713118, IIS-1065618 and NIH grant 5R01HG004175-03. Also supported in part by the Israeli Science Foundation.

References

[1] D. Allouche, S. de Givry, and T. Schiex. Towards parallel non serial dynamic programming for solving hard weighted CSP. In *CP*, pages 53–60, 2010.

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

[3] A. Chechetka and K. P. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *AAMAS*, pages 1427–1429, 2006.

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

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

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

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

[8] G. Elidan, A. Globerson, and U. Heinemann. PASCAL 2011 probabilistic inference challenge. `http://www.cs.huji.ac.il/project/PASCAL/`.

[9] F. Hutter, H. H. Hoos, and T. Stützle. Efficient stochastic local search for MPE solving. In *IJCAI*, pages 169–174, 2005.

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

[11] 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.

[12] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree decompositions for reasoning in graphical models. *Artif. Intell.*, 166(1-2):165–193, 2005.

[13] K. Kask, A. Gelfand, L. Otten, and R. Dechter. Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In *AAAI*, 2011.

[14] U. Kjaerulff. Triangulation of graphs – algorithms giving small total state space. Technical report, Aalborg University, 1990.

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

[16] 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.

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

[18] P. Meseguer. Interleaved depth-first search. In *IJCAI*, pages 1382–1387, 1997.

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

[20] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[21] R. J. Wallace. Analysis of heuristic methods for partial constraint satisfaction problems. In *CP*, pages 482–496, 1996.

[22] C. Yanover, O. Schueler-Furman, and Y. Weiss. Minimizing and learning energy functions for side-chain prediction. *Journal of Computational Biology*, 15(7):899–911, 2008.

[23] W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound dcop algorithm. *J. Artif. Intell. Res. (JAIR)*, 38:85–133, 2010.

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