Advancing AND/OR Search for Optimization Using Diverse Principles

Radu Marinescu and Rina Dechter

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

Abstract. In recent years, several Branch-and-Bound and best-first search algorithms were developed to explore the AND/OR search graph for solving general constraint optimization problems. Previous work showed the tremendous gain obtained by exploiting problem's decomposition (using AND nodes), equivalence (by caching) and irrelevance (via the power of lower bound heuristics). In this paper, we show the additional improvements that can be gained by bringing together all the above, as well as diverse refinements and optimizing principles such as exploiting determinism via constraint propagation, using good initial upper bounds generated via stochastic local search and improving the quality of the guiding pseudo tree. We illustrate our results using a number of benchmark networks, including the very challenging ones that arise in genetic linkage analysis.

1 Introduction

Constraint satisfaction problems (CSPs) provide a formalism for formulating many interesting real world problems as an assignment of values to variables, subject to a set of constraints. A constraint optimization problem (COP) is defined as a regular CSP augmented with a set of cost functions (called soft constraints) indicating preferences. The aim of constraint optimization is to find a solution to the problem whose cost, expressed as the sum of the cost functions, is minimized or maximized. The classical approach to solving COPs is the Branch-and-Bound method which maintains the best solution found so far and discards any partial solution which cannot improve on the best.

The AND/OR Branch-and-Bound search (AOBB) introduced in [1] is a Branch-and-Bound algorithm that explores an AND/OR search tree for graphical models [2], in a depth-first manner. The AND/OR Branch-and-Bound search with caching (AOBB-C) [3] allows saving previously computed results and retrieving them when the same subproblem is encountered again. The algorithm explores the context minimal AND/OR graph. A best-first AND/OR search algorithm (AOBF-C) that traverses the AND/OR graph was also explored [4]. Earlier empirical evaluations demonstrated (1) the impact of AND decomposition, (2) the impact of caching, (3) the impact of some dynamic variable ordering heuristics, (4) the impact of the lower bound strength, as well as (5) the impact of best-first versus depth-first search regimes [1, 3, 4].

In this paper, we want to take these classes of algorithms as much further as we can by including additional known principles of problem solving and examine their interactive impact on performance. We investigate three key factors that impact the performance of any search algorithm: (1) the availability of hard constraints ($i.e.$, determinism) in the problem (2) the availability of a good initial upper bound provided to the algorithm, and (3) the availability of good quality guiding pseudo trees. We therefore extend AOBB-C (and whenever relevant, AOBF-C) to exploit explicitly the computational power of hard constraints by incorporating standard constraint propagation techniques such as unit resolution. We provide AOBB-C with a non-trivial initial uppers bound computed by local search. Finally, we investigate randomized orderings generated via two heuristics for constructing small induced width/depth pseudo trees.

We evaluate the impact and interaction of these extensions on the optimization problem of finding the most probable explanation in belief networks using a variety of random and real-world benchmarks. We show that exploiting the determinism as well as good quality initial upper bounds and pseudo trees improves the performance dramatically in many cases.

2 Background

2.1 Constraint Optimization Problems

A finite *Constraint Optimization Problem* (COP) is a triple $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, where $\mathbf{X} = \{X_1, ..., X_n\}$ is a set of variables, $\mathbf{D} = \{D_1, ..., D_n\}$ is a set of finite domains and $\mathbf{F} = \{f_1, ..., f_r\}$ is a set of cost functions. Cost functions can be either *soft* or *hard* (constraints). Without loss of generality we assume that hard constraints are represented as (bi-valued) cost functions. Allowed and forbidden tuples have cost 0 and ∞, respectively. The scope of function f_i, denoted $scope(f_i) \subseteq \mathbf{X}$, is the set of arguments of f_i. The goal is to find a complete value assignment to the variables that minimizes the global cost function $f(\mathbf{X}) = \sum_{i=1}^{r} f_i$, namely to find $x = \arg\min_{\mathbf{X}} \sum_{i=1}^{r} f_i$.

Given a COP instance, its *primal graph* G associates each variable with a node and connects any two nodes whose variables appear in the scope of the same function. The *induced graph* of G relative to an ordering d of its variables, denoted $G^*(d)$, is obtained by processing the nodes in reverse order of d. For each node all its earlier neighbors are connected, including neighbors connected by previously added edges. Given a graph and an ordering of its nodes, the *width* of a node is the number of edges connecting it to nodes lower in the ordering. The *induced width* (also equal to the *treewidth*) of a graph along d, denoted $w^*(d)$, is the maximum width of nodes in the induced graph.

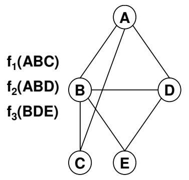
Belief networks [5] provide a formalism for reasoning under conditions of uncertainty. A belief network represents a joint probability distribution over the variables of interest. A function of the model encodes the conditional probability distribution of a variable given its parents in the graph (also viewed as a cost function were each tuple has associated a real cost between 0 and 1). The most common constraint optimization task over belief networks if finding the *Most Probable Explanation* (MPE), namely finding a complete assignment with maximum probability that is consistent with the evidence in the network. It appears in applications such as speech recognition or medical diagnosis. The task can also be formulated using the semi-ring framework introduced in [6].

A	B	C	$f_1(ABC)$
0	0	0	∞
0	0	1	∞
0	1	0	∞
0	1	1	2
1	0	0	∞
1	0	1	2
1	1	0	∞
1	1	1	2

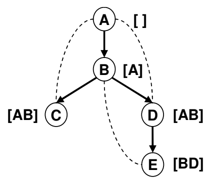
A	B	D	$f_2(ABD)$
0	0	0	1
0	0	1	∞
0	1	0	0
0	1	1	2
1	0	0	6
1	0	1	5
1	1	0	6
1	1	1	5

B	D	E	$f_3(BDE)$
0	0	0	∞
0	0	1	3
0	1	0	∞
0	1	1	4
1	0	0	∞
1	0	1	3
1	1	0	∞
1	1	1	4

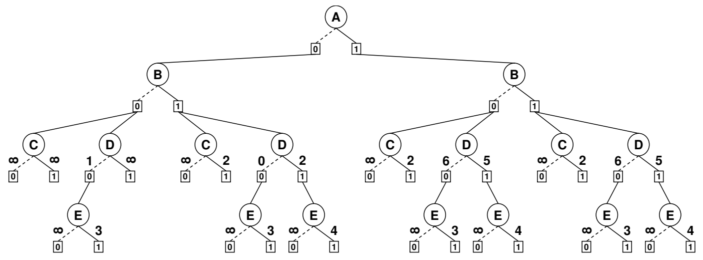
(a) Functions



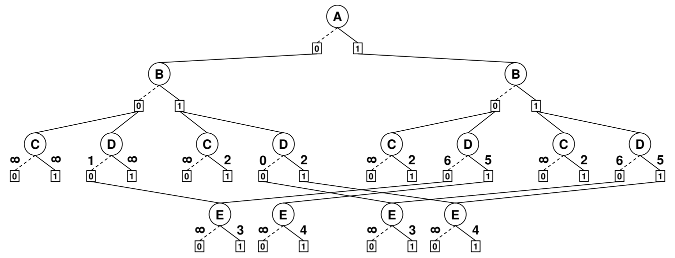
(b) Primal graph



(c) Pseudo tree



(d) AND/OR search tree



(e) AND/OR search graph

Fig. 1. AND/OR search spaces for constraint optimization.

2.2 AND/OR Search Spaces for Constraint Optimization

The AND/OR search space [2] is a unifying framework for advanced algorithmic schemes for graphical models, including belief networks, constraint networks and cost networks. Its main virtue consists in exploiting conditional independencies between variables during search, which can provide exponential speedups over traditional structure-blind search methods. The search space is defined using a backbone *pseudo tree* [7].

Definition 1 (pseudo tree). *Given an undirected graph $G = (\mathbf{X}, E)$, a directed rooted tree $\mathcal{T} = (\mathbf{X}, E')$ defined on all its nodes is called* pseudo-tree *if any edge of G that is not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node to an ancestor in $\mathcal{T}$.*

AND/OR Search Trees. Given a COP instance $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ its primal graph G and a pseudo tree $\mathcal{T}$ of G, the associated AND/OR search tree, $S_{\mathcal{T}}$, has alternating levels of OR and AND nodes. The OR nodes are labeled X_i and correspond to the variables. The AND nodes are labeled $\langle X_i, x_i \rangle$ (or just x_i) and correspond to value assignments of the variables. The structure of the AND/OR search tree 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 X_i are AND nodes labeled with assignments

$\langle X_i, x_i \rangle$ that are consistent with the assignments along the path from the root. The children of an AND node $\langle X_i, x_i \rangle$ are OR nodes labeled with the children of variable X_i in $\mathcal{T}$. The AND/OR search tree can be traversed by a depth first search (DFS) algorithm, thus using linear space to compute the value of the root node. It was shown in that [7, 2] that given a COP instance $\mathcal{P}$ and a pseudo tree $\mathcal{T}$ of depth m, the size of the AND/OR search tree based on $\mathcal{T}$ is $O(n \cdot k^m)$, where k bounds the domains of variables.

AND/OR Search Graphs. The AND/OR search tree may contain nodes that root identical conditioned subproblems. Such nodes can be merged yielding an AND/OR graph. Its size becomes smaller at the expense of using additional memory by the search algorithm. Some mergeable nodes can be identified based on their *contexts*.

Given a pseudo tree $\mathcal{T}$ of an AND/OR search space, the *context* of an OR node X, denoted by $context(X) = [X_1 \dots X_p]$, is the set of ancestors of X in $\mathcal{T}$ ordered descendingly, that are connected in the primal graph to X or to descendants of X. The context of X separates the subproblem below X from the rest of the network. The *context minimal* AND/OR graph [2] is obtained from the AND/OR search tree by merging all the context mergeable nodes.

It can be shown [2] that given a COP $\mathcal{P}$, its primal graph G and a pseudo tree $\mathcal{T}$, the size of the context minimal AND/OR search graph is $O(n \cdot k^{w^*_{\mathcal{T}}(G)})$, where $w^*_{\mathcal{T}}(G)$ is the induced width of G over the DFS traversal of $\mathcal{T}$, and k bounds the domain size.

Weighted AND/OR Search Graphs. The OR-to-AND arcs from nodes X_i to x_i in an AND/OR search tree or graph are annotated by *weights* derived from the cost functions in **F**. The *weight* $w(X_i, x_i)$ of the arc from the OR node X_i to the AND node x_i is the sum of all the 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 the path.

Given a weighted AND/OR search graph, each of its nodes can be associated with a *value*. The value $v(n)$ of a node n is the minimal cost solution to the subproblem rooted at n, subject to the current variable instantiation along the path from the root to n. It can be computed recursively using the values of n's successors (see also [2] for details).

Example 1. Figure 1 shows an example of AND/OR search spaces for a COP with binary variables. The cost functions are given in Figure 1(a). The value ∞ indicates a hard constraint. The primal graph is given in Figure 1(b), and the pseudo tree in Figure 1(c). The square brackets indicate the context of the variables. The AND/OR search tree is given in Figure 1(d). The numbers on the OR-to-AND arcs are the weights corresponding to the function values. Note that the tree is pruned whenever a weight shows inconsistency (e.g., for $A = 0, B = 0, D = 1$ there is no need to visit variable E, due to the value $f_2(0, 0, 1) = \infty$). The context minimal AND/OR graph is given in Figure 1(e). Note that only the cache table of E will get cache hits during the depth first search traversal (E is the only level of OR nodes that has more than one incoming arc). It can be determined from the pseudo tree inspection that all variables except for E generate *dead-caches* [2] and their cache tables need not be stored.

3 AND/OR Search Algorithms for Constraint Optimization

In recent years several depth-first Branch-and-Bound and best-first search algorithms were developed to search the context minimal AND/OR graph for solving COPs [3, 4].

We next briefly overview these two classes of algorithms.

AND/OR Branch-and-Bound (AOBB-C) traverses the context minimal AND/OR graph in a depth-first manner via full caching. It interleaves forward expansion of the current partial solution tree with a backward cost revision step that updates node values, until search terminates. The efficiency of the algorithm also depends on the strength of its heuristic evaluation function ($i.e.$, lower bound). Specifically, each node n in the search graph has an associated heuristic function $h(n)$ underestimating $v(n)$ that can be computed efficiently when the node n is first expanded. The algorithm then computes the heuristic evaluation function $f(T')$ of the current partial solution T' and uses it to prune irrelevant portions of the search space, as part of a Branch-and-Bound scheme.

In the forward step the algorithm expands alternating levels of OR and AND nodes. Before expanding an OR node, its cache table is checked. If the same context was encountered before, it is retrieved from the cache, and its successors set is set to empty which will trigger the cost revision step. If an OR node is not found in the cache, it is expanded in the usual way. Before expanding an AND node n, the algorithm computes the heuristic evaluation function $f(T'_m)$ for every partial solution subtree T'_m rooted at each OR ancestor m of n along the search path. The search is terminated below n, if, for some m, $f(T'_m)$ is greater or equal to the best cost solution found at m.

The backward cost revision step is triggered when a closed node has an empty set of successors. This means that all its children have been evaluated, and its final value can now be computed. If the current node is the root, then the search terminates with its value. OR nodes update their values by minimization, while AND nodes combine their children values by summation.

Best-First AND/OR Search (AOBF-C) explores the context minimal AND/OR graph and interleaves forward expansion of the best partial solution tree with a cost revision step that updates the node values. First, a top-down, graph growing operation finds the best partial solution tree by tracing down through the marked arcs of the explicit AND/OR search graph $C'_{\mathcal{T}}$. These previously computed marks indicate the current best partial solution tree from each node in $C'_{\mathcal{T}}$. One of the nonterminal leaf nodes n of this best partial solution tree is then expanded and a heuristic estimate $h(n_i)$, underestimating $v(n_i)$, is assigned to its successors.

The second operation in AOBF-C is a bottom-up, cost revision, arc marking, SOLVE-labeling procedure. Starting with the node just expanded n, the procedure revises its value $v(n)$ (using the newly computed values of its successors) and marks the outgoing arcs on the estimated best path to terminal nodes. OR nodes revise their values by minimization, while AND node by summation. This revised value is then propagated upwards in the graph. The revised cost $v(n)$ is an updated lower bound on the cost of an optimal solution to the subproblem rooted at n. During the bottom-up step, AOBF-C labels an AND node as SOLVED if all of its OR child nodes are solved, and labels an OR node as SOLVED if its marked AND child is also solved. The optimal cost solution to the initial problem is obtained when the root node is labeled SOLVED.

Mini-Bucket Heuristics. The effectiveness of both depth-first and best-first AND/OR search algorithms greatly depends on the quality of the lower bound heuristic evaluation functions. The primary heuristic that we used in our experiments is the Mini-Bucket heuristic, which was presented in [3, 4]. It was shown that the intermediate functions generated by the Mini-Bucket algorithm MBE(i) [8] can be used to compute a heuristic function that underestimates the minimal cost solution to the current subproblem in the AND/OR graph.

4 Improving AND/OR Branch-and-Bound Search

In this section we overview several principled improvements to the AND/OR Branch-and-Bound algorithm that we will incorporate.

4.1 Exploiting Determinism

When the functions of the COP instance express both hard constraints and general cost functions, it may be beneficial to exploit the computational power of the constraints explicitly via constraint propagation [9, 10]. In belief networks, for example, the hard constraints are represented by the zero probability tuples of the CPTs. We note that the use of constraint propagation via directional resolution [11] or generalized arc consistency has been explored in [12], in the context of variable elimination algorithms. The approach we take for handling the determinism in COP is based on the known technique of *unit resolution* for Boolean Satisfiability (SAT) over a logical knowledge base (KB) in the form of propositional clauses (CNF) representing the hard constraints.

One common way of encoding hard constraints as a CNF formula is the *direct encoding* [13]. Given a COP instance, we associate a propositional variable x_{ij} with each value j that can be assigned to the COP variable X_i. We then have clauses that ensures each COP variable is given a value: for each i, $x_{i1} \vee ... \vee x_{im}$. We optionally have clauses that ensure each variable takes no more than one values: for each i, j, k with $j \neq k$, $\neg x_{ij} \vee \neg x_{ik}$. Finally, we have clauses that rule out any no-goods. For example, if $X_1 = 2$ and $X_3 = 1$ is not allowed then we have the clause $\neg x_{12} \vee \neg x_{31}$.

The changes needed in the AND/OR Branch-and-Bound procedure are then as follows. Upon expanding an AND node $\langle X_i, x_j \rangle$ the corresponding SAT instantiation is asserted in KB, namely x_{ij} is set to *true*. If the unit resolution leads to a contradiction, then the current AND node is marked as dead-end and the search continues by expanding the next node on the search stack. Whenever the algorithm backtracks to the previous level, it also retracts any SAT instantiations recorded by unit resolution. Notice that the algorithm is capable of pruning the domains of future variables in the current subproblem due to conflicts detected during unit propagation.

4.2 Exploiting Good Initial Upper Bounds via Local Search

The AND/OR Branch-and-Bound algorithm assumed a trivial initial upper bound ($i.e.$, ∞), which effectively guarantees that the optimal solution will be computed, however it provides limited pruning in the initial phase. We therefore can incorporate a more informed upper bound, obtained by solving the problem via a local search scheme. This approach is often used by state-of-the-art constraint optimization solvers.

One of the most popular local search algorithms for COP is the *Guided Local Search* (GLS) method [14]. GLS is a penalty-based meta-heuristic, which works by augmenting the objective function of a local search algorithm ($e.g.$ hill climbing) with penalties, to help guide them out of local minima. GLS has been shown to be successful in solving a number of practical real life problems, such as the traveling salesman problem, radio link frequency assignment problem and vehicle routing. It was also applied to solving the MPE in belief networks [15] as well as weighted MAX-SAT problems [16].

4.3 Exploiting the Pseudo Tree Quality

The performance of the AND/OR search algorithms can be heavily influenced by the quality of the guiding pseudo tree. Finding the minimal depth or induced width pseudo tree is a hard problem [7, 17]. We describe next two heuristics for generating pseudo trees which we used in our experiments.

Min-Fill Heuristic. The *Min-Fill* ordering [18] is generated by placing the variable with the smallest *fill set* ($i.e.$, number of induced edges that need be added to fully connect the neighbors of a node) at the end of the ordering, connecting all of its neighbors and then removing the variable from the graph. The process continues until all variables have been eliminated. Once an elimination order is given, the pseudo tree can be extracted as a depth-first traversal of the min-fill induced graph, starting with the variable that initiated the ordering, always preferring as successor of a node the earliest adjacent node in the induced graph. An ordering uniquely determines a pseudo tree. This approach was first used by [17].

Hypergraph Decomposition Heuristic. An alternative heuristic for generating a low height balanced pseudo tree is based on the recursive decomposition of the dual hypergraph associated with the COP instance. The dual hypergraph of a COP $\langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ is a pair $(\mathbf{V}, \mathbf{E})$ where each function in $\mathbf{F}$ is a vertex $v_i \in \mathbf{V}$ and each variable in $\mathbf{X}$ is a hyperedge $e_j \in \mathbf{E}$ connecting all the functions (vertices) in which it appears.

Generating heuristically good hypergraph separators can be done using a package called `hMeTiS` (*available at: http://www-users.cs.umn.edu/karypis/metis/hmetis*), which we used following [19]. The vertices of the hypergraph are partitioned into two balanced (roughly equal-sized) parts, denoted by $\mathcal{H}_{left}$ and $\mathcal{H}_{right}$ respectively, while minimizing the number of hyperedges across. A small number of crossing edges translates into a small number of variables shared between the two sets of functions. $\mathcal{H}_{left}$ and $\mathcal{H}_{right}$ are then each recursively partitioned in the same fashion, until they contain a single vertex. The result of this process is a tree of hypergraph separators which can be shown to also be a pseudo tree of the original model where each separator corresponds to a subset of variables chained together.

Randomization. Both the min-fill and hypergraph partitioning heuristics can randomize their tie breaking rules, yielding varying qualities of the generated pseudo tree.

5 Experiments

In order to empirically evaluate the performance of the proposed improvements to AOBB-C algorithms, we have conducted a number of experiments on the optimization problem of finding the most probable explanation in belief networks. We implemented our algorithms in C++ and ran all experiments on a 2.4GHz Pentium 4 with 2GB of RAM running Windows XP.

5.1 Overview and Methodology

Algorithms. We evaluated the following AND/OR Branch-and-Bound hybrid algorithms with full caching and static mini-bucket heuristics:

- AOBB-C+SAT+SMB(i), which exploits the determinism in the network by applying unit resolution over the CNF encoding of the zero probability tuples of the probability tables. We used a unit resolution scheme based on the one available in the `zChaff` SAT solver [20].
- AOBB-C+GLS+SMB(i), which exploits a good initial upper bound obtained by a guided local search algorithm. We used the GLS implementation for belief networks available from [15].
- AOBB-C+SAT+GLS+SMB(i), which combines the previous two approaches.

We compare these algorithms against the baseline AND/OR Branch-and-Bound with full caching and mini-bucket heuristics, AOBB-C+SMB(i). We also ran the best-first search version of the algorithm, denoted by AOBF-C+SMB(i), but the algorithm did not exploit any of the above principles. The guiding pseudo trees were constructed using both the min-fill and the hypergraph partitioning heuristics, described earlier.

We also compared with the SamIam version 2.3.2 software package (*available at http://reasoning.cs.ucla.edu*). SamIam is a public implementation of Recursive Conditioning [19] which can also be viewed as an AND/OR search algorithm, namely it explores a context minimal AND/OR graph [2]. Since any MPE problem instance can be converted into an equivalent 0-1 Integer Linear Program [21], we also ran the ILOG CPLEX 11.0 solver, with default settings ($i.e.$, best-bound control strategy, strong branching based variable ordering heuristic, and the cutting planes engine turned on).

Benchmarks. We tested the performance of the AND/OR search algorithms on random grid networks, belief networks derived from the ISCAS'89 digital circuits, genetic linkage analysis networks, and relational belief networks that model the popular game of Mastermind. All of these networks contain a significant amount of determinism.

Measures of performance. We report CPU time in seconds and the number of nodes visited. We also specify the the number of variables (n), number of evidence variables (e), maximum domain size (k), the depth of the pseudo tree (h) and the induced width of the graph (w^*) for each problem instance. When evidence is asserted in the network, the w^* and h are computed after the evidence variables are removed from the graph. We also report the time required by GLS to compute the initial upper bound. Note that in each domain we ran GLS for a fixed number of flips. Moreover, AOBB-C+GLS+SMB(i) and AOBB-C+SAT+GLS+SMB(i) do not include the GLS running time, because GLS can be tuned independently for each problem class to minimize its running time. The best performance points are highlighted. In each table, ”-” denotes that the respective algorithm exceeded the time limit. Similarly, ”out” stands for exceeding the 2GB memory limit. A ”*” by the GLS running time indicates that it found the optimal solution to the respective problem instance.

minfill pseudo tree without randomization											
grid (w*, h) (n, e)	SamIam CPLEX GLS	AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=12		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=14		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=16		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=18		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=20	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
		-	-	1273.09	9,047,518	596.27	4,923,760	70.42	473,675	74.99	412,291
90-24-1	out	687.96	4,823,044	202.05	1,564,800	172.31	1,370,222	55.52	401,294	69.53	386,785
(33, 111)	8.43	-	-	66.20	425,585	20.16	93,911	11.17	7,850	28.16	27,868
(576, 20)	0.53	473.64	3,181,352	19.09	131,546	8.41	49,054	$\boxed{\textbf{5.45}}$	6,891	23.87	39,175
		out		21.94	75,637	10.59	33,770	6.06	5,144	23.80	17,291
		146.97	878,874	152.80	962,484	4.36	15,632	12.92	46,489	22.13	2,242
90-26-1	out	32.67	230,030	53.11	360,612	$\boxed{\textbf{3.58}}$	11,620	11.95	40,075	22.02	1,858
(36, 113)	7.87	36.94	252,380	87.02	559,518	4.17	14,580	7.86	6,310	22.00	1,894
(676, 40)	0.56	15.09	104,775	32.85	219,037	$\boxed{\textbf{3.58}}$	10,932	8.06	8,128	24.42	1,658
		19.06	65,271	24.39	79,619	4.27	7,190	8.05	3,777	22.44	1,435
		652.15	3,882,300	165.74	1,070,823	155.20	956,837	40.14	212,963	59.28	174,715
90-30-1	out	117.25	771,233	66.66	453,095	50.94	341,670	30.69	168,928	42.86	88,004
(43, 150)	$\boxed{\textbf{6.32}}$	263.32	1,498,756	74.95	446,498	68.16	376,916	23.88	95,136	53.92	148,540
(900, 60)	0.72	89.94	561,397	38.92	247,271	28.67	176,330	15.50	52,260	40.52	72,053
		158.97	534,385	46.73	157,187	47.27	154,496	21.06	45,201	57.97	100,800
		-	-	-	-	-	-	-	-	369.36	823,604
90-34-1	out	-	-	-	-	-	-	-	-	132.84	271,609
(45, 153)	$\boxed{\textbf{17.02}}$	-	-	-	-	1096.14	5,569,276	1772.51	5,516,888	294.11	630,406
(1154, 80)	1.31	-	-	-	-	550.55	2,944,055	651.04	2,614,171	124.16	238,333
		out		out		243.63	596,978	270.88	667,013	71.19	67,611
		969.02	2,623,971	1753.10	3,794,053	203.67	614,868	165.45	488,873	113.06	214,919
90-38-1	out	141.89	577,763	204.69	593,809	86.16	319,185	102.03	312,473	85.74	142,589
(47, 163)	$\boxed{\textbf{12.48}}$	854.61	2,498,702	1822.71	3,792,826	212.63	647,089	164.43	484,815	109.77	211,740
(1444, 120)	1.11	138.44	573,923	204.68	597,751	96.27	339,729	98.21	311,072	85.50	140,581
		101.69	174,786	103.80	146,237	54.00	95,511	53.44	78,431	73.10	59,856

Table 1. CPU time and nodes visited for solving deterministic grid networks. Time limit 1 hour. Number of flips for GLS is 50,000. Pseudo tree generated by a single run of the min-fill heuristic, without randomization. SamIam ran out of memory on all test instances.

5.2 Empirical Results

Deterministic Grid Networks. In grid networks, the nodes are arranged in an $N \times N$ square and each CPT is generated uniformly randomly. We experimented with problem instances initially developed by [22] for the task of weighted model counting. For these problems, N ranged between 24 and 38, and, for each instance, 90% of the probability tables were deterministic, namely they contained only 0 and 1 probability entries.

Table 1 shows the results for experiments with 5 deterministic grid networks. The columns are indexed by the mini-bucket i-bound which we varied between 12 and 20. We generated a single MPE query with e variables picked randomly and instantiated as evidence. When comparing the AND/OR Branch-and-Bound algorithms, we observe that AOBB-C+SAT+SMB(i) improves considerably over AOBB-C+SMB(i), especially for relatively small i-bounds ($e.g.$, $i \in \{12, 14\}$) which correspond to relatively weak heuristic estimates. For example, on the 90-38-1 grid instance, AOBB-C+SAT+SMB(14) is about 9 times faster than AOBB-C+SMB(14) and explores 6 times fewer nodes. Similarly, AOBB-C+GLS+SMB(i) which exploits a non-trivial initial upper bound is better than AOBB-C+SMB(i) for relatively small i-bounds. Overall, AOBB-C+SAT+GLS+SMB(i), which exploits both the determinism and informed initial upper bounds, is the best performing among the Branch-and-Bound algorithms and in some cases it is able to outperform AOBF-C+SMB(i), across all reported i-bounds ($e.g.$, 90-30-1). As the i-bound increases and the heuristics become strong enough to cut the search space significantly, the difference between AOBB-C+SMB(i) and its competitors decreases. The mini-bucket heuristic already does a level of constraint propagation. Notice that CPLEX achieves the best performance on 3 test instances.

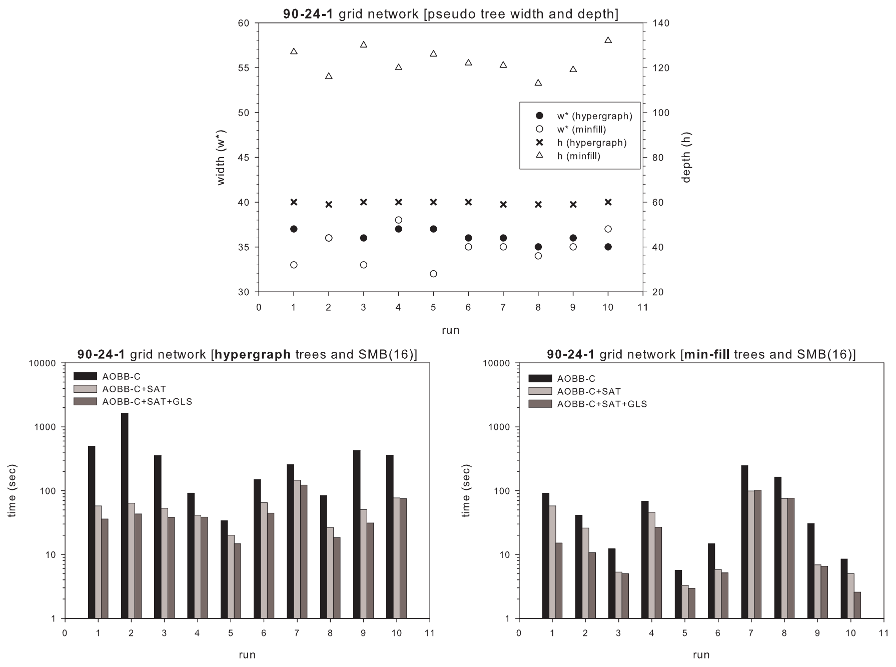


Fig. 2. Detailed execution of AOBB-C+SMB(16), AOBB-C+SAT+SMB(16) and AOBB-C+SAT+GLS+SMB(16) on the 90-24-1 grid network over 10 runs using randomized min-fill and hypergraph based pseudo trees. Pseudo tree depth is plotted on a different scale to the right.

Figure 2 shows the execution of AOBB-C+SMB(16), AOBB-C+SAT+SMB(16) and AOBB-C+SAT+GLS+SMB(16) on the 90-24-1 network with randomized min-fill and hypergraph pseudo trees, over 10 independent runs. For each run we also report the induced width and depth of the corresponding pseudo trees. We see that the best performance in this case is obtained for min-fill trees. This is probably because the mini-bucket heuristics were more accurate when computed along the min-fill ordering which has smaller induced width than the hypergraph partitioning based ordering.

ISCAS'89 Circuits (*available at http://www.fm.vslib.cz/kes/asic/iscas*) are a common benchmark used in formal verification and diagnosis. For our purpose, we converted each of these circuits into a belief network by removing flip-flops and buffers in a standard way, creating a deterministic conditional probability table for each gate and putting uniform distributions on the input signals.

Table 2 displays the results obtained on 5 ISCAS'89 circuits. We see that constraint propagation via unit resolution plays a dramatic role on this domain rendering the search space almost backtrack-free, across all reported i-bounds. For instance, on the s953 circuit, AOBB-C+SAT+SMB(6) is 3 orders of magnitude faster than AOBB-C+SMB(6) and 2 orders of magnitude faster than CPLEX, respectively, while AOBF-C+SMB(6) exceeded the memory limit. When looking at the AND/OR Branch-and-Bound algorithms that exploit a local search based initial upper bound, namely AOBB-C+GLS+SMB(i) and AOBB-C+SAT+GLS+SMB(i), we see that they did not expand

minfill pseudo tree without randomization											
iscas89 (w*, h) (n, d)	SamIam CPLEX GLS	AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=6		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=8		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=10		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=12		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=14	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
		-	-	-	-	182.53	2,316,024	0.16	432	0.24	432
c432	out	374.29	4,336,403	189.13	2,043,475	1.02	9,512	0.16	432	0.25	432
(27, 45)	20.54	$\boxed{\mathbf{0.05}}$	0	0.06	0	0.09	0	0.13	0	0.19	0
(432, 2)	0.08*	0.06	0	0.08	0	0.09	0	0.13	0	0.20	0
		out		out		106.27	488,462	0.20	432	0.28	432
		899.63	7,715,133	17.99	155,865	48.13	417,924	17.00	132,139	2.19	13,039
s953	out	0.19	829	0.16	667	0.20	685	0.31	623	0.74	623
(66, 101)	12.14	$\boxed{\mathbf{0.12}}$	0	0.13	0	0.17	0	0.28	0	0.69	0
(440, 2)	0.05*	0.13	0	0.13	0	0.17	0	0.30	0	0.70	0
		out		41.03	150,598	110.45	408,828	36.50	113,322	4.06	12,256
		18.05	104,316	124.53	686,069	3.69	26,847	14.23	94,985	9.47	62,883
s1196	out	0.19	565	0.19	565	0.23	565	0.38	565	0.92	565
(54, 97)	92.19	0.14	0	0.16	0	0.20	0	0.34	0	0.89	0
(560, 2)	0.08*	$\boxed{\mathbf{0.13}}$	0	0.14	0	0.20	0	0.34	0	0.87	0
		26.16	77,019	158.19	372,129	7.22	23,348	26.97	80,264	17.64	48,114
		13.22	82,294	1.02	5,920	2.50	15,621	1.19	6,024	1.47	3,516
s1488	out	0.20	708	0.20	667	0.25	667	0.44	667	1.06	667
(47, 67)	33.48	0.14	0	0.16	0	0.22	0	0.44	0	0.99	0
(667, 2)	0.13*	$\boxed{\mathbf{0.13}}$	0	0.16	0	0.20	0	0.47	0	0.99	0
		21.75	74,658	1.67	5,499	4.22	14,445	1.84	5,372	1.80	3,124
		7.30	41,798	19.69	108,768	4.81	27,711	7.00	41,977	2.06	8,104
s1494	out	0.20	665	0.22	665	0.27	665	0.45	665	1.11	665
(48, 69)	42.1	$\boxed{\mathbf{0.16}}$	0	0.17	0	0.22	0	0.41	0	1.09	0
(661, 2)	0.11*	$\boxed{\mathbf{0.16}}$	0	0.17	0	0.22	0	0.42	0	1.22	0
		9.67	24,849	27.28	65,859	7.86	19,678	11.48	28,793	3.03	6,484

Table 2. CPU time and nodes visited for solving belief networks derived from the ISCAS'89 circuits. Time limit 30 minutes. Number of flips for GLS is 10,000. Pseudo tree generated by a single run of the min-fill heuristic, without randomization. SamIam ran out of memory.

any nodes. This is because the upper bound obtained by GLS, which was the optimal solution in this case, was equal to the the mini-bucket lower bound computed at the root node. The best performance on this domain were achieved by AOBB-C+SAT+SMB(i) and AOBB-C+SAT+GLS+SMB(i), respectively, for the smallest reported i-bound ($i = 6$). Notice the poor performance of SamIam which ran out of memory on all tests.

Genetic Linkage Analysis. The *maximum likelihood haplotype* problem in genetic linkage analysis is the task of finding a joint haplotype configuration for all members of the pedigree which maximizes the probability of data. It is equivalent to finding the most probable explanation of a belief network which represents the pedigree data [23].

Table 3 displays the results obtained for 6 hard linkage analysis networks (*available at http://bioinfo.cs.technion.ac.il/superlink*) using randomized min-fill and hypergraph partitioning based pseudo trees. We selected the hypergraph based tree having the smallest depth over 100 independent runs (ties were broken on the smallest induced width). Similarly, the min-fill based tree was the one having the smallest induced width out of 100 tries (ties were broken on the smallest depth). For comparison, we also include results obtained with Superlink version 1.6 [23]. Superlink is currently one of the most efficient solvers for genetic linkage analysis, is dedicated to this domain, uses a combination of variable elimination and conditioning, and takes advantage of the determinism present in the network. To the best of our knowledge, these networks were never solved before for the maximum likelihood haplotype task ($i.e.$, the MPE task).

We see that the AND/OR Branch-and-Bound algorithms are the only ones that could solve all the problem instances, especially when guided by hypergraph partitioning based pseudo trees. This can be explained by the much smaller depth of these pseudo trees compared with the min-fill ones, which overcame the relatively poor quality of

		hypergraph pseudo tree					**min-fill pseudo tree**				
pedigree (n, d)	SamIam Superlink CPLEX GLS	(w*, h)	MBE(i) AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBF-C+SMB(i) i=20		MBE(i) AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBF-C+SMB(i) i=22		(w*, h)	MBE(i) AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBF-C+SMB(i) i=20		MBE(i) AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBF-C+SMB(i) i=22	
			time	nodes	time	nodes		time	nodes	time	nodes
	out		25.26		164.49			117.03		out	
ped7	-		30504.84	285,084,124	3005.66	27,761,219		-	-		
(868, 4)	out	(36, 60)	31701.54	285,084,124	3116.07	27,761,219	(32, 133)	-	-		
	13.32		30349.92	284,635,328	$\boxed{\mathbf{2955.06}}$	27,371,526		-	-		
			out		out			out			
	out		67.93		300.06			76.31		out	
ped9	-		8922.81	117,328,162	$\boxed{\mathbf{3292.30}}$	40,251,723		1434.74	15,825,340		
(936, 7)	out	(35, 58)	10075.90	117,328,162	3657.91	40,251,723	(27, 130)	1515.50	15,825,340		
	12.85*		8866.40	117,011,941	3336.86	40,251,661		$\boxed{\mathbf{1163.09}}$	12,444,961		
			out		out			out			
	out		59.31		150.38			out		out	
ped19	-		45075.31	466,748,365	8321.42	90,665,870					
(693, 5)	out	(35, 53)	47986.66	466,748,365	8774.51	90,665,870	(24, 122)				
	10.23		44585.84	459,741,495	$\boxed{\mathbf{8070.95}}$	87,060,723					
			out		out						
	out		42.21		209.51			out		out	
ped34	-		67647.42	1,293,350,829	11719.28	220,199,927					
(923, 4)	out	(34, 60)	74020.63	1,293,350,829	12847.33	220,199,927	(32, 127)				
	13.99		64136.36	1,230,870,576	$\boxed{\mathbf{11005.18}}$	218,890,668					
			out		out						
	out		35.41		111.24			out		out	
ped41	-		3891.86	31,731,270	380.01	2,318,544					
(886, 5)	out	(36, 61)	4055.15	31,731,270	390.93	2,318,544	(33, 128)				
	12.28*		3869.31	31,729,654	$\boxed{\mathbf{374.95}}$	2,317,321					
			out		out						
	out		32.92		140.81			57.88		344.68	
ped44	-		3597.12	62,385,573	204.96	1,355,595		112.60	1,114,641	385.30	668,737
(644, 4)	out	(31, 52)	3904.39	60,709,547	215.46	1,355,595	(26, 73)	127.42	1,114,641	385.47	668,737
	9.84		3580.32	62,392,439	$\boxed{\mathbf{196.57}}$	1,213,051		$\boxed{\mathbf{95.09}}$	752,970	366.18	447,514
			out		out			out		out	

Table 3. CPU time and nodes visited for solving genetic linkage analysis networks. Pseudo trees created using randomized min-fill and hypergraph partitioning heuristics. Time limit 24 hours. SamIam, CPLEX and AOBF-C+SMB(i) ran out of memory, whereas Superlink exceeded the time limit. The maximum number of flips for GLS was set to 1,000,000.

the mini-bucket heuristics obtained on these highly connected networks. Exploiting the GLS initial upper bound improved slightly the performance of AOBB-C+SMB(i) ($e.g.$, AOBB-C+GLS+SMB(22) improves over AOBB-C+SMB(22) on ped34 with a margin of 7% only). This was probably because AOBB-C+SMB(i) found the optimal or very close to optimal solutions quite early in the search. Similarly, we observe that applying unit resolution was not cost effective in this case, namely AOBB-C+SMB(i) and AOBB-C+SAT+SMB(i) expanded the same number of nodes. Notice also that Superlink exceeded the 24 hour time limit, whereas SamIam, CPLEX and AOBF-C+SMB(i) ran out of memory on all test instances.

Figure 3 shows the execution of AOBB-C+SMB(22), AOBB-C+SAT+SMB(22) and AOBB-C+GLS+SMB(22) on the ped44 network with randomized min-fill and hypergraph pseudo trees, over 10 independent runs. For each run we also report the induced width and depth of the corresponding pseudo trees. We see that the hypergraph pseudo trees, which have small depths, offer the best performance in this case. This can be explained by the large induced width ($i.e.$, context) which in this case renders most of the cache entries dead. Therefore, the AND/OR graph explored effectively is very close to a tree and the dominant factor that impacts performance is the depth of the pseudo tree.

Mastermind Game Instances. Each of these networks is a ground instance of a relational belief network that models differing sizes of the popular game of Mastermind, and was produced by the PRIMULA System (*http://www.cs.auc.dk/jaeger/Primula*).

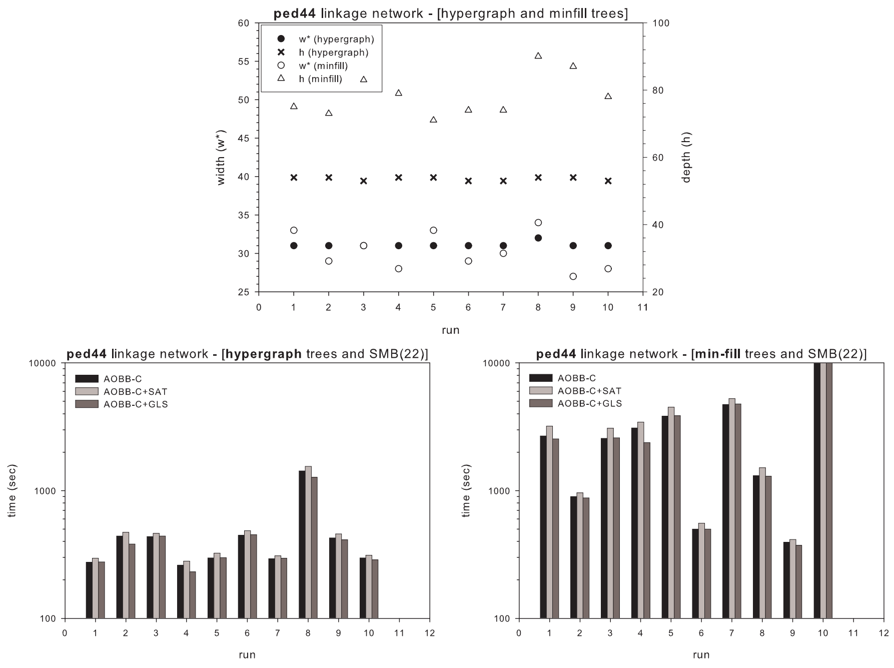


Fig. 3. Detailed execution of AOBB-C+SMB(22), AOBB-C+SAT+SMB(22) and AOBB-C+GLS+SMB(22) on the ped44 linkage network over 10 runs using randomized min-fill and hypergraph based pseudo trees. Pseudo tree depth is plotted on a different scale to the right.

Table 4 reports the results obtained on 6 Mastermind networks. We generated a single MPE query with e random evidence variables. We see that AOBB-C+SAT+SMB(i) is far better than AOBB-C+SMB(i), especially for relatively small i-bounds. For example, on the mm-03-08-05 network, AOBB-C+SAT+SMB(14) is 7 times faster than AOBB-C+SMB(14), 3 times faster than AOBF-C+SMB(14), and 58 times faster than CPLEX, respectively. When looking at the effect of the GLS based upper bound, which was equal to the optimal solution in this case, we see no noticeable difference between AOBB-C+GLS+SMB(i) and AOBB-C+SMB(i). This is because AOBB-C+SMB(i) was able to find close to optimal solutions quite fast. Notice that SamIam exceeded the memory bound on all instances, whereas CPLEX obtained the best performance on only one instance, namely mm-10-08-03.

Summary of Empirical Results. Summarizing our empirical observations we see that exploiting the hard constraints present in the problem improves in many cases significantly the performance of the AND/OR Branch-and-Bound algorithms with mini-bucket heuristics ($e.g.$, ISCAS'89 circuits). The impact of the local search based initial upper bound is more prominent at relatively small i-bounds, when the corresponding heuristics are relatively weak and the algorithm does not find close to optimal solution early enough in the search ($e.g.$, grid networks). The two heuristics for generating pseudo trees do not dominate each other. When the induced width is small enough, which is typically the case for min-fill pseudo trees, the strength of the mini-bucket

minfill pseudo tree without randomization											
mastermind (w*, h) (n, e)	SamIam CPLEX GLS	AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=12		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=14		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=16		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=18		AOBB-C+SMB(i) AOBB-C+SAT+SMB(i) AOBB-C+GLS+SMB(i) AOBB-C+SAT+GLS+SMB(i) AOBF-C+SMB(i) i=20	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
		1.88	9,552	0.98	2,470	1.83	2,418	4.19	2,427	7.80	2,349
mm-03-08-03	out	1.58	9,287	$\boxed{\mathbf{0.92}}$	2,464	1.83	2,412	4.09	2,426	7.87	2,349
(20, 56)	13.42	1.94	9,552	1.03	2,470	1.91	2,418	4.14	2,427	7.75	2,349
(1221, 30)	4.23*	1.59	9,287	0.94	2,464	1.84	2,412	4.14	2,426	7.80	2,349
		1.83	3,833	1.00	1,191	1.95	1,194	4.20	1,191	7.83	1,191
		1174.31	1,231,505	63.45	99,228	50.89	84,757	44.39	53,972	72.52	46,762
mm-03-08-04	out	52.97	210,793	16.25	44,888	16.33	40,016	26.03	34,467	61.86	35,973
(31, 84)	894.43	1167.09	1,231,505	55.08	86,115	51.10	84,619	44.75	53,916	72.88	45,672
(2288, 30)	21.95*	53.17	210,793	14.33	40,372	16.64	40,016	26.48	34,467	62.80	35,662
		1058.78	650,509	$\boxed{\mathbf{3.59}}$	2,543	6.41	2,563	17.81	2,443	53.99	8,125
		943.92	907,020	293.66	467,949	580.10	621,834	93.89	94,795	167.25	122,672
mm-03-08-05	out	74.88	141,075	$\boxed{\mathbf{43.88}}$	125,914	247.27	399,764	43.88	46,596	121.38	89,194
(38, 102)	2478.3	948.12	907,009	281.55	455,075	576.24	621,834	91.05	91,329	166.46	122,654
(3691, 60)	44.28*	73.86	141,075	40.84	120,896	252.69	399,764	43.05	44,961	120.05	89,194
		308.33	251,406	135.08	112,264	543.87	308,627	54.45	26,426	152.50	53,561
		2.44	12,177	2.58	8,221	3.00	4,138	8.63	3,925	19.84	3,842
mm-04-08-03	out	$\boxed{\mathbf{1.42}}$	8,331	2.11	7,991	2.78	4,060	8.52	3,907	19.36	3,836
(23, 61)	47.14	2.45	12,091	2.56	8,108	3.03	4,138	8.66	3,925	19.70	3,842
(1422, 30)	4.76*	1.44	8,331	2.16	7,991	2.89	4,060	8.67	3,907	19.50	3,836
		2.25	4,021	2.24	3,127	2.85	1,503	8.59	1,410	20.10	1,410
		-	-	2199.25	7,127,831	501.60	996,770	2391.51	4,841,644	1067.00	3,303,366
mm-04-08-04	out	1126.50	1,705,023	1031.31	5,137,324	$\boxed{\mathbf{215.24}}$	774,176	1015.98	3,292,823	699.65	3,166,653
(36, 87)	654.82	-	-	2200.43	7,127,248	500.30	996,515	2409.06	4,841,578	1063.08	3,302,875
(2615, 30)	32.34*	1133.77	1,705,023	1052.40	5,137,324	217.44	774,176	1019.18	3,292,823	697.10	3,166,420
		635.01	353,630	515.58	427,707	276.71	249,872	412.15	364,113	241.72	251,018
		693.32	1,115,426	453.65	1,074,834	494.44	920,849	442.88	446,455	252.89	236,756
mm-10-08-03	out	406.60	901,908	308.30	905,773	342.90	830,171	319.94	410,324	197.93	222,644
(42, 99)	$\boxed{\mathbf{38.89}}$	707.96	1,115,426	467.79	1,074,834	502.10	920,849	450.83	446,455	255.52	236,702
(2604, 60)	29.31*	408.43	901,908	310.91	905,773	347.16	830,171	324.33	410,324	199.58	222,644
		464.40	444,873	269.13	398,706	353.30	383,310	335.76	193,385	198.97	95,488

Table 4. CPU time and nodes visited for solving Mastermind game instances. Time limit 1 hour. Number of flips for GLS is 200,000. Pseudo tree generated by a single run of the min-fill heuristic, without randomization. SamIam ran out of memory on all test instances.

heuristics compiled along these orderings usually determines the performance ($e.g.$, grid networks). However, when the graph is highly connected, the relatively large induced width causes the AND/OR algorithms to traverse a search space that is very close to a tree (due to dead caches) and therefore the hypergraph based pseudo trees which have far smaller depths improve performance substantially ($e.g.$, genetic linkage).

6 Conclusion

The paper rests on three key contributions. First, we propose a principled approach for handling hard constraints in COPs within the AND/OR search framework, which builds upon progress made in the SAT community. Second, we allow for exploiting non-trivial initial upper bounds which are obtained by a local search scheme. Third, we investigate two heuristics for generating good quality pseudo trees: the min-fill heuristic which minimizes the induced width, and the hypergraph partitioning heuristic that minimizes the depth of the pseudo tree. We demonstrated empirically the impact of these factors on hard benchmarks, including some very challenging networks from the field of genetic linkage analysis.

References

1. R. Marinescu and R. Dechter. And/or branch-and-bound for graphical models. In *International Joint Conference on Artificial Intelligence (IJCAI-2005)*, pages 224–229, 2005.

2. R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.
3. R. Marinescu and R. Dechter. Memory intensive branch-and-bound search for graphical models. In *National Conference on Artificial Intelligence (AAAI)*, 2006.
4. R. Marinescu and R. Dechter. Best-first and/or search for graphical models. In *National Conference on Artificial Intelligence (AAAI)*, 2007.
5. J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.
6. S. Bistarelli, U. Montanari, and F. Rossi. Semiring based constraint solving and optimization. *Journal of ACM*, 44(2):309–315, 1997.
7. E. C. Freuder and M. J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1076–1078, 1985.
8. R. Dechter. Mini-buckets: A general scheme of generating approximations in automated reasoning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1297–1302, 1997.
9. D. Larkin and R. Dechter. Bayesian inference in the presence of determinism. In *International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2003.
10. D. Allen and A. Darwiche. New advances in inference using recursive conditioning. In *Uncertainty in Artificial Intelligence (UAI-2003)*, pages 2–10, 2003.
11. I. Rish and R. Dechter. Resolution vs. search: two strategies for sat. *Journal of Automated Reasoning*, 24(1-2):225–275, 2000.
12. R. Dechter and D. Larkin. Hybrid processing of beliefs and constraints. In *Uncertainty in Artificial Intelligence (UAI)*, pages 112–119, 2001.
13. T. Walsh. Sat v csp. In *Principles and Practice of Constraint Programming (CP)*, pages 441–456, 2000.
14. C. Voudouris. Guided local search for combinatorial optimization problems. Technical report, PhD Thesis. University of Essex, 1997.
15. F. Hutter, H. Hoos, and T. Stutzle. Efficient stochastic local search for mpe solving. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 169–174, 2005.
16. P. Mills and E. Tsang. Guided local search for solving sat and weighted max-sat problems. *Journal of Automated Reasoning (JAR)*, 2000.
17. R. Bayardo and D. P. Miranker. On the space-time trade-off in solving constraint satisfaction problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 558–562, 1995.
18. U. Kjæaerulff. Triangulation of graph-based algorithms giving small total space. *Technical Report, University of Aalborg, Denmark*, 1990.
19. A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 125(1-2):5–41, 2001.
20. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. *Design and Automation Conference*, 2001.
21. E. Santos. On the generation of alternative explanations with implications for belief revision. In *Uncertainty in Artificial Intelligence (UAI)*, pages 339–347, 1991.
22. T. Sang, P. Beame, and H. Kautz. Solving Bayesian networks by weighted model counting. In *National Conference of Artificial Intelligence (AAAI)*, pages 475–482, 2005.
23. M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. *Bioinformatics*, 18(1):189–198, 2002.