AND/OR Branch-and-Bound Search for Combinatorial Optimization in Graphical Models

Radu Marinescu, Rina Dechter

Donald Bren School of Information and Computer Science
University of California, Irvine, CA 92697, USA

Abstract

We introduce a new generation of depth-first Branch-and-Bound algorithms that explore the AND/OR search tree using static and dynamic variable orderings for solving general constraint optimization problems. The virtue of the AND/OR representation of the search space is that its size may be far smaller than that of a traditional OR representation, which can translate into significant time savings for search algorithms. The focus of this paper is on linear space search which explores the AND/OR search tree rather than the search graph and therefore make no attempt to cache information. We investigate the power of the mini-bucket heuristics within the AND/OR search space, in both static and dynamic setups. We focus on two most common optimization problems in graphical models: finding the Most Probable Explanation (MPE) in Bayesian networks and solving Weighted CSPs (WCSP). In extensive empirical evaluations we demonstrate that the new AND/OR Branch-and-Bound approach improves considerably over the traditional OR search strategy and show how various variable ordering schemes impact the performance of the AND/OR search scheme.

Key words: search, AND/OR search, decomposition, graphical models, Bayesian networks, constraint networks, constraint optimization

1 Introduction

Graphical models such as Bayesian networks or constraint networks are a widely used representation framework for reasoning with probabilistic and deterministic information. These models use graphs to capture conditional independencies between variables, allowing a concise representation of the knowledge as well as efficient graph-based query processing algorithms. Optimization problems such as

Email addresses: radum@ics.uci.edu (Radu Marinescu), dechter@ics.uci.edu (Rina Dechter).

finding the most likely state of a Bayesian network or finding a solution that violates the least number of constraints can be defined within this framework and they are typically tackled with either *inference* or *search* algorithms.

Inference-based algorithms (*e.g.*, Variable Elimination, Tree Clustering) were always known to be good at exploiting the independencies captured by the underlying graphical model. They provide worst case time guarantees exponential in the treewidth of the underlying graph. Unfortunately, any method that is time-exponential in the treewidth is also space exponential in the treewidth or separator width, therefore not practical for models with large treewidth.

Search-based algorithms (*e.g.*, depth-first Branch-and-Bound search) traverse the model's search space where each path represents a partial or full solution. The linear structure of such traditional search spaces does not retain the independencies represented in the underlying graphical models and, therefore, search-based algorithms may not be nearly as effective as inference-based algorithms in using this information and therefore do not accommodate informative performance guarantees. This situation has changed in the past few years with the introduction of AND/OR search algorithms for graphical models. In addition, search methods require only an implicit, generative, specification of the functional relationships (that may be given in a procedural or functional form) while inference schemes often rely on an explicit tabular representation over the (discrete) variables. For these reasons, search-based algorithms are the only choice available for models with large treewidth and with implicit representation.

The AND/OR search space for graphical models [1] is a new framework that is sensitive to the independencies in the model, often resulting in exponentially reduced complexities. It is guided by a *pseudo tree* [2,3] that captures independencies in the graphical model, resulting in a search space exponential in the depth of the pseudo tree, rather than in the number of variables.

In this paper we present a new generation of AND/OR Branch-and-Bound algorithms (AOBB) that explore the AND/OR search tree in a depth-first manner for solving optimization problems in graphical models. As in traditional Branch-and-Bound search, the efficiency of these algorithm depends heavily also on its guiding heuristic function. A class of partitioning-based heuristic functions, based on the Mini-Bucket approximation [4] and known as *static mini-bucket heuristics* was shown to be powerful for optimization problems [5] in the context of OR search spaces. The Mini-Bucket algorithm provides a scheme for extracting heuristic information from the functional specification of the graphical model, which is applicable to any graphical model. The accuracy of the Mini-Bucket algorithm is controlled by a bounding parameter, called i-bound, which allows varying degrees of heuristics accuracy and results in a spectrum of search algorithms that can trade off heuristic strength and search [5]. We show here how the pre-computed mini-bucket heuristic as well as any other heuristic information can be incorporated in AND/OR

search, then we introduce *dynamic mini-bucket heuristics*, which are computed dynamically at each node of the search tree.

Since variable orderings can influence dramatically the search performance, we also introduce a collection of *dynamic* AND/OR Branch-and-Bound algorithms that combine the AND/OR decomposition principle with dynamic variable ordering heuristics.

We apply the depth-first AND/OR Branch-and-Bound approach to two common optimization problems in graphical models: finding the Most Probable Explanation (MPE) in Bayesian networks [6] and solving Weighted Constraint Satisfaction Problems (WCSP) [7]. We experiment with both random models and real-world benchmarks. Our results show conclusively that the new depth-first AND/OR Branch-and-Bound algorithms improve dramatically over traditional ones exploring the OR search space, especially when the heuristic estimates are inaccurate and the algorithms rely primarily on search and cannot prune the search space efficiently.

Following preliminary notations and definitions (Section 2), Sections 3, 4 and 5 provide background on graphical models, on the classic OR Branch-and-Bound approach, and the AND/OR representation of the search space. Section 6 presents our new depth-first AND/OR Branch-and-Bound algorithm. Section 7 describes its extension with dynamic variable ordering heuristics. Section 8 presents several general purpose heuristic functions that can guide the search focusing on the mini-bucket heuristics. Section 9 is dedicated to an extensive empirical evaluation, Section 10 overviews related work and Section 11 provides a summary and concluding remarks.

2 Preliminaries

2.1 Notations

A reasoning problem is defined in terms of a set of variables taking values on finite domains and a set of functions defined over these variables. We denote variables by uppercase letters ($e.g.$, $X, Y, Z, ...$), subsets of variables by bold faced uppercase letters ($e.g.$, $\mathbf{X}, \mathbf{Y}, \mathbf{Z}, ...$) and values of variables by lower case letters ($e.g.$, $x, y, z, ...$). An assignment $(X_1 = x_1, ..., X_n = x_n)$ can be abbreviated as $x = (\langle X_1, x_1 \rangle, ..., \langle X_n, x_n \rangle)$ or $x = (x_1, ..., x_n)$. For a subset of variables $\mathbf{Y}$, $D_{\mathbf{Y}}$ denotes the Cartesian product of the domains of variables in $\mathbf{Y}$. $x_{\mathbf{Y}}$ and $x[\mathbf{Y}]$ are both used as the projection of $x = (x_1, ..., x_n)$ over a subset $\mathbf{Y}$. We denote functions by letters f, h, g etc., and the scope (set of arguments) of a function f by $scope(f)$.

2.2 Graph Concepts

DEFINITION **1 (directed, undirected graphs)** *A* directed graph *is defined by a pair* $G = \{\mathbf{V}, \mathbf{E}\}$*, where* $\mathbf{V} = \{X_1, ..., X_n\}$ *is a set of vertices (nodes), and* $\mathbf{E} = \{(X_i, X_j) | X_i, X_j \in V\}$ *is a set of edges (arcs). If* $(X_i, X_j) \in \mathbf{E}$*, we say that* X_i *points to* X_j*. The degree of a vertex is the number of incident arcs to it. For each vertex* X_i*,* $pa(X_i)$ *or* pa_i*, is the set of vertices pointing to* X_i *in* G*, while the set of child vertices of* X_i*, denoted* $ch(X_i)$*, comprises the variables that* X_i *points to. The family of* X_i*, denoted* F_i*, includes* X_i *and its parent vertices. A directed graph is acyclic if it has no directed cycles. An* undirected *graph is defined similarly to a directed graph, but there is no directionality associated with the edges.*

DEFINITION **2 (induced width)** *An* ordered graph *is a pair* (G, d) *where* G *is an undirected graph, and* $d = X_1, ..., X_n$ *is an ordering of the nodes. The* width of a node *is the number of the node's neighbors that precede it in the ordering. The* width of an ordering d *is the maximum width over all nodes. The* induced width of an ordered graph*, denoted by* $w^*(d)$*, is the width of the induced ordered graph obtained as follows: nodes are processed from last to first; when node* X_i *is processed, all its preceding neighbors are connected. The* induced width *of a graph, denoted by* w^**, is the minimal induced width over all its orderings.*

DEFINITION **3 (hypergraph)** *A* hypergraph *is a pair* $H = (\mathbf{X}, \mathbf{S})$*, where* $\mathbf{S} = \{S_1, ..., S_t\}$ *is a set of subsets of* $\mathbf{X}$*, called hyperedges.*

DEFINITION **4 (tree decomposition)** *A* tree decomposition *of a hypergraph* $H = (\mathbf{X}, \mathbf{S})$*, is a tree* $T = (\mathbf{V}, \mathbf{E})$*, where* $\mathbf{V}$ *is a set of nodes, also called "clusters", and* $\mathbf{E}$ *is a set of edges, together with a labeling function* χ *that associates with each vertex* $v \in \mathbf{V}$ *a set* $\chi(v) \subseteq \mathbf{X}$ *satisfying:*

(1) For each $S_i \in \mathbf{S}$ *there exists a vertex* $v \in \mathbf{V}$ *such that* $S_i \subseteq \chi(v)$*;*
(2) (running intersection property) For each $X_i \in \mathbf{X}$*, the set* $\{v \in \mathbf{V} | X_i \in \chi(v)\}$ *induces a connected subtree of* T*.*

DEFINITION **5 (treewidth, pathwidth)** *The* width *of a tree decomposition of a hypergraph is the size of the largest cluster minus 1 (i.e.,* $max_v |\chi(v) - 1|$*). The* treewidth *of a hypergraph is the minimum width along all possible tree decompositions. The* pathwidth *is the treewidth over the restricted class of chain decompositions.*

2.3 AND/OR Search Spaces

An AND/OR state space representation of a problem is a 4-tuple $\langle S, O, S_g, s_0 \rangle$ [8]. S is a set of states which can be either OR or AND states (the OR states repre-

sent alternative ways for solving the problem while the AND states often represent problem decomposition into subproblems, all of which need to be solved). O is a set of operators. An OR operator transforms an OR state into another state, and an AND operator transforms an AND state into a set of states. There is a set of goal states $S_g \subseteq S$ and a start node $s_0 \in S$.

The AND/OR state space model induces an explicit AND/OR search *graph*. Each state is a node and child nodes are obtained by applicable AND or OR operators. The search graph includes a *start* node. The terminal nodes (having no children) are labeled as SOLVED or UNSOLVED.

A *solution tree* of an AND/OR search graph G is a subtree which: (1) contains the start node s_0; (2) if n in the tree is an OR node then it contains one of its child nodes in G, and if n is an AND node it contains all its children in G; (3) all its terminal nodes are SOLVED.

3 Graphical Models

Graphical models include constraint networks defined by relations of allowed tuples, directed or undirected probabilistic networks and cost networks defined by cost functions. Each graphical model comes with its specific optimization queries such as finding a solution of a constraint network that violates the least number of constraints, finding the most probable assignment given some evidence, posed over probabilistic networks or finding the optimal solution for cost networks.

In general, a graphical model is defined by a collection of functions $\mathbf{F}$, over a set of variables $\mathbf{X}$, conveying probabilistic or deterministic information, whose structure is captured by a graph.

DEFINITION **6 (graphical model)** *A* graphical model $\mathcal{R}$ *is defined by a 4-tuple* $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \otimes \rangle$*, where:*

(1) $\mathbf{X} = \{X_1, ..., X_n\}$ *is a set of variables;*
(2) $\mathbf{D} = \{D_1, ..., D_n\}$ *is the set of their respective finite domains of values;*
(3) $\mathbf{F} = \{f_1, ..., f_r\}$ *is a set of real-valued functions, each defined over a subset of variables* $S_i \subseteq \mathbf{X}$ *(i.e., the scope);*
(4) $\otimes_i f_i \in \{\prod_i f_i, \sum_i f_i\}$ *is a combination operator.*

The graphical model represents the combination of all its functions: $\otimes_{i=1}^{r} f_i$.

DEFINITION **7 (cost of a full and partial assignment)** *Given a graphical model* $\mathcal{R}$*, the cost of a full assignment* $x = (x_1, ..., x_n)$ *is defined by:*

$$c(x) = \otimes_{f \in \mathbf{F}} f(x[scope(f)])$$

Given a subset of variables $\mathbf{Y} \subseteq \mathbf{X}$, the cost of a partial assignment y is the combination of all the functions whose scopes are included in $\mathbf{Y}$, namely $\mathbf{F_Y}$, evaluated at the assigned values. Namely, $c(y) = \otimes_{f \in \mathbf{F_Y}} f(y[scope(f)])$. We will often abuse notation writing $c(y) = \otimes_{f \in F_\mathbf{Y}} f(y)$ instead.

DEFINITION **8 (primal graph)** *The* primal graph *of a graphical model has the variables as its nodes and an edge connects any two variables that appear in the scope of the same function.*

There are various queries (tasks) that can be posed over graphical models. We refer to all as *automated reasoning problems*. In general, an optimization task is a reasoning problem defined as a function from a graphical model to a set of elements, most commonly, the real numbers.

DEFINITION **9 (constraint optimization problem)** *A* constraint optimization problem (COP) *is a pair $\mathcal{P} = \langle \mathcal{R}, \Downarrow_\mathbf{X} \rangle$, where $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \otimes \rangle$ is a graphical model. If S is the scope of function $f \in \mathbf{F}$ and $\Downarrow_S f \in \{max_S f, min_S f\}$. The optimization problem is to compute $\Downarrow_\mathbf{X} \otimes_{i=1}^{r} f_i$.*

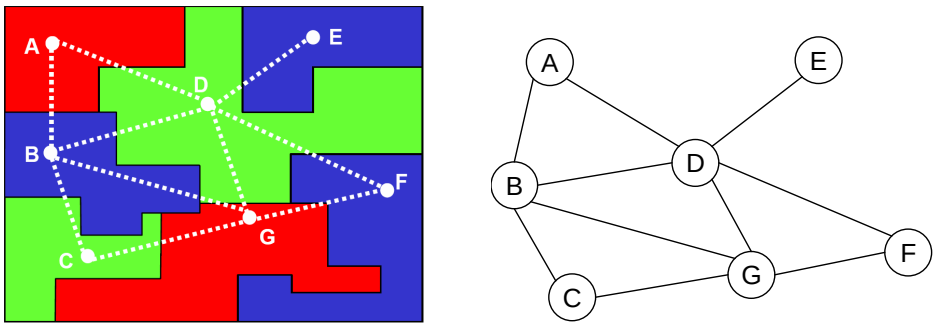
The min/max ($\Downarrow$) operator is sometimes called an *elimination* operator because it removes the arguments from the input functions' scopes.

We next elaborate on several popular graphical models of constraint networks, cost networks and belief networks which will be the primary focus of this paper.

3.1 Constraint Networks

Constraint Satisfaction is a framework for formulating real-world problems as a set of constraints between variables. The task is to find an assignment of values to variables that does not violate any constraint, or else to conclude that problem is inconsistent. Such problems are graphically represented by nodes corresponding to variables and edges corresponding to constraints between variables.

DEFINITION **10 (constraint network)** *A* constraint network *is a graphical model $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C}, \bowtie \rangle$, where $\mathbf{X} = \{X_1, ..., X_n\}$ is a set of variables, associated with discrete-valued domains $\mathbf{D} = \{D_1, ..., D_n\}$, and a set of constraints $\mathbf{C} = \{C_1, ..., C_r\}$. Each constraint C_i is a pair (S_i, R_i), where R_i is a relation $R_i \subseteq D_{S_i}$ defined on a subset of variables $S_i \subseteq \mathbf{X}$. The relation denotes all compatible tuples of D_{S_i} allowed by the constraint. The combination operator $\otimes$ is* join, $\bowtie$. *The primal graph of a constraint network is called a* constraint graph. *A solution is an assignment of values to all variables $x = (x_1, ..., x_n)$, $x_i \in D_i$, such that $\forall C_i \in \mathbf{C}$, $x_{S_i} \in R_i$. The constraint network represents its set of solutions, $\bowtie_i C_i$. The elimination operator in this case is* projection.



(a) Graph coloring problem (b) Constraint graph

Fig. 1. Constraint network.

Example 1 *Figure 1(a) shows a graph coloring problem that can be modeled by a constraint network. Given a map of regions, the problem is to color each region by one of the given colors {red, green, blue}, such that neighboring regions have different colors. The variables of the problem are the regions, and each one has the domain {red,green,blue}. The constraints are the relation "different" between neighboring regions. Figure 1(b) shows the constraint graph, and a solution* $(A = red, B = blue, C = green, D = green, E = blue, F = blue, G = red)$ *is given in Figure 1(a).*

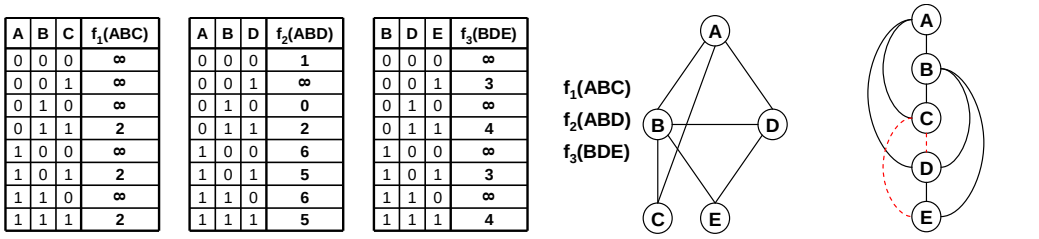
3.2 Cost Networks

An immediate extension of constraint networks are *cost networks* where the set of functions are real-valued cost functions, the combination and elimination operators are *summation* and *minimization*, respectively, and the primary constraint optimization task is to find a solution with minimum cost.

A special class of COPs which has gained a lot of interest in recent years is the Weighted Constraint Satisfaction Problem (WCSP). WCSP extends the classical CSP formalism with *soft constraints* which are represented as integer-valued cost functions. Formally,

DEFINITION **11 (WCSP)** *A* Weighted Constraint Satisfaction Problem (WCSP) *is a graphical model* $\langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$ *where each of the functions* $f_i \in \mathbf{F}$ *assigns "0" (no penalty) to allowed tuples and a positive integer penalty cost to the forbidden tuples. Namely,* $f_i : D_{S_{i_1}} \times ... \times D_{S_{i_t}} \rightarrow \mathbb{N}$*, where* $S_i = \{S_{i_1}, ..., S_{i_t}\}$ *is the scope of the function. The optimization problem is to find a value assignment to the variables with minimum penalty cost, namely finding* $\Downarrow_{\mathbf{X}} \otimes_i f_i = min_{\mathbf{X}} \sum_i f_i$.

DEFINITION **12 (MAX-CSP)** *A* MAX-CSP *is a WCSP with all penalty costs equal to* 1*. Namely,* $\forall f_i \in \mathbf{F}$*,* $f_i : D_{S_{i_1}} \times ... \times D_{S_{i_t}} \rightarrow \{0, 1\}$*, where* $S_i = \{S_{i_1}, ..., S_{i_t}\}$ *is the scope of* f_i.



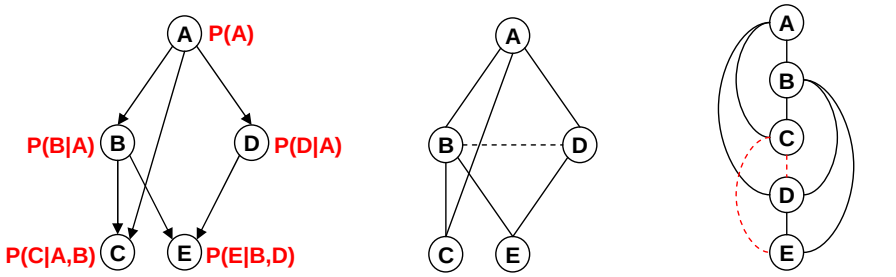
(a) Functions (b) Primal graph (c) Induced graph

Fig. 2. A WCSP instance with cost functions $f_1(A, B, C)$, $f_2(A, B, D)$ and $f_3(B, D, E)$.

Solving a MAX-CSP task can also be interpreted as finding an assignment that violates the minimum number of constraints (or maximizes the number of satisfied constraints). Many real-world problems can be formulated as MAX-CSP/WCSPs, including resource allocation problems [9], scheduling [10], bioinformatics [11,12], combinatorial auctions [13,14] or maximum satisfiability problems [15].

Example 2 *Figure 2 shows an example of a Weighted CSP instance with bi-valued variables. The cost functions are given in Figure 2(a). The value ∞ indicates an inconsistent tuple. Figures 2(b) and 2(c) depict the primal and the induced graph along the ordering $d = (A, B, C, D, E, F)$, respectively. The induced graph is obtained by adding the dotted-arcs. It can be shown that the minimal cost solution is 5 and corresponds to the assignment $(A = 0, B = 1, C = 1, D = 0, E = 1)$.*

Overview of Related Work on MAX-CSP/WCSP. MAX-CSP and WCSP can also be formulated using the semi-ring framework introduced by [7]. As an optimization version of constraint satisfaction, MAX-CSP/WCSP is NP-hard. A number of complete and incomplete algorithms have been developed for MAX-CSP and WCSP, respectively. Stochastic Local Search (SLS) algorithms, such as GSAT [16,17], developed for Boolean Satisfiability and Constraint Satisfaction can be directly applied to MAX-CSP [18]. Since they are incomplete, SLS algorithms cannot guarantee an optimal solution, but they have been successful in practice on many classes of SAT and CSP problems. A number of search-based complete algorithms, using partial forward checking [19] for heuristic computation, have been developed [20,21]. The Branch-and-Bound algorithm proposed by [5] uses bounded mini-bucket inference to compute the guiding heuristic function. More recently, [22–24] introduced a family of depth-first Branch-and-Bound algorithms that maintain various levels of directional soft arc-consistency for solving WCSPs. The optimization method, called *Backtracking with Tree Decomposition* (BTD), developed by [25] uses a tree decomposition of the graphical model to capture the problem structure and guide the search more effectively. BTD can also be interpreted as traversing an AND/OR search graph using substantial caching [1].



(a) Belief network (b) Moral graph (c) Induced graph

Fig. 3. An example of a belief network.

3.3 Belief Networks

Belief networks [6] provide a formalism for reasoning about partial beliefs under conditions of uncertainty. They are defined by a directed acyclic graph over vertices representing variables of interest ($e.g.$, the temperature of a device, the gender of a patient, a feature of an object, the occurrence of an event). The arcs signify the existence of direct causal influences between linked variables quantified by conditional probabilities that are attached to each cluster of parents-child vertices in the network.

DEFINITION **13 (belief network)** *A* belief network (BN) *is a graphical model* $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}_G, \prod \rangle$*, where* $\mathbf{X} = \{X_1, ..., X_n\}$ *is a set of variables over multi-valued domains* $\mathbf{D} = \{D_1, ..., D_n\}$*. Given a directed acyclic graph* G *over* $\mathbf{X}$ *as nodes,* $\mathbf{P}_G = \{P_i\}$*, where* $P_i = \{P(X_i|pa(X_i))\}$ *are conditional probability tables (CPTs for short) associated with each variable* X_i*, and* $pa(X_i)$ *are the parents of* X_i *in the acyclic graph* G*. A belief network represents a joint probability distribution over* $\mathbf{X}$*,* $P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i|x_{pa(X_i)})$*. An evidence set* e *is an instantiated subset of variables.*

When formulated as a graphical model, the functions in $\mathbf{P}_G$ denote conditional probability tables and the scopes of these functions are determined by the directed acyclic graph G: each function f_i ranges over variable X_i and its parents in G. The combination operator is multiplication, namely $\otimes_j = \prod_j$. The primal graph of a belief network is called a *moral graph*. It connects any two variables appearing in the same probability table.

Example 3 *An example of a belief network is given in Figure 3(a). This belief network represents the joint probability distribution* $P(A, B, C, D, E) = P(A) \cdot P(B|A) \cdot P(C|A, B) \cdot P(D|A) \cdot P(E|B, D)$*. In this case,* $pa(E) = \{B, D\}$*,* $pa(B) = \{A\}$*,* $pa(A) = \emptyset$*,* $ch(A) = \{B, C, D\}$*. We see that the moral graph shown in Figure 3(b) is identical to the graph in Figure 2(b), and therefore, the induced width along the ordering* $d = (A, B, C, D, E)$ *is identical. Namely, the width and induced width of the ordered moral graph is 3.*

DEFINITION **14 (most probable explanation)** *Given a belief network and evidence* e*, the* Most Probable Explanation *(MPE) task is to find a complete assignment which agrees with the evidence, and which has the highest probability among all such assignments. Namely, to find an assignment* $(x_1^o, ..., x_n^o)$ *such that:*

$$P(x_1^o, ..., x_n^o) = max_{x_1,...,x_n} \prod_{k=1}^{n} P(x_k, e|x_{pa_k})$$

As a reasoning problem, the MPE task is to find $\Downarrow_{\mathbf{X}} \otimes_i f_i = max_{\mathbf{X}} \prod_{i=1}^{n} P_i$.

The MPE task appears in applications such as diagnosis, abduction, and explanation. For example, given data on clinical findings, MPE can postulate on a patient's probable affliction. In decoding, the task is to identify the most likely input message transmitted over a noisy channel given the observed output. Researchers in natural language consider the understanding of text to consist of finding the most likely facts (in internal representation) that explain the existence of the given text. In computer vision and image understanding, researchers formulate the problem in terms of finding the most likely set of objects that explain the image.

Overview of Related Work on MPE. It is known that solving the MPE task is NP-hard [26]. Complete algorithms use either the cycle cutset technique (also called conditioning) [6], the join-tree-clustering technique [27,28], or the bucket elimination scheme [29]. However, these methods work well only if the network is sparse enough to allow small cutsets or small clusters. The complexity of algorithms based on the cycle cutset idea is time exponential in the cutset size but require only linear space. The complexity of join-tree-clustering and bucket elimination algorithms are both time and space exponential in the cluster size that equals the induced-width of the network's moral graph. Following Pearl's stochastic simulation algorithms [6], the suitability of Stochastic Local Search (SLS) algorithms for MPE was studied in the context of medical diagnosis applications [30] and more recently in [31–33]. Best-First search algorithms were proposed [34] as well as algorithms based on linear programming [35]. Some extensions are also available for the task of finding the k most-likely explanations [36,37]. Recently, [5,38] introduced a collection of depth-first Branch-and-Bound algorithms that use bounded inference, in particular the Mini-Bucket approximation [4], for computing the guiding heuristic function.

In the next section we present some of these known approaches on which we build in this paper.

4 Search and Inference for Combinatorial Optimization

4.1 Bucket and Mini-Bucket Elimination

Bucket Elimination (BE) is a unifying algorithmic framework for dynamic programming algorithms applicable to probabilistic and deterministic reasoning [29].

The input to a bucket elimination algorithm is an optimization problem, namely a collection of cost functions or relations. Given a variable ordering, the algorithm partitions the functions into buckets, each associated with a single variable. A function is placed in the bucket of its argument that appears latest in the ordering. The algorithm has two phases. During the first, top-down phase, it processes each bucket, from last to first by a variable elimination procedure that computes a new function which is placed in a lower bucket. The variable elimination procedure computes the combination of all functions and eliminates the bucket's variable. During the second, bottom-up phase, the algorithm constructs a solution by assigning a value to each variable along the ordering, consulting the functions created during the top-down phase. It can be shown that:

THEOREM **1 (complexity of BE [29])** *The time and space complexity of BE applied along order d is $O(r \cdot k^{(w^*+1)})$ and $O(n \cdot k^{w^*})$ respectively, where w^* is the induced width of the primal graph along the ordering d, r is the number of functions, n is the number of variables and k bounds the domain size.*

Bucket Elimination can be viewed as message passing from leaves to root along a bucket tree [39]. Let $\{B(X_1), ..., B(X_n)\}$ denote a set of buckets, one for each variable, along an ordering $d = (X_1, ..., X_n)$. A *bucket tree* of $\mathcal{R}$ has buckets as its nodes. Bucket $B(X)$ is connected to bucket $B(Y)$ if the function generated in bucket $B(X)$ by BE is placed in $B(Y)$. The variables of $B(X)$, are those appearing in the scopes of any of its new and old functions. Therefore, in a bucket tree, every vertex $B(X)$ other than the root, has one parent vertex $B(Y)$ and possibly several child vertices $B(Z_1), ..., B(Z_t)$.

Mini-Bucket Elimination (MBE) is an approximation designed to avoid the space and time problem of full bucket elimination [4] by partitioning large buckets into smaller subsets, called *mini-buckets* which are processed independently. Here is the rationale. Consider an optimization problem over a graphical model $\langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ with *summation* and *minimization* as the combination and elimination operators, respectively. Let $h_1, ..., h_j$ be the functions in bucket $B(X_p)$ of variable X_p. When BE processes $B(X_p)$, it computes the function h^p: $h^p = min_{X_p} \sum_{i=1}^{j} h_i$, where $scope(h^p) = \cup_{i=1}^{j} S_i - \{X_p\}$. The *Mini-Bucket* algorithm, on the other hand, creates a partition $Q' = \{Q_1, ..., Q_t\}$ where the mini-bucket Q_l contains the functions $h_{l_1}, ..., h_{l_t}$. The approximation processes each mini-bucket separately, there-

fore computing $g^p = \sum_{l=1}^{t} min_{X_p} \sum_{i=1}^{t} h_{l_i}$. Clearly, g^p is a lower bound on h^p, namely $g^p \leq h^p$ (for maximization, g^p is an upper bound). Therefore, the bound computed in each bucket yields an overall bound on the cost of the solution. An upper bound (resp. lower bound for maximization problems) can be obtained by the cost of the assignment computed when going forward, from the first bucket to the last, consulting the generated functions.

The quality of the bound depends on the degree of partitioning into mini-buckets. Given a bounding parameter i (called here i-bound), the algorithm creates an i-partitioning, where each mini-bucket includes no more than i variables. Algorithm MBE(i) is parameterized by this i-bound. It outputs not only a lower bound on the cost of the optimal solution and an assignment, but also the collection of the augmented buckets. By comparing the bound computed by MBE(i) to the cost of the assignment output by MBE(i), we can always have an interval bound on the error for that given instance.

The complexity of the algorithm is time and space $O(exp(i))$ where $i < n$. It can be viewed as solving by bucket elimination a simplified problem that is sparser [5]. When the i-bound is large enough ($i.e.$, $i \geq w^*$), the Mini-Bucket algorithm coincides with full bucket elimination on the original problem. In summary,

THEOREM **2 (complexity of MBE(i) [4])** *Algorithm MBE(i) generates an interval bound of the optimal solution, and its time and space complexity are $O(r \cdot k^i)$ and $O(r \cdot k^{i-1})$ respectively, where r is the number of functions and k bounds the domain size.*

4.2 Branch-and-Bound Search with Mini-Bucket Heuristics

Most exact search algorithms for solving optimization problems in graphical models follow a *Branch-and-Bound* schema [40]. These algorithms perform a depth-first traversal on the search tree defined by the problem, where internal nodes represent partial assignments and leaf nodes stand for complete ones. Throughout the search, the algorithm maintains a global bound on the cost of the optimal solution, which corresponds to the cost of the best full variable instantiation found thus far. At each node, the algorithm computes a heuristic estimate of the best solution extending the current partial assignment and prunes the respective subtree if the heuristic estimate is not better than the current global bound (that is - not greater for maximization problems, not smaller for minimization problems).

The algorithm requires only a limited amount of memory and can be used as an anytime scheme, namely whenever interrupted, Branch-and-Bound outputs the best solution found so far.

The effectiveness of Branch-and-Bound depends on the quality of the heuristic

function. A heuristic function that is accurate, but not too hard to compute, is desirable. The most common class of Branch-and-Bound algorithms was developed for integer programming where the heuristic function is generated by solving a linear relaxation of the problem via the simplex algorithm [41,40,42].

In the past few years [5] showed that the intermediate functions generated by MBE(i) can be used to create a heuristic function that provides an optimistic estimate of the best cost extension of any partial variable assignment, and therefore can guide Branch-and-Bound search. Since these heuristics have varying strengths depending on the Mini-Bucket i-bound, they allow a controlled tradeoff between pre-processing (for heuristics generation) and search. The following shows how a heuristic evaluation function can be extracted from the functions generated by the Mini-Bucket algorithm.

DEFINITION **15 (mini-bucket heuristic evaluation function [5])** *Given an ordered set of augmented buckets $\{B(X_1), ..., B(X_p), ..., B(X_n)\}$ generated by the Mini-Bucket algorithm MBE(i) along the ordering $d = (X_1, ..., X_p, ..., X_n)$, and given a partial assignment $\bar{x}^p = (x_1, ..., x_p)$, the heuristic evaluation function $f(\bar{x}^p) = g(\bar{x}^p) + h(\bar{x}^p)$ is defined follows:*

1. *$g(\bar{x}^p) = (\sum_{f_i \in B(X_1..X_p)} f_i)(\bar{x}^p)$ is the combination of all the input functions that are fully instantiated along the current path, where $B(X_1..X_p)$ denotes the buckets $B(X_1)$ through $B(X_p)$ in the ordering d;*
2. *The* mini-bucket heuristic *function $h(\bar{x}^p)$ is defined as the sum of all the intermediate functions h_j^k that satisfy the following properties:*
   - *They are generated in buckets $B(X_{p+1})$ through $B(X_n)$,*
   - *They reside in buckets $B(X_1)$ through $B(X_p)$.*

Kask and Dechter showed [5] that for any partial assignment $\bar{x}^p = (x_1, ..., x_p)$ of the first p variables in the ordering, the evaluation function $f(\bar{x}^p) = g(\bar{x}^p) + h(\bar{x}^p)$ is *admissible* and *monotonic* [8].

Branch-and-Bound guided by the *Mini-Bucket heuristics* is denoted by BBMB(i). The algorithm was introduced for a static variable ordering and has a space complexity dominated by the pre-processing step which is exponential in the i-bound [5]. BBMB(i) was evaluated extensively for probabilistic and deterministic optimization tasks. The results showed conclusively that the scheme overcomes partially the memory explosion of bucket elimination allowing a gradual tradeoff of space for time, and of time for accuracy when used as an anytime scheme.

Subsequently, [43,38] explored the feasibility of generating partition-based heuristics during search, rather than in a pre-processing manner. This allows dynamic variable and value ordering, a feature that can have tremendous impact on search. The dynamic generation of these heuristics is facilitated by Mini-Bucket-Tree Elimination, MBTE(i), a partition-based approximation defined over cluster-trees [43]. MBTE(i) outputs multiple (lower or upper) bounds for each possible variable and

value extension at once, which is much faster than running MBE(i) n times, once for each variable.

The resulting *Branch-and-Bound with Mini-Bucket-Tree heuristics* [43,38], called BBBT(i), applies the MBTE(i) heuristic computation at each node of the search tree. Clearly, the algorithm has a higher time overhead compared with BBMB(i) for the same i-bound, which computes the mini-buckets once. It is exponential in the i-bound multiplied by the number of nodes visited, but it can prune the search space much more effectively. Experimental results on probabilistic and deterministic graphical models showed that the power of BBBT(i) is more pronounced over BBMB(i) at relatively small i-bounds, which is significant because small i-bounds require restricted space.

5 AND/OR Search Trees for Graphical Models

In this section we overview the AND/OR search space for graphical models, which forms the core of our work in this paper.

The usual way to do search in graphical models is to instantiate variables in turn, following a static/dynamic variable ordering. In the simplest case, this process defines a search tree (called here OR search tree), whose state nodes represent partial variable assignments. Since this search space does not capture the structure of the underlying graphical model an AND/OR search space recently introduced for general graphical models [1] can be used instead. The AND/OR search space is defined using a backbone *pseudo tree* [2,3].

DEFINITION **16 (pseudo tree, extended graph)** *Given an undirected graph $G = (\mathbf{V}, \mathbf{E})$, a directed rooted tree $\mathcal{T} = (\mathbf{V}, \mathbf{E}')$ defined on all its nodes is called* pseudo tree *if any arc of G which is not included in E' is a back-arc, namely it connects a node to an ancestor in $\mathcal{T}$. Given a pseudo tree $\mathcal{T}$ of G, the* extended graph *of G relative to $\mathcal{T}$ is defined as $G^{\mathcal{T}} = (\mathbf{V}, \mathbf{E} \cup \mathbf{E}')$ (see Example 4 ahead).*

We next define the notion of AND/OR search tree for a graphical model.

DEFINITION **17 (AND/OR search tree [1])** *Given a graphical model $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, its primal graph G and a backbone pseudo tree $\mathcal{T}$ of G, the associated AND/OR search tree, denoted $S_{\mathcal{T}}(\mathcal{R})$, has alternating levels of AND and OR 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 simply x_i) and correspond to value assignments in the domains of the variables. The structure of the AND/OR search tree is based on the underlying backbone pseudo tree $\mathcal{T}$. The root of the AND/OR search tree is an OR node labeled with the root of $\mathcal{T}$. A path from the root of the search tree $S_{\mathcal{T}}(\mathcal{R})$ to a node n is denoted by π_n. If n is labeled X_i or x_i the path will be denoted $\pi_n(X_i)$ or $\pi_n(x_i)$,*

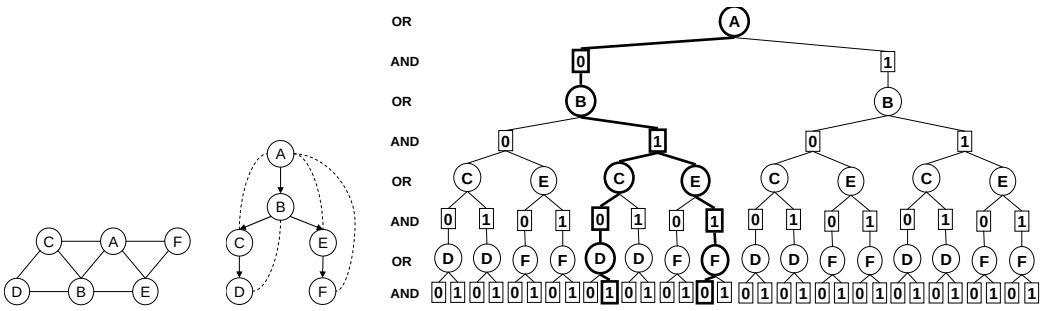


Fig. 4. AND/OR search spaces for graphical models.

respectively. The assignment sequence along path π_n, denoted $asgn(\pi_n)$, is the set of value assignments associated with the AND nodes along π_n:

$$asgn(\pi_n(X_i)) = \{\langle X_1, x_1\rangle, \langle X_2, x_2\rangle, ..., \langle X_{i-1}, x_{i-1}\rangle\}$$
$$asgn(\pi_n(x_i)) = \{\langle X_1, x_1\rangle, \langle X_2, x_2\rangle, ..., \langle X_i, x_i\rangle\}$$

The set of variables associated with OR nodes along the path π_n is denoted by $var(\pi_n)$: $var(\pi_n(X_i)) = \{X_1, ..., X_{i-1}\}$, $var(\pi_n(x_i)) = \{X_1, ..., X_i\}$. The parent-child relationship between nodes in the search space are defined as follows:

(1) An OR node, n, labeled by X_i has a child AND node labeled $\langle X_i, x_i\rangle$ iff $\langle X_i, x_i\rangle$ is consistent with $asgn(\pi_n)$, relative to the hard constraints.
(2) An AND node, n, labeled by $\langle X_i, x_i\rangle$ has a child OR node labeled Y iff Y is a child of X_i in the backbone pseudo tree $\mathcal{T}$. Each OR arc, emanating from an OR to an AND node is associated with a weight to be defined shortly.

Semantically, the OR states in the AND/OR search tree represent alternative ways of solving a problem, whereas the AND states represent problem decomposition into independent subproblems, conditioned on the assignment above them, all of which need to be solved.

Following the general definition of a solution tree for AND/OR search graphs [8] we have here that:

DEFINITION **18 (solution tree)** *A* solution tree *of an AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$ is an AND/OR subtree T such that:*

(1) It contains the root of $S_{\mathcal{T}}(\mathcal{R})$, s;
(2) If a non-terminal AND node $n \in S_{\mathcal{T}}(\mathcal{R})$ is in T then all of its children are in T;
(3) If a non-terminal OR node $n \in S_{\mathcal{T}}(\mathcal{R})$ is in T then exactly one of its children is in T;
(4) All its leaf (terminal) nodes are consistent.

Example 4 *Figure 4(a) shows the primal graph of cost network with 6 bi-valued variables A, B, C, D, E and F, and 9 binary cost functions. Figure 4(b) displays a pseudo tree together with the back-arcs (dotted lines). Figure 4(c) shows the AND/OR search tree based on the pseudo tree. A solution tree is highlighted. Notice that once variables A and B are instantiated, the search space below the AND node $\langle B, 0\rangle$ decomposes into two independent subproblems, one that is rooted at C and one that is rooted at E, respectively.*

The virtue of an AND/OR search tree representation is that its size may be far smaller than the traditional OR search tree. It was shown that:

THEOREM **3 (size of AND/OR search trees [1])** *Given a graphical model $\mathcal{R}$ and a backbone pseudo tree $\mathcal{T}$, its AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$ is sound and complete, and its size is $O(l \cdot k^m)$ where m is the depth of the pseudo tree, l bounds its number of leaves, and k bounds the domain size.*

Given a *tree decomposition* of the primal graph G having n nodes, whose treewidth is w^*, it is known there exists a pseudo tree $\mathcal{T}$ of G whose depth, m, satisfies: $m \leq w^* \cdot logn$ [44,45]. Therefore,

THEOREM **4 ([1])** *A graphical model that has a treewidth w^* has an AND/OR search tree whose size is $O(n \cdot k^{w^* \cdot logn})$, where k bounds the domain size and n is the number of variables.*

The arcs in the AND/OR trees are associated with weights that are defined based on the graphical model's functions and the combination operator. We next define arc weights for any graphical model using the notion of *buckets of functions*.

DEFINITION **19 (buckets relative to a pseudo tree)** *Given a graphical model $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}\rangle$ and a backbone pseudo tree $\mathcal{T}$, the bucket of X_i relative to $\mathcal{T}$, denoted $B_{\mathcal{T}}(X_i)$, is the set of functions whose scopes contain X_i and are included in $path_{\mathcal{T}}(X_i)$, which is the set of variables from the root to X_i in $\mathcal{T}$. Namely,*

$$B_{\mathcal{T}}(X_i) = \{f \in \mathbf{F} | X_i \in scope(f), scope(f) \subseteq path_{\mathcal{T}}(X_i)\}$$

For simplicity and without loss of generality we consider in the remainder of the paper a graphical model $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}\rangle$ for which the combination and elimination operators are *summation* and *minimization*, respectively.

DEFINITION **20 (OR-to-AND weights)** *Given an AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$, of a graphical model $\mathcal{R}$, the weight $w_{(n,m)}(X_i, x_i)$ (or simply $w(X_i, x_i)$) of arc (n, m), where X_i labels n and x_i labels m, is the* combination *of all the functions in $B_{\mathcal{T}}(X_i)$ assigned by values along π_m. Formally,*

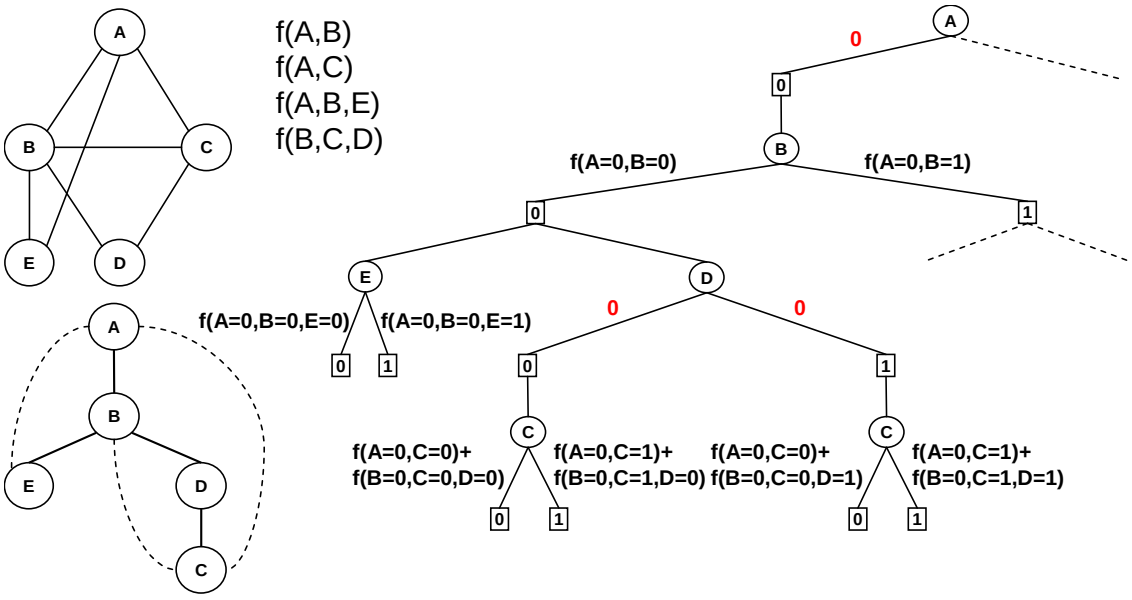


Fig. 5. Arc weights for a cost network with 5 variables and 4 cost functions.

$$w(X_i, x_i) = \begin{cases} 0 & \text{, if } B_{\mathcal{T}}(X_i) = \emptyset \\ \sum_{f \in B_{\mathcal{T}}(X_i)} f(asgn(\pi_m)) & \text{, otherwise} \end{cases}$$

DEFINITION **21 (cost of a solution tree)** *Given a weighted AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$, of a graphical model $\mathcal{R}$, and given a solution tree T having OR-to-AND set of arcs $arcs(T)$, the cost of T is defined by $f(T) = \sum_{e \in arcs(T)} w(e)$.*

Let T_n be the subtree of T rooted at node n in T. The cost $f(T)$ can be computed recursively, as follows:

1. *If T_n consists only of a terminal AND node n, then $f(T_n) = 0$.*
2. *If T_n is rooted at an OR node having an AND child m in T_n, then $f(T_n) = w(n, m) + f(T_m)$.*
3. *If T_n is rooted at an AND node having OR children $m_1, ..., m_k$ in T_n, then $f(T_n) = \sum_{i=1}^{k} f(T_{m_i})$.*

Example 5 *Figure 5 shows the primal graph of a cost network with functions $\{f(A, B), f(A, C), f(A, B, E), f(B, C, D)\}$, a pseudo tree that drives its weighted AND/OR search tree, and a portion of the AND/OR search tree with appropriate weights on the arcs expressed symbolically. In this case the bucket of E contains the function $f(A, B, E)$, the bucket of C contains two functions $f(A, C)$ and $f(B, C, D)$ and the bucket of B contains the function $f(A, B)$. We see indeed that the weights on the arcs from the OR node E to any of its AND value assignments include only the instantiated function $f(A, B, E)$, while the weights on the arcs connecting C to its AND child nodes are the sum of the two functions in its bucket instantiated appropriately. Notice that the buckets of A and D are empty and there-*

fore the weights associated with the respective arcs are 0.

With each node n of the search tree we can associate a value $v(n)$ which stands for the answer to the particular query restricted to the subproblem below n [1].

DEFINITION **22 (node value)** *Given an optimization problem* $\mathcal{P} = \langle \mathcal{R}, min \rangle$ *over a graphical model* $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum \rangle$*, the* value *of a node* n *in the AND/OR search tree* $S_{\mathcal{T}}(\mathcal{R})$ *is the optimal cost to the subproblem below* n.

The value of a node can be computed recursively, as follows: it is 0 *for terminal AND nodes and* ∞ *for terminal OR nodes, respectively. The value of an internal OR node is obtained by* combining (summing) *the value of each AND child node with the weight on its incoming arc and then* optimize (minimize) *over all AND children. The value of an internal AND node is the* combination (summation) *of values of its OR children. Formally, if* $succ(n)$ *denotes the children of the node* n *in the AND/OR search tree, then:*

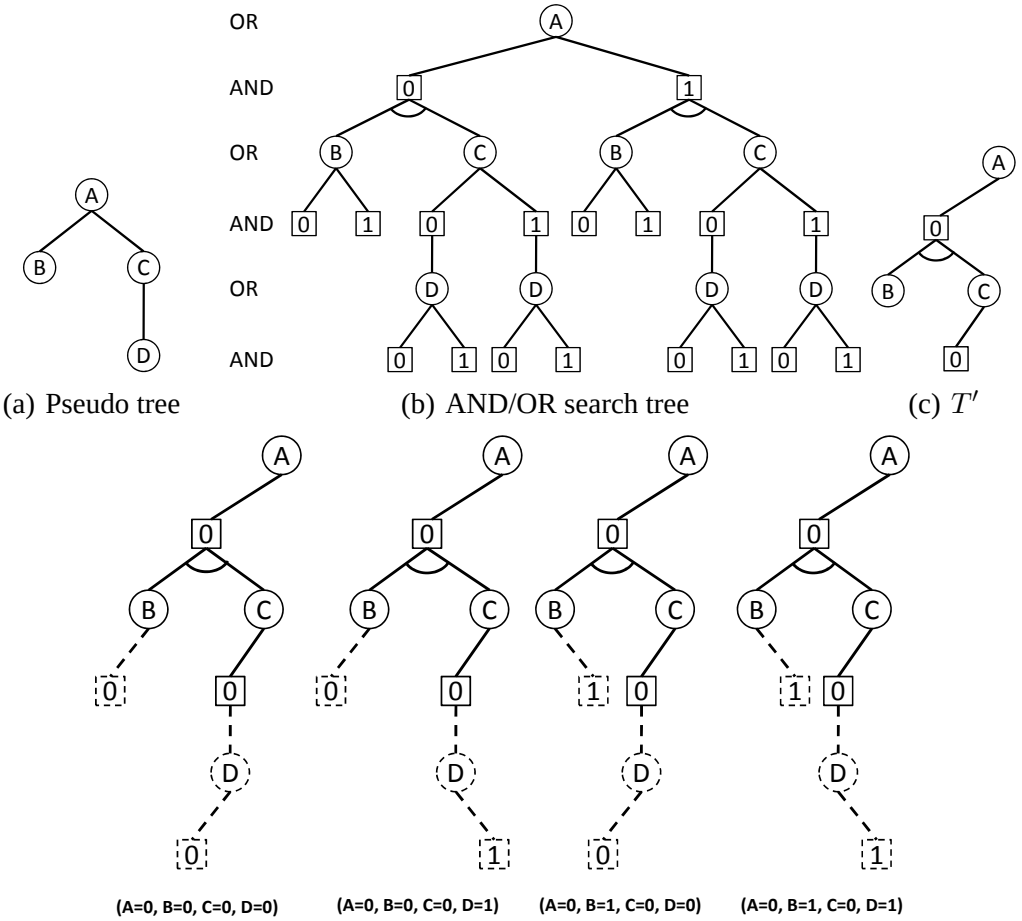
$$v(n) = \begin{cases} 0 & \text{, if } n = \langle X, x \rangle \text{ is a terminal AND node} \\ \infty & \text{, if } n = X \text{ is a terminal OR node} \\ \sum_{m \in succ(n)} v(m) & \text{, if } n = \langle X, x \rangle \text{ is an AND node} \\ \min_{m \in succ(n)} (w(n,m) + v(m)) & \text{, if } n = X \text{ is an OR node} \end{cases} \tag{1}$$

If n is the root of $S_{\mathcal{T}}(\mathcal{R})$, then $v(n)$ is the minimal cost solution to the initial problem. Alternatively, the value $v(n)$ can also be interpreted as the minimum of the costs of the solution trees rooted at n. Therefore, search algorithms that traverse the AND/OR search space can compute the value of the root node yielding the answer to the problem. It can be immediately inferred from Theorems 3 and 4 that:

THEOREM **5 (complexity [1])** *A depth-first search algorithm traversing an AND/OR search tree for finding the minimal cost solution is time* $O(n \cdot k^m)$*, where* k *bounds the domain size and* m *is the depth of the pseudo tree, and may use linear space. If the primal graph has a tree decomposition with treewidth* w^**, there there exists a pseudo tree* $\mathcal{T}$ *for which the time complexity is* $O(n \cdot k^{w^* \cdot log n})$.

6 AND/OR Branch-and-Bound Search

This section introduces the main contribution of the paper which is an AND/OR Branch-and-Bound algorithm for AND/OR search spaces for graphical models. Traversing AND/OR search spaces by best-first algorithms or depth-first Branch-



(a) Pseudo tree (b) AND/OR search tree (c) T'

(d) Solution trees represented by T': $extension(T')$

Fig. 6. A partial solution tree and possible extensions to solution trees.

and-Bound was described as early as [8,46,47]. Here we adapt these algorithms to graphical models. We will revisit next the notion of partial solution trees [8] to represent sets of solution trees which will be used in our description.

DEFINITION **23 (partial solution tree)** *A* partial solution tree T' *of an AND/OR search tree $S_{\mathcal{T}}$ is a subtree which: (1) contains the root node s of $S_{\mathcal{T}}$; (2) if n in T' is an OR node then it contains at most one of its AND child nodes in $S_{\mathcal{T}}$, and if n is an AND node then it contains all its OR children in $S_{\mathcal{T}}$ or it has no child nodes. A node in T' is called a* tip *node if it has no children in T'. A tip node is either a* terminal *node (if it has no children in $S_{\mathcal{T}}$), or a* non-terminal *node (if it has children in $S_{\mathcal{T}}$).*

A partial solution tree can be extended (possibly in several ways) to a full solution tree. It represents $extension(T')$, the set of all full solution trees which can extend it. Clearly, a partial solution tree all of whose tip nodes are terminal in $S_{\mathcal{T}}$ is a solution tree.

Example 6 *Figure 6(c) shows a partial solution tree T' of the AND/OR search*

Algorithm 1: AO: Depth-first AND/OR tree search

Input: An optimization problem $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum, min \rangle$, pseudo-tree $\mathcal{T}$ rooted at X_1.
Output: Minimal cost solution to $\mathcal{P}$ and an optimal solution tree.

```
1  v(s) ← ∞; ST(s) ← ∅; OPEN ← {s}                                  // Initialize the root node
2  while OPEN ≠ ∅ do
3      n ← top(OPEN); remove n from OPEN                              // EXPAND
4      succ(n) ← ∅
5      if n is an OR node, labeled X_i then
6          foreach x_i ∈ D_i do
7              create an AND node n' labeled by ⟨X_i, x_i⟩
8              v(n') ← 0; ST(n') ← ∅
9              w(n, n') ← Σ_{f ∈ B_T(X_i)} f(asgn(π_n))          // Compute the OR-to-AND arc weight
10             succ(n) ← succ(n) ∪ {n'}
11     else if n is an AND node, labeled ⟨X_i, x_i⟩ then
12         foreach X_j ∈ children_T(X_i) do
13             create an OR node n' labeled by X_j
14             v(n') ← ∞; ST(n') ← ∅
15             succ(n) ← succ(n) ∪ {n'}
16     Add succ(n) on top of OPEN
                                                                      // PROPAGATE
17     while succ(n) == ∅ do
18         let p be the parent of n
19         if n is an OR node, labeled X_i then
20             if X_i == X_1 then
21                 return (v(n), ST(n))                              // Search terminates
22             v(p) ← v(p) + v(n)                                    // Update AND value
23             ST(p) ← ST(p) ∪ ST(n)                  // Update solution tree below AND node
24         else if n is an AND node, labeled ⟨X_i, x_i⟩ then
25             if v(p) > (w(p, n) + v(n)) then
26                 v(p) ← w(p, n) + v(n)                             // Update OR value
27                 ST(p) ← ST(n) ∪ {(X_i, x_i)}          // Update solution tree below OR node
28         remove n from succ(p)
29         n ← p
```

tree of Figure 6(b) relative to the pseudo tree displayed in Figure 6(a). The set of solution trees represented by T' is given in Figure 6(d) and corresponds to the following assignments: $(A = 0, B = 0, C = 0, D = 0)$, $(A = 0, B = 0, C = 0, D = 1)$, $(A = 0, B = 1, C = 0, D = 0)$ *and* $(A = 0, B = 1, C = 0, D = 1)$.

Brute-force Depth-First AND/OR Tree Search. A simple depth-first search algorithm, called AO, that traverses the AND/OR search tree is described in Algorithm 1. The algorithm maintains the current partial solution being explored and will compute the value of each node (see Definition 22) in a depth-first manner. The value of the root node is the optimal cost. The algorithm also returns the optimal solution tree. It interleaves a forward expansion of the current partial solution tree (EXPAND) with a cost revision step (PROPAGATE) that updates the node values. The search stack is maintained by the OPEN list, n denotes the current node and p its parent in the search tree. Each node n in the search tree maintains its current value $v(n)$, which is updated based on the values of its children. For OR nodes, the current $v(n)$ is an upper bound on the optimal solution cost below n. Initially, $v(n)$ is set to ∞ if n is OR, and 0 if n is AND, respectively. A data structure $ST(n)$ maintains the actual best solution found in the subtree of n.

EXPAND selects a tip node n of the current partial solution tree and expands it by generating its successors. If n is an OR node, labeled X_i, then its successors are AND nodes represented by the values x_i in variable X_i's domain (lines 5–10). Each OR-to-AND arc is associated with the appropriate weight (see Definition 20). Similarly, if n is an AND node, labeled $\langle X_i, x_i \rangle$, then its successors are OR nodes labeled by the child variables of X_i in $\mathcal{T}$ (lines 11–15). There are no weights associated with AND-to-OR arcs.

PROPAGATE propagates node values bottom up in the search tree. Is is triggered when a node has an empty set of descendants (note that as each successor is evaluated, it is removed from the set of successors in line 28). This means that all its children have been evaluated, and their final values are already determined. If the current node is the root, then the search terminates with its value and an optimal solution tree (line 21). If n is an OR node, then its parent p is an AND node, and p updates its current value $v(p)$ by summation with the value of n (line 22). An AND node n propagates its value to its parent p in a similar way, by minimization (lines 25–27). Finally, the current node n is set to its parent p (line 29), because n was completely evaluated. Each node in the search tree also records the current best assignment to the variables of the subproblem below it and when the algorithm terminates it contains an optimal solution tree. Specifically, if n is an AND node, then $ST(n)$ is the union of the optimal solution trees propagated from n's OR children (line 23). If n is an OR node and n' is its AND child such that $n' = argmin_{m \in succ(n)}(w(n,m) + v(m))$, then $ST(n)$ is obtained from the label of n' combined with the optimal solution tree below n' (line 27). Search continues either with a *propagation* step (if conditions are met) or with an *expansion* step.

Heuristic Lower Bounds on Partial Solution Trees. A regular OR Branch-and-Bound algorithm traverses the space of partial assignments in a depth-first manner and discards any partial assignment that cannot lead to a superior solution than the current best one found so far. This is normally achieved by using an evaluation function that underestimates (for minimization tasks) the best possible extension of the current partial path. Thus, when the estimated lower bound, called also heuristic evaluation function, is higher than the best current solution (upper bound), search terminates below this path.

We will now extend the brute-force AO algorithm into a Branch-and-Bound scheme, guided by a lower bound heuristic evaluation function. For that, we first define the exact evaluation function of a partial solution tree, and will then derive the notion of a lower bound for it. Like in OR search, we assume a given heuristic evaluation function $h(n)$ associated with each node n in the AND/OR search tree such that $h(n) \leq h^*(n)$, where $h^*(n)$ is the best cost extension of the subproblem below n (namely, $h^*(n) = v(n)$). We call $h(n)$ a *node-based heuristic function*.

DEFINITION **24 (exact evaluation function of a partial solution tree)** *The* exact evaluation function $f^*(T')$ *of a partial solution tree* T' *is the minimum of the costs*

of all solution trees represented by T', namely:

$$f^*(T') = min\{f(T) \mid T \in extension(T')\}$$

We define $f^*(T'_n)$ the exact evaluation function of a partial solution tree rooted at node n. Then $f^*(T'_n)$ can be computed recursively, as follows:

1. *If T'_n consists of a single node n, then $f^*(T'_n) = v(n)$.*
2. *If n is an OR node having the AND child m in T'_n, then $f^*(T'_n) = w(n,m) + f^*(T'_m)$, where T'_m is the partial solution subtree of T'_n that is rooted at m.*
3. *If n is an AND node having OR children $m_1, ..., m_k$ in T'_n, then $f^*(T'_n) = \sum_{i=1}^{k} f^*(T'_{m_i})$, where T'_{m_i} is the partial solution subtree of T'_n rooted at m_i.*

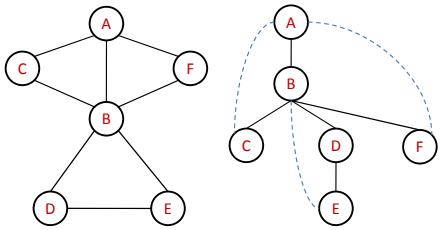
Clearly, we are interested to find the $f^*(T')$ of a partial solution tree T' rooted at the root s. If each non-terminal tip node n of T' is assigned a heuristic lower bound estimate $h(n)$ of $v(n)$, then it induces a heuristic evaluation function on the minimal cost extension of T', as follows.

DEFINITION **25 (heuristic evaluation function of a partial solution tree)** *Given a node-based heuristic function $h(m)$ which is a lower bound on the optimal cost below any node m, namely $h(m) \leq v(m)$, and given a partial solution tree T'_n rooted at node n in the AND/OR search tree $S_{\mathcal{T}}$, the* tree-based heuristic evaluation function *$f(T'_n)$ of T'_n, is defined recursively by:*

1. *If T'_n consists of a single node n then $f(T'_n) = h(n)$.*
2. *If n is an OR node having the AND child m in T'_n, then $f(T'_n) = w(n,m) + f(T'_m)$, where T'_m is the partial solution subtree of T'_n that is rooted at m.*
3. *If n is an AND node having OR children $m_1, ..., m_k$ in T'_n, then $f(T'_n) = \sum_{i=1}^{k} f(T'_{m_i})$, where T'_{m_i} is the partial solution subtree of T'_n rooted at m_i.*

PROPOSITION **1** *Clearly, by definition $f(T'_n) \leq f^*(T'_n)$. If n is the root of the AND/OR search tree, then $f(T') \leq f^*(T')$.*

Example 7 *Consider the cost network with bi-valued variables A, B, C, D, E and F in Figure 7(a). The cost functions $f_1(A,B,C)$, $f_2(A,B,F)$ and $f_3(B,D,E)$ are given in Figure 7(b). A partially explored AND/OR search tree relative to the pseudo tree from Figure 7(a) is displayed in Figure 7(c). The current partial solution tree T' is highlighted. It contains the nodes: A, $\langle A,0\rangle$, B, $\langle B,1\rangle$, C, $\langle C,0\rangle$, D, $\langle D,0\rangle$ and F. The nodes labeled by $\langle D,0\rangle$ and by F are non-terminal tip nodes and their corresponding heuristic estimates are $h(\langle D,0\rangle) = 4$ and $h(F) = 5$, respectively. The node labeled by $\langle C,0\rangle$ is a terminal tip node of T'. The subtree rooted at $\langle B,0\rangle$ along the path $(A, \langle A,0\rangle, B, \langle B,0\rangle)$ is fully explored, yielding the current best solution cost found so far equal to 9. We assume that the search is currently at the tip node labeled by $\langle D,0\rangle$ of T'. The heuristic evaluation function of T' is computed recursively as follows:*

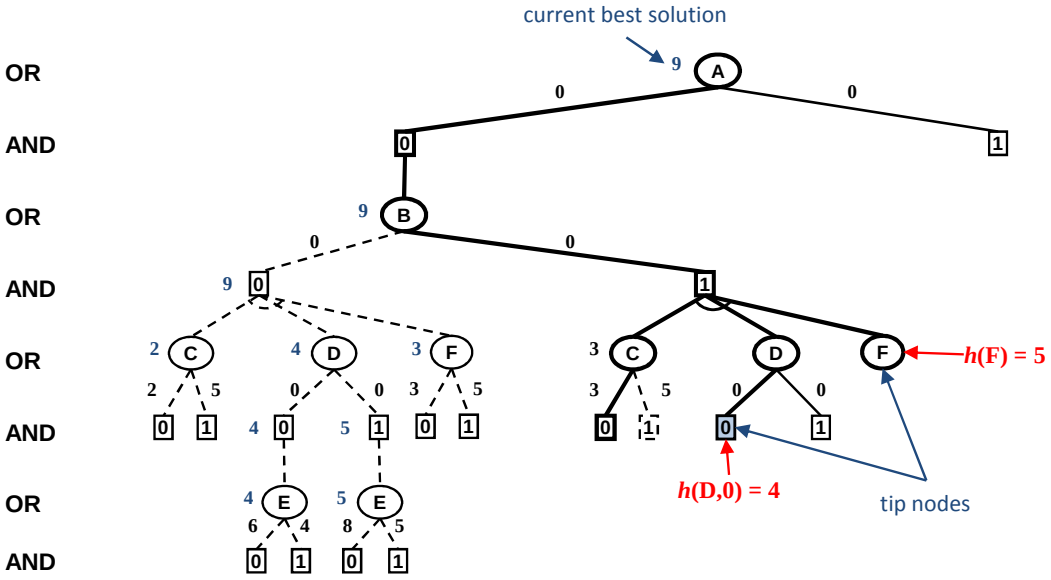


A	B	C	f_1(ABC)
0	0	0	2
0	0	1	5
0	1	0	3
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	2

A	B	F	f_2(ABF)
0	0	0	3
0	0	1	5
0	1	0	1
0	1	1	4
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	6
0	0	1	4
0	1	0	8
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	4

(a) Cost network and pseudo tree (b) Cost functions



(c) Partial solution tree

Fig. 7. Cost of a partial solution tree.

$$
\begin{aligned}
f(T') &= w(A,0) + f(T'_{\langle A,0\rangle}) \\
&= w(A,0) + f(T'_B) \\
&= w(A,0) + w(B,1) + f(T'_{\langle B,1\rangle}) \\
&= w(A,0) + w(B,1) + f(T'_C) + f(T'_D) + f(T'_F) \\
&= w(A,0) + w(B,1) + w(C,0) + f(T'_{\langle C,0\rangle}) + w(D,0) + f(T'_{\langle D,0\rangle}) + h(F) \\
&= w(A,0) + w(B,1) + w(C,0) + 0 + w(D,0) + h(\langle D,0\rangle) + h(F) \\
&= 0 + 0 + 3 + 0 + 0 + 4 + 5 \\
&= 12
\end{aligned}
$$

Notice that if the pseudo tree $\mathcal{T}$ is a chain, then a partial tree T' is also a chain and corresponds to the partial assignment $\bar{x}^p = (x_1, ..., x_p)$. In this case, $f(T')$ is equivalent to the classical definition of the heuristic evaluation function of $\bar{x}^p$. Namely, $f(T')$ is the sum of the cost of the partial solution $\bar{x}^p$, $g(\bar{x}^p)$, and the heuristic estimate of the optimal cost extension of $\bar{x}^p$ to a complete solution.

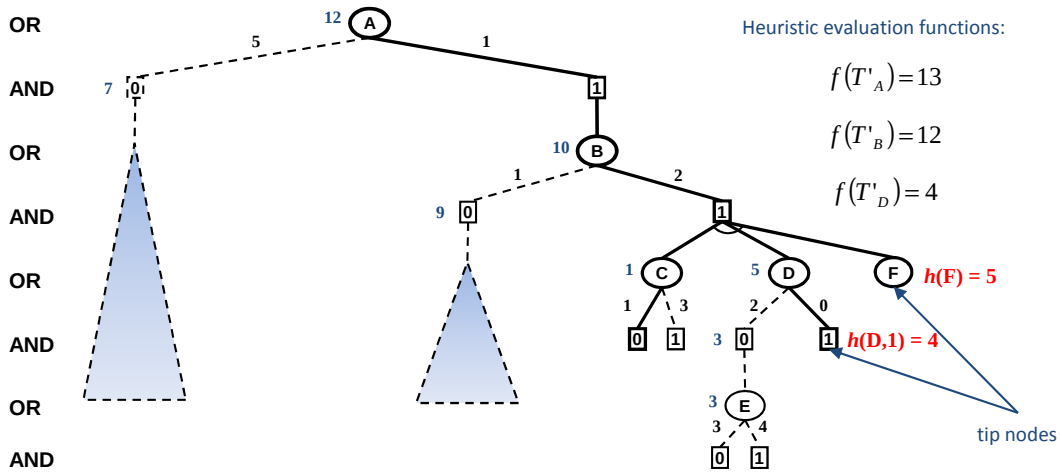


Fig. 8. Illustration of the pruning mechanism.

During search we maintain an upper bound $ub(s)$ on the optimal solution $v(s)$ as well as the heuristic evaluation function of the current partial solution tree $f(T')$, and we can prune the search space by comparing these two measures, as is common in Branch-and-Bound search. Namely, if $f(T') \geq ub(s)$, then searching below the current tip node t of T' is guaranteed not to reduce $ub(s)$ and therefore, the search space below t can be pruned.

Example 8 *For illustration, consider again the partially explored AND/OR search tree from Example 7 (see Figure 7(c)). In this case, the current best solution found after exploring the subtree below $\langle B, 0\rangle$, which ends the path $(A, \langle A, 0\rangle, B, \langle B, 0\rangle)$, is 9. Since we computed $f(T') = 12$ for the current partial solution tree highlighted in Figure 7(c), then exploring the subtree rooted at $\langle D, 0\rangle$, which is the current tip node, cannot yield a better solution and search can be pruned.*

Up until now we considered the case when the best solution found so far is maintained at the root node of the search tree. It is also possible to maintain the current best solutions for all the OR nodes along the active path between the tip node t of T' and s. Then, if $f(T'_m) \geq ub(m)$, where m is an OR ancestor of t in T' and T'_m is the subtree of T' rooted at m, it is also safe to prune the search tree below t. This provides an efficient mechanism to discover that the search space below a node can be pruned more quickly.

Example 9 *Consider the partially explored weighted AND/OR search tree in Figure 8, relative to the pseudo tree from Figure 7(a). The current partial solution tree T' is highlighted. It contains the nodes: A, $\langle A, 1\rangle$, B, $\langle B, 1\rangle$, C, $\langle C, 0\rangle$, D, $\langle D, 1\rangle$ and F. The nodes labeled by $\langle D, 1\rangle$ and by F are non-terminal tip nodes and their corresponding heuristic estimates are $h(\langle D, 1\rangle) = 4$ and $h(F) = 5$, respectively. The subtrees rooted at the AND nodes labeled $\langle A, 0\rangle$, $\langle B, 0\rangle$ and $\langle D, 0\rangle$ are fully evaluated, and therefore the current upper bounds of the OR nodes labeled A, B and D, along the active path, are $ub(A) = 12$, $ub(B) = 10$ and $ub(D) = 5$, respectively. Moreover, the heuristic evaluation functions of the partial solution*

Algorithm 2: AOBB: Depth-first AND/OR Branch-and-Bound search

Input: An optimization problem $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \sum, min \rangle$, pseudo-tree $\mathcal{T}$ rooted at X_1, heuristic function $h(n)$.
Output: Minimal cost solution to $\mathcal{P}$ and an optimal solution tree.

```
1  v(s) ← ∞; ST(s) ← ∅; OPEN ← {s}                                   // Initialize the root node
2  while OPEN ≠ ∅ do
3      n ← top(OPEN); remove n from OPEN                              // EXPAND
4      succ(n) ← ∅
5      if n is an OR node, labeled X_i then
6          foreach x_i ∈ D_i do
7              create an AND node n' labeled by ⟨X_i, x_i⟩
8              v(n') ← 0; ST(n') ← ∅
9              w(n, n') ← Σ_{f∈B_T(X_i)} f(asgn(π_n))              // Compute the OR-to-AND arc weight
10             succ(n) ← succ(n) ∪ {n'}
11     else if n is an AND node, labeled ⟨X_i, x_i⟩ then
12         deadend ← false
13         foreach OR ancestor m of n do
14             f(T'_m) ← evalPartialSolutionTree(T'_m)
15             if f(T'_m) ≥ v(m) then
16                 deadend ← true          // Pruning the subtree below the current tip node
17                 break
18         if deadend == false then
19             foreach X_j ∈ children_T(X_i) do
20                 create an OR node n' labeled by X_j
21                 v(n') ← ∞; ST(n') ← ∅
22                 succ(n) ← succ(n) ∪ {n'}
23         else
24             p ← parent(n)
25             succ(p) ← succ(p) − {n}
26     Add succ(n) on top of OPEN
                                                                      // PROPAGATE
27     while succ(n) == ∅ do
28         let p be the parent of n
29         if n is an OR node, labeled X_i then
30             if X_i == X_1 then
31                 return (v(n), ST(n))                              // Search terminates
32             v(p) ← v(p) + v(n)                                    // Update AND value
33             ST(p) ← ST(p) ∪ ST(n)                  // Update solution tree below AND node
34         if n is an AND node, labeled ⟨X_i, x_i⟩ then
35             if v(p) > (w(p, n) + v(n)) then
36                 v(p) ← w(p, n) + v(n)                             // Update OR value
37                 ST(p) ← ST(n) ∪ {(X_i, x_i)}          // Update solution tree below OR node
38         remove n from succ(p)
39         n ← p
```

subtrees rooted at the OR nodes along the current path can be computed recursively based on Definition 25, namely $f(T'_A) = 13$, $f(T'_B) = 12$ *and* $f(T'_D) = 4$, *respectively. Notice that while we could prune the subtree below* $\langle D, 1 \rangle$ *because* $f(T'_A) > ub(A)$, *we could discover this pruning earlier by looking at node* B *only, because* $f(T'_B) > ub(B)$. *Therefore, the partial solution tree* T'_A *need not be consulted in this case.*

Depth-First AND/OR Branch-and-Bound Tree Search. The *AND/OR Branch-and-Bound* algorithm, AOBB, for searching AND/OR trees for graphical models, is described by Algorithm 2. Like AO, it interleaves a forward expansion of the cur-

Algorithm 3: Recursive computation of the heuristic evaluation function.

```
function: evalPartialSolutionTree(T'_n)
Input: Partial solution subtree T'_n rooted at node n.
Output: Return heuristic evaluation function f(T'_n).
1  if succ(n) == ∅ then
2      if n is an AND node then
3          return 0
4      else
5          return h(n)
6  else
7      if n is an AND node then
8          let m_1, ..., m_k be the OR children of n
9          return Σ_{i=1}^{k} evalPartialSolutionTree(T'_{m_i})
10     else if n is an OR node then
11         let m be the AND child of m
12         return w(n, m) + evalPartialSolutionTree(T'_m)
```

rent partial solution tree with a backward propagation step that updates the nodes upper-bounds of values. The fringe of the search is maintained by a stack called OPEN, the current node is n, its parent p, and the current path π_n. As before, $ST(n)$ accumulates the current best solution tree below n. The node-based heuristic function $h(n)$ of $v(n)$ is assumed to be available to the algorithm, either retrieved from a cache or computed during search.

Before expanding the current AND node n, labeled $\langle X_i, x_i \rangle$, the algorithm computes the heuristic evaluation function for every partial solution subtree rooted at the OR ancestors of n along the path from the root (lines 11–17). The search below n is terminated if, for some OR ancestor m, $f(T'_m) \geq v(m)$, where $v(m)$ is the current best upper bound on the optimal cost below m. The recursive computation of $f(T'_m)$ based on Definition 25 is described in Algorithm 3. Notice also that for any OR node n, labeled X_i in the search tree, $v(n)$ is trivially initialized to ∞ and is updated in line 36.

The node values are updated by the propagation step, in the usual way (lines 24–40): OR nodes by minimization, while AND nodes by summation. The search terminates when the root node is evaluated in line 32.

THEOREM **6** *The time complexity of the depth-first AND/OR Branch-and-Bound algorithm (AOBB) is $O(n \cdot k^m)$, where m is the depth of the pseudo tree, k bounds the domain size and n is the number of variables, and it can use linear space.*

Proof. The time complexity follows immediately from the size of the AND/OR search tree explored (see Theorem 3). Since only the current partial solution tree needs to be stored in memory, the algorithm can operate in linear space. □

7 Lower Bound Heuristics for AND/OR Search

The effectiveness of any Branch-and-Bound search strategy greatly depends on the quality of the heuristic evaluation function. Naturally, more accurate heuristic estimates may yield a smaller search space, possibly at a much higher computational cost for computing the lower bound heuristic function. The right tradeoff between the computational overhead and the pruning power exhibited during search may be hard to predict. One of the primary heuristics we used is the Mini-Bucket heuristic introduced in [5] for OR search spaces. In the following subsections we discuss its extension to AND/OR search spaces. We also extend the local consistency based lower bound developed in [22–24] to AND/OR search spaces. Both of these heuristic functions were used in our experiments.

7.1 Static Mini-Bucket Heuristics

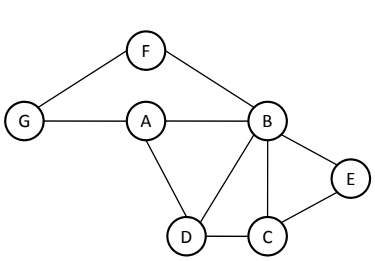
Consider the cost network and pseudo tree shown in Figures 9(a) and 9(b), respectively, and consider also the variable ordering $d = (A, B, C, D, E, F, G)$ and the bucket and mini-buckets configuration in the output as displayed in Figures 9(c) and 9(d), respectively (see Sections 4.1 and 4.2 for more details). For clarity, we display the execution of the bucket and mini-bucket elimination along the bucket tree corresponding to the given elimination ordering. The bucket tree is also a pseudo tree [1]. The functions denoted on the arcs are those messages sent from a bucket node to its parent in the tree.

Let us assume, without loss of generality, that variables A and B have been instantiated during search. Let $h^*(a, b, c)$ be the minimal cost solution of the subproblem rooted at node C in the pseudo tree, conditioned on $(A = a, B = b, C = c)$. In the AND/OR search tree, this is represented by the subproblem rooted at the AND node labeled $\langle C, c\rangle$, ending the path $\{A, \langle A, a\rangle, B, \langle B, b\rangle, C, \langle C, c\rangle\}$. By definition,

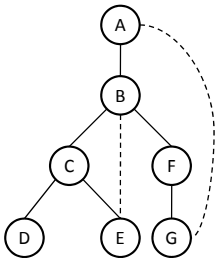
$$h^*(a, b, c) = min_{d,e}(f(c, e) + f(b, e) + f(a, d) + f(c, d) + f(b, d)) \qquad (2)$$

Notice that we restrict ourselves to the subproblem over variables D and E only. Therefore, we obtain:

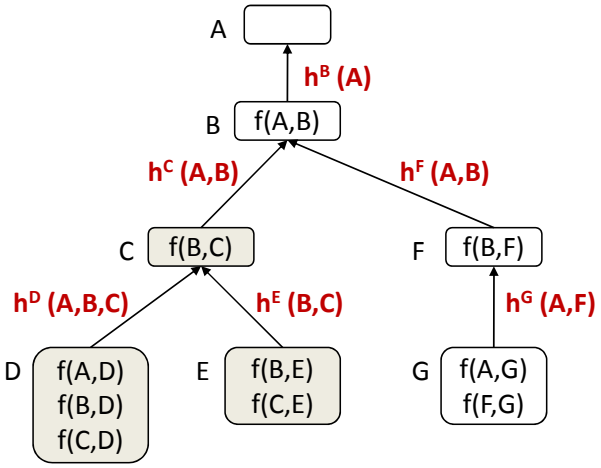
$$\begin{aligned} h^*(a, b, c) &= min_d(f(a, d) + f(c, d) + f(b, d) + min_e(f(c, e) + f(b, e))) \\ &= min_d(f(a, d) + f(c, d) + f(b, d)) + min_e(f(c, e) + f(b, e)) \\ &= h^D(a, b, c) + h^E(b, c) \end{aligned}$$



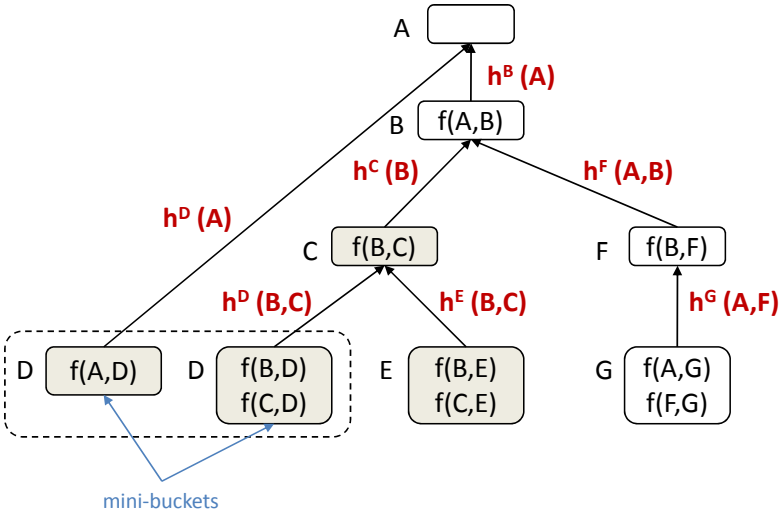
(a) Cost network



(b) Pseudo tree



(c) Bucket Elimination



(d) Mini-Bucket Elimination MBE(3)

Fig. 9. Static mini-bucket heuristics for $i = 3$.

where,

$$h^D(a,b,c) = min_d(f(a,d) + f(c,d) + f(b,d))$$
$$h^E(b,c) = min_e(f(c,e) + f(b,e))$$

Notice that the functions $h^D(a,b,c)$ and $h^E(b,c)$ are produced by the bucket elimination algorithm shown in Figure 9(c). Specifically, the function $h^D(a,b,c)$, generated in bucket of D by bucket elimination, is the result of a minimization operation over variable D. In practice, however, this function may be too hard to compute as it requires processing a function on four variables. It can be replaced by a partition-based approximation ($e.g.$, the minimization is split into two parts). This yields a lower bound approximation, denoted by $h(a,b,c)$, namely:

$$\begin{aligned} h^*(a,b,c) &= min_d(f(a,d)+f(c,d)+f(b,d))+h^E(b,c) \\ &\geq min_d f(a,d)+min_d(f(c,d)+f(b,d))+h^E(b,c) \\ &= h^D(a)+h^D(b,c)+h^E(b,c) \\ &\triangleq h(a,b,c) \end{aligned}$$

where,

$$\begin{aligned} h^D(a) &= min_d f(a,d) \\ h^D(c,b) &= min_d(f(c,d)+f(b,d)) \end{aligned}$$

The functions $h^D(a)$ and $h^D(b,c)$ are the ones computed by the Mini-Bucket algorithm MBE(3), shown in Figure 9(d). Therefore, the function $h(a,b,c)$ can be constructed during search from the pre-compiled mini-buckets, yielding a lower bound on the minimal cost of the respective subproblem.

For OR nodes, such as n, labeled by C, ending the path $\{A,\langle A,a\rangle,B,\langle B,b\rangle,C\}$, $h(n)$ can be obtained by minimizing over the values $c \in D_C$ the sum between the weight $w(n,m)$ and the heuristic estimate $h(m)$ below the AND child m of n. Namely, $h(n)=min_m(w(n,m)+h(m))$.

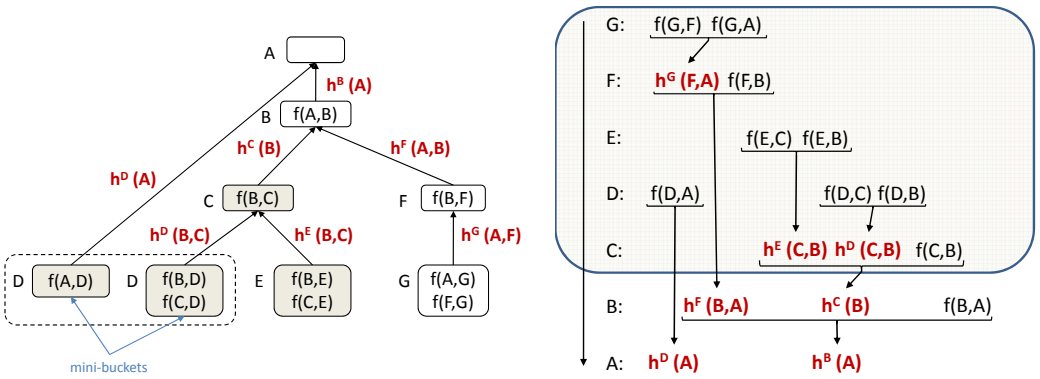
In summary, similarly to [5], we can show that the mini-bucket heuristic associated with any node in the AND/OR search tree can be obtained from the the pre-compiled mini-bucket functions.

DEFINITION **26 (static mini-bucket heuristic)** *Given an ordered set of augmented buckets $\{B(X_1),...,B(X_n)\}$ generated by the Mini-Bucket algorithm MBE(i) along the bucket tree $\mathcal{T}$, and given a node n in the AND/OR search tree, the* static mini-bucket heuristic *function $h(n)$ is computed as follows:*

(1) If n is an AND node, labeled by $\langle X_p,x_p\rangle$, then:

$$h(n)=\sum_{h_j^k\in\{B(X_p)\cup B(X_p^1..X_p^q)\}} h_j^k$$

Namely, it is the sum of the intermediate functions h_j^k that satisfy the following



(a) AND/OR static mini-buckets (b) OR static mini-buckets

Fig. 10. AND/OR versus OR static mini-bucket heuristics for $i = 3$.

two properties:

- *They are generated in buckets $B(X_k)$, where X_k is any descendant of X_p in the bucket tree $\mathcal{T}$,*
- *They reside in bucket $B(X_p)$ or the buckets $B(X_p^1..X_p^q) = \{B(X_p^1), ..., B(X_p^q)\}$ that correspond to the ancestors $\{X_p^1, ..., X_p^q\}$ of X_p in $\mathcal{T}$.*

(2) If n is an OR node, labeled by X_p, then:

$$h(n) = min_m(w(n,m) + h(m))$$

where m is the AND child of n labeled with value x_p of X_p.

Example 10 *Figure 9(d) shows the bucket tree for the cost network in Figure 9(a) together with the intermediate functions generated by MBE(3) along the ordering $d = (A, B, C, D, E, F, G)$. The static mini-bucket function $h(a', b', c')$ associated with the AND node labeled $\langle C, c' \rangle$ ending the path $(A = a', B = b', C = c')$ in the AND/OR search tree is by definition $h(a', b', c') = h^D(a') + h^D(c', b') + h^E(b', c')$. The intermediate functions $h^D(c', b')$ and $h^E(b', c')$ are generated in buckets D and E, respectively, and reside in bucket C. The function $h^D(a')$ is also generated in bucket D, but it resides in bucket A, which is an ancestor of C in the bucket tree.*

We see that the computation of the static mini-bucket heuristic of a node n in the AND/OR search tree is identical to the OR case (see Definition 15), except that it only considers the intermediate functions generated by the buckets corresponding to the current conditioned subproblem rooted at n.

Example 11 *For example, consider again the cost network in Figure 9(a). Figures 10(a) (which repeats Figure 9(d)) and 10(b) show the compiled bucket structure obtained by MBE(3) along the given elimination order $d = (A, B, C, D, E, F, G)$, for the AND/OR and OR spaces, respectively. The static mini-bucket heuristic function underestimating the minimal cost extension of the partial assignment $(A = a', B = b', C = c')$ in the OR search space is $h(a', b', c') = h^D(a') + h^D(c', b') + h^E(b', c') + h^F(b', a')$. Namely, it involves the extra function $h^F(b', a')$ which was generated in bucket F and resides in bucket B, as shown in Figure 10(b). This is because, in the*

OR space, variables F and G are part of the subproblem rooted at C, unlike the AND/OR search space.

7.2 Dynamic Mini-Bucket Heuristics

It is also possible to generate the mini-bucket heuristic information dynamically during search, as we show next. The idea is to compute MBE(i) conditioned on the current partial assignment.

DEFINITION **27 (dynamic mini-bucket heuristics)** *Given a bucket tree $\mathcal{T}$ with buckets $\{B(X_1), ..., B(X_n)\}$, a node n in the AND/OR search tree and given the current partial assignment $asgn(\pi_n)$ along the path to n, the* dynamic mini-bucket heuristic *function $h(n)$ is computed as follows:*

(1) If n is an AND node labeled by $\langle X_p, x_p \rangle$, then:

$$h(n) = \sum_{h_j^k \in B(X_p)} h_j^k$$

Namely, it is the sum of the intermediate functions h_j^k that reside in bucket $B(X_p)$ and were generated by MBE(i), conditioned on $asgn(\pi_n)$, in buckets $B(X_p^1)$ through $B(X_p^q)$, where $\{X_p^1, ..., X_p^q\}$ are the descendants of X_p in $\mathcal{T}$.

(2) If n is an OR node labeled by X_p, then:

$$h(n) = min_m(w(n,m) + h(m))$$

where m is the AND child of n labeled with value x_p of X_p.

Given an i-bound, the dynamic mini-bucket heuristic implies a much higher computational effort compared with the static version. However, the bounds generated dynamically may be far more accurate since some of the variables are assigned and will therefore yield smaller functions and less partitioning. More importantly, the dynamic mini-bucket heuristic can be used with dynamic variable ordering heuristics, unlike the pre-compiled one, which restricts search to be conducted in an order that respects a static pseudo tree structure.

Example 12 *Figure 11 shows the bucket tree structure corresponding to the binary cost network instance displayed in Figure 9(a), along the elimination ordering (A, B, C, D, E, F, G). The dynamic mini-bucket heuristic estimate $h(a', b', c')$ of the AND node labeled $\langle C, c' \rangle$ ending the path $\{A, \langle A, a' \rangle, B, \langle B, b' \rangle, C, \langle C, c' \rangle\}$ is computed by MBE(3) on the subproblem represented by the buckets D and E, conditioned on the partial assignment $(A = a', B = b', C = c')$. Namely, MBE(3) processes buckets D and E by eliminating the respective variables, and generates two new functions: $h^D(c')$ and $h^E(c')$, as illustrated in Figure 11. These new functions are in fact constants since variables A, B and C are assigned in the scopes of*

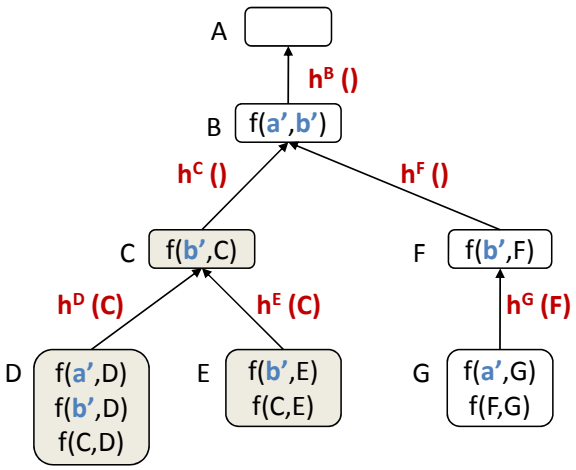


Fig. 11. Dynamic mini-bucket heuristics for $i = 3$.

the input functions that constitute the conditioned subproblem: $f(a', D)$, $f(b', D)$, $f(c', D)$, $f(b', E)$ *and* $f(c', E)$, *respectively. Therefore* $h(a', b', c') = h^D(c') + h^E(c')$ *and it equals the exact* $h^*(a', b', c')$ *in this case.*

7.3 Local Consistency Based Heuristics for AND/OR Search

Another class of heuristic lower bounds developed for guiding Branch-and-Bound search for solving binary Weighted CSPs is based on exploiting local consistency algorithms for cost functions. In the next section we overview the basic principles behind these types of heuristics and discuss their extension to AND/OR trees.

7.3.1 Review of Local Consistency for Weighted CSPs

As in the classical CSP, enforcing soft local consistency on the initial problem provides in polynomial time an *equivalent* problem defining the same cost distribution on complete assignments, with possible smaller domains [22–24].

Assume a binary Weighted CSP $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, where $\mathbf{X} = \{X_1, ..., X_n\}$ and $\mathbf{D} = \{D_1, ..., D_n\}$ are the variables and their corresponding domains. $\mathbf{C}$ is the set of binary and unary cost functions (or soft constraints). A binary soft constraint $C_{ij}(X_i, X_j) \in \mathbf{C}$ (or C_{ij} in short) is $C_{ij}(X_i, X_j) : D_i \times D_j \rightarrow \mathbb{N}$. A unary soft constraint $C_i(X_i) \in \mathbf{C}$ (or C_i in short) is $C_i(X_i) : D_i \rightarrow \mathbb{N}$. We assume the existence of a unary constraint $C_i(X_i)$ for every variable, and a zero-arity constraint, denoted by $C_\emptyset$. If no such constraints are defined, we can always define dummy ones, as $C_i(x_i) = 0, \forall x_i \in D_i$ or $C_\emptyset = 0$. We denote by $\top$, the maximum allowed cost (*e.g.*, $\top = \infty$). The cost of a tuple $x = (x_1, ..., x_n)$, denoted by $cost(x)$, is defined by:

$$cost(x) = \sum_{C_{ij} \in \mathbf{C}} C_{ij}(x[i,j]) + \sum_{C_i \in \mathbf{C}} C_{X_i}(x[i]) + C_\emptyset$$

For completeness, we define next some local consistencies in WCSP, in particular *node, arc* and *directional arc consistency*, as in [22,23]. We assume that the set of variables $\mathbf{X}$ is totally ordered. We note that there are several stronger local consistencies which were defined in recent years, such as *full directional arc consistency* (FDAC) [22,23] or *existential directional arc consistency* (EDAC) [24].

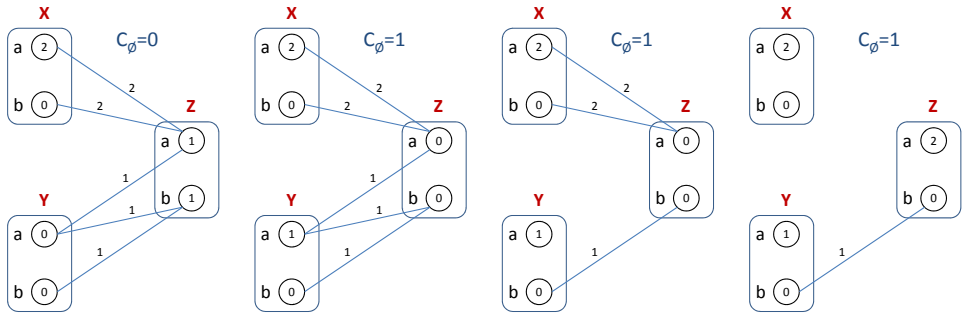
DEFINITION **28 (soft node consistency [22,23])** *Let $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$ be a binary WCSP. (X_i, x_i) is star node consistent (NC^*) if $C_\emptyset + C_i(x_i) < \top$. Variable X_i is NC^* if: i) all its values are NC^* and ii) there exists a value $x_i \in D_i$ such that $C_i(x_i) = 0$. Value x_i is a* support *for variable X_i. $\mathcal{R}$ is NC^* if every variable is NC^*.*

DEFINITION **29 (soft arc consistency [22,23])** *Let $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$ be a binary WCSP. (X_i, x_i) is arc consistent (AC) with respect to constraint $C_{ij}(X_i, X_j)$ if there exists a value $x_j \in D_j$ such that $C_{ij}(x_i, x_j) = 0$. Value x_j is called a* support *for the value x_i. Variable X_i is AC if all its values are AC wrt. every binary constraint affecting X_i. $\mathcal{R}$ is AC^* if every variable is AC and NC^*.*

DEFINITION **30 (soft directional arc consistency [22,23])** *Let $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$ be a binary WCSP. (X_i, x_i) is directional arc consistent (DAC) with respect to constraint $C_{ij}(X_i, X_j)$, $i < j$, if there exists a value $x_j \in D_j$ such that $C_{ij}(x_i, x_j) + C_j(x_j) = 0$. Value x_j is called a* full support *of x_j. Variable X_i is DAC is all its values are DAC wrt. every $C_{ij}(X_i, X_j)$, $i < j$. $\mathcal{R}$ is DAC^* if every variable is DAC and NC^*.*

For our purpose, we point out that enforcing such local consistencies is done by the repeated application of atomic operations called *arc equivalence preserving transformations* [48]. This process may increase the value of $C_\emptyset$ and the unary costs $C_i(x_i)$ associated with domain values. The zero-arity cost function $C_\emptyset$ defines a *strong lower bound* which can be exploited by Branch-and-Bound algorithms while the updated $C_i(x_i)$ can inform variable and value orderings [22–24].

If we consider two cost functions $C_{ij}(X_i, X_j)$, defined over variables X_i and X_j, and $C_i(X_i)$, defined over variable X_i, a value $x_i \in D_i$ and a cost α, we can add α to $C_i(x_i)$ and subtract α from every $C_{ij}(x_i, x_j)$ for all $x_j \in D_j$. Simple arithmetics shows that the global cost distribution is unchanged while costs may have moved from the binary to the unary level (if $\alpha > 0$, this is called a *projection*) or from the unary to the binary level (if $\alpha < 0$, this is called an *extension*). In these operations, any cost above $\top$, the maximum allowed cost, can be considered as infinite and is thus unaffected by subtraction. If no negative cost appears and if all costs above $\top$ are set to $\top$, the remaining problem is always a valid and equivalent WCSP. The same mechanism, at the unary level, can be used to move costs from the $C_i(X_i)$ to



(a) WCSP (b) NC^* (c) DAC^* (d) AC^*

Fig. 12. Four equivalent WCSPs (for $\top = 4$) [22].

$C_\emptyset$. Finally, any value a such that $C_i(x_i) + C_\emptyset$ is equal to $\top$ can be deleted. For a detailed description of these operations, we refer the reader to [22–24].

Example 13 *Figure 12(a) shows a WCSP with a sets of costs* $[0, ..., 4]$ *and with* $\top = 4$*. The network has three variables* $\mathbf{X} = \{X, Y, Z\}$*, each with values* $\{a, b\}$*. There are 2 binary constraints* $C(X, Z)$*,* $C(Y, Z)$ *and two non-trivial unary constraints* $C(X)$ *and* $C(Z)$*. Unary costs are depicted inside their domain value. Binary costs are depicted as labeled edges connecting the corresponding pair of values. Zero costs are not shown. Initially,* $C_\emptyset$ *is set to 0. One optimal solution is* $(X = b, Y = b, Z = b)$*, with cost 2.*

The problem in Figure 12(a) is not NC^* *since* Z *has no support. To enforce* NC^* *we must force a support for* Z *by projecting* $C_Z(Z)$ *onto* $C_\emptyset$*. The resulting problem in Figure 12(b) is* NC^* *but not* AC^**. To enforce* AC^**, it suffices to enforce a support for* (Y, a) *and* (Z, a)*, as follows: we project* $C_{YZ}(Y, Z)$ *over* (Y, a) *by adding 1 to* $C_Y(a)$ *and subtracting 1 from* $C_{YZ}(a, a)$ *and* $C_{YZ}(a, b)$*, and similarly project* $C_{XZ}(X, Z)$ *over* (Z, a)*. Consequently, we get problem 12(d) which is* AC^**. Observe also that problem 12(b) is not* DAC^* *for order* (X, Y, Z) *since* (Y, a) *has no full support on* Z*. Problem 12(c) is an equivalent* DAC^* *problem.*

7.3.2 Extension of Local Consistency to AND/OR Search Spaces

As mentioned earlier, the zero-arity constraint $C_\emptyset$ which is obtained by enforcing local consistency, can be used as a heuristic function to guide Branch-and-Bound search. The extension of this heuristic to AND/OR search spaces is fairly straightforward and is similar to the extension of the mini-bucket heuristics from OR to AND/OR spaces. Consider P_n, the subproblem rooted at the AND node n, labeled $\langle X_i, x_i \rangle$, in the AND/OR search tree defined by a pseudo tree $\mathcal{T}$. The heuristic function $h(n)$ underestimating $v(n)$ is the zero-arity cost function $C_\emptyset^n$ resulted from enforcing soft arc consistency over P_n only, subject to the current partial instantiation of the variables along the path from the root of the search tree. Note that P_n is defined by the variables and cost functions corresponding to the subtree rooted at X_i in $\mathcal{T}$. If n is an OR node labeled X_i then $h(n)$ is computed in the usual way,

namely $h(n) = min_m(w(n,m) + h(m))$, where m is the AND child of n, labeled with value x_i of X_i. Notice that in this case the weights associated with the OR-to-AND arcs are computed now relative to the equivalent subproblem resulted from enforcing arc consistency.

There is a strong relation between directional arc consistency and mini-buckets. It was shown in [22] that given a WCSP with $\top = \infty$, and a variable ordering, the lower bound induced by mini-buckets involving at most 2 variables is the same as the lower bound induced by $C_\emptyset$ after the problem is made directional arc consistent. However, the mini-bucket computation provides only a lower bound while DAC enforcing provides both a lower bound and a directional arc consistent equivalent problem. All the work done to compute the lower bound is captured in this problem which offers the opportunity to perform incremental updates of the lower bound.

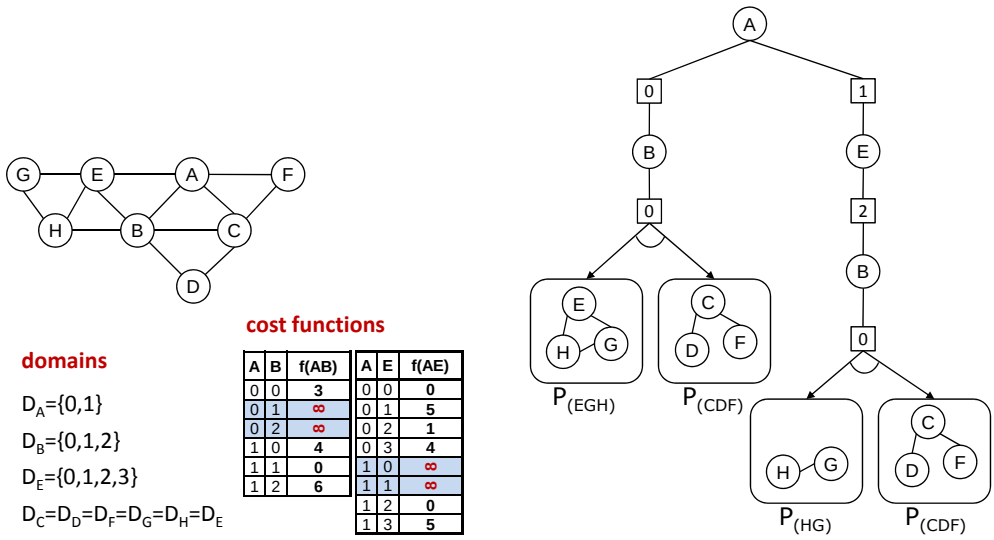
8 Dynamic Variable Orderings

The depth-first AND/OR Branch-and-Bound algorithm introduced in Section 6 assumed a static variable ordering determined by the underlying pseudo tree of the primal graph. In classical CSPs, dynamic variable ordering is known to have a significant impact on the size of the search space explored [14]. Well known variable ordering heuristics, such as *min-domain* [49], *min-dom/ddeg* [50], *brelaz* [51] and *min-dom/wdeg* [52,53] were shown to improve dramatically the performance of systematic search algorithms. In this section we discuss some strategies that allow dynamic variable orderings in AND/OR search.

We distinguish two classes of variable ordering heuristics:

(1) *Graph*-based heuristics ($e.g.$, pseudo tree) that try to maximize problem decomposition, and
(2) *Semantic*-based heuristics ($e.g.$, min-domain) that aim at shrinking the search space, based on context and current value assignment.

These two approaches are orthogonal, namely we can use one as the primary guide and break ties based on the other. We present three schemes of combining these heuristics. For simplicity and without loss of generality we consider the *min-domain* as our semantic variable ordering heuristic. It selects the next variable to instantiate as the one having the smallest current domain among the uninstantiated (future) variables. Clearly, it can be replaced by any other heuristic.



(a) Cost network (b) AND/OR search tree

Fig. 13. Full dynamic variable ordering for AND/OR Branch-and-Bound search.

8.1 Partial Variable Ordering (PVO)

The first approach, called *AND/OR Branch-and-Bound with Partial Variable Ordering* and denoted by AOBB+PVO uses the static graph-based decomposition given by a pseudo tree with a dynamic semantic ordering heuristic applied over chain portions of the pseudo tree. It is an adaptation of the ordering heuristics developed in [54,55] which were used for solving large-scale SAT problem instances.

Consider the pseudo tree from Figure 4(a) inducing the following variable groups (or chains): $\{A, B\}$, $\{C, D\}$ and $\{E, F\}$, respectively. This implies that variables $\{A, B\}$ should be considered before $\{C, D\}$ and $\{E, F\}$. The variables in each group can be dynamically ordered based on a second, independent heuristic. Notice that once variables $\{A, B\}$ are instantiated, the problem decomposes into independent components that can be solved separately.

AOBB+PVO can be derived from Algorithm 2 with some simple modifications. As usual, the algorithm traverses an AND/OR search tree in a depth-first manner, guided by a pre-computed pseudo tree $\mathcal{T}$. When the current AND node n, labeled $\langle X_i, x_i \rangle$ is expanded in the forward step (line 9), the algorithm generates its OR successor, labeled by X_j, based on the semantic variable ordering heuristic (line 12). Specifically, the OR node m, labeled X_j corresponds to the uninstantiated variable with the smallest current domain in the current pseudo tree chain. If there are no uninstantiated variables left in the current chain, namely variable X_i was instantiated last, then the OR successors of n are labeled by the variables with the smallest domain from the variable chains rooted by X_i in $\mathcal{T}$.

A second, orthogonal approach to partial variable orderings, called *AND/OR Branch-and-Bound with Full Dynamic Variable Ordering* and denoted by DVO+AOBB, gives priority to the dynamic semantic variable ordering heuristic and applies static problem decomposition as a secondary principle during search. This idea was also explored in [56] for model counting, and more recently in [57] for weighted model counting.

For illustration, consider the cost network with 8 variables $\{A, B, C, D, E, F, G, H\}$, 13 binary cost functions, and the domains given in Figure 13(a), as follows: $D_A = \{0, 1\}$, $D_B = \{0, 1, 2\}$, and $D_C = D_D = D_E = D_F = D_G = D_H = \{0, 1, 2, 3\}$, respectively. Each of the cost functions $f(A, B)$ and $f(A, E)$ assigns an ∞ cost to two of their corresponding tuples, whereas the remaining 11 functions do not contain such tuples.

During search, variables are instantiated in min-domain order. However, after each variable assignment we test for problem decomposition and solve the remaining subproblems independently. Figure 13(b) shows the partial AND/OR search tree obtained after several variable instantiations based on the min-degree ordering heuristic. Notice that, depending on the order in which the variables are instantiated, the primal graph may decompose into independent components *higher* or *deeper* in the search tree. For instance, after instantiating A to 0, the values $\{1, 2\}$ can be removed from the domain of B, because the corresponding tuples have cost ∞ in the cost function $f(A, B)$ (see Figure 13(a)). Therefore, B is the next variable to be instantiated, at which point the problem decomposes into independent components, as shown in Figure 13(b). Similarly, when A is instantiated to 1, values $\{0, 1\}$ can also be removed from the domain of E, because of the cost function $f(A, E)$. Then, variable E, having 2 values left in its domain, is selected next in the min-domain order, followed by B with domain size 3.

DVO+AOBB can be expressed by modifying Algorithm 2 as follows. It instantiates the variables dynamically using the min-domain ordering heuristic while maintaining the current graph structure. Specifically, after the current AND node n, labeled $\langle X_i, x_i \rangle$, is expanded, DVO+AOBB tentatively removes from the primal graph all nodes corresponding to the instantiated variables together with their incoming arcs. If disconnected components are detected, their corresponding subproblems are then solved separately and the results combined in an AND/OR manner. In this case a variable selection may yield a significant impact on tightening the search space, yet, it may not yield a good decomposition for the remaining problem.

8.3 Dynamic Separator Ordering (DSO)

The third approach, *AND/OR Branch-and-Bound with Dynamic Separator Ordering* (AOBB+DSO), exploits constraint propagation which can be used for dynamic graph-based decomposition with a dynamic semantic variable ordering, giving priority to the first. At each AND node we apply a lookahead procedure hoping to detect singleton variables ($i.e.$, with only one feasible value left in their domains). When the value of a variable is known, it can be removed from the corresponding subproblem, yielding a stronger decomposition of the simplified primal graph.

AOBB+DSO defined on top of Algorithm 2 creates and maintains a separator S of the current primal graph. A graph separator can be computed using the hypergraph partitioning method presented in [55]. The next variable is chosen dynamically from S by the min-domain ordering heuristic until S is fully instantiated and the current problem decomposes into several independent subproblems, which are then solved separately. The separator of each component is created from a simplified subgraph resulted from previous constraint propagation steps and it may differ for different value assignments. Clearly, if no singleton variables are discovered by the lookahead steps this approach is computationally identical to AOBB+PVO, although it may have a higher overhead due to the dynamic generation of the separators.

9 Experimental Results

We have conducted a number of experiments on two common optimization problem classes in graphical models: finding the Most Probable Explanation in Bayesian networks and solving Weighted CSPs. We implemented our algorithms in C++ and carried out all experiments on a 1.8GHz dual-core Athlon64 with 2GB of RAM running Ubuntu Linux 7.04.

9.1 Overview and Methodology

Bayesian Networks. For the MPE task, we tested the performance of the AND/OR Branch-and-Bound algorithms on the following types of problems: random Bayesian networks, random coding networks, grid networks, Bayesian networks derived from the ISCAS'89 digital circuit benchmark, genetic linkage analysis networks, networks from the Bayesian Network Repository, and Bayesian networks from the UAI'06 Inference Evaluation Dataset. We report here some of the results and place the rest in the Appendix.

The detailed outline of the experimental evaluation for Bayesian networks is given

Table 1
Detailed outline of the experimental evaluation for Bayesian networks.

Benchmarks	static mini-buckets BB+SMB(i) AOBB+SMB(i)	dynamic mini-buckets BB+DMB(i) AOBB+DMB(i)	min-fill vs. hypergraph pseudo trees	constraint propagation	SamIam	Superlink
main						
Random BN	√	√	√	-	-	-
Coding	√	√	√	-	√	-
Grids	√	√	√	√	√	-
Linkage	√	-	√	√	√	√
appendix						
ISCAS'89	√	√	√	√	√	-
UAI'06 Dataset	√	-	√	-	√	-
BN Repository	√	√	-	-	√	-

in Table 1. We evaluated the two classes of depth-first AND/OR Branch-and-Bound search algorithms, guided by the static and dynamic mini-bucket heuristics, denoted by `AOBB+SMB(`i`)` and `AOBB+DMB(`i`)`, respectively. We compare these algorithms against traditional depth-first OR Branch-and-Bound algorithms with static and dynamic mini-bucket heuristics introduced in [5,38], denoted by `BB+SMB(`i`)` and `BB+DMB(`i`)`, respectively, which were among the best-performing complete search algorithms for this domain at the time. The parameter i represents the mini-bucket i-bound and controls the accuracy of the heuristic. The pseudo trees that guide AND/OR search algorithms were generated using the min-fill and hypergraph partitioning heuristics, described later in this section. We also consider an extension of the AND/OR Branch-and-Bound that exploits the determinism present in the Bayesian network by constraint propagation.

Since the pre-compiled mini-bucket heuristics require a static variable ordering, the corresponding OR and AND/OR search algorithms used the variable ordering as well derived from a depth-first traversal of the guiding pseudo tree. When we applied dynamic variable orderings with dynamic mini-bucket heuristics we observed that the computational overhead was prohibitively large compared with the static variable ordering setup. We therefore do not report these. We note however that the `AOBB+SMB(`i`)` and `AOBB+DMB(`i`)` algorithms support a restricted form of dynamic variable and value ordering. Namely, there is a dynamic internal ordering of the successors of the node just expanded, before placing them onto the search stack. Specifically, in line 26 of Algorithm 2, if the current node n is AND, then the independent subproblems rooted by its OR children can be solved in decreasing order of their corresponding heuristic estimates (variable ordering). Alternatively, if n is OR, then its AND children corresponding to domain values can also be sorted in decreasing order of their heuristic estimates (value ordering).

We compared our algorithms with the SAMIAM version 2.3.2 software package [1].

[1] Available at http://reasoning.cs.ucla.edu/samiam. We used the `batchtool 1.5` pro-

Table 2
Detailed outline of the experimental evaluation for Weighted CSPs.

Benchmarks	static mini-buckets BB+SMB(i) AOBB+SMB(i)	dynamic mini-buckets BB+DMB(i) AOBB+DMB(i)	min-fill vs. hypergraph pseudo trees	EDAC heuristics BBEDAC AOEDAC, PVO, DVO, DSO	toolbar
main					
SPOT5	√	√	√	√	√
ISCAS'89	√	√	√	√	√
Mastermind	√	-	√	√	√
CELAR	-	-	-	√	√

SAMIAM is a public implementation of Recursive Conditioning [58] which can also be viewed as an AND/OR search algorithm. The algorithm uses a context-based caching mechanism that records the optimal solution of the subproblems and retrieves the saved values when the same subproblems are encountered again during search. This version of recursive conditioning traverses a context minimal AND/OR search graph [1], rather than a tree, and its space complexity is exponential in the treewidth. Note that when we use mini-bucket heuristics with high values of i, we use space exponential in i for the heuristic calculation and storing. Our search regime however does not consume any additional space.

Weighted CSPs. For WCSPs we evaluated the performance of the AND/OR Branch-and-Bound algorithms on: random binary WCSPs, scheduling problems from the SPOT5 benchmark, networks derived from the ISCAS'89 digital circuits, radio link frequency assignment problems and instances of the Mastermind game.

The outline of the experimental evaluation for Weighted CSPs is detailed in Table 2. In addition to the mini-bucket heuristics, we also consider a heuristic evaluation function that is computed by maintaining Existential Directional Arc-Consistency (EDAC) [24]. `AOBB` with this heuristic is called `AOEDAC`. We also consider the extension of `AOEDAC` that incorporates dynamic variable orderings heuristics described earlier yielding: `AOEDAC+PVO` (partial variable ordering - Section 8.1), `DVO+AOEDAC` (full dynamic variable ordering - Section 8.2) and `AOEDAC+DSO` (dynamic separator ordering - Section 8.3). For comparison, we report results obtained with our implementation of the classic OR Branch-and-Bound with EDAC, denoted here by `BBEDAC`.

For reference, we also ran `toolbar` [2], an OR Branch-and-Bound that maintains EDAC during search and uses dynamic variable orderings. `toolbar` was introduced in [24] and is currently one of the best performing solvers for binary WCSPs.

The semantic-based dynamic variable ordering heuristic used by both the OR and AND/OR Branch-and-Bound algorithms with EDAC based heuristics was the *min-*

vided with the package.

[2] Available at: http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP

dom/ddeg heuristic, which selects the variable with the smallest ratio of the current domain size divided by the future degree. Ties were broken lexicographically.

Measures of Performance. In all our experiments we report the average CPU time in seconds and the number of nodes visited, required for proving optimality. We also specify the number of variables (n), number of evidence variables (e), maximum domain size (k), number of functions (c), maximum arity of the functions (r), 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, w^* and h are computed after the evidence nodes are removed from the graph. We also report the time required by the Mini-Bucket algorithm MBE(i) to pre-compile the heuristic information. 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.

9.2 Finding Good Pseudo Trees

The performance of the AND/OR Branch-and-Bound search algorithms is influenced by the quality of the guiding pseudo tree. Finding the minimal depth/induced width pseudo tree is a hard problem [2,44,3]. We describe next two heuristics for generating pseudo trees with relatively small depths/induced widths which we used in our experiments.

Min-Fill Heuristic. *Min-Fill* [59] is one of the best and most widely used heuristics for creating small induced width elimination orders. An ordering is generated by placing the variable with the smallest *fill set* ($i.e.$, number of induced edges that need be added to fully connect the neighbors of a node) at the end of the ordering, connecting all of its neighbors and then removing the variable from the graph. The process continues until all variables have been eliminated.

Once an elimination order is given, the pseudo tree can be extracted as a depth-first traversal of the min-fill induced graph, starting with the variable that initiated the ordering, always preferring as successor of a node the earliest adjacent node in the induced graph. An ordering uniquely determines a pseudo tree. This approach was first used by [3].

To improve orderings, we can run the min-fill ordering several times by randomizing the tie breaking. In our experiments, we ran the min-fill heuristic just once and broke the ties lexicographically.

Hypergraph Decomposition Heuristic. An alternative heuristic for generating a low height balanced pseudo tree is based on the recursive decomposition of the

Table 3
Bayesian Networks Repository (left); SPOT5 benchmarks (right).

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

dual hypergraph associated with the graphical model.

DEFINITION **31 (dual hypergraph)** *The* dual hypergraph *of a graphical model* $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$*, is a pair* $\mathcal{H}(\mathcal{R}) = (\mathbf{V}, \mathbf{E})$*, where each function in* $\mathbf{F}$ *is a vertex* $v_i \in \mathbf{V}$ *and each variable in* $\mathbf{X}$ *is an edge* $e_j \in \mathbf{E}$ *connecting all the functions (vertices) in which it appears.*

DEFINITION **32 (hypergraph separators)** *Given a dual hypergraph* $\mathcal{H} = (\mathbf{V}, \mathbf{E})$ *of a graphical model, a* hypergraph separator decomposition *is a triple* $\langle \mathcal{H}, \mathcal{S}, \alpha \rangle$ *where:*

(1) $\mathcal{S} \subset \mathbf{E}$*, and the removal of* $\mathcal{S}$ *separates* $\mathcal{H}$ *into* k *disconnected components (subgraphs);*
(2) α *is a relation over the size of the disjoint subgraphs (i.e., balance factor).*

It is well known that the problem of finding the minimal size hypergraph separator is hard. However heuristic approaches were developed over the years. A good approach is packaged in `hMeTiS` [3].

We will use this software as a basis for our pseudo tree generation. Following [58], generating a pseudo tree $\mathcal{T}$ for $\mathcal{R}$ using `hMeTiS` is fairly straightforward. The vertices of the hypergraph are partitioned into two balanced (roughly equal-sized) parts, denoted by $\mathcal{H}_{left}$ and $\mathcal{H}_{right}$ respectively, while minimizing the number of hyperedges across. A small number of crossing edges translates into a small number

[3] Available at: http://www-users.cs.umn.edu/karypis/metis/hmetis

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.

Since the hypergraph partitioning heuristic uses a non-deterministic algorithm ($i.e.$, `hMeTiS`), the depth and induced width of the resulting pseudo tree may vary significantly from one run to the next. In our experiments we picked the pseudo tree with the smallest depth out of 10 independent runs.

In Table3 we illustrate the induced width and depth of the pseudo tree obtained with the hypergraph and min-fill heuristics for 10 belief networks from the Bayesian Networks Repository [4] and 10 constraint networks derived from the SPOT5 benchmark [10]. From this and the experiments presented in the remainder of this section, we observe that the min-fill heuristic generates lower induced width pseudo trees, while the hypergraph heuristic produces much smaller depth pseudo trees. Therefore, perhaps the hypergraph based pseudo trees appear to be favorable for tree search algorithms guided by heuristics that are not sensitive to the treewidth ($e.g.$, local consistency based heuristics), while the min-fill pseudo trees, which minimize the treewidth, are more appropriate for search algorithms whose guiding heuristic is sensitive to the treewidth ($e.g.$, mini-bucket heuristics).

9.3 Results for Empirical Evaluation of Bayesian Networks

9.3.1 Random Bayesian Networks

The random Bayesian networks were generated using parameters (n, k, c, p), where n is the number of variables, k is the domain size, c is the number of conditional probability tables (CPTs) and p is the number of parents in each CPT. The structure of the network is created by randomly picking c variables out of n and, for each, randomly picking p parents from their preceding variables, relative to some ordering. The remaining $n - c$ variables are called *root* nodes. The entries of each probability table are generated randomly using a uniform distribution, and the table is then normalized.

Table 4 shows detailed results for solving a class of random belief networks using min-fill and hypergraph partitioning based pseudo trees. The columns are indexed by the mini-bucket i-bound. For each domain size we generated 20 random instances and in each test case $e = 10$ variables were chosen randomly as evidence.

We observe that `AOBB+SMB(`i`)` is better than `BB+SMB(`i`)` at relatively small i-

[4] Available at: http://www.cs.huji.ac.il/labs/compbio/Repository

Table 4
CPU time in seconds and number of nodes explored for solving **random belief networks** with $n = 100$ nodes, $p = 2$ parents per CPT, $c = 90$ CPTs and domain sizes $k \in \{2, 3, 4, 5\}$. Each test case had $e = 10$ variables chosen randomly as evidence. The time limits are 180 seconds for $k \in \{2, 3\}$ and 300 seconds for $k \in \{4, 5\}$, respectively. Pseudo trees generated by min-fill and hypergraph heuristics.

minfill pseudo tree													
k	(w*, h)	MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=2		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=4		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=6		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=8		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=10		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=12	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
		0.43		0.43		0.44		0.43		0.44		0.45	
		174.86	2,109,890	89.33	1,088,420	38.19	488,197	3.28	41,539	0.90	12,918	1.06	15,021
2	(14, 25)	12.23	308,536	1.01	25,706	0.70	17,124	0.17	4,273	0.07	1,666	**0.06**	1,103
		25.86	62,466	3.13	10,737	2.75	10,289	2.71	10,653	2.91	11,570	2.59	10,153
		3.07	11,023	0.50	1,365	0.24	635	0.15	489	0.17	450	0.18	347
		0.43		0.43		0.44		0.47		0.69		2.12	
		-	-	-	-	122.00	1,061,530	37.44	344,128	7.23	67,299	3.55	21,341
3	(14, 25)	-	-	100.47	1,950,280	40.54	722,818	19.78	384,609	2.37	39,318	**2.26**	13,957
		163.72	208,945	31.09	24,603	23.00	19,753	23.50	19,293	28.24	17,787	44.43	18,994
		137.61	357,485	24.93	34,127	16.17	6,283	16.40	1,613	20.85	702	34.96	478
		0.50		0.50		0.52		0.80		3.93		39.22	
		-	-	-	-	251.01	1,724,330	107.49	742,803	**20.31**	137,357	43.14	42,869
4	(14, 25)	-	-	283.61	4,585,420	188.38	2,922,760	85.19	1,326,610	23.38	303,695	41.27	51,276
		-	-	162.86	48,281	157.93	31,620	170.88	28,508	218.89	27,731	323.48	13,235
		-	-	155.49	85,964	146.72	7,891	161.38	1,367	211.84	697	317.11	218
		0.49		0.49		0.58		2.20		33.18			
		-	-	-	-	298.49	1,645,150	174.05	998,579	**116.31**	572,171		
5	(14, 25)	-	-	-	-	267.68	3,804,650	185.49	2,540,320	127.26	1,218,160		
		-	-	277.68	51,702	288.91	42,167	293.88	38,522	-	-		
		-	-	270.10	69,453	282.30	5,623	291.07	1,054	-	-		
hypergraph pseudo tree													
		0.43		0.43		0.44		0.43		0.44		0.45	
		178.94	2,076,390	143.48	1,739,470	121.20	1,495,580	67.72	858,691	24.85	319,742	7.63	99,539
2	(14, 20)	18.87	453,372	2.37	44,796	0.83	9,181	0.73	7,135	0.54	2,415	**0.50**	1,242
		120.80	203,392	8.83	15,798	3.65	9,299	3.47	9,134	3.41	9,013	3.47	9,163
		3.64	11,524	0.85	899	0.63	480	0.58	363	0.60	336	0.66	294
		0.43		0.43		0.44		0.47		0.69		2.12	
		-	-	-	-	-	-	172.16	1,508,000	119.81	1,066,200	81.45	717,941
3	(14, 20)	178.35	3,965,780	137.11	2,558,520	67.95	1,078,460	14.27	198,026	5.10	68,847	**2.94**	13,396
		-	-	67.56	53,725	29.66	24,415	21.68	20,004	29.79	19,347	49.22	17,425
		129.58	490,813	16.66	9,164	10.57	1,409	8.39	640	16.64	469	35.47	349
		0.50		0.50		0.52		0.80		3.93		39.22	
		-	-	-	-	-	-	-	-	243.82	1,685,500	157.19	848,755
4	(14, 20)	-	-	284.29	4,679,600	176.11	2,478,050	89.32	1,196,610	**35.50**	409,701	41.73	30,918
		-	-	167.98	52,789	141.18	32,760	164.00	30,774	213.91	31,316	300.53	13,787
		287.64	666,192	142.71	18,706	125.39	2,834	139.73	785	196.69	502	303.70	195
		0.49		0.49		0.58		2.20		33.18			
		-	-	-	-	-	-	-	-	295.99	1,524,180		
5	(14, 20)	-	-	-	-	257.71	2,955,420	152.83	1,365,200	**102.25**	586,760		
		-	-	287.11	59,292	289.47	40,179	-	-	-	-		
		-	-	254.74	30,200	253.84	1,933	279.00	645	-	-		

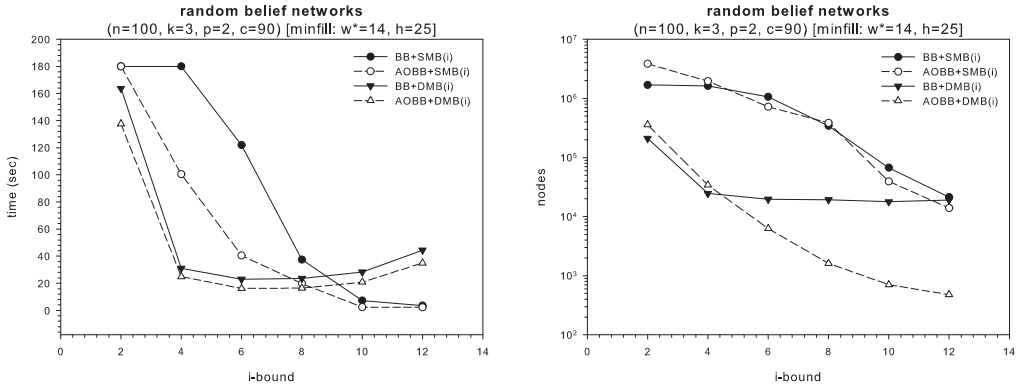


Fig. 14. Comparison of the impact of static and dynamic mini-bucket heuristics on **random belief networks** with parameters $(n = 100, k = 3, p = 2, c = 90)$ from Table 4.

bounds ($i.e.$, $i \in \{2, 4, 6\}$) when the heuristic is weak. This demonstrates the benefit of AND/OR over classical OR search when the heuristic estimates are relatively weak and the algorithms rely primarily on search rather than on pruning via the heuristic evaluation function. As the i-bound increases ($e.g.$, $i \geq 8$) and the heuristic estimates become strong enough to cut the search space substantially, the difference between the AND/OR and OR Branch-and-Bound decreases.

When focusing on dynamic mini-bucket heuristics, we observe that AOBB+DMB(i) is better than BB+DMB(i) at relatively small i-bounds, but the difference is not that prominent as in the static case. This is probably because these heuristics are far more accurate compared with the pre-compiled version and the savings in number of nodes caused by traversing the AND/OR search tree do not translate into additional time savings. When comparing the static and dynamic mini-bucket heuristics, we see that the latter is competitive only for relatively small i-bounds, because of the high overhead of the dynamic mini-bucket. This may be significant because small i-bounds usually require restricted space. At higher levels of the i-bound the accuracy of the dynamic mini-bucket heuristic does not outweigh its overhead.

In some exceptional cases the OR Branch-and-Bound explored fewer nodes than the AND/OR counterpart. For example, on problem class displayed in the third horizontal block of Table 4, the search space explored by AOBB+DMB(4) was almost two times larger that that explored by BB+DMB(4). Similarly, AOBB+SMB(8) expanded almost two times more nodes than BB+SMB(8) on this problem class. This can be explained by the internal dynamic ordering used by AND/OR Branch-and-Bound to solve independent subproblems rooted at the AND nodes in the search tree, which did not pay off in this case. We also see that even though BB+SMB(i) (resp. BB+DMB(i)) traversed a smaller search space than AOBB+SMB(i) (resp. AOBB+DMB(i)), the runtime of the AND/OR algorithms was actually better. This is because the computational overhead of the mini-bucket heuristics was much smaller for AND/OR search than for OR search, and, therefore, the AND/OR algorithms were able to overcome the increase in size of the search space.

Figure 14 plots the running time and number of nodes visited by AOBB+SMB(i) and AOBB+DMB(i) (resp. BB+SMB(i) and BB+DMB(i)) as a function of the mini-bucket i-bound for solving the random belief networks with parameters $(n = 100, k = 3, p = 2, c = 90)$ ($i.e.$, corresponding to the second horizontal block from Table 4). It shows explicitly how the performance of Branch-and-Bound changes with the mini-bucket strength for both types of heuristics. We see that i-bound of 6 is most cost effective for dynamic mini-buckets, while i-bound of 12 yields best performance for static mini-buckets. We see clearly that the dynamic mini-bucket heuristic is more accurate yielding smaller search spaces. It also demonstrates that the dynamic mini-bucket heuristics are cost effective at small i-bounds, whereas the pre-compiled version is more powerful for larger i-bounds. This behavior is typical for all instances presented in the subsequent sections.

When comparing the min-fill versus hypergraph heuristics for generating pseudo trees, we observe that the hypergraph based pseudo trees have smaller depths. However, min-fill trees appear to be favorable to AOBB+SMB(i). This may be explained by the fact that pre-compiling the mini-bucket heuristic using a min-fill based elimination ordering tends to generate more accurate estimates. For AOBB+DMB(i) the picture is sometimes reversed, but not in a significant way.

9.3.2 Random Coding Networks

We experimented with random coding networks from the class of *linear block codes* [60–62]. They can be represented as 4-layer belief networks with K nodes in each layer ($i.e.$, the number of input bits). The second and third layers correspond to input information bits and parity check bits respectively. Each parity check bit represents an XOR function of the input bits. The first and last layers correspond to transmitted information and parity check bits respectively. Input information and parity check nodes are binary, while the output nodes are real-valued. Given a number of input bits K, number of parents P for each XOR bit, and channel noise variance σ^2, a coding network structure is generated by randomly picking parents for each XOR node. Then we simulate an input signal by assuming a uniform random distribution of information bits, compute the corresponding values of the parity check bits, and generate an assignment to the output nodes by adding Gaussian noise to each information and parity check bit. The decoding algorithm takes as input the coding network and the observed real-valued output assignment and recovers the original input bit-vector by computing an MPE assignment.

Table 5 displays the results using min-fill and hypergraph based pseudo trees for solving a classes of random coding networks with $K = 128$ input bits. The number of parents for each XOR bit was $P = 4$ and we chose the channel noise variance $\sigma^2 \in \{0.22, 0.36\}$. For each value combination of the parameters we generated 20 random instances.

Table 5
CPU time and nodes visited for solving **random coding networks** with 128 bits, 4 parents per XOR bit and channel noise variance $\sigma^2 \in \{0.22, 0.36\}$. Time limit 5 minutes. The pseudo trees were generated by the min-fill and hypergraph heuristics.

minfill pseudo tree												
(K, N)	(w*, h)	Samlam	MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=4		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=8		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=12		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=16		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=20	
			time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
			0.05		0.06		0.18		1.80		25.65	
		-	-		257.42	1,581,950	52.69	345,028	3.53	12,513	25.75	2,065
(128, 256)	(53, 71)	-	-	-	229.02	3,227,110	16.67	206,004	**3.51**	22,644	25.87	3,081
$\sigma^2 = 0.22$			196.64	41,359	48.80	4,178	17.86	726	130.95	588	-	-
			195.82	121,822	48.17	9,391	17.15	500	129.38	388	-	-
			0.05		0.06		0.18		1.80		25.39	
			-	-	-	-	271.29	1,717,770	211.88	1,452,980	**99.14**	598,738
(128, 256)	(53, 71)	-	-	-	291.61	4,309,160	240.74	3,409,580	188.44	2,617,880	110.89	1,137,120
$\sigma^2 = 0.36$			289.06	65,591	230.23	22,617	234.33	6,857	276.40	1,957	-	-
			289.09	223,938	229.91	46,768	233.96	7,947	276.31	953	-	-
hypergraph pseudo tree												
			0.73		0.74		0.86		2.49		27.13	
			-	-	285.82	1,765,300	184.90	1,264,890	94.43	677,488	31.72	36,604
(128, 256)	(53, 63)	-	-	-	238.91	3,070,670	125.01	1,252,930	38.12	404,160	**27.28**	1,658
$\sigma^2 = 0.22$			277.94	133,702	152.10	21,264	27.63	942	90.89	376	-	-
			282.15	126,614	84.82	6,358	73.46	1,307	166.75	409	-	-
			0.73		0.74		0.86		2.51		25.95	
			-	-	-	-	296.69	1,948,930	285.70	2,009,240	210.16	1,360,710
(128, 256)	(53, 63)	-	-	-	-	-	296.02	3,583,930	251.96	2,969,470	**142.85**	1,340,740
$\sigma^2 = 0.36$			-	-	287.30	32,456	269.73	5,269	292.08	2,308	-	-
			-	-	261.00	58,212	269.14	4,614	282.24	823	-	-

We see that `AOBB+SMB(`i`)` and `AOBB+DMB(`i`)` are slightly faster than `BB+SMB(`i`)` and `BB+DMB(`i`)`, respectively, only for relatively small i-bounds. In several test cases, however, the search space explored by the AND/OR algorithms was larger than the corresponding OR space. For instance, on the problem class with $\sigma^2 = 0.36$ shown in the second horizontal block of Table 5, `AOBB+SMB(12)` expanded almost 2 times more nodes than `BB+SMB(12)`. This was caused again by the internal dynamic variable ordering used by the AND/OR algorithms. We also see that the overhead of the mini-bucket heuristic was smaller in the AND/OR than the OR case, which paid off in some test cases.

When looking at the impact of the min-fill versus the hypergraph based pseudo trees we see that, even though the hypergraph trees were shallower than the min-fill ones, the mini-bucket heuristics generated relative to min-fill orderings were more accurate than those corresponding to hypergraph partitioning based orderings. In some cases this translated into significant time savings. For example, on the problem class with $\sigma^2 = 0.22$, the min-fill pseudo tree causes an 8-fold speedup over the hypergraph tree, for `AOBB+SMB(12)`. A similar behavior can be observed for dynamic mini-bucket heuristics, as well.

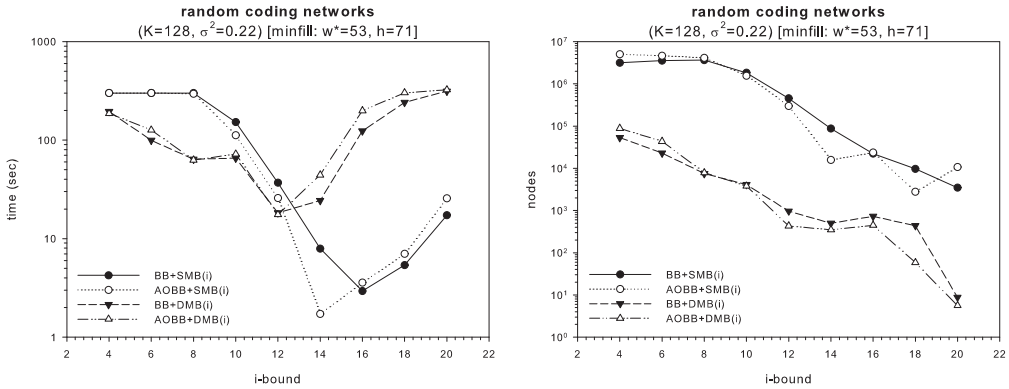


Fig. 15. Comparison of the impact of static and dynamic mini-bucket heuristics on **random coding networks** with parameters $(K = 128, \sigma^2 = 0.22)$ from Table 5.

Figure 15 plots the running time and number of nodes visited by AOBB+SMB(i) and AOBB+DMB(i) (resp. BB+SMB(i) and BB+DMB(i)), for solving the coding networks with parameters $(K = 128, \sigma^2 = 0.22)$ ($i.e.$, corresponding to the first horizontal block from Table 5). We see that as the i-bound increases, the mini-bucket heuristics become more accurate and the performance of Branch-and-Bound improves. For example, i-bound of 14 yields the best performance for AOBB+SMB(i), whereas AOBB+DMB(i) achieves the best performance at $i = 12$. For even larger i-bounds however, the overhead of both the pre-compiled and dynamic heuristics deteriorates the performance of the algorithms. The dynamic mini-bucket heuristics are better for relatively small i-bounds, whereas relatively larger i-bounds are cost effective for the pre-compiled heuristics.

9.3.3 Grid Networks

In random grid networks, the nodes are arranged in an $N \times N$ square and each CPT is generated uniformly at random. We experimented with problem instances having bi-valued variables that were initially developed in [63] for the task of weighted model counting. For these problems N ranges between 10 and 38, and, for each instance, 90% of the CPTs are deterministic (having only 0 and 1 probability entries).

Table 6 displays the results for experiments with 8 grids of increasing difficulty, using min-fill based pseudo trees. For each test instance we ran a single MPE query with e evidence variables picked randomly. We see again the superiority of AOBB+SMB(i) over the OR counterpart, especially on the harder instances. For example, on the 90-30-1 grid, AOBB+SMB(20) finds the MPE in about 87 seconds, whereas BB+SMB(20) exceeds the 1 hour time limit. The AND/OR Branch-and-Bound algorithms with dynamic mini-bucket heuristics as well as SamIam are able to solve relatively efficiently only the first 3 test instances.

Figure 16 plots the running time and number of nodes visited by AOBB+SMB(i) and AOBB+DMB(i) (resp. BB+SMB(i) and BB+DMB(i)), for solving the 90-14-1

Table 6
CPU time in seconds and nodes visited for solving **grid networks**. Time limit 1 hour.

minfill pseudo tree													
grid (w*, h) (n, e)	SamIam time	MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=8 time	nodes	MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=10 time	nodes	MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=12 time	nodes	MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=14 time	nodes	MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=16 time	nodes	MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=18 time	nodes
		0.01		0.02		0.04		0.07		0.07		0.08	
90-10-1		0.12	3,348	**0.04**	424	0.05	153	0.07	153	0.08	153	0.09	153
(13, 39)	0.13	0.17	8,080	0.06	2,052	0.05	101	0.07	101	0.08	101	0.08	101
(100, 0)		0.87	543	0.57	250	0.48	153	0.54	153	0.54	153	0.54	153
		0.34	344	0.33	241	0.32	101	0.39	101	0.39	101	0.39	101
		0.02		0.04		0.11		0.22		0.72		2.71	
90-14-1		75.71	1,235,366	71.98	1,320,090	1.07	18,852	0.54	5,035	0.90	2,826	2.78	1,075
(22, 66)	11.97	4.27	130,619	3.44	100,696	0.61	17,479	**0.32**	3,321	0.81	2,938	2.80	3,386
(196, 0)		149.44	16,415	52.34	2,894	12.46	537	13.71	211	19.22	199	38.05	198
		65.74	31,476	33.57	4,137	7.50	397	12.00	211	17.65	199	36.87	198
		0.03		0.05		0.14		0.46		1.01		4.36	
90-16-1		-	-	-	-	23.74	347,479	1.85	18,855	**1.44**	6,098	4.53	1,894
(24, 82)	147.19	362.66	10,104,350	91.03	2,600,690	7.53	193,440	1.89	39,825	1.78	23,421	4.55	5,842
(256, 0)		771.73	43,366	553.08	13,363	172.14	2,011	166.61	1,169	65.15	414	181.71	414
		1114.19	462,180	410.87	47,121	109.11	3,227	80.57	719	40.68	260	109.76	260

		i=10		i=12		i=14		i=16		i=18		i=20	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
		0.14		0.33		0.89		2.69		7.61		31.26	
90-24-1		-	-	-	-	-	-	-	-	-	-	-	-
(33, 111)	-	-	-	-	-	1500.66	24,117,151	921.96	18,238,983	**93.73**	1,413,764	111.46	1,308,009
(576, 20)		-	-	-	-	-	-	-	-	-	-	-	-
		-	-	-	-	-	-	1367.38	2,739	1979.42	1,228	2637.71	598
		0.16		0.37		1.02		3.39		11.74		36.16	
90-26-1		-	-	-	-	-	-	324.30	2,234,558	-	-	70.53	327,859
(36, 113)	-	1533.11	17,899,574	206.93	2,903,489	242.37	3,205,257	**7.43**	59,055	21.48	165,182	36.49	5,777
(676, 40)		-	-	-	-	-	-	-	-	-	-	-	-
		1852.27	177,661	-	-	-	-	1514.18	2,545	2889.49	1,191	-	-
		0.25		0.53		1.35		4.36		13.34		50.53	
90-30-1		-	-	-	-	-	-	-	-	-	-	-	-
(43, 150)	-	-	-	742.51	9,445,224	239.08	3,324,942	215.56	3,039,966	101.10	1,358,569	**87.68**	485,300
(900, 60)		-	-	-	-	-	-	-	-	-	-	-	-
		-	-	-	-	-	-	-	-	-	-	-	-
		0.33		0.66		1.60		5.35		18.42		62.17	
90-34-1		-	-	-	-	-	-	-	-	-	-	-	-
(45, 153)	-	-	-	-	-	-	-	-	-	-	-	**257.14**	1,549,829
(1154, 80)		-	-	-	-	-	-	-	-	-	-	-	-
		-	-	-	-	-	-	-	-	-	-	-	-
		0.41		0.82		2.16		6.43		20.46		72.10	
90-38-1		-	-	-	-	-	-	-	-	-	-	-	-
(47, 163)	-	-	-	936.65	6,835,745	1858.99	12,321,175	341.05	2,850,393	252.67	2,079,146	**199.44**	1,038,065
(1444, 120)		-	-	-	-	-	-	-	-	-	-	-	-
		-	-	-	-	-	-	-	-	-	-	-	-

grid network ($i.e.$, corresponding to the second horizontal block from Table 6). Focusing on `AOBB+SMB(`i`)` (resp. `BB+SMB(`i`)`) we see that its running time, as a function of i, forms a U-shaped curve. At first ($i = 4$) it is high, then as the i-bound increases the total time decreases (when $i = 10$ the time is 3.44 for `AOBB+SMB(10)` and 71.98 for `BB+SMB(10)`, respectively), but then as i in-

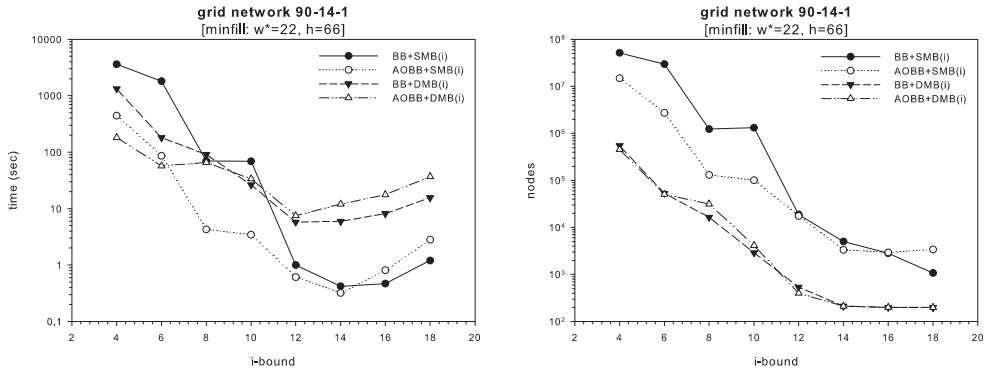


Fig. 16. Comparison of the impact of static and dynamic mini-bucket heuristics on the **`90-14-1` grid network** from Table 6.

creases further the time starts to increase again. The same behavior can be observed in the case of `AOBB+DMB(`i`)` (resp. `BB+DMB(`i`)`) as well.

Figure 17 displays the runtime distribution of `AOBB+SMB(`i`)` using hypergraph based pseudo trees for 4 grid networks. For each reported i-bound, the corresponding data point and error bar show the average as well as the minimum and maximum runtime obtained over 20 independent runs of the algorithm with a 30 minute time limit. We also record the average induced width and depth obtained for the hypergraph pseudo trees (see the header of each plot in Figure 17). As observed earlier, the hypergraph based pseudo trees are significantly shallower compared with the min-fill ones, and in some cases they are able to improve performance dramatically, especially at relatively small i-bounds. For example, on the grid `90-24-1`, `AOBB+SMB(14)` guided by a hypergraph pseudo tree is about 2 orders of magnitude faster than `AOBB+SMB(14)` using a min-fill pseudo tree. At larger i-bounds, the pre-compiled mini-bucket heuristic benefits from the small induced width which normally is obtained with the min-fill ordering. Therefore `AOBB+SMB(`i`)` using min-fill based trees is generally faster than `AOBB+SMB(`i`)` guided by hypergraph based trees ($e.g.$, `90-26-1`).

9.3.4 Genetic Linkage Analysis

In human genetic linkage analysis [64], the *haplotype* is the sequence of alleles at different loci inherited by an individual from one parent, and the two haplotypes (maternal and paternal) of an individual constitute this individual's *genotype*. When genotypes are measured by standard procedures, the result is a list of unordered pairs of alleles, one pair for each locus. The *maximum likelihood haplotype* problem consists of finding a joint haplotype configuration for all members of the pedigree which maximizes the probability of data.

The pedigree data can be represented as a belief network with three types of random variables: *genetic loci* variables which represent the genotypes of the individuals in

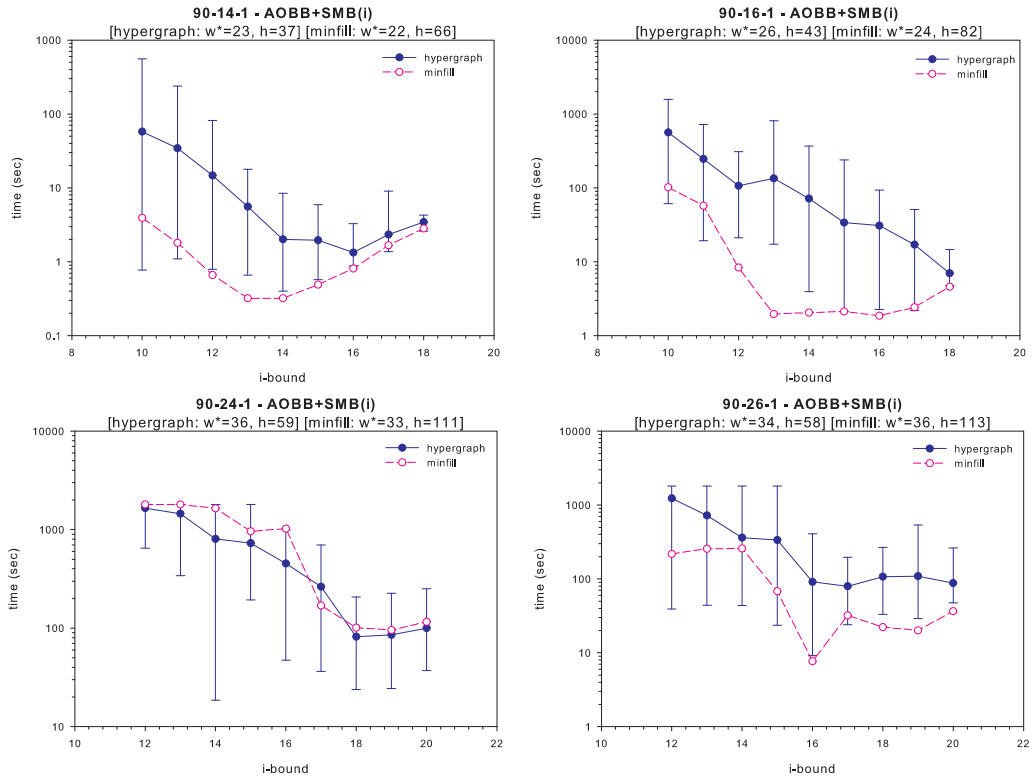


Fig. 17. Min-Fill versus Hypergraph partitioning heuristics for pseudo tree construction. CPU time in seconds for solving **grid networks** with AOBB+SMB(i).

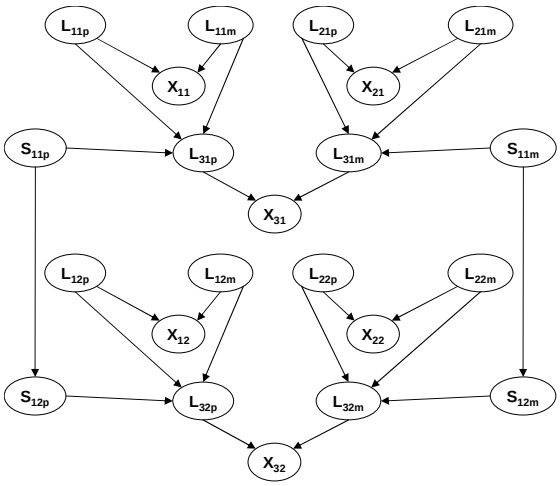


Fig. 18. A fragment of a belief network used in genetic linkage analysis.

the pedigree (two genetic loci variables per individual per locus, one for the paternal allele and one for the maternal allele), *phenotype* variables, and *selector* variables which are auxiliary variables used to represent the gene flow in the pedigree. Figure 18 shows a fragment of a network that describes parents-child interactions in a simple 2-loci analysis. The genetic loci variables of individual i at locus j are denoted by $L_{i,jp}$ and $L_{i,jm}$. Variables $X_{i,j}$, $S_{i,jp}$ and $S_{i,jm}$ denote the phenotype

Table 7
CPU time and nodes visited for solving **genetic linkage networks**. Time limit 3 hours.

min-fill pseudo tree												
			MBE(i)		MBE(i)		MBE(i)		MBE(i)		MBE(i)	
pedigree	Superlink	SamIam	BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)	
(n, k)			AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)	
(w*, h)			i=6		i=8		i=10		i=12		i=14	
			time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
ped1			0.05		0.05		0.11		0.31		0.97	
(299, 5)	54.73	5.44	-	-	-	-	6.34	37,657	7.33	42,447	8.30	41,134
(15, 61)			24.30	416,326	13.17	206,439	**1.58**	24,361	1.84	25,674	1.89	15,156
ped38			0.12		0.45		2.20		60.97		out	
(582, 5)	**28.36**	out	-	-	-	-	-	-	-	-		
(17, 59)			-	-	8120.58	85,367,022	-	-	3040.60	35,394,461		
ped50			0.11		0.74		5.38		37.19		out	
(479, 5)	-	out	-	-	-	-	-	-	-	-		
(18, 58)			-	-	-	-	476.77	5,566,578	**104.00**	748,792		

			i=10		i=12		i=14		i=16		i=18	
			time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
ped23			0.42		2.33		11.33		274.75		out	
(310, 5)	9146.19	out	-	-	-	-	3176.72	14,044,797	343.52	358,604		
(27, 71)			498.05	6,623,197	**15.45**	154,676	16.28	67,456	286.11	117,308		
ped37			0.67		5.16		21.53		58.59		out	
(1032, 5)	**64.17**	out	-	-	-	-	-	-	-	-		
(21, 61)			273.39	3,191,218	1682.09	25,729,009	1096.79	15,598,863	128.16	953,061		

			i=12		i=14		i=16		i=18		i=20	
			time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
ped18			0.51		1.42		4.59		12.87		19.30	
(1184, 5)	139.06	157.05	-	-	-	-	-	-	-	-	-	-
(21, 119)			-	-	2177.81	28,651,103	270.96	2,555,078	100.61	682,175	**20.27**	7,689
ped20			1.42		5.11		37.53		410.96		out	
(388, 5)	**14.72**	out	-	-	-	-	-	-	-	-		
(24, 66)			3793.31	54,941,659	1293.76	18,449,393	1259.05	17,810,674	1080.05	9,151,195		
ped25			0.34		0.72		2.27		6.56		29.30	
(994, 5)	-	out	-	-	-	-	-	-	-	-	-	-
(34, 89)			-	-	-	-	9399.28	111,301,168	3607.82	34,306,937	**2965.60**	28,326,541
ped30			0.42		0.83		1.78		5.75		21.30	
(1016, 5)	13095.83	out	-	-	-	-	-	-	-	-	-	-
(23, 118)			-	-	-	-	-	-	214.10	1,379,131	**91.92**	685,661
ped33			0.58		2.31		7.84		33.44		112.83	
(581, 4)	-	out	-	-	-	-	-	-	-	-	-	-
(37, 165)			2804.61	34,229,495	737.96	9,114,411	3896.98	50,072,988	**159.50**	1,647,488	2956.47	35,903,215
ped39			0.52		2.32		8.41		33.15		81.27	
(1272, 5)	322.14	out	-	-	-	-	-	-	-	-	-	-
(23, 94)			-	-	-	-	4041.56	52,804,044	386.13	2,171,470	**141.23**	407,280
ped42			4.20		31.33		206.40		out		out	
(448, 5)	**561.31**	out	-	-	-	-	-	-				
(25, 76)			-	-	-	-	-	-				

variable, the paternal selector variable and the maternal selector variable of individual i at locus j, respectively. The conditional probability tables that correspond to the selector variables are parameterized by the *recombination ratio* θ [65]. The remaining tables contain only deterministic information. It can be shown that given the pedigree data, the haplotyping problem is equivalent to computing the Most Probable Explanation (MPE) of the corresponding belief network [65,66].

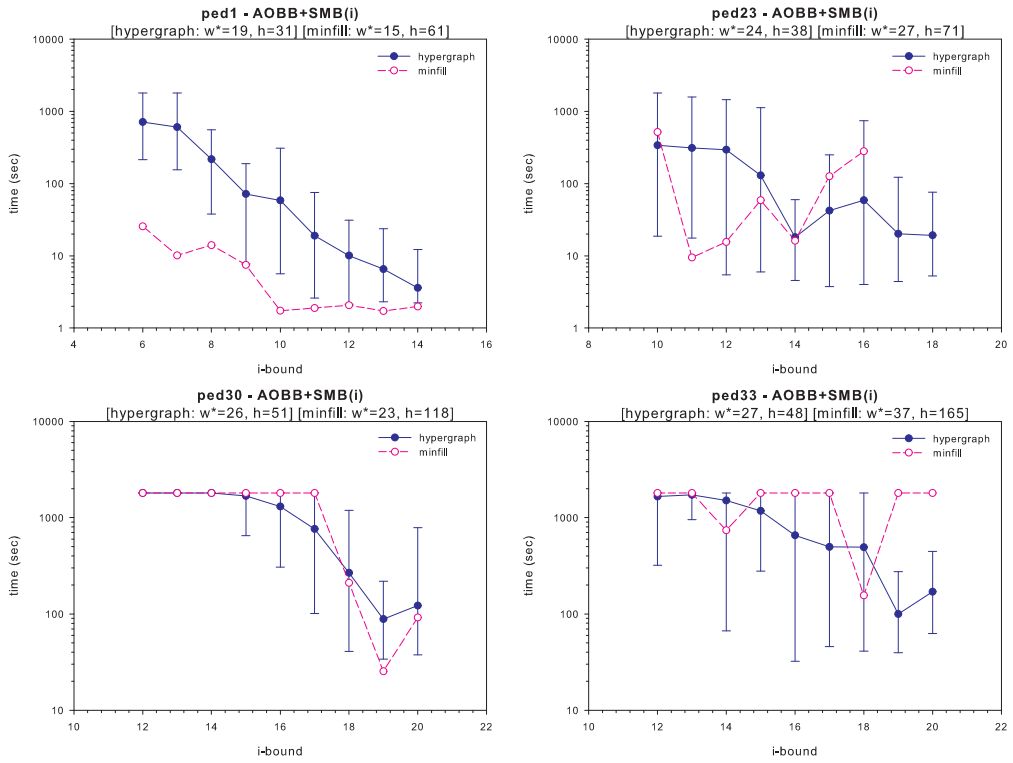


Fig. 19. Min-Fill versus Hypergraph partitioning heuristics for pseudo tree construction. CPU time in seconds for solving **genetic linkage networks** with AOBB+SMB(i).

Table 7 shows results with 12 genetic linkage networks [5]. For comparison, we include results obtained with SUPERLINK 1.6. SUPERLINK [65,66] which is currently one of the most efficient solvers for genetic linkage analysis, uses a combination of variable elimination and conditioning, and takes advantage of the determinism in the network. We did not run AOBB+DMB(i) (resp. BB+DMB(i)) on this domain because of its prohibitively high computational overhead associated with relatively large i-bounds.

We observe again that AOBB+SMB(i) is the best performing algorithm, outperforming its competitors on 8 out of the 12 test networks. For example, on the ped23 instance, AOBB+SMB(12) is 2 orders of magnitude faster than SUPERLINK, whereas SAMIAM and BB+SMB(i) exceed the 2GB memory bound and the 3 hour time limit, respectively. Similarly, on the ped30 instance, AOBB+SMB(20) outperforms SUPERLINK with about 2 orders of magnitude, while neither SAMIAM nor BB+SMB(20) are able to solve the problem instance. Notice that the ped42 instance is solved only by SUPERLINK.

Figure 19 displays the runtime distribution of AOBB+SMB(i) with hypergraph based pseudo trees over 20 independent runs, for 4 linkage instances. Again, we

[5] Available at http://bioinfo.cs.technion.ac.il/superlink/. The corresponding belief network of the pedigree data was extracted using the export feature of the SUPERLINK 1.6 program.

see that the hypergraph partitioning heuristic generates pseudo trees having average depths almost two times smaller than those of the min-fill based ones. Therefore, using hypergraph based pseudo trees improves sometimes significantly the performance for relatively small i-bounds ($e.g.$, `ped23`, `ped33`).

In the appendix we provide additional empirical results over coding networks (Section A.1), circuit diagnosis networks (Section A.2), problem instances from the Bayesian Networks Repository (Section A.4), and networks from the UAI'06 Evaluation Dataset (Section A.3).

9.4 The Impact of Determinism in Bayesian Networks

In general, when the functions of the graphical model express both hard constraints and general cost functions, it is beneficial to exploit the computational power of the constraints explicitly via constraint propagation [67–70]. For Bayesian networks, the hard constraints are represented by the zero probability tuples of the CPTs. We note that the use of constraint propagation via directional resolution [71] or generalized arc consistency has been explored in [67,68], in the context of variable elimination algorithms where the constraints are also extracted based on the zero probabilities in the Bayesian network. The approach we take for handling the determinism in belief networks is based on the known technique of *unit resolution* for Boolean Satisfiability (SAT). The idea of using unit resolution during search for Bayesian networks was first explored in [69]. A detailed description of the CNF encoding based on the zero probability tuples in the Bayesian network is provided in Appendix (Section A.5).

We evaluated the AND/OR Branch-and-Bound algorithms with static and dynamic mini-bucket heuristics on selected classes of Bayesian networks containing deterministic conditional probability tables ($i.e.$, zero probability tuples). The algorithms exploit the determinism present in the networks by applying unit resolution over the CNF encoding of the zero-probability tuples, at each node in the search tree. They are denoted by `AOBB+SAT+SMB(`i`)` and `AOBB+SAT+DMB(`i`)`, respectively. We used a unit resolution scheme similar to the one employed by `zChaff`, a state-of-the-art SAT solver introduced by [72]. These experiments were performed on a 2.4GHz Pentium IV with 2GB of RAM running Windows XP, and therefore the CPU times reported here may be slower than those in the previous sections.

Table 8 shows the results for experiments with the grid networks from Section 9.3.3. As mentioned earlier, these networks have a high degree of determinism encoded in their CPTs. Specifically, 90% of the probability tables are deterministic, containing only 0 and 1 probability entries.

We observe that `AOBB+SAT+SMB(`i`)` improves significantly over `AOBB+SMB(`i`)`, especially at relatively small i-bounds. For example, on the `90-26-1` grid in-

Table 8
CPU time and nodes visited for solving **deterministic grid networks**. Time limit 1 hour.

min-fill pseudo tree

grid (w*, h) (n, e)	AOBB+SMB(i) AOBB+SAT+SMB(i) AOBB+DMB(i) AOBB+SAT+DMB(i) i=8 time	nodes	i=10 time	nodes	i=12 time	nodes	i=14 time	nodes	i=16 time	nodes	i=18 time	nodes
	0.31	8,080	0.11	2,052	**0.02**	101	0.05	101	0.05	101	0.06	101
90-10-1	0.28	7,909	0.09	2,050	0.05	101	0.06	101	0.06	101	0.06	101
(13, 39)	0.31	344	0.30	241	0.24	101	0.30	101	0.30	101	0.28	101
(100, 0)	0.52	344	0.47	241	0.39	101	0.47	101	0.47	101	0.47	101
	7.84	130,619	6.42	100,696	1.03	17,479	0.34	3,321	0.61	2,938	1.81	3,386
90-14-1	2.36	45,870	2.52	46,064	0.66	11,914	**0.31**	3,286	0.61	2,922	1.78	3,359
(22, 66)	62.17	31,476	25.22	4,137	5.05	397	7.61	211	10.67	199	21.23	198
(196, 0)	33.03	10,135	16.08	3,270	4.92	396	7.72	211	10.88	199	21.64	198
	646.83	10,104,350	164.02	2,600,690	13.14	193,440	2.92	39,825	2.08	23,421	2.92	5,842
90-16-1	121.24	2,209,097	78.97	1,416,247	6.99	121,595	2.25	35,376	**1.84**	22,986	2.84	5,609
(24, 82)	1030.41	462,180	316.77	47,121	75.13	3,227	52.16	719	25.63	260	65.05	260
(256, 0)	841.32	452,923	248.38	37,670	55.86	2,264	49.99	719	25.03	260	64.99	260

	i=10 time	nodes	i=12 time	nodes	i=14 time	nodes	i=16 time	nodes	i=18 time	nodes	i=20 time	nodes
	-	-	-	-	2214.12	24,117,151	1479.15	18,238,983	132.35	1,413,764	135.72	1,308,009
90-24-1	1529.21	18,103,859	2605.56	30,929,553	689.47	9,868,626	738.17	11,100,088	**106.00**	1,282,902	121.67	1,273,738
(33, 111)	-	-	-	-	-	-	884.41	2,739	1223.18	1,228	1634.57	598
(576, 20)	-	-	-	-	-	-	843.79	2,739	1173.48	1,228	1611.74	598
	2217.15	17,899,574	314.88	2,903,489	382.22	3,205,257	8.42	59,055	23.14	165,182	22.22	5,777
90-26-1	233.94	2,527,496	103.56	1,264,309	167.27	1,805,787	**6.20**	43,798	19.36	150,345	22.11	4,935
(36, 113)	1420.21	177,661	-	-	-	-	938.98	2,545	1701.64	1,191	2638.95	691
(676, 40)	1099.87	171,961	1592.53	108,694	1034.26	12,819	862.38	2,545	1583.37	1,191	2478.19	691
	-	-	1125.40	9,445,224	379.14	3,324,942	339.66	3,039,966	147.99	1,358,569	93.63	485,300
90-30-1	754.427	7,050,411	367.41	3,723,781	190.38	2,002,447	164.39	1,734,294	107.95	1,150,182	**70.14**	387,242
(43, 150)	-	-	-	-	-	-	-	-	-	-	-	-
(900, 60)	-	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	462.41	1,549,829
90-34-1	-	-	-	-	-	-	-	-	-	-	**255.08**	981,831
(45, 153)	-	-	-	-	-	-	-	-	-	-	-	-
(1154, 80)	-	-	-	-	-	-	-	-	-	-	-	-
	-	-	2007.47	6,835,745	3589.43	12,321,175	800.72	2,850,393	566.11	2,079,146	368.60	1,038,065
90-38-1	1128.56	5,121,466	410.94	1,972,430	578.54	2,339,244	270.05	1,349,223	278.11	1,249,270	**204.56**	702,806
(47, 163)	-	-	-	-	-	-	-	-	-	-	-	-
(1444, 120)	-	-	-	-	-	-	-	-	-	-	-	-

stance, AOBB+SAT+SMB(10) is 9 times faster than AOBB+SMB(10). As the i-bound increases and the search space is pruned more effectively, the difference between AOBB+SMB(i) and AOBB+SAT+SMB(i) decreases because the heuristics are strong enough to cut the search space significantly. The mini-bucket heuristic already does some level of constraint propagation.

When comparing the AND/OR search algorithms with dynamic mini-bucket heuristics, we see that the difference between AOBB+DMB(i) and AOBB+SAT+DMB(i) is again more pronounced at relatively small i-bounds. For more experiments on deterministic Bayesian networks see Section A.5 in the appendix.

Figure 20 displays the CPU time and number of nodes visited, as a function of

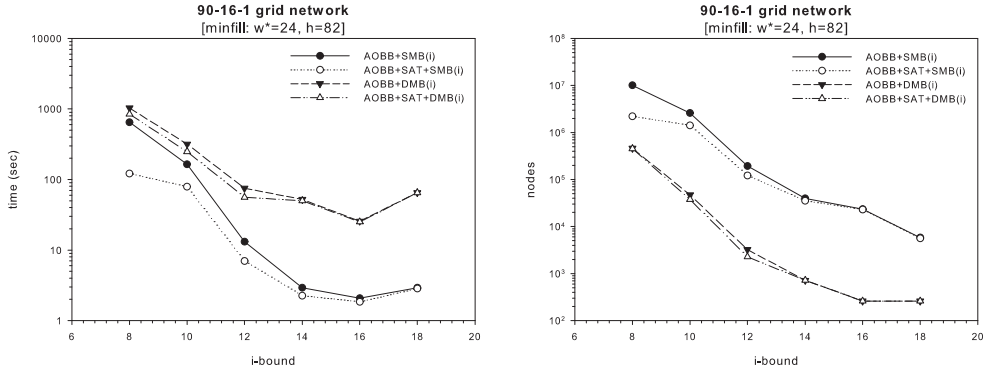


Fig. 20. Comparison of the impact of static and dynamic mini-bucket heuristics on the **`90-16-1` deterministic grid network** from Table 8.

the mini-bucket i-bound, on the `90-16-1` grid network ($i.e.$, corresponding to the third horizontal block from Table 8). We notice again the U-shaped curve of the running time for all algorithms.

9.5 Results for Empirical Evaluation of Weighted CSPs

In this section we focus on both mini-bucket and EDAC heuristics when problems are solved in a static variable ordering. We also evaluate the impact of dynamic variable orderings when using EDAC based heuristics.

9.5.1 SPOT5 Benchmark

SPOT5 benchmark contains a collection of large real scheduling problems for the daily management of Earth observing satellites [10]. These problems can be described as follows:

- Given a set $\mathbf{P}$ of photographs which can be taken the next day from at least one of the three instruments, w.r.t. the satellite trajectory;
- Given, for each photograph, a weight expressing its importance;
- Given a set of imperative constraints: non overlapping and minimal transition time between two successive photographs on the same instrument, limitation on the instantaneous data flow through the satellite telemetry;
- The goal is to find an admissible subset $\mathbf{P}'$ of $\mathbf{P}$ which maximizes the sum of the weights of the photographs in $\mathbf{P}'$ when all imperative constraints are satisfied.

They can be casted as WCSPs by:

- Associating a variable X_i with each photograph $p_i \in \mathbf{P}$;
- Associating with X_i a domain D_i to express the different ways of achieving p_i and adding to D_i a special value, called *rejection* value, to express the possibility

Table 9
CPU time and nodes visited for solving **SPOT5 networks**. Time limit 2 hours.

minfill pseudo tree												
	MBE(i)		MBE(i)		MBE(i)		MBE(i)		MBE(i)		AOEDAC	
spot5	BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		toolbar	
	AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)			
(w*, h)	BB+DMB(i)		BB+DMB(i)		BB+DMB(i)		BB+DMB(i)		BB+DMB(i)			
(n, k, c)	AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)			
	i=4		i=6		i=8		i=12		i=14			
	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
	0.01		0.03		0.34		21.72		147.66		613.79	8,997,894
29	-	-	-	-	-	-	25.69	5,095	148.27	632	4.56	218,846
(14, 42)	8.44	86,058	4.83	45,509	**0.64**	2,738	21.74	246	147.69	481		
(83, 4, 476)	44.42	12,007	131.64	9,713	57.22	541	678.22	507	1758.78	507		
	28.27	14,438	65.91	11,850	53.72	364	630.09	330	1675.74	330		
	0.01		0.11		0.50		28.81		223.14		-	-
42b	-	-	-	-	2154.64	9,655,444	148.11	712,685	228.17	12,255	-	-
(18, 62)	-	-	-	-	1790.76	9,606,846	**131.34**	689,402	223.64	4,189		
(191, 4, 1341)	-	-	-	-	-	-	-	-	-	-		
	-	-	-	-	-	-	-	-	-	-		
	0.01		0.02		0.09		1.25		1.23		31.34	823,326
54	668.77	6,352,998	2.98	27,383	0.59	4,996	1.28	921	1.52	921	**0.31**	21,939
(11, 33)	105.99	1,106,598	1.50	17,757	0.34	3,616	1.28	329	1.27	329		
(68, 4, 283)	1150.54	163,993	52.44	2,469	38.63	921	464.58	921	465.35	921		
	204.11	69,362	27.27	2,188	21.91	329	266.55	329	265.89	329		
	0.01		0.02		0.09		1.09		4.03		255.83	3,260,610
404	-	-	-	-	-	-	4009.57	32,763,223	1827.05	15,265,025	151.11	6,215,135
(19, 42)	413.18	3,969,398	146.05	1,373,846	14.08	144,535	**1.39**	3,273	4.06	367		
(100, 4, 710)	-	-	-	-	-	-	-	-	1964.20	2,015		
	238.97	156,338	272.46	39,144	215.17	5,612	565.06	1,327	167.90	220		
	0.02		0.08		0.31		8.30		35.22		-	-
408b	-	-	-	-	-	-	-	-	-	-	-	-
(24, 59)	-	-	-	-	-	-	682.12	4,784,407	**124.67**	567,407		
(201, 4, 1847)	-	-	-	-	-	-	-	-	-	-		
	-	-	-	-	-	-	-	-	-	-		
	0.01		0.03		0.14		0.39		0.39		-	-
503	-	-	-	-	-	-	1.22	5,229	1.22	5,229	-	-
(9, 39)	-	-	412.63	5,102,299	397.77	4,990,898	**0.44**	641	**0.44**	641		
(144, 4, 639)	-	-	-	-	-	-	690.44	5,229	694.86	5,229		
	-	-	-	-	-	-	64.02	641	64.52	641		
	0.01		0.01		0.12		48.20		372.27		-	-
505b	-	-	-	-	-	-	-	-	-	-	-	-
(16, 98)	-	-	-	-	-	-	-	-	**392.08**	143,371		
(240, 4, 1721)	-	-	-	-	-	-	-	-	-	-		
	-	-	-	-	-	-	-	-	-	-		

of not selecting the photograph p_i;

- Associating with every X_i an unary constraint forbidding the rejection value, with a valuation equal to the weight of p_i;
- Translating as imperative constraints (binary or ternary) the constraints of non overlapping and minimal transition time between two (or three) photographs on the same instrument, and of limitation on the instantaneous data flow. Each imperative constraint is defined over a subset of two or three photographs and for each value combination of its scope variables it associates a high penalty cost (10^6) if the corresponding photographs cannot be taken simultaneously, on the same instrument.

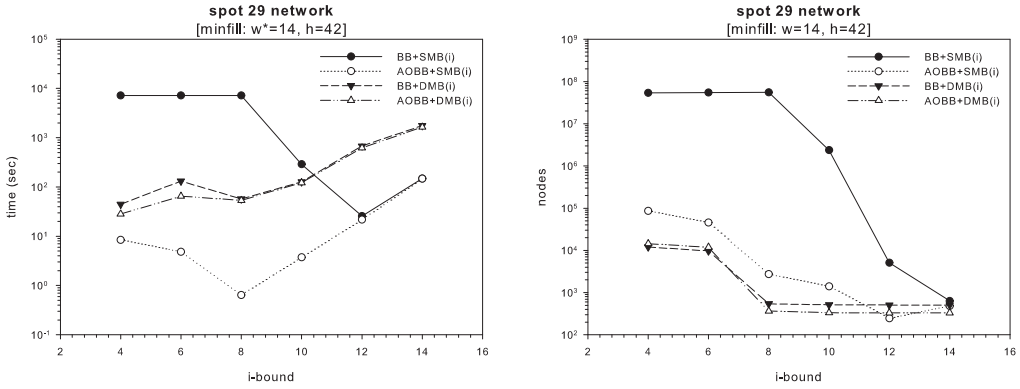


Fig. 21. Comparison of the impact of static and dynamic mini-bucket heuristics on the **29 SPOT5 instance** from Table 9.

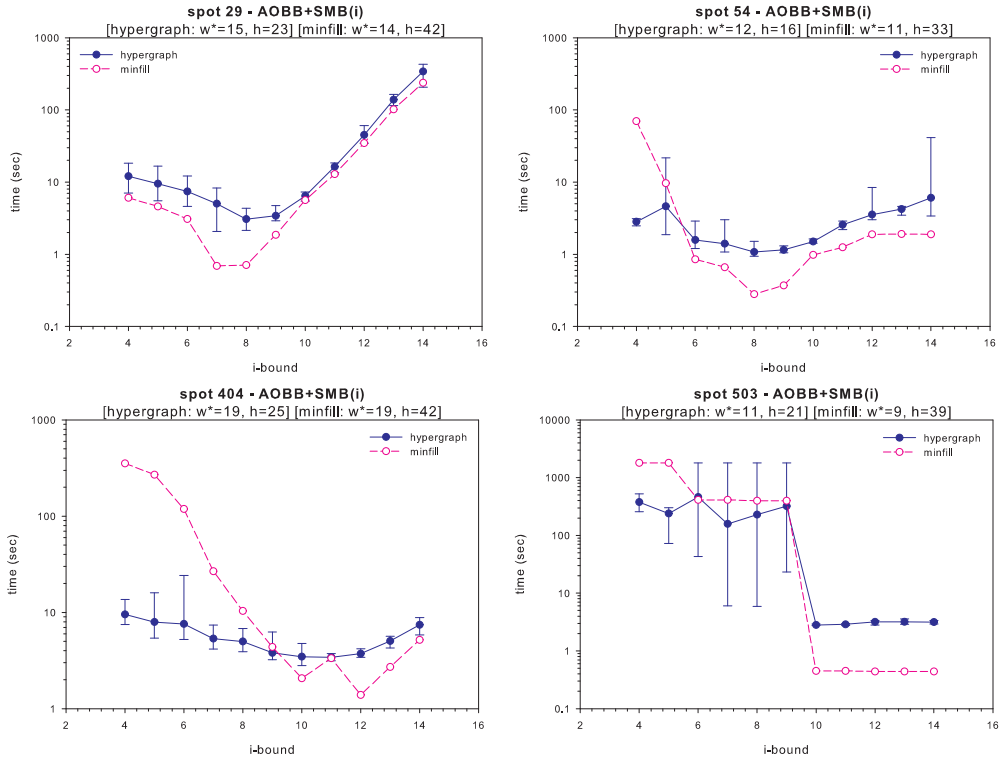


Fig. 22. Min-Fill versus Hypergraph partitioning heuristics for pseudo tree construction. CPU time in seconds for solving **SPOT5 networks** with AOBB+SMB(i).

The task is to compute: $min_{\mathbf{X}} \sum_{i=1}^{r} f_i$, where r is the number of unary, binary and ternary cost functions in the problem.

Table 9 reports the results obtained for experiments with 7 SPOT5 networks, using min-fill pseudo trees. We see that AOBB+SMB(i) is the best performing algorithm on this dataset. The overhead of the dynamic mini-bucket heuristics outweighs search pruning here. For instance, on the 404 network, the difference between AOBB+SMB(12) and BB+SMB(12), in terms of runtime and size of the search space explored, is up to 3 orders of magnitude. The best performances on this do-

main are obtained by AOBB+SMB(i) at relatively large i-bounds which generate very accurate heuristic estimates. For example, AOBB+SMB(14) is the only algorithm able to solve the 505b network. AOEDAC and toolbar were able to solve relatively efficiently only 3 out of the 7 test instances ($e.g.$, 29, 54 and 404).

In Figure 21 we plot the running time and number of nodes visited by AOBB+SMB(i) and AOBB+DMB(i) (resp. BB+SMB(i) and BB+DMB(i)), as a function of the i-bound, on the 29 SPOT5 network ($i.e.$, corresponding to the first horizontal block from Table 9). In this case AOBB+DMB(i) (resp. BB+DMB(i)) is inferior to AOBB+SMB(i) (resp. BB+DMB(i)) across all reported i-bounds. We see that AOBB+SMB(i) achieves the best performance at $i = 8$, whereas AOBB+DMB(i) performs best only at the smallest reported i-bound, namely $i = 4$.

Figure 22 displays the runtime distribution of AOBB+SMB(i) guided by hypergraph based pseudo trees, over 20 independent runs. Hypergraph based trees have far smaller depths than the min-fill ones, and therefore are again able to improve the runtime over min-fill based ones only at relatively small i-bounds ($e.g.$, 404). On average, however, the min-fill pseudo trees generally yield a more robust performance, especially for larger i-bounds of the mini-bucket heuristics ($e.g.$, 503).

9.5.2 ISCAS'89 Benchmark

ISCAS'89 circuits are a common benchmark used in formal verification and diagnosis. For our purpose, we converted each of these circuits into a non-binary WCSP instance by removing flip-flops and buffers in a standard way and creating for each gate a cost function that assigns a high penalty cost (1000) to the forbidden tuples. For each of the input signals we created, in addition, a unary cost function with penalty costs distributed uniformly at random between 1 and 10.

Table 10 shows the results for experiments with 10 circuits, using min-fill pseudo trees. The EDAC based algorithms performed very poorly on this dataset and could not solve any of the test instances within the 30 minute time limit. This was due to the relatively large arity of the constraints, with up to 10 variables in their scope.

AOBB+SMB(i) is superior, especially at relatively large i-bounds. For example, on the s1238 circuit, AOBB+SMB(16) finds the optimal solution in about 26 seconds, whereas BB+SMB(16) as well as AOBB+DMB(16) and BB+DMB(16) exceed the time limit. In this case, AOBB+DMB(i) is competitive at relatively small i-bounds, which cause a relatively small computational overhead. For instance, AOBB+DMB(6) is the best performing algorithm on the s953 network. It is 18 times faster and expands 14 times fewer nodes than BB+DMB(6).

In Figure 23 we show the running time and size of the search space explored by AOBB+SMB(i) and AOBB+DMB(i) (resp. BB+SMB(i) and BB+DMB(i)), as a function of the i-bound, on the s1494 ISCAS'89 circuit ($i.e.$, corresponding to

Table 10
CPU time and nodes visited for solving **ISCAS'89 circuits**. Time limit 1 hour. `AOEDAC` and `toolbar` were not able to solve any of the test instances within the time limit.

minfill pseudo tree												
	MBE(i)		MBE(i)		MBE(i)		MBE(i)		MBE(i)		MBE(i)	
	BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)	
iscas	AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)	
	BB+DMB(i)		BB+DMB(i)		BB+DMB(i)		BB+DMB(i)		BB+DMB(i)		BB+DMB(i)	
(w*, h)	AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)	
(n, k)	i=6		i=8		i=10		i=12		i=14		i=16	
	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
	0.06		0.07		0.09		0.14		0.27		0.89	
c432	-	-	-	-	-	-	13.27	103,088	13.29	102,546	6.79	34,671
(27, 45)	-	-	1373.07	23,355,897	96.29	1,713,265	3.85	76,346	3.89	75,420	**0.97**	1,958
(432, 2)	-	-	104.57	35,073	770.42	107,126	485.61	70,401	125.95	35,502	122.09	35,609
	-	-	3.04	1,578	39.65	34,904	26.26	16,482	9.17	1,070	6.67	692
	0.03		0.04		0.05		0.14		0.36		0.99	
c499	-	-	-	-	-	-	11.71	53,171	9.62	63,177	5.80	24,397
(23, 55)	4.72	117,563	61.48	1,265,425	25.15	526,517	**0.83**	18,851	24.18	486,656	1.80	22,065
(499, 2)	56.49	29,664	141.89	78,830	110.56	56,256	65.42	40,123	132.20	56,002	203.74	76,832
	3.87	10,147	23.31	13,529	15.67	6,101	5.71	1,002	37.34	3,353	87.99	1,736
	0.06		0.07		0.10		0.16		0.49		1.48	
c880	-	-	-	-	-	-	-	-	-	-	816.47	4,953,611
(27, 67)	2284.65	39,448,762	957.25	19,992,512	737.90	15,247,946	275.51	5,835,825	607.43	13,568,696	137.31	2,837,010
(880, 2)	2463.80	321,585	-	-	-	-	2461.68	270,166	3532.50	410,360	2817.47	238,297
	28.43	40,057	809.53	796,699	709.79	569,471	101.88	32,748	232.97	36,187	625.50	20,357
	0.01		0.01		0.03		0.06		0.19		0.46	
s386	3.26	31,903	0.48	5,118	0.50	5,655	0.51	5,108	0.61	4,543	0.86	4,543
(19, 44)	0.12	3,705	**0.07**	2,073	0.19	4,867	0.14	2,699	0.22	1,420	0.49	1,420
(172, 2)	2.92	4,543	3.14	4,543	3.67	4,543	4.46	4,543	5.92	4,543	8.64	4,543
	0.42	1,420	0.65	1,420	1.17	1,420	1.98	1,420	3.44	1,420	6.13	1,420
	0.06		0.07		0.13		0.31		1.00		3.35	
s953	-	-	-	-	-	-	-	-	-	-	-	-
(66, 101)	-	-	1734.71	21,438,706	225.16	3,074,516	-	-	28.40	348,699	7.14	51,441
(440, 2)	110.11	100,180	125.49	103,086	394.09	107,405	466.71	106,825	1412.68	107,063	1094.88	103,383
	6.44	6,885	17.49	7,400	277.03	10,250	350.17	9,164	1294.39	11,164	984.06	8,377
	0.06		0.08		0.15		0.37		1.27		4.51	
s1196	-	-	-	-	-	-	-	-	-	-	-	-
(54, 97)	-	-	-	-	920.11	12,392,442	3146.04	34,576,509	1281.38	15,775,180	269.73	3,318,953
(560, 21)	828.59	217,500	1126.06	216,777	2147.95	207,317	-	-	-	-	-	-
	39.22	26,501	62.99	21,849	147.88	17,524	355.39	15,443	1443.72	13,687	-	-
	0.06		0.09		0.15		0.41		1.25		4.72	
s1238	-	-	-	-	-	-	-	-	-	-	-	-
(59, 94)	2245.60	32,501,292	-	-	-	-	1061.12	18,302,873	821.55	14,213,319	**26.13**	360,788
(540, 2)	2744.88	294,977	1661.09	141,562	1708.45	103,045	-	-	-	-	-	-
	142.51	44,980	288.25	39,493	250.61	21,252	844.40	20,945	1449.22	13,857	-	-
	0.04		0.05		0.08		0.12		0.33		0.94	
s1423	-	-	-	-	-	-	-	-	-	-	167.07	448,044
(24, 54)	25.97	309,520	51.60	648,520	18.23	228,634	**5.03**	68,102	5.50	70,043	7.62	87,483
(748, 2)	-	-	-	-	2056.98	566,007	1969.46	539,925	2056.07	565,423	2156.59	579,511
	57.03	52,996	27.67	26,772	31.98	17,801	38.85	19,719	31.92	3,513	56.80	4,323
	0.06		0.09		0.17		0.45		1.50		5.43	
s1488	-	-	-	-	-	-	20.49	58,330	21.56	58,859	23.59	50,080
(47, 67)	1076.11	13,244,002	4.79	50,613	32.33	430,141	**3.08**	29,729	4.28	33,827	6.63	17,904
(667, 2)	192.51	48,822	204.68	49,417	221.85	49,547	286.90	50,803	495.13	50,803	1205.42	50,803
	11.58	15,025	18.02	15,064	35.61	13,279	94.05	13,762	304.60	13,762	1022.09	13,762
	0.08		0.10		0.18		0.50		1.57		5.66	
s1494	3483.40	11,667,673	94.08	362,002	1600.16	5,437,947	396.38	1,544,960	22.78	66,745	26.81	68,848
(48, 69)	345.91	3,076,992	91.55	833,720	43.87	455,131	343.58	3,207,718	**9.06**	83,318	17.01	124,765
(661, 2)	233.36	55,236	279.75	59,161	270.84	53,969	350.23	53,067	391.96	47,139	1431.41	48,119
	41.40	21,156	64.60	21,743	77.64	18,671	162.70	15,699	232.34	9,706	1260.97	9,913

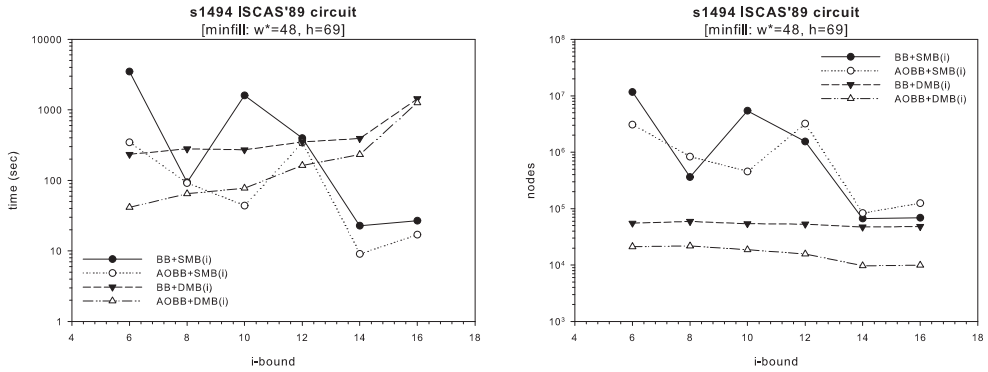


Fig. 23. Comparison of the impact of static and dynamic mini-bucket heuristics on the **s1494 ISCAS'89 circuit** from Table 10.

the last horizontal block from Table 10). We see that the power of the dynamic mini-bucket heuristics is again more prominent for relatively small i-bounds. At larger i-bounds, the static mini-bucket heuristics are cost effective, namely the difference in running time between `AOBB+SMB(`i`)` and `AOBB+DMB(`i`)` (resp. between `BB+SMB(`i`)` and `BB+DMB(`i`)`) is about two orders of magnitude in favor of the former.

Figure 24 depicts the runtime distribution of `AOBB+SMB(`i`)` guided by hypergraph based pseudo trees on the instances: `c499`, `c880`, `s1238` and `s1488`, respectively. In some cases ($e.g.$, `s1238`), using hypergraph pseudo trees improves the runtime up to one order of magnitude, compared with min-fill ones.

9.5.3 Mastermind Games

Each of these networks is a ground instance of a relational Bayesian network that models differing sizes of the popular game of Mastermind. These networks were produced by the PRIMULA System [6] and used in experimental results from [73]. For our purpose, we converted these networks into equivalent WCSP instances by taking the negative log probability of each conditional probability table entry and rounding it to the nearest integer. The resulting WCSP instances are quite large with the number of bi-valued variables n ranging between 1220 and 3692, and containing n unary and ternary cost functions.

Table 11 shows the results for experiments with 6 game instances of increasing difficulty, using min-fill based pseudo trees. As before, `AOBB+SMB(`i`)` offers the overall best performance. For example, `AOBB+SMB(10)` solves the `mm-04-08-03` instance in about 3 seconds, whereas `BB+SMB(10)` exceeds the 1 hour time limit. We did not report results with dynamic mini-bucket heuristics because of the prohibitively large computational overhead associated with relatively large i-bounds. We also note that the EDAC based algorithms were not able to solve any of these

[6] http://www.cs.auc.dk/jaeger/Primula

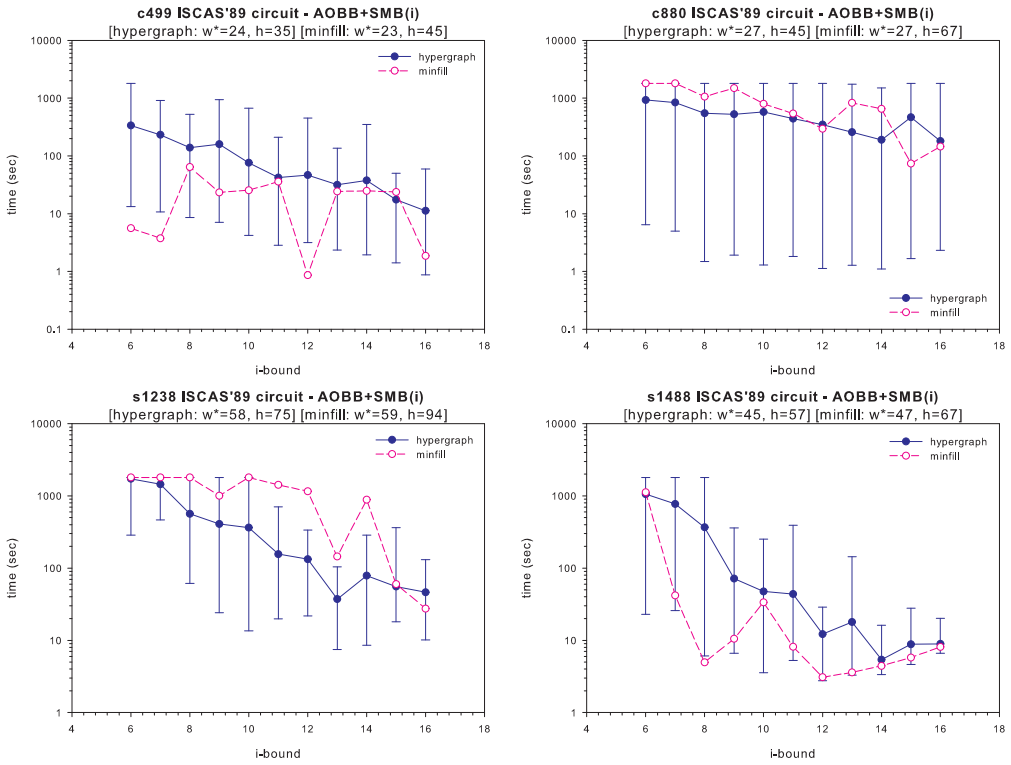


Fig. 24. Min-Fill versus Hypergraph partitioning heuristics for pseudo tree construction. CPU time in seconds for solving **ISCAS'89 networks** with AOBB+SMB(i).

instances within the alloted time bound (not shown in the table).

In Figure 25 we display the runtime distribution of AOBB+SMB(i) guided by hypergraph based pseudo trees over 20 independent runs, for 4 test instances. The spectrum of results is similar to what we observed earlier.

9.6 The Impact of Dynamic Variable Orderings

In this section we evaluate the impact of dynamic variable orderings on AND/OR Branch-and-Bound search guided by local consistency (EDAC) based heuristics. We did not use dynamic variable orderings with dynamic mini-bucket heuristics because of the prohibitively large computational overhead.

SPOT5 Benchmark. Table 12 shows the results for experiments with the 7 SPOT5 networks described in Section 9.5.1. We see that variable ordering can have a tremendous impact on performance. AOEDAC+DSO is the best performing among the EDAC based algorithms, and is able to solve 6 out of 7 test instances. The second best algorithm in this category is DVO+AOEDAC which solves relatively efficiently 3 test networks. This demonstrates the benefit of using variable order-

Table 11
CPU time and nodes visited for solving **Mastermind game instances**. Time limit 1 hour. `AOEDAC` and `toolbar` did not solve any of the test instances within the time limit.

minfill pseudo tree												
mastermind	MBE(i) BB+SMB(i) AOBB+SMB(i)		MBE(i) BB+SMB(i) AOBB+SMB(i)		MBE(i) BB+SMB(i) AOBB+SMB(i)		MBE(i) BB+SMB(i) AOBB+SMB(i)		MBE(i) BB+SMB(i) AOBB+SMB(i)		MBE(i) BB+SMB(i) AOBB+SMB(i)	
(w*, h)	i=8		i=10		i=12		i=14		i=16		i=18	
(n, r, k)	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
mm-03-08-03	0.17		0.22		0.35		0.91		2.83		7.99	
(20, 57)	-	-	-	-	-	-	897.87	873,606	946.84	915,095	738.13	720,764
(1220, 3, 2)	1.16	10,369	**0.88**	7,075	0.93	6,349	1.23	3,830	3.11	3,420	8.25	3,153
mm-03-08-04	0.48		0.60		0.89		2.08		6.45		25.15	
(33, 87)	-	-	-	-	-	-	-	-	-	-	-	-
(2288, 3, 2)	72.37	150,642	66.69	193,805	36.22	71,622	**10.15**	31,177	25.16	63,669	29.27	13,870
mm-04-08-03	0.21		0.27		0.48		1.06		3.54		12.52	
(26, 72)	-	-	-	-	1609.86	1,315,415	1603.71	1,175,430	1157.09	901,309	1924.02	1,451,854
(1418, 3, 2)	8.20	68,929	**3.05**	26,111	4.23	34,445	3.10	17,255	5.29	15,443	13.71	10,570

	i=12		i=14		i=16		i=18		i=20		i=22	
	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
mm-04-08-04	1.19		2.35		6.85		26.47		106.37		395.57	
(39, 103)	-	-	-	-	-	-	-	-	-	-	-	-
(2616, 3, 2)	324.06	744,993	166.67	447,464	310.06	798,507	**64.72**	107,463	192.39	242,865	414.54	62,964
mm-03-08-05	2.14		4.54		11.82		39.01		134.46		497.45	
(41, 111)	-	-	-	-	-	-	-	-	-	-	-	-
(3692, 3, 2)	-	-	-	-	-	-	**835.90**	1,122,008	1162.22	1,185,327	1200.65	1,372,324
mm-10-08-03	1.48		3.78		11.39		34.53		127.55		593.25	
(51, 132)	-	-	-	-	-	-	-	-	-	-	-	-
(2606, 3, 2)	109.50	290,594	128.29	326,662	**64.31**	151,128	74.14	127,130	169.84	133,112	623.83	79,724

ing heuristics within AND/OR Branch-and-Bound search. We also observe that the best performance points highlighted in Table 12 are inferior to those from Table 9 corresponding to `AOBB+SMB(`i`)`. For example, on the `42b` network, the difference in runtime and size of the search space explored between `AOBB+SMB(12)` and `AOEDAC+DSO` is up to one order of magnitude in favor of the former. Similarly, the `505b` network could not be solved by any of the EDAC based algorithms, whereas `AOBB+SMB(14)` finds the optimal solution in about 6 minutes. Notice that `toolbar` is much better than `BBEDAC` in all test cases. This can be explained by a more careful and optimized implementation of EDAC which is available in `toolbar`.

In Figure 26 we show the runtime distribution of `AOEDAC+PVO` with hypergraph pseudo trees on 20 independent runs. In this case, the difference between the minfill and the hypergraph case is dramatic, resulting in up to three orders of magnitude in favor of the latter.

CELAR Benchmark. Radio Link Frequency Assignment Problem (RLFAP) is a communication problem where the goal is to assign frequencies to a set of radio links in such a way that all links may operate together without noticeable interferences [9]. It can be naturally casted as a binary WCSP where each forbidden tuple has an associated penalty cost.

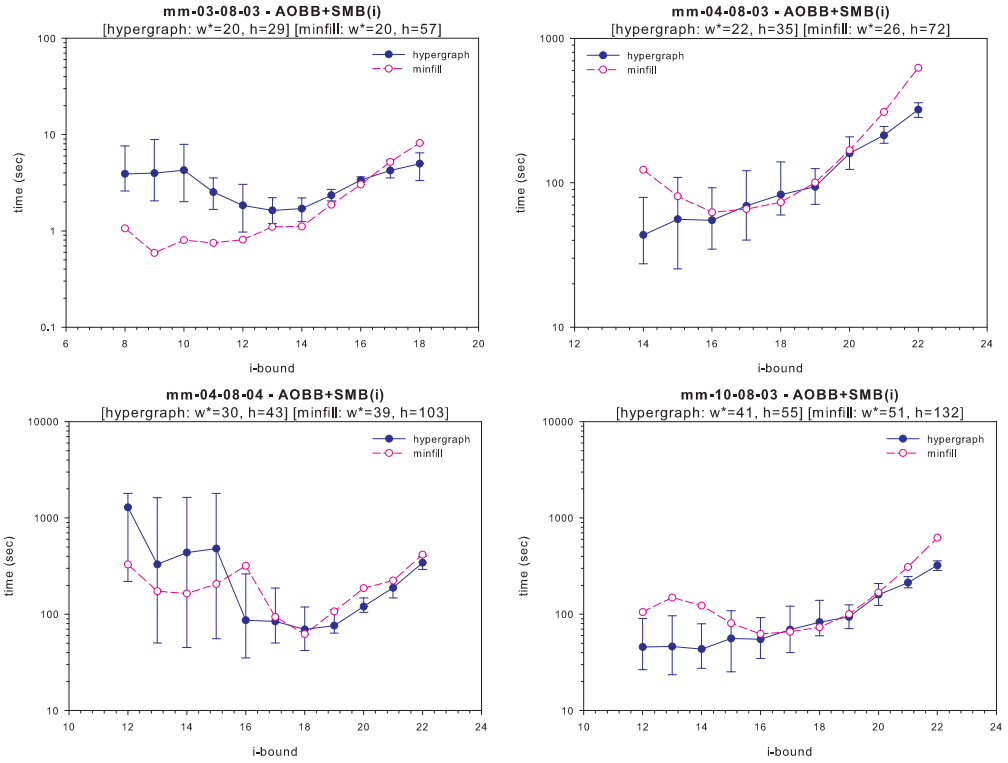


Fig. 25. Min-Fill versus Hypergraph partitioning heuristics for pseudo tree construction. CPU time in seconds for solving **Mastermind networks** with AOBB+SMB(i).

Table 12
CPU time and nodes visited for solving **SPOT5 benchmarks** with EDAC heuristics and dynamic variable orderings. Time limit 2 hours.

minfill pseudo tree									
spot5	n c	w* h		toolbar	BBEDAC	AOEDAC	AOEDAC+PVO	DVO+AOEDAC	AOEDAC+DSO
29	16	7	time	4.56	109.66	613.79	545.43	**0.83**	11.36
	57	8	nodes	218,846	710,122	8,997,894	7,837,447	8,698	92,970
42b	14	9	time	-	-	-	-	-	**6825.4**
	75	9	nodes	-	-	-	-	-	27,698,614
54	14	9	time	0.31	0.97	31.34	9.11	**0.06**	0.75
	75	9	nodes	21,939	8,270	823,326	90,495	688	6,614
404	16	10	time	151.11	2232.89	255.83	152.81	12.09	**1.74**
	89	12	nodes	6,215,135	7,598,995	3,260,610	1,984,747	88,079	14,844
408b	18	10	time	-	-	-	-	-	**747.71**
	106	13	nodes	-	-	-	-	-	2,134,472
503	22	11	time	-	-	-	-	-	**53.72**
	131	15	nodes	-	-	-	-	-	231,480
505b	16	9	time	-	-	-	-	-	-
	70	10	nodes	-	-	-	-	-	-

Table 13 shows detailed results for experiments with CELAR6 and CELAR7 subinstances. We considered only the OR and AND/OR algorithms using EDAC heuristics. The performance of the mini-bucket based algorithms was quite poor on this domain, due to the very low quality of the heuristic estimates resulted from approximating subproblems with very large domains (up to 44 values).

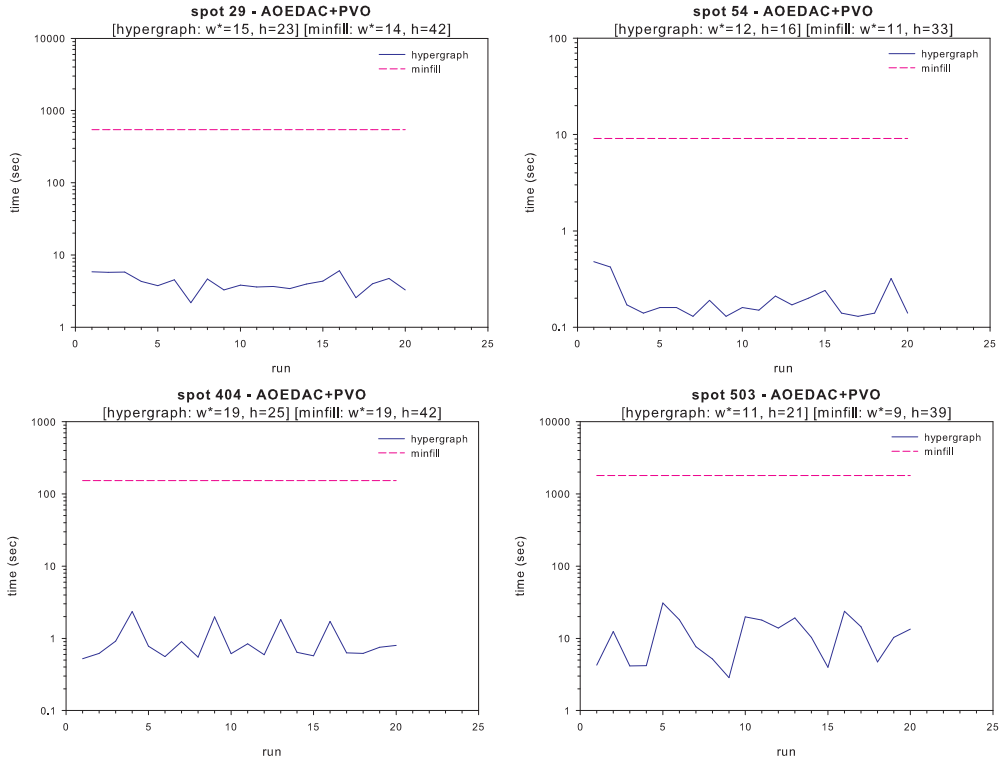


Fig. 26. Min-Fill versus Hypergraph partitioning heuristics for pseudo tree construction. CPU time in seconds for solving **SPOT5 networks** with AOEDAC+PVO.

We observe that `toolbar` is the overall best performing algorithm on this dataset. One reason is that h is close to n, so the AND/OR search is close to OR search. When looking at the AND/OR algorithms we notice that DVO+AOEDAC offers the best performance. On average, the speedups caused by DVO+AOBB over the other algorithms are as follows: 1.9x over AOEDAC, 1.6x over AOEDAC+PVO and 2.5x over BBEDAC. Furthermore, AOEDAC+DSO performs similarly to AOEDAC+PVO indicating that the quality of the dynamic problem decomposition is comparable to the static one.

10 Related Work

The idea of exploiting structural properties of the problem in order to enhance the performance of search algorithms in constraint satisfaction is not new. Freuder and Quinn [2] introduced the concept of pseudo tree arrangement of a constraint graph as a way of capturing independencies between subsets of variables. Subsequently, *pseudo tree search* [2] is conducted over a pseudo tree arrangement of the problem which allows the detection of independent subproblems that are solved separately. More recently, [74] extended pseudo tree search [2] to optimization tasks in order to boost the Russian Doll search [20] for solving Weighted CSPs. Our AND/OR

Table 13
CPU time and nodes visited for solving **CELAR6** and **CELAR7** sub-instances with EDAC heuristics and dynamic variable orderings. Time limit 10 hours.

minfill pseudo tree									
celar	n c	w* h		toolbar	BBEDAC	AOEDAC	AOEDAC+PVO	DVO+AOEDAC	AOEDAC+DSO
celar6-sub0	16	7	time	**0.66**	0.88	1.20	0.79	0.82	0.67
	57	8	nodes	8,952	2,985	2,901	1,565	2,652	1,633
celar6-sub1	14	9	time	**488.58**	5079.28	6693.33	4972.42	4961.16	4999.17
	75	9	nodes	7,521,496	6,381,472	5,558,900	4,376,510	4,420,050	4,326,480
celar6-sub1-24	14	9	time	**47.80**	269.88	319.20	251.11	248.55	252.65
	75	9	nodes	1,028,814	716,746	512,419	446,808	440,238	440,857
celar6-sub2	16	10	time	**1887.40**	6579.99	23896.83	12026.15	6097.33	11323.30
	89	12	nodes	30,223,624	10,941,839	21,750,156	8,380,049	6,700,589	5,584,139
celar6-sub3	18	10	time	**4376.37**	14686.60	32439.00	28251.70	11131.00	28407.40
	106	13	nodes	61,700,735	63,304,285	39,352,900	32,467,100	28,803,649	32,451,800
celar6-sub4-20	22	11	time	**27.76**	1671.55	277.51	415.02	268.57	413.48
	131	15	nodes	167,960	8,970,211	522,981	952,894	893,609	1,256,102
celar7-sub0	16	9	time	**1.11**	4.56	6.20	5.00	4.64	4.71
	70	10	nodes	6,898	9,146	10,248	10,198	9,151	9,761
celar7-sub1	14	9	time	**23.86**	188.11	470.36	239.20	189.15	245.41
	75	9	nodes	134,404	501,145	589,117	329,236	372,790	318,351
celar7-sub1-20	14	9	time	**0.67**	3.49	14.09	3.56	3.30	3.33
	75	9	nodes	10,438	18,959	27,805	15,860	15,637	14,351
celar7-sub2	16	10	time	**627.97**	4822.89	7850.10	5424.98	4727.30	5545.80
	89	11	nodes	1,833,808	4,026,263	7,644,780	3,454,750	3,326,511	2,654,120
celar7-sub3	18	10	time	**6944.96**	-	-	-	-	-
	106	13	nodes	14,754,723					
celar7-sub4-22	22	11	time	3604.47	23882.20	26210.05	7958.44	23166.40	**2999.55**
	129	15	nodes	6,391,923	23,700,235	34,941,835	11,533,163	23,674,049	3,429,708

Branch-and-Bound algorithm is also related to the Branch-and-Bound method proposed by [47] for acyclic AND/OR graphs and game trees.

Dechter's graph-based back-jumping algorithm [75] uses a depth-first (DFS) spanning tree to extract knowledge about dependencies in the graph. The notion of DFS-based search was also used by [76] for a distributed constraint satisfaction algorithm. Bayardo and Miranker [3] reformulated the pseudo tree search algorithm in terms of back-jumping and showed that the depth of a pseudo-tree arrangement is always within a logarithmic factor off the induced width of the graph.

Recursive Conditioning (RC) [58] is based on the divide and conquer paradigm. Rather than instantiating variables to obtain a tree structured network like the cycle cutset scheme, RC instantiates variables with the purpose of breaking the network into independent subproblems, on which it can recurse using the same technique. The computation is driven by a data-structure called *dtree*, which is a full binary tree, the leaves of which correspond to the network CPTs. It can be shown that RC explores an AND/OR space [1]. A pseudo tree can be generated from the static ordering of RC dictated by the dtree. This ensures that whenever RC splits the problem into independent subproblems, the same happens in the AND/OR space.

Backtracking with Tree-Decomposition (BTD) [25] is a memory intensive method for solving constraint satisfaction (or optimization) problems which combines search

techniques with the notion of tree decomposition. This mixed approach can in fact be viewed as searching an AND/OR search space whose backbone pseudo tree is defined by and structured along the tree decomposition. What is defined in [25] as structural goods, that is parts of the search space that would not be visited again as soon as their consistency (or optimal value) is known, corresponds precisely to the decomposition of the AND/OR space at the level of AND nodes, which root independent subproblems. The BTD algorithm is not linear space, it uses substantial caching and may be exponential in the induced width.

11 Summary and Conclusion

The paper investigates the impact of AND/OR search spaces perspective on solving general constraint optimization problems in graphical models. In contrast to the traditional OR search, the new AND/OR search is sensitive the problem's structure. The linear space AND/OR tree search algorithms can be exponentially better (and never worse) than the linear space OR tree search algorithms. Specifically, the size of the AND/OR search tree is exponential in the depth of the guiding pseudo tree rather than the number of variables, as in the OR case.

We introduced a general AND/OR Branch-and-Bound algorithm that explores the AND/OR search tree in a depth-first manner. It can be guided by any heuristic function. We investigated extensively the mini-bucket heuristic and showed that it can prune the search space very effectively. The mini-bucket heuristics can be either pre-compiled (static mini-buckets) or generated dynamically at each node in the search tree (dynamic mini-buckets). They are parameterized by the Mini-Bucket i-bound which allows for a controllable trade-off between heuristic strength and computational overhead. In conjunction with the mini-bucket heuristics we also explored the effectiveness of another class of heuristic lower bounds that is based on exploiting local consistency algorithms for cost functions, in the context of WCSPs.

Since variable ordering can influence dramatically the search performance, we also introduced several ordering schemes that combine the AND/OR decomposition principle with dynamic variable ordering heuristics. There are three approaches to incorporating dynamic orderings into AND/OR Branch-and-Bound search. The first one applies an independent semantic variable ordering heuristic whenever the partial order dictated by the static decomposition principle allows. The second, orthogonal approach gives priority to the semantic variable ordering heuristic and applies problem decomposition as a secondary principle. Since the structure of the problem may change dramatically during search we presented a third approach that uses a dynamic decomposition method coupled with semantic variable ordering heuristics.

We focused our empirical evaluation on two common optimization problems in

graphical models: finding the MPE in Bayesian networks and solving WCSPs. Our results demonstrated conclusively that in many cases the depth-first AND/OR Branch-and-Bound algorithms guided by either mini-bucket or local consistency based heuristics improve dramatically over traditional OR Branch-and-Bound search, especially for relatively weak guiding heuristic estimates. We summarize next the most important aspects reflecting the better performance of the AND/OR algorithms, including the mini-bucket i-bound, dynamic variable orderings, constraint propagation and the quality of the guiding pseudo tree.

- **Impact of the mini-bucket i-bound.** Our results show conclusively that when enough memory is available the static mini-bucket heuristics with relatively large i-bounds are cost effective ($e.g.$, genetic linkage analysis networks from Table 7, Mastermind game instances from Table 11). However, if space is restricted, the dynamic mini-bucket heuristics, which exploit the partial assignment along the search path, appear to be the preferred choice, especially for relatively small i-bounds ($e.g.$, ISCAS'89 networks from Tables 10 and A.2). This is because these heuristics are far more accurate for the same i-bound than the pre-compiled version and the savings in number of nodes explored translate into important time savings.
- **Impact of dynamic variable ordering.** Our dynamic AND/OR search approach was shown to be powerful especially when used in conjunction with local consistency based heuristics. The AND/OR Branch-and-Bound algorithms with EDAC based heuristics and dynamic variable orderings were sometimes able to outperform the Branch-and-Bound counterpart with static variable orderings by two orders of magnitude in terms of running time ($e.g.$, see for example the `503` SPOT5 network from Table 12).
- **Impact of determinism.** When the graphical model contains both deterministic information (hard constraints) as well as general cost functions, we demonstrated that is beneficial to exploit the computational power of the constraints explicitly, via constraint propagation. Our experiments on selected classes of deterministic Bayesian networks showed that enforcing a form of constraint propagation, called unit resolution, over the CNF encoding of the determinism present in the network was able in some cases to render the search space almost backtrack-free ($e.g.$, ISCAS'89 networks from Table A.6). This caused a tremendous reduction in running time for the corresponding AND/OR algorithms ($e.g.$, see for example the `s953` network from Table A.6).
- **Impact of the static variable ordering via the pseudo tree.** The performance of the AND/OR search algorithms is highly influenced by the quality of the guiding pseudo tree. We investigated two heuristics for generating small induced width/depth pseudo trees. The min-fill based pseudo trees usually have small induced width but significantly larger depth, whereas the hypergraph partitioning heuristic produces much smaller depth trees but with larger induced widths. Our experiments demonstrated that the AND/OR algorithms using mini-bucket heuristics benefit, on average, from the min-fill based pseudo trees because the guiding mini-bucket heuristic is sensitive to the induced width size which is ob-

tained for these types of pseudo trees. In some exceptional cases however, the hypergraph partitioning based pseudo trees were able to improve significantly the search performance, especially at relatively small i-bounds ($e.g.$, see for example the `s1238` network from Figure 24), because in those cases the smaller depth guarantees a smaller AND/OR search tree. The picture is reversed for the AND/OR algorithms that enforce local consistency, which is not sensitive to the problem's induced width. Here, the hypergraph based trees were able to improve performance up to 3 orders of magnitude over the min-fill based trees ($e.g.$, SPOT5 networks from Figure 26).

Our current depth-first AND/OR Branch-and-Bound approach leaves room for future improvements which are likely to make it more efficient in practice. For instance, one could incorporate good initial upper bound techniques (using incomplete schemes), which in some cases can allow a best-first performance using depth-first AND/OR Branch-and-Bound search. Our approach for handling the deterministic information present in the graphical model is based on a restricted form of relational arc consistency, namely unit resolution. Therefore, it would be interesting to exploit more powerful constraint propagation schemes such as generalized arc or path consistency. The recent improvement of the Mini-Bucket algorithm, called *Depth-First Mini-Bucket Elimination* [77], could be explored further in the context of AND/OR search in order to enhance the AND/OR Branch-and-Bound guided by dynamic mini-bucket heuristics.

In a subsequent article we will continue the investigation of the AND/OR search space perspective for optimization in graphical models. Our focus will be on memory intensive AND/OR search algorithms that explore an AND/OR *graph*, rather than the tree, by equipping them with a context-based adaptive caching scheme similar to good and no-good recording mechanism as well as recent schemes appearing in Recursive Conditioning [58] and Backtracking with Tree Decompositions [25]. In addition to depth-first we will also explore a *best-first* control strategy. Under conditions of admissibility and monotonicity of the heuristic function, best-first search is known to expand the minimal number of nodes, at the expense of using additional memory [78]. In practice, these savings in number of nodes may often translate into time savings as well.

Acknowledgments

This work was partially supported by the NSF grants IIS-0086529 and IIS-0412854, and by the MURI ONR award N00014-00-1-0617.

References

[1] R. Dechter and R. Mateescu. And/or search spaces for graphical models. *Artificial Intelligence*, 171(1):73–106, 2007.

[2] E. Freuder and M. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *International Joint Conference on Artificial Intelligence (IJCAI-1985)*, pages 1076–1078, 1985.

[3] R. Bayardo and D. Miranker. On the space-time trade-off in solving constraint satisfaction problems. In *International Joint Conference on Artificial Intelligence (IJCAI-1995)*, pages 558–562, 1995.

[4] Rina Dechter and Irina Rish. Mini-buckets: A general scheme for approximating inference. *ACM*, 2(50):107–153, 2003.

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

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

[7] S. Bistarelli, U. Montanari, and F. Rossi. Semiring based constraint solving and optimization. *Journal of ACM*, 44(2):309–315, 1997.

[8] Nils J. Nilsson. *Principles of Artificial Intelligence.* Tioga, 1980.

[9] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. Warners. Radio link frequency assignment. *Constraints*, 4(1):79–89, 1999.

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

[11] S. de Givry, I. Palhiere, Z. Vitezica, and T. Schiex. Mendelian error detection in complex pedigree using weighted constraint satisfaction techniques. In *ICLP Workshop on Constraint Based Methods for Bioinformatics*, 2005.

[12] P. Thbault, S. de Givry, T. Schiex, and C. Gaspin. Combining constraint processing and pattern matching to describe and locate structured motifs in genomic sequences. In *Fifth IJCAI-05 Workshop on Modelling and Solving Problems with Constraints*, 2005.

[13] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *International Joint Conference on Artificial Intelligence (IJCAI-1999)*, pages 542–547, 1999.

[14] Rina Dechter. *Constraint Processing*. MIT Press, 2003.

[15] S. de Givry, J. Larrosa, and T. Schiex. Solving max-sat as weighted csp. In *Principles and Practice of Constraint Programming (CP-2003)*, pages 363–376, 2003.

[16] S. Minton, M.D. Johnston, A.B. Philips, and P. Laired. Solving large scale constraint satisfaction and scheduling problems using heuristic repair methods. In *National Conference on Artificial Intelligence (AAAI-1990)*, pages 17–24, 1990.

[17] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *National Conference on Artificial Intelligence (AAAI-1992)*, pages 440–446, 1992.

[18] R. Wallace. Analysis of heuristic methods for partial constraint satisfaction problems. In *In Principles and Practice of Constraint Programming (CP-1996)*, pages 482–496, 1996.

[19] E.C. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1–3):21–70, 1992.

[20] M. Lemaitre G. Verfaillie and T. Schiex. Russian doll search for solving constraint optimization problems. In *National Conference on Artificial Intelligence (AAAI)*, pages 298–304, 1996.

[21] J. Larrosa and P. Meseguer. Partition-based lower bound for max-csp. In *Principles and Practice of Constraint Programming (CP-1999)*, pages 303–315, 1999.

[22] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted csps. In *National Conference of Artificial Intelligence (AAAI-2003)*, pages 631–637, 2003.

[23] J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.

[24] S. de Givry, F. Heras, J. Larrosa, and M. Zytnicki. Existential arc consistency: getting closer to full arc consistency in weighted csps. In *International Joint Conference in Artificial Intelligence (IJCAI-2005)*, pages 84–89, 2005.

[25] P. Jegou and C. Terrioux. Decomposition and good recording for solving max-csps. In *European Conference on Artificial Intelligence (ECAI 2004)*, pages 196–200, 2004.

[26] P. Dagum and M. Luby. Approximating probabilistic inference in bayesian belief networks is np-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

[27] P. Shenoy and G. Shafer. Propagating belief functions with local computations. *IEEE Expert*, 4(1):43–52, 1986.

[28] F.V. Jensen, S. Lauritzen, and K. Olesen. Bayesian updating in recursive graphical models by local computation. *In Computational Statistics Quarterly*, 4(1):269–282, 1990.

[29] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[30] Y. Peng and J.A. Reggia. A connectionist model for diagnostic problem solving. *IEEE Transactions on Systems, Man and Cybernetics*, 1989.

[31] K. Kask and R. Dechter. Stochastic local search for bayesian networks. In *In Workshop on AI and Statistics (AI-STAT-1999)*, pages 113–122, 1999.

[32] J. Park. Using weighted max-sat engines to solve mpe. In *National Conference of Artificial Intelligence (AAAI-2002)*, pages 682–687, 2002.

[33] F. Hutter, H. Hoos, and T. Stutzle. Efficient stochastic local search for mpe solving. In *International Joint Conference on Artificial Intelligence (IJCAI-2005)*, pages 169–174, 2005.

[34] S.E. Shimony and E. Charniak. A new algorithm for finding map assignments to belief networks. In *Uncertainty in Artificial Intelligence (UAI-1991)*, pages 185–193, 1991.

[35] E. Santos. On the generation of alternative explanations with implications for belief revision. In *Uncertainty in Artificial Intelligence (UAI-1991)*, pages 339–347, 1991.

[36] Z. Li and B. DAmbrosio. An efficient approach for finding the mpe in belief networks. In *Uncertainty in Artificial Intelligence (UAI-1993)*, pages 342–349, 1993.

[37] B. K. Sy. Reasoning mpe to multiply connected belief networks using message-passing. In *National Conference of Artificial Intelligence (AAAI-1992)*, pages 570–576, 1992.

[38] R. Marinescu, K. Kask, and R. Dechter. Systematic vs non-systematic algorithms for solving the mpe task. In *Uncertainty in Artificial Intelligence (UAI-2003)*, pages 394–402, 2003.

[39] J. Larrosa K. Kask, R. Dechter and A. Dechter. Unifying cluster-tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166(1-2):225–275, 2005.

[40] E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[41] G.B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, 1951.

[42] G. Nemhauser and L. Wolsey. *Integer and combinatorial optimization.* Wiley, 1988.

[43] R. Dechter, K. Kask, and J. Larrosa. A general scheme for multiple lower bound computation in constraint optimization. In *Principles and Practice of Constraint Programming (CP)*, pages 346–360, 2001.

[44] H. Bodlaender and J. Gilbert. Approximating treewidth, pathwidth and minimum elimination tree-height. *In Technical Report, Utrecht University*, 1991.

[45] R. Bayardo and D. Miranker. A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. In *National Conference on Artificial Intelligence (AAAI)*, pages 298–304, 1996.

[46] J. Pearl. *Heuristics: Intelligent search strategies for computer problem solving.* Addison-Welsey, 1984.

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

[48] M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2):199–227, 2003.

[49] R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.

[50] C. Bessiere and J-C. Regin. Mac and combined heuristics: two reasons to forsake fc (and cbj) on hard problems. In *Principles and Practice of Constraint Programming (CP-1996)*, pages 61–75, 1996.

[51] D. Brelaz. New method to color the vertices of a graph. *Communications of the ACM*, 4(22):251–256, 1979.

[52] C. Lecoutre F. Boussemart, F. Hemery and L. Sais. Boosting systematic search by weighting constraints. In *European Conference on Artificial Intelligence (ECAI-2004)*, pages 146–150, 2004.

[53] F. Boussemart C. Lecoutre and F. Hemery. Backjump-based techniques versus conflict directed heuristics. In *International Conference on Tools with Artificial Intelligence (ICTAI-2004)*, pages 549–557, 2004.

[54] J. Huang and A. Darwiche. A structure-based variable ordering heuristic. In *International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 1167–1172, 2003.

[55] W. Li and P. van Beek. Guiding real-world sat solving with dynamic hypergraph separator decomposition. In *International Conference on Tools with Artificial Intelligence (ICTAI'04)*, pages 542–548, 2004.

[56] R. Bayardo and J. D. Pehoushek. Counting models using connected components. In *National Conference of Artificial Intelligence (AAAI-2000)*, pages 157–162, 2000.

[57] P. Beame T. Sang and H. Kautz. A dynamic approach to mpe and weighted max-sat. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 549–557, 2007.

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

[59] U. Kjæaerulff. Triangulation of graph-based algorithms giving small total space. *Technical Report, University of Aalborg, Denmark*, 1990.

[60] R.J. McEliece, D.J.C. MacKay, and J.F. Cheng. Turbo decoding as an instance of pearls belief propagation algorithm. *In IEEE Journal of Selected Areas in Communication*, 16(2):140–152, 1998.

[61] F. R. Kschischang and B. H. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *In IEEE Journal of Selected Areas in Communication*, 16(2):219–230, 1998.

[62] D.J.C. MacKay and R.M. Neal. Near shannon limit performance of low density parity check codes. *In Electronic Letters*, 33(1):457–458, 1996.

[63] T. Sang, P. Beame, and H. Kautz. Solving Bayesian networks by weighted model counting. In *National Conference of Artificial Intelligence (AAAI-2005)*, pages 475–482, 2005.

[64] Jurg Ott. *Analysis of Human Genetic Linkage*. The Johns Hopkins University Press, 1999.

[65] M. Fishelson and D. Geiger. Exact genetic linkage computations for general pedigrees. In *International Conference on Intelligent Systems for Molecular Biology*, pages 189–198, 2002.

[66] M. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. *Human Heredity*, 2005.

[67] R. Dechter and D. Larkin. Hybrid processing of beliefs and constraints. In *Uncertainty in Artificial Intelligence (UAI-2001)*, pages 112–119, 2001.

[68] D. Larkin and R. Dechter. Bayesian inference in the presence of determinism. In *Artificial Intelligence and Statistics (AISTAT-2003)*, 2003.

[69] D. Allen and A. Darwiche. New advances in inference using recursive conditioning. In *Uncertainty in Artificial Intelligence (UAI-2003)*, pages 2–10, 2003.

[70] R. Dechter and R. Mateescu. Mixtures of deterministic-probabilistic networks. In *Uncertainty in Artificial Intelligence (UAI)*, pages 120–129, 2004.

[71] I. Rish and R. Dechter. Resolution vs. search: two strategies for sat. *Journal of Automated Reasoning*, 24(1-2):225–275, 2000.

[72] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Design Automation Conference (DAC-2001)*, 2001.

[73] M. Chavira, A. Darwiche, and M. Jaeger. Compiling relational bayesian networks for exact inference. *International Journal of Approximate Reasoning*, 42(1–2):4–20, 2006.

[74] J. Larrosa, P. Meseguer, and M. Sanchez. Pseudo-tree search with soft constraints. In *European Conference on Artificial Intelligence (ECAI-2002)*, pages 131–135, 2002.

[75] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

[76] Z. Collin, R. Dechter, and S. Katz. On the feasibility of distributed constraint satisfaction. In *International Joint Conference on Artificial Intelligence (IJCAI-1991)*, pages 318–324, 1991.

[77] E. Rollon and J. Larrosa. Depth-first mini-bucket elimination. In *Principles and Practice of Constraint Programming (CP)*, pages 563–577, 2005.

[78] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of a*. *ACM*, 32(3):505–536, 1985.

Table A.1
CPU time and nodes visited for solving **random coding networks** with 64 bits, 4 parents per XOR bit and channel noise variance $\sigma^2 \in \{0.22, 0.36\}$. Time limit 5 minutes. The pseudo trees were generated by the min-fill and hypergraph partitioning heuristics.

minfill pseudo tree												
(K, N)	(w*, h)	SamIam	MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=4		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=8		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=12		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=16		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=20	
			time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
			0.02		0.02		0.07		0.68		8.33	
			-	-	19.71	203,028	0.09	184	0.71	153	8.51	153
(64, 128)	(27, 40)	-	287.10	5,052,010	6.58	119,289	**0.08**	152	0.68	129	8.34	129
$\sigma^2 = 0.22$			23.42	9,932	0.43	232	1.43	153	12.76	153	121.90	153
			23.62	20,008	0.35	185	1.37	129	12.77	129	121.12	129
			0.02		0.02		0.07		0.68		8.32	
			-	-	82.60	850,665	1.16	12,190	**0.81**	1,463	8.35	227
(64, 128)	(27, 40)	-	277.41	5,250,380	47.80	834,680	1.23	22,406	0.84	3,096	8.33	160
$\sigma^2 = 0.36$			48.81	19,489	5.38	1,504	5.71	618	15.70	240	123.76	192
			48.71	44,734	5.17	1,864	5.53	512	15.53	164	122.90	144
hypergraph pseudo tree												
			0.32		0.33		0.38		1.02		8.91	
			-	-	24.29	287,699	0.59	2,259	1.06	156	8.97	156
(64, 128)	(27, 34)	-	-	-	4.76	61,426	**0.40**	381	1.03	142	8.92	129
$\sigma^2 = 0.22$			35.71	20,678	0.77	263	1.71	163	12.02	163	107.08	163
			31.46	17,224	0.59	160	1.60	129	11.69	129	102.38	129
			0.32		0.33		0.38		1.05		9.39	
			-	-	113.04	1,391,480	22.26	275,844	1.74	9,039	9.40	295
(64, 128)	(27, 34)	-	-	-	34.73	489,614	1.82	19,040	**1.69**	9,494	9.40	295
$\sigma^2 = 0.36$			92.76	50,006	3.34	1,134	3.67	408	14.80	307	105.92	185
			54.25	26,031	5.55	1,312	7.91	472	12.52	143	105.76	142

A Experiments - Bayesian Networks

A.1 Random Coding Networks

Table A.1 displays the results using min-fill and hypergraph based pseudo trees for solving a classes of random coding networks with $K = 64$ input bits, $P = 4$ parents per XOR bit and channel noise variance $\sigma^2 \in \{0.22, 0.36\}$. The spectrum of results is similar to that observed in Section 9.3.2, namely `AOBB+SMB(`i`)` is slightly better than `BB+SMB(`i`)` only for relatively small i-bounds, while for dynamic mini-bucket heuristics there is no noticeable difference between the OR and AND/OR algorithms, across i-bounds.

Table A.2
CPU time and nodes visited for solving Bayesian networks derived from the **ISCAS'89 circuits**. Time limit 30 minutes.

minfill pseudo tree													
		MBE(i)		MBE(i)		MBE(i)		MBE(i)		MBE(i)		MBE(i)	
		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)	
iscas89		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)	
	SamIam	BB+DMB(i)		BB+DMB(i)		BB+DMB(i)		BB+DMB(i)		BB+DMB(i)		BB+DMB(i)	
(w*, h)		AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)		AOBB+DMB(i)	
(n, d)		i=6		i=8		i=10		i=12		i=14		i=16	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
		0.04		0.05		0.08		0.12		0.26		0.88	
c432		-	-	-	-	-	-	0.29	432	0.42	432	1.04	432
(27, 45)	out	-	-	-	-	605.79	20,751,699	**0.13**	432	0.28	432	0.89	432
(432, 2)		-	-	132.19	21,215	2.23	432	3.44	432	5.85	432	13.46	432
		1422.98	4,438,597	24.03	39,711	1.15	432	2.23	432	4.52	432	12.06	432
		0.02		0.03		0.05		0.14		0.37		1.01	
c499		0.16	499	0.17	499	0.19	499	0.28	499	0.50	499	1.13	499
(23, 55)	139.89	**0.04**	499	0.05	499	0.06	499	0.15	499	0.38	499	1.02	499
(499, 2)		1.09	499	1.32	499	2.00	499	4.01	499	8.92	499	28.35	499
		0.39	499	0.63	499	1.31	499	3.32	499	8.21	499	27.67	499
		0.09		0.09		0.11		0.18		0.51		1.49	
c880		-	-	0.59	881	0.60	881	0.66	881	0.99	881	1.97	881
(27, 67)	out	**0.13**	884	**0.13**	881	0.15	881	0.21	881	0.55	881	1.53	881
(880, 2)		4.49	881	5.82	881	8.07	881	12.78	881	20.99	881	42.16	881
		0.78	881	1.14	881	2.16	881	4.98	881	13.19	881	34.42	881
		0.01		0.02		0.03		0.08		0.20		0.47	
s386		0.10	1,358	0.06	677	0.05	172	0.10	172	0.22	172	0.50	172
(19, 44)	3.66	**0.02**	257	**0.02**	257	0.03	172	0.08	172	0.21	172	0.48	172
(172, 2)		0.15	172	0.21	172	0.42	172	0.78	172	1.56	172	3.13	172
		0.09	172	0.16	172	0.36	172	0.72	172	1.50	172	3.09	172
		0.06		0.07		0.12		0.31		1.01		3.40	
s953		-	-	-	-	-	-	-	-	601.69	4,031,967	449.40	3,075,116
(66, 101)	out	715.60	9,919,295	15.25	238,780	37.11	549,181	22.83	434,481	**2.30**	21,499	4.42	19,117
(440, 2)		27.12	2,737	18.84	912	64.12	1,009	25.28	467	221.17	577	211.70	447
		26.48	2,738	18.30	913	63.44	1,010	24.75	468	220.97	578	211.44	447
		0.07		0.10		0.16		0.39		1.30		4.60	
s1196		-	-	-	-	-	-	-	-	-	-	-	-
(54, 97)	out	21.75	316,875	215.81	3,682,077	4.57	77,205	19.81	320,205	16.64	289,873	9.81	99,935
(560, 2)		2.57	580	4.34	568	49.30	924	126.85	863	582.66	1,008	1413.31	817
		1.20	660	2.59	568	45.90	924	118.16	863	571.79	1,008	1404.57	817
		0.07		0.09		0.17		0.42		1.26		4.74	
s1238		-	-	-	-	272.63	2,078,885	144.85	1,094,713	585.48	4,305,175	38.57	253,706
(59, 94)	out	2.63	57,355	8.32	187,499	2.14	47,340	**1.49**	25,538	2.12	20,689	5.27	13,032
(540, 2)		32.17	5,841	6.59	601	370.26	17,278	52.28	651	120.20	558	353.63	551
		2.04	1,089	4.02	795	17.44	1,824	40.35	849	95.84	744	313.29	737
		0.06		0.06		0.09		0.13		0.35		0.96	
s1423		-	-	-	-	-	-	0.46	762	0.67	749	1.29	749
(24, 54)	107.48	**0.14**	1,986	0.30	5,171	0.32	5,078	0.17	866	0.37	749	0.99	749
(748, 2)		2.95	751	3.37	749	4.05	749	5.50	749	9.62	749	20.82	749
		0.55	751	0.76	749	1.35	749	2.81	749	6.93	749	18.16	749
		0.08		0.10		0.18		0.46		1.50		5.49	
s1488		11.91	92,764	1.65	12,080	2.19	17,410	1.26	6,480	2.17	5,327	5.78	830
(47, 67)	out	11.83	135,563	1.48	17,170	2.29	28,420	1.25	12,285	2.26	12,370	5.53	964
(667, 2)		2.31	670	3.14	670	5.43	668	13.11	667	41.43	667	143.13	667
		0.83	670	1.64	670	3.92	668	11.67	667	40.17	667	142.68	667
		0.07		0.09		0.17		0.49		1.57		5.69	
s1494		8.64	64,629	524.05	3,410,547	130.92	815,326	**4.43**	33,373	43.54	268,421	6.07	2,504
(48, 69)	out	9.63	158,070	28.14	476,874	7.09	118,372	11.87	198,912	2.75	21,137	5.83	3,061
(661, 2)		6.29	873	6.23	711	9.81	681	26.60	680	93.29	686	207.20	667
		4.88	873	4.77	711	8.36	681	25.10	680	91.70	686	205.19	667

A.2 ISCAS'89 Benchmark

ISCAS'89 circuits [7] 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 probabilistic tables for each gate and putting uniform distributions on the input signals.

Table A.2 shows the results for experiments with 10 circuits, using min-fill based pseudo trees. As usual, for each test instance we generated a single MPE query without any evidence. When comparing the algorithms using static mini-bucket heuristics we observe again the superiority of the AND/OR over OR Branch-and-Bound search in almost all test cases, across i-bounds. For instance, on the `c880` circuit, `AOBB+SMB(4)` proves optimality in less than a second, while `BB+SMB(4)` exceeds the 30 minute time limit. Similarly, on the `s953` circuit, `AOBB+SMB(14)` is 300 times faster than `BB+SMB(14)` and explores a search space 180 times smaller. Using the dynamic mini-bucket heuristics does pay off in some test cases. For example, on the `s1196` circuit, `AOBB+DMB(4)` causes a speedup of 2 over `BB+DMB(4)` and 45 over `AOBB+SMB(4)`, while `BB+SMB(4)` exceeds the time limit. The overall impact of the AND/OR algorithms versus the OR ones can be explained by the relatively shallow pseudo trees. In summary, the dynamic mini-bucket heuristics were inferior to the corresponding static ones for large i-bounds, however, smaller i-bound dynamic mini-buckets were overall more cost-effective. Notice that SAMIAM is able to solve only 2 out of 10 test instances.

Figure A.1 shows the runtime distribution of `AOBB+SMB(`i`)` with hypergraph based pseudo trees, over 20 independent runs. We observe again that in several cases ($e.g.$, `s1196`) the hypergraph pseudo trees are able to improve performance with up to 3 orders of magnitude, at relatively small i-bounds.

A.3 UAI'06 Evaluation Dataset

The UAI 2006 Evaluation Dataset [8] contains a collection of random as well as real-world belief networks that were used during the first UAI 2006 Inference Evaluation contest.

Table A.3 shows the results for experiments with 14 networks, using min-fill based pseudo trees. Instances `BN_31` through `BN_41` are random grid networks with deterministic CPTs, while instances `BN_126` through `BN_133` represent random coding networks with 128 input bits, 4 parents per XOR bit and channel noise vari-

[7] Available at http://www.fm.vslib.cz/kes/asic/iscas/

[8] http://ssli.ee.washington.edu/bilmes/uai06InferenceEvaluation

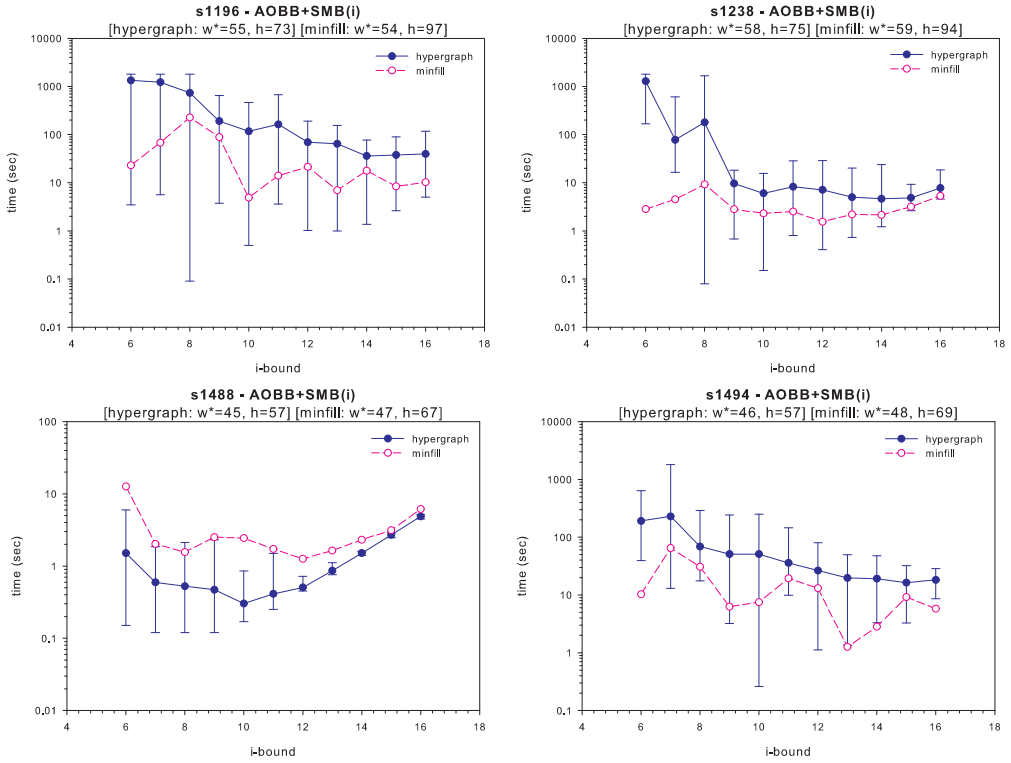


Fig. A.1. Min-Fill versus Hypergraph partitioning heuristics for pseudo tree construction. CPU time in seconds for solving **ISCAS'89 networks** with AOBB+SMB(i).

ance $\sigma^2 = 0.40$. We report only on the Branch-and-Bound algorithms using static mini-buckets. The dynamic mini-buckets were not competitive due to their much higher computational overhead at relatively large i-bounds. We notice again that AOBB+SMB(i) clearly outperforms BB+SMB(i) at all reported i-bounds, especially on the first set of grid networks ($e.g.$, BN_31, ..., BN_41). For instance, on the BN_37, AOBB+SMB(19) finds the MPE solution in about 80 seconds, whereas its OR counterpart BB+SMB(19) exceeds the 30 minute time limit. This is in contrast to what we see on the second set of coding networks ($e.g.$, BN_126, ..., BN_133), where the best performance is offered by the OR algorithm BB+SMB(i).

Figure A.2 shows the runtime distribution of AOBB+SMB(i) with hypergraph pseudo trees, over 20 independent runs. We see that the hypergraph pseudo trees improve slightly the performance compared with min-fill ones.

A.4 Bayesian Network Repository

The Bayesian Network Repository [9] contains a collection of belief networks extracted from various real-life domains which are often used for benchmarking prob-

[9] http://www.cs.huji.ac.il/compbio/Repository/

Table A.3
CPU time and nodes visited for solving **UAI'06 instances**. Time limit 30 minutes.

min-fill pseudo tree											
		MBE(i)		MBE(i)		MBE(i)		MBE(i)		MBE(i)	
bn		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)		BB+SMB(i)	
(w*, h)	SamIam	AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)	
(n, d)		i=17		i=18		i=19		i=20		i=21	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
BN_31		10.31		20.06		34.15		74.17		121.5	
(46, 160)	out	-	-	-	-	-	-	-	-	-	-
(1156, 2)		828.60	4,741,037	1229.64	7,895,304	594.36	3,988,933	646.67	4,293,760	**178.87**	380,470
BN_33		13.84		26.27		48.61		90.35		159.67	
(43, 163)	-	-	-	-	-	-	-	-	-	-	-
(1444, 2)		865.01	3,540,778	**193.79**	685,246	395.99	1,441,245	308.14	1,018,353	230.53	360,880
BN_35		14.27		24.62		47.50		77.66		124.17	
(41, 168)	-	-	-	-	-	-	-	-	-	-	-
(1444, 2)		335.43	1,755,561	390.04	1,954,720	247.73	1,108,708	**191.03**	663,784	234.97	622,551
BN_37		13.82		26.58		44.21		85.17		170.20	
(45, 159)	-	-	-	-	-	-	-	-	-	-	-
(1444, 2)		94.27	428,643	82.15	298,477	**79.99**	183,016	100.41	89,948	196.06	168,957
BN_39		12.95		26.10		51.51		87.16		148.40	
(48, 164)	-	-	-	-	-	-	-	-	-	-	-
(1444, 2)		-	-	-	-	-	-	-	-	**837.58**	3,366,427
BN_41		13.41		23.51		42.01		71.77		125.97	
(49, 164)	-	-	-	-	-	-	-	-	-	-	-
(1444, 2)		125.27	486,844	107.81	364,363	**79.23**	168,340	115.18	195,506	161.10	162,274
BN_126		6.76		13.75		24.62		49.11		98.43	
(54, 70)	-	336.88	2,101,962	871.17	6,677,492	628.26	3,717,027	97.21	350,841	105.54	71,919
(512, 2)		351.91	4,459,174	918.04	10,991,861	126.49	1,333,266	**75.40**	386,490	108.20	150,391
BN_127		7.15		14.26		30.82		56.12		98.82	
(57, 74)	out	-	-	-	-	-	-	-	-	**180.57**	639,878
(512, 2)		-	-	-	-	-	-	-	-	200.14	1,384,957
BN_128		7.77		15.38		28.49		58.08		99.85	
(48, 73)	out	8.19	3,476	15.66	2,645	34.14	36,025	58.54	831	100.29	4,857
(512, 2)		**8.11**	5,587	15.48	1,712	29.64	18,734	58.12	625	100.18	5,823
BN_129		7.39		11.83		24.96		55.28		96.60	
(52, 68)	out					**188.49**	1,605,045	1423.49	11,860,050	343.68	2,049,880
(512, 2)		827.37	11,469,012	-	-	198.24	1,999,591	1796.81	22,855,693	297.90	2,542,057
BN_130		6.29		13.24		22.63		53.68		94.78	
(54, 67)	out	**25.42**	184,439	-	-	918.48	7,317,237	-	-	105.43	110,193
(512, 2)		29.52	348,660	-	-	981.08	10,905,151	-	-	108.25	205,010
BN_131		7.16		13.72		23.36		44.94		82.36	
(48, 72)	out	**21.55**	142,487	47.11	328,560	1216.80	10,249,055	73.25	235,433	-	-
(512, 2)		26.44	296,576	58.78	677,149	1695.44	24,678,072	87.01	673,358	-	-
BN_132		6.16		11.63		22.31		52.78		91.20	
(49, 71)	out	-	-	-	-	-	-	792.42	6,596,296	**644.01**	4,829,396
(512, 2)		-	-	-	-	-	-	886.31	10,251,600	809.86	10,207,347
BN_133		7.60		14.43		27.55		56.54		106.24	
(54, 71)	out	-	-	**24.18**	105,920	46.69	174,274	157.04	932,745	110.05	32,041
(512, 2)		-	-	25.55	169,574	48.53	272,258	184.94	1,859,117	110.87	71,195

abilistic inference algorithms.

Table A.4 displays the results for experiments with 15 belief networks from the repository. We set the time limit to 10 minutes and for each test instance we generated a single MPE query without evidence. We observe again a considerable improvement of the new AND/OR Branch-and-Bound algorithms over the corresponding OR ones. For example, on the `cpcs360b` network, `AOBB+SMB(5)`

Table A.4
CPU time in seconds and number of nodes visited for solving **Bayesian Network Repository** instances. Time limit 10 minutes.

min-fill pseudo tree													
bn (w*, h) (n, d)	SamIam	MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=2		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=3		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=4		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=5		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=6		MBE(i) BB+SMB(i) AOBB+SMB(i) BB+DMB(i) AOBB+DMB(i) i=7	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
		0.01		0.01		0.01		0.01		0.01		0.01	
cpcs54		10.41	141,260	18.26	252,886	0.54	8,072	2.18	30,912	0.62	9,237	0.47	6,955
(14, 23)	0.16	0.66	16,030	0.34	8,621	0.27	6,761	0.39	10,485	0.13	3,672	**0.10**	2,674
(54, 2)		1.99	2,493	1.45	2,214	0.70	1,003	0.53	848	0.34	532	0.33	510
		1.48	2,339	1.16	1,889	0.86	798	0.42	419	0.29	159	0.29	131
		0.09		0.09		0.09		0.09		0.09		0.09	
cpcs360b		72.21	336,720	66.86	317,249	65.05	316,991	61.38	297,313	63.82	314,173	64.59	318,067
(20, 27)	18.91	0.45	10,027	0.44	9,827	0.44	9,809	**0.40**	8,947	0.43	9,771	0.44	9,847
(360, 2)		377.73	308,339	373.48	307,084	373.23	307,083	373.96	307,083	373.34	307,078	373.67	307,078
		4.36	9,383	4.15	9,309	4.06	9,313	4.20	9,285	4.18	9,181	4.40	9,217
		1.58		1.58		1.58		1.58		1.58		1.58	
cpcs422b		57.43	204,209	56.60	203,448	55.61	203,410	54.27	203,410	54.34	203,409	53.98	203,370
(23, 36)	112.78	1.80	3,557	1.78	3,409	1.77	3,409	1.77	3,409	1.78	3,568	**1.76**	3,316
(422, 2)		-	-	-	-	-	-	-	-	-	-	-	-
		54.48	3,140	54.41	3,142	54.98	3,094	54.98	3,029	55.03	2,998	55.14	2,969
		0.01		0.01		0.01		0.01		0.01		0.01	
Insurance		0.14	1,877	0.06	962	69.56	1,749,933	35.70	910,498	0.02	160	0.03	136
(7, 14)	0.08	0.04	977	0.02	453	0.02	411	**0.01**	255	**0.01**	62	0.03	80
(27, 5)		0.13	364	0.03	89	0.03	87	0.08	87	0.16	87	0.52	125
		0.11	299	0.02	36	0.03	33	0.08	33	0.15	33	0.51	62
		0.02		0.02		0.03		0.06		0.19		0.71	
Munin1		-	-	-	-	-	-	-	-	10.16	81,982	11.46	88,836
(12, 28)	out	6.32	102,540	2.79	44,071	**1.32**	22,934	2.00	42,484	1.79	38,669	2.66	48,302
(189, 21)		-	-	256.48	80,411	228.91	66,583	62.08	15,523	65.29	15,513	77.73	15,514
		45.76	84,788	25.46	27,217	18.15	11,230	9.45	2,557	12.30	2,547	25.61	2,548
		0.14		0.16		0.20		0.32		0.46		0.69	
Munin2		-	-	-	-	-	-	-	-	-	-	-	-
(9, 32)	4.30	-	-	-	-	137.72	712,814	30.53	174,333	2.57	15,978	**2.08**	9,961
(1003, 21)		-	-	-	-	-	-	-	-	-	-	-	-
		-	-	-	-	-	-	208.47	13,459	167.27	9,360	122.50	4,806
		0.15		0.15		0.18		0.28		0.40		0.74	
Munin3		-	-	-	-	-	-	-	-	-	-	-	-
(9, 32)	7.28	-	-	-	-	15.20	152,191	1.02	6,440	**0.63**	1,945	0.88	1,180
(1044, 21)		-	-	-	-	-	-	-	-	-	-	-	-
		-	-	345.26	146,866	28.54	2,573	12.11	1,319	10.50	1,180	14.76	1,180
		0.16		0.15		0.19		0.32		0.86		2.22	
Munin4		-	-	-	-	-	-	-	-	-	-	-	-
(9, 35)	26.19	-	-	-	-	-	-	-	-	292.30	3,183,146	**16.83**	125,480
(1041, 21)		-	-	-	-	-	-	-	-	-	-	-	-
		-	-	-	-	-	-	-	-	-	-	-	-
		0.03		0.04		0.04		0.04		0.05		0.06	
Pigs		-	-	0.50	6,060,855	0.48	6,446,055	0.48	5,956,733	0.48	81,982	0.50	88,836
(11, 26)	1.14	-	-	**0.06**	455	**0.06**	455	**0.06**	455	0.07	455	0.09	455
(441, 3)				7.98	1,984	8.58	1,984	8.66	1,984	8.79	1,984	9.09	1,984
				0.31	455	0.39	455	0.49	455	0.63	455	0.93	455
		0.01		0.01		0.01		0.02		0.03		0.05	
Water		78.53	1,658,313	78.02	1,670,307	3.47	53,784	0.34	5,202	0.45	6,769	0.11	973
(10, 15)	3.03	0.67	17,210	1.07	24,527	0.80	19,193	0.14	3,005	0.14	2,658	**0.08**	856
(32, 4)		344.89	697,777	4.39	1,932	0.92	535	0.67	235	0.98	468	2.31	369
		8.49	11,125	3.97	1,622	0.82	193	0.61	153	0.88	113	2.26	136

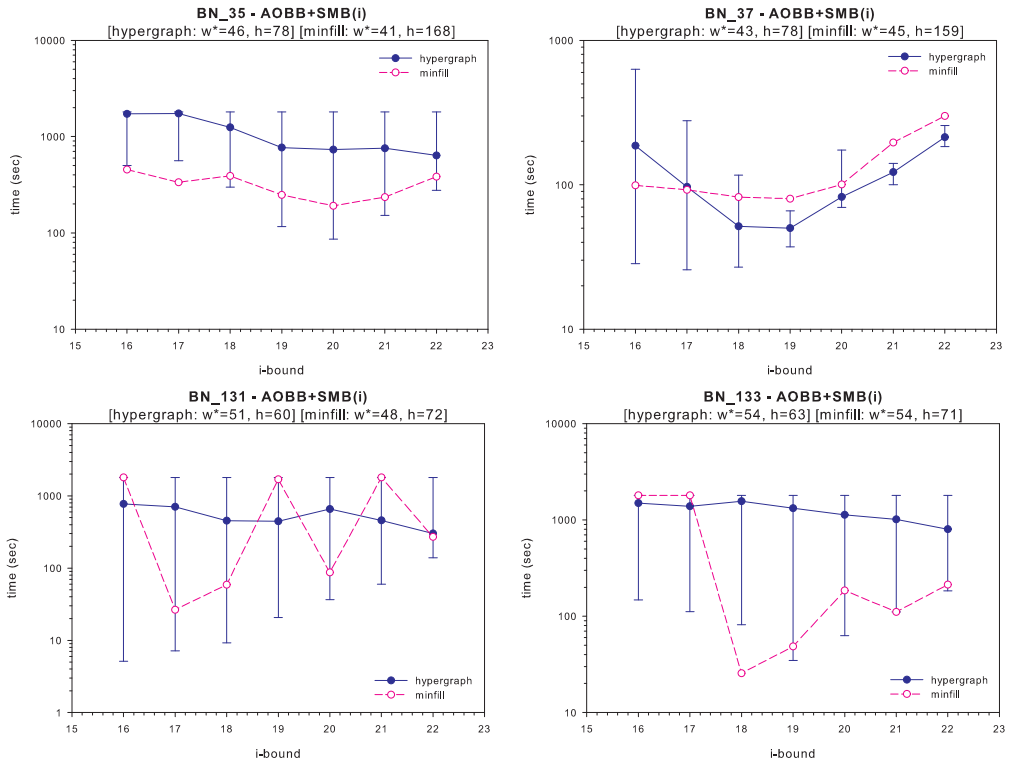


Fig. A.2. Min-Fill versus Hypergraph partitioning heuristics for pseudo tree construction. CPU time in seconds for solving **UAI'06 networks** with AOBB+SMB(i).

causes a CPU speedup of 153 over BB+SMB(5), while exploring a search space 33 times smaller. Similarly, AOBB+DMB(5) is 89 times faster than BB+DMB(5) and expands about 33 times less nodes. Overall, AOBB+SMB(i) is the best performing algorithm for this domain. In particular, for networks with relatively low connectivity and large domain sizes (e.g., Munin networks) the difference between AOBB+SMB(i) and BB+SMB(i) is up to several orders of magnitude in terms of both running time and size of the search space explored.

A.5 The Impact of Determinism in Bayesian Networks

In this section we present a detailed description of the CNF formula which encodes the determinism in the Bayesian network. It is created based on the zero probability table entries, as follows.

SAT Variables: Given a Bayesian network $\mathcal{P} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, the CNF is defined over the multi-valued variables $\{X_1, ..., X_n\}$. Its propositions are L_{X_i,x_i}, where $x_i \in D_i$. The proposition is *true* if X_i is assigned value $x_i \in D_i$ and is *false* otherwise.

SAT Clauses: The CNF is augmented with a collection of 2-CNFs for each variable

Table A.5
Deterministic CPT $P(C|A,B)$

A	B	C	$P(C\|A,B)$	Clauses
1	1	1	1	
1	1	2	0	$(\neg L_{A,1} \vee \neg L_{B,1} \vee \neg L_{C,2})$
1	1	3	0	$(\neg L_{A,1} \vee \neg L_{B,1} \vee \neg L_{C,3})$
1	2	1	0	$(\neg L_{A,1} \vee \neg L_{B,2} \vee \neg L_{C,1})$
1	2	2	1	
1	2	3	0	$(\neg L_{A,1} \vee \neg L_{B,2} \vee \neg L_{C,3})$
2	1	1	.2	
2	1	2	.8	
2	1	3	0	$(\neg L_{A,2} \vee \neg L_{B,1} \vee \neg L_{C,3})$
2	2	1	.7	
2	2	2	.3	
2	2	3	0	$(\neg L_{A,2} \vee \neg L_{B,2} \vee \neg L_{C,3})$

X_i in the network, called *at-most-one* clauses, that forbids the assignments of more than one value to a variable. Formally,

DEFINITION **33 (at-most-one clause)** *Given a variable $X_i \in \mathbf{X}$ with domain $D_i = \{x_{i_1}, ..., x_{i_d}\}$, its corresponding* at-most-one *clauses have the following form:*

$$\neg L_{X_i,x_{i_p}} \vee \neg L_{X_i,x_{i_q}}$$

for every pair $(x_{i_p}, x_{i_q}) \in D_i \times D_i$, where $1 \leq p < q \leq d$.

In addition, we will add to the CNF a set of *at-least-one* clauses to ensure that each variable in the network is assigned at least one value from its domain:

DEFINITION **34 (at-least-one clause)** *Given a variable $X_i \in \mathbf{X}$ with domain $D_i \in \{X_{i_1}, ..., X_{i_d}\}$, its corresponding* at-least-one *clause is of the following form:*

$$L_{X_i,x_{i_1}} \vee L_{X_i,x_{i_2}} ... \vee L_{X_i,x_{i_d}}$$

The remaining clauses are generated from the zero probability tuples in the network's CPTs.

DEFINITION **35 (no-good clauses)** *Given a conditional probability table, denoted by $P(X_i|pa(X_i))$, each entry in the CPT having $P(x_i|x_{pa_i}) = 0$, where $pa(X_i) = \{Y_1, ..., Y_t\}$ are X_i's parents and $x_{pa_i} = (y_1, ..., y_t)$ is their corresponding value assignment, can be translated to a* no-good *clause of the form:*

$$\neg L_{Y_1,y_1} \vee ... \vee \neg L_{Y_t,y_t} \vee \neg L_{X_i,x_i}$$

Example 14 *Consider a belief network over variables* $\{A, B, C\}$ *with domains* $D_A = \{1, 2\}$, $D_B = \{1, 2\}$ *and* $D_C = \{1, 2, 3\}$, *and probability tables:* $P(A)$, $P(B)$ *and* $P(C|A, B)$, *respectively. The deterministic CPT* $P(C|A, B)$ *is given in Table A.5. The corresponding CNF encoding has the following Boolean variables:* $L_{A,1}$, $L_{A,2}$, $L_{B,1}$, $L_{B,2}$, $L_{C,1}$, $L_{C,2}$ *and* $L_{C,3}$. *Variable* $L_{A,1}$ *is* true *if the network variable* A *takes value* 1, *and* false *otherwise.*

To generate the no-good clauses in the knowledge base, we begin by iterating through the parent instantiations of the CPT for variable C. *Whenever a state* $c \in D_C$ *has a probability of 0 we will generate a clause. This clause contains the negative literal* $\neg L_{C,c}$, *as well as the negative literals* $\{\neg L_{A,a}, \neg L_{B,b}\}$ *where* $(A = a, B = b)$ *is the corresponding parent instantiation. These clauses are given in the last column of Table A.5.*

The remaining at-least-one and at-most-one clauses are given in the table below:

at-least-one	at-most-one
$(L_{A,1} \vee L_{A,2})$	$(\neg L_{A,1} \vee \neg L_{A,2})$
$(L_{B,1} \vee L_{B,2})$	$(\neg L_{B,1} \vee \neg L_{B,2})$
$(L_{C,1} \vee L_{C,2} \vee L_{C,3})$	$(\neg L_{C,1} \vee \neg L_{C,2})$
	$(\neg L_{C,1} \vee \neg L_{C,3})$
	$(\neg L_{C,2} \vee L_{C,3})$

Table A.6 displays the results obtained for the 10 ISCAS'89 circuits used in Section A.2. Constraint propagation via unit resolution plays a dramatic role on this domain rendering the search space almost backtrack-free for both static and dynamic mini-bucket heuristics, at all reported i-bounds. For instance, on the `s953` circuit, `AOBB+SAT+SMB(6)` is 3 orders of magnitude faster than `AOBB+SMB(6)` and the search space explored is about 4 orders of magnitude smaller. Similarly, on the same network, `AOBB+SAT+DMB(6)` is 12 times faster than `AOBB+DMB(4)` and explores about 5 times fewer nodes. Notice that in the case of dynamic mini-bucket heuristics, the difference between `AOBB+SAT+DMB(`i`)` and `AOBB+DMB(`i`)` is not too prominent as in the static case, because the heuristic estimates prune the search space quite effectively in this case.

Table A.7 shows the results for experiments with the 12 genetic linkage analysis networks from Section 9.3.4. In this case, we observe that applying unit resolution was not cost effective.

Table A.6
CPU time in seconds and number of nodes visited for solving Bayesian networks derived from **ISCAS'89 circuits** using constraint propagation. Time limit 30 minutes.

min-fill pseudo tree												
iscas89 (w*, h) (n, d)	AOBB+SMB(i) AOBB+SAT+SMB(i) AOBB+DMB(i) AOBB+SAT+DMB(i) i=6		AOBB+SMB(i) AOBB+SAT+SMB(i) AOBB+DMB(i) AOBB+SAT+DMB(i) i=8		AOBB+SMB(i) AOBB+SAT+SMB(i) AOBB+DMB(i) AOBB+SAT+DMB(i) i=10		AOBB+SMB(i) AOBB+SAT+SMB(i) AOBB+DMB(i) AOBB+SAT+DMB(i) i=12		AOBB+SMB(i) AOBB+SAT+SMB(i) AOBB+DMB(i) AOBB+SAT+DMB(i) i=14		AOBB+SMB(i) AOBB+SAT+SMB(i) AOBB+DMB(i) AOBB+SAT+DMB(i) i=16	
	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
	-	-	-	-	1079.59	20,751,699	**0.14**	432	0.24	432	0.59	432
c432	1658.62	37,492,131	873.71	19,423,461	4.52	89,632	0.16	432	0.23	432	0.61	432
(27, 45)	-	-	30.08	39,711	1.03	432	1.75	432	3.20	432	7.74	432
(432, 2)	0.56	434	0.69	433	1.00	432	1.70	432	3.09	432	7.61	432
	0.11	499	**0.09**	499	0.11	499	0.17	499	0.30	499	0.66	499
c499	0.10	499	**0.09**	499	0.11	499	0.17	499	0.30	499	0.69	499
(23, 55)	0.59	499	0.75	499	1.22	499	2.55	499	5.55	499	17.16	499
(499, 2)	0.59	499	0.77	499	1.19	499	2.56	499	5.59	499	17.45	499
	0.22	884	0.23	881	0.23	881	0.28	881	0.48	881	1.06	881
c880	**0.22**	881	**0.22**	881	0.25	881	0.28	881	0.47	881	1.08	881
(27, 67)	1.17	881	1.41	881	2.14	881	4.08	881	9.33	881	22.25	881
(880, 2)	1.19	881	1.35	881	2.25	881	4.03	881	9.67	881	22.92	881
	0.03	257	0.05	257	**0.03**	172	0.06	172	0.14	172	0.31	172
s386	**0.03**	172	**0.03**	172	0.05	172	0.08	172	0.14	172	0.30	172
(19, 44)	0.14	172	0.17	172	0.31	172	0.53	172	1.03	172	2.02	172
(172, 2)	0.11	172	0.16	172	0.30	172	0.52	172	1.02	172	1.98	172
	1019.87	9,919,295	22.50	238,780	54.77	549,181	34.74	434,481	2.61	21,499	3.67	19,117
s953	**0.19**	829	**0.19**	667	0.22	685	0.33	623	0.74	623	2.14	599
(66, 101)	33.03	2,738	16.52	913	48.61	1,010	17.23	468	146.66	578	132.69	447
(440, 2)	2.64	543	4.31	525	12.53	550	14.56	459	98.31	527	105.45	441
	33.00	316,875	343.50	3,682,077	7.22	77,205	31.25	320,205	26.80	289,873	11.61	99,935
s1196	**0.19**	565	0.20	565	0.23	565	0.38	565	0.92	565	2.97	565
(54, 97)	1.59	660	2.50	568	35.47	924	81.63	863	369.30	1,008	886.47	817
(560, 21)	1.17	564	2.00	563	4.61	563	13.05	563	42.02	563	144.13	563
	4.31	57,355	13.73	187,499	3.55	47,340	2.16	25,538	2.41	20,689	3.88	13,032
s1238	**0.20**	771	0.30	2,053	0.34	2,053	0.49	2,037	1.00	2,037	3.09	2,037
(59, 94)	2.66	1,089	3.81	795	13.77	1,824	28.03	849	62.30	744	194.78	737
(540, 2)	1.63	748	2.48	734	7.44	1,655	19.41	802	52.86	736	171.33	735
	0.27	1,986	0.47	5,171	0.48	5,078	**0.22**	866	0.34	749	0.70	749
s1423	0.24	1,903	0.45	4,918	0.45	4,896	**0.22**	860	0.36	749	0.70	749
(24, 54)	0.83	751	0.97	749	1.36	749	2.33	749	4.92	749	11.75	749
(748, 2)	0.81	751	0.97	749	1.37	749	2.34	749	4.92	749	11.89	749
	15.95	135,563	2.09	17,170	3.24	28,420	1.56	12,285	1.64	12,370	3.42	964
s1488	**0.22**	1,115	**0.22**	667	0.27	667	0.44	667	1.05	667	3.36	667
(47, 67)	1.14	670	1.67	670	3.25	668	8.11	667	25.55	667	86.67	667
(667, 2)	0.89	667	1.30	667	2.63	667	6.61	667	20.641	667	69.88	667
	15.13	158,070	43.58	476,874	11.30	118,372	17.48	198,912	3.00	21,137	3.53	3,061
s1494	**0.20**	665	0.22	665	0.25	665	0.45	665	1.11	665	3.42	665
(48, 69)	7.20	873	2.77	711	11.38	681	19.70	680	58.78	686	126.70	667
(661, 2)	1.11	665	1.75	665	3.92	665	10.41	665	31.11	665	101.39	665

Table A.7
CPU time and nodes visited for solving **genetic linkage networks** using constraint propagation via unit resolution. Time limit 3 hours.

min-fill pseudo tree											
		MBE(i)		MBE(i)		MBE(i)		MBE(i)		MBE(i)	
	Superlink	AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)		AOBB+SMB(i)	
pedigree	SamIam	AOBB+SAT+SMB(i)		AOBB+SAT+SMB(i)		AOBB+SAT+SMB(i)		AOBB+SAT+SMB(i)		AOBB+SAT+SMB(i)	
		i=6		i=8		i=10		i=12		i=14	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
ped1		0.05		0.05		0.11		0.31		0.97	
(299, 5)	54.73	24.30	416,326	13.17	206,439	**1.58**	24,361	1.84	25,674	1.89	15,156
(15, 61)	5.44	24.72	414,239	12.97	205,887	1.59	24,361	1.86	25,674	1.89	15,156
ped38		0.12		0.45		2.20		60.97		out	
(582, 5)	**28.36**	-	-	8120.58	85,367,022	-	-	3040.60	35,394,461		
(17, 59)	out	-	-	7663.89	83,808,576	-	-	3094.33	35,394,277		
ped50		0.11		0.74		5.38		37.19		out	
(479, 5)	-	-	-	-	-	476.77	5,566,578	**104.00**	748,792		
(18, 58)	out	-	-	-	-	497.30	5,566,344	107.11	748,792		
		i=10		i=12		i=14		i=16		i=18	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
ped23		0.42		2.33		11.33		274.75		out	
(310, 5)	9146.19	498.05	6,623,197	**15.45**	154,676	16.28	67,456	286.11	117,308		
(27, 71)	out	514.33	6,618,811	15.89	154,666	17.87	67,456	270.05	117,308		
ped37		0.67		5.16		21.53		58.59		out	
(1032, 5)	**64.17**	273.39	3,191,218	1682.09	25,729,009	1096.79	15,598,863	128.16	953,061		
(21, 61)	out	282.83	3,189,847	1674.54	25,280,466	1066.79	15,372,724	131.56	953,061		
		i=12		i=14		i=16		i=18		i=20	
		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
ped18		0.51		1.42		4.59		12.87		19.30	
(1184, 5)	139.06	-	-	2177.81	28,651,103	270.96	2,555,078	100.61	682,175	**20.27**	7,689
(21, 119)	157.05	-	-	2199.44	28,651,103	285.03	2,555,078	103.89	682,175	20.41	7,689
ped20		1.42		5.11		37.53		410.96		out	
(388, 5)	**14.72**	3793.31	54,941,659	1293.76	18,449,393	1259.05	17,810,674	1080.05	9,151,195		
(24, 66)	out	3953.23	54,941,659	1349.51	18,449,393	1301.26	17,810,674	1112.49	9,151,195		
ped25		0.34		0.72		2.27		6.56		29.30	
(994, 5)	-	-	-	-	-	9399.28	111,301,168	3607.82	34,306,937	**2965.60**	28,326,541
(34, 89)	out	-	-	-	-	9690.70	111,301,168	3427.79	34,306,937	2987.50	28,326,541
ped30		0.42		0.83		1.78		5.75		21.30	
(1016, 5)	13095.83	-	-	-	-	-	-	214.10	1,379,131	**91.92**	685,661
(23, 118)	out	-	-	-	-	-	-	225.67	1,379,131	96.16	685,661
ped33		0.58		2.31		7.84		33.44		112.83	
(581, 4)	-	2804.61	34,229,495	737.96	9,114,411	3896.98	50,072,988	**159.50**	1,647,488	2956.47	35,903,215
(37, 165)	out	3051.15	34,218,037	796.58	9,113,615	4290.28	50,071,828	171.31	1,647,488	3216.04	35,884,557
ped39		0.52		2.32		8.41		33.15		81.27	
(1272, 5)	322.14	-	-	-	-	4041.56	52,804,044	386.13	2,171,470	**141.23**	407,280
(23, 94)	out	-	-	-	-	4242.59	52,804,044	405.08	2,171,470	145.03	407,280
ped42		4.20		31.33		206.40		out		out	
(448, 5)	**561.31**	-	-	-	-	-	-				
(25, 76)	out	-	-	-	-	-	-				