Available online at www.sciencedirect.com
ScienceDirect
Artificial Intelligence 171 (2007) 73–106
Artificial Intelligence
www.elsevier.com/locate/artint

AND/OR search spaces for graphical models

Rina Dechter, Robert Mateescu *

Donald Bren School of Information and Computer Science, University of California, Irvine, CA 92697-3425, USA

Received 10 October 2005; received in revised form 8 November 2006; accepted 9 November 2006

Available online 17 January 2007

Abstract

The paper introduces an AND/OR search space perspective for graphical models that include probabilistic networks (directed or undirected) and constraint networks. In contrast to the traditional (OR) search space view, the AND/OR search tree displays some of the independencies present in the graphical model explicitly and may sometimes reduce the search space exponentially. Indeed, most algorithmic advances in search-based constraint processing and probabilistic inference can be viewed as searching an AND/OR search tree or graph. Familiar parameters such as the depth of a spanning tree, treewidth and pathwidth are shown to play a key role in characterizing the effect of AND/OR search graphs vs. the traditional OR search graphs. We compare memory intensive AND/OR graph search with inference methods, and place various existing algorithms within the AND/OR search space.

Keywords: Search; AND/OR search; Decomposition; Graphical models; Bayesian networks; Constraint networks

1. Introduction

Bayesian networks, constraint networks, Markov random fields and influence diagrams, commonly referred to as graphical models, are all languages for knowledge representation that use graphs to capture conditional independencies between variables. These independencies allow both the concise representation of knowledge and the use of efficient graph-based algorithms for query processing. Algorithms for processing graphical models fall into two general types: inference-based and search-based. Inference-based algorithms (e.g., Variable Elimination, Tree Clustering) are better at exploiting the independencies captured by the underlying graphical model. They provide a superior worst case time guarantee, as they are time exponential in the treewidth of the graph. Unfortunately, any method that is time-exponential in the treewidth is also space exponential in the treewidth or separator width and, therefore, not practical for models with large treewidth.

Search-based algorithms (e.g., depth-first branch-and-bound, best-first search) traverse the model's search space where each path represents a partial or full solution. The linear structure of 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. On the other hand, the space requirements of search-based algorithms may be much less severe than those of inference-based algorithms and they can accommo-

* Corresponding author.
E-mail addresses: dechter@ics.uci.edu (R. Dechter), mateescu@ics.uci.edu (R. Mateescu).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2006.11.003

date a wide spectrum of space-bounded algorithms, from linear space to treewidth bounded space. In addition, search methods require only an implicit, generative, specification of the functional relationship (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.

In this paper we propose to use the well-known idea of an AND/OR search space, originally developed for heuristic search [1], to generate search procedures that take advantage of information encoded in the graphical model. We demonstrate how the independencies captured by the graphical model may be used to yield AND/OR search trees that are exponentially smaller than the standard search tree (that can be thought of as an OR tree). Specifically, we show that the size of the AND/OR search tree is bounded exponentially by the depth of a spanning pseudo tree over the graphical model. Subsequently, we move from AND/OR search trees to AND/OR search graphs. Algorithms that explore the search graph involve controlled memory management that allows improving their time-performance by increasing their use of memory. The transition from a search tree to a search graph in AND/OR representations also yields significant savings compared to the same transition in the original OR space. In particular, we show that the size of the minimal AND/OR graph is bounded exponentially by the treewidth, while for OR graphs it is bounded exponentially by the pathwidth.

Our idea of the AND/OR search space is inspired by search advances introduced sporadically in the past three decades for constraint satisfaction and more recently for probabilistic inference and for optimization tasks. Specifically, it resembles pseudo tree rearrangement [2,3], briefly introduced two decades ago, which was adapted subsequently for distributed constraint satisfaction [4,5] and more recently in [6], and was also shown to be related to graph-based backjumping [7]. This work was extended in [8] and more recently applied to optimization tasks [9]. Another version that can be viewed as exploring the AND/OR graphs was presented recently for constraint satisfaction [10] and for optimization [11]. Similar principles were introduced recently for probabilistic inference (in algorithm Recursive Conditioning [12] as well as in Value Elimination [13,14]) and currently provide the backbones of the most advanced SAT solvers [15]. An important contribution of this paper is in showing that all these seemingly different ideas can be cast as simple traversal of AND/OR search spaces. We will also elaborate on the relationship between this scheme and Variable Elimination [16]. We will discuss the relationship with Ordered Binary Decision Diagrams (OBDD) [17], disjunctive Decomposable Negational Normal Forms (d-DNNF) and their extension to arithmetic circuits for Bayesian networks [18,19], as well as with the recent work in [20–23].

The structure of the paper is as follows. Section 2 contains preliminary notations and definitions. Section 3 describes graphical models. Section 4 introduces the AND/OR search tree that can be traversed by a linear space search algorithm. Section 5 presents the AND/OR search graph that can be traversed by memory intensive search algorithms. Section 6 shows how to use the AND/OR graphs to solve a reasoning problem, and gives the AND/OR search algorithm for counting and belief updating. Section 7 is dedicated to a detailed comparison of AND/OR search and other new algorithmic advances in graphical models as well as compilation schemes. Finally, Section 8 provides concluding remarks. All the proofs are given in an appendix at the end.

2. Preliminaries

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 or subsets of variables by uppercase letters (e.g., $X, Y, Z, S, R \ldots$) and values of variables by lower case letters (e.g., x, y, z, s). An assignment $(X_1 = x_1, \ldots, X_n = x_n)$ can be abbreviated as $x = (\langle X_1, x_1\rangle, \ldots, \langle X_n, x_n\rangle)$ or $x = (x_1, \ldots, x_n)$. For a subset of variables Y, D_Y denotes the Cartesian product of the domains of variables in Y. x_Y and $x[Y]$ are both used as the projection of $x = (x_1, \ldots, x_n)$ over a subset Y. We will also denote by $Y = y$ (or y for short) the assignment of values to variables in Y from their respective domains. We denote functions by letters f, g, h etc., and the scope (set of arguments) of the function f by $scope(f)$.

Definition 1 *(functional operators).* Given a function h defined over a subset of variables S, where $X \in S$, functions $(\min_X h)$, $(\max_X h)$, and $(\sum_X h)$ are defined over $U = S - \{X\}$ as follows: For every $U = u$, and denoting by (u, x) the extension of tuple u by assignment $X = x$, $(\min_X h)(u) = \min_x h(u, x)$, $(\max_X h)(u) = \max_x h(u, x)$, and $(\sum_X h)(u) = \sum_x h(u, x)$. Given a set of functions $h_1, \ldots, h_k$ defined over the subsets $S_1, \ldots, S_k$, the product func-

tion, $\Pi_j h_j$, and summation function, $\sum_j h_j$, are defined over $U = \bigcup_j S_j$. For every $U = u$, $(\Pi_j h_j)(u) = \Pi_j h_j(u_{S_j})$, and $(\sum_j h_j)(u) = \sum_j h_j(u_{S_j})$.

Definition 2 *(graph concepts).* A *directed graph* is a pair $G = \{V, E\}$, where $V = \{X_1, \ldots, X_n\}$ is a set of vertices, and $E = \{(X_i, X_j) \mid X_i, X_j \in V\}$ is the set of edges (arcs). If $(X_i, X_j) \in E$, we say that X_i *points to* X_j. The degree of a variable is the number of arcs incident to it. For each variable X_i, $pa(X_i)$ or pa_i, is the set of variables 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, F_i, includes X_i and its parent variables. 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 3 *(induced width).* An *ordered graph* is a pair (G, d) where G is an undirected graph, and $d = X_1, \ldots, X_n$ 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*, $w^*(d)$, is the width of the induced ordered graph obtained as follows: nodes are processed from last to first; when node X 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 4 *(hypergraph).* A *hypergraph* is a pair $H = (X, S)$, where $S = \{S_1, \ldots, S_t\}$ is a set of subsets of V called *hyperedges*.

Definition 5 *(tree decomposition).* A *tree decomposition* of a hypergraph $H = (X, S)$ is a tree $T = (V, E)$ (V is the set of nodes, also called "clusters", and E is the set of edges) together with a labeling function χ that associates with each vertex $v \in V$ a set $\chi(v) \subseteq X$ satisfying:

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

Definition 6 *(treewidth, pathwidth).* The *width* of a tree decomposition of a hypergraph is the size of its largest cluster minus 1 ($\max_v |\chi(v)| - 1$). 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.

It is easy to see that given an induced graph, the set of maximal cliques (also called clusters) provide a tree decomposition of the graph, namely the clusters can be connected in a tree structure that satisfies the running intersection property. It is well known that the induced width of a graph is identical to its treewidth [24]. For various relationships between these and other graph parameters see [25–27].

2.1. AND/OR search graphs

AND/OR search spaces. An AND/OR state space representation of a problem is defined by a 4-tuple $\langle S, O, S_g, s_0 \rangle$. S is a set of states which can be either OR or AND states (the OR states represent 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 $s_0 \in S$. Example problem domains modeled by AND/OR graphs are two-player games, parsing sentences and Tower of Hanoi [1].

The AND/OR state space model induces an explicit AND/OR search *graph*. Each state is a node and its child nodes are those obtained by applicable AND or OR operators. The search graph includes a *start* node. The terminal nodes (having no child nodes) are marked as Solved (S), or Unsolved (U).

A *solution subtree* of an AND/OR search graph G is a subtree which: (1) contains the start node s_0; (2) if n in the subtree 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" (S). AND/OR graphs can have a cost associated with each arc, and the cost of a solution subtree is a function (e.g., sum-cost) of the arcs included in the solution subtree. In this case we may

seek a solution subtree with optimal (maximum or minimum) cost. Other tasks that enumerate all solution subtrees (e.g., counting solutions) can also be defined.

3. Graphical models

Graphical models include constraint networks defined by relations of allowed tuples, (directed or undirected) probabilistic networks, defined by conditional probability tables over subsets of variables, cost networks defined by cost functions and influence diagrams which include both probabilistic functions and cost functions (i.e., utilities) [28]. Each graphical model comes with its typical queries, such as finding a solution, or an optimal one (over constraint networks), finding the most probable assignment or updating the posterior probabilities given evidence, posed over probabilistic networks, or finding optimal solutions for cost networks. The task for influence diagrams is to choose a sequence of actions that maximizes the expected utility. Markov random fields are the undirected counterparts of probabilistic networks. They are defined by a collection of probabilistic functions called potentials, over arbitrary subsets of variables. The framework presented in this paper is applicable across all graphical models that have discrete variables, however we will draw most of our examples from constraint networks and directed probabilistic networks.

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

Definition 7 *(graphical models).* A *graphical model* $\mathcal{R}$ is a 4-tuple, $\mathcal{R} = \langle X, D, F, \otimes \rangle$, where:

(1) $X = \{X_1, \ldots, X_n\}$ is a set of variables;
(2) $D = \{D_1, \ldots, D_n\}$ is the set of their respective finite domains of values;
(3) $F = \{f_1, \ldots, f_r\}$ is a set of real-valued functions each defined over a subset of variables $S_i \subseteq X$, called its scope, and sometimes denoted by $scope(f_i)$;
(4) $\bigotimes_i f_i \in \{\prod_i f_i, \sum_i f_i, \bowtie_i f_i\}$ is a *combination* operator.[1]

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

Next, we introduce the notion of *universal* graphical model which is defined by a single function.

Definition 8 *(universal equivalent graphical model).* Given a graphical model $\mathcal{R} = \langle X, D, F, \otimes \rangle$, the *universal equivalent graphical model* of $\mathcal{R}$ is $u(\mathcal{R}) = \langle X, D, F = \{\bigotimes_{i=1}^{r} f_i\}, \otimes \rangle$.

Two graphical models are *equivalent* if they have the same universal model.

Definition 9 *(cost of a full and a partial assignment).* Given a graphical model $\mathcal{R}$, the *cost of a full assignment* $x = (x_1, \ldots, x_n)$ is defined by $c(x) = \bigotimes_{f \in F} f(x[scope(f)])$. Given a subset of variables $Y \subseteq X$, the *cost of a partial assignment* y is the combination of all the functions whose scopes are included in Y (F_Y) evaluated at the assigned values. Namely, $c(y) = \bigotimes_{f \in F_Y} f(y[scope(f)])$.

We can restrict a graphical model by conditioning on a partial assignment.

Definition 10 *(conditioned graphical model).* Given a graphical model $\mathcal{R} = \langle X, D, F, \otimes \rangle$ and given a partial assignment $Y = y$, $Y \subset X$, the *conditioned graphical model* is $\mathcal{R}|_y = \langle X, D|_y, F|_y, \otimes \rangle$, where $D|_y = \{D_i \in D, X_i \notin Y\}$ and $F|_y = \{f|_{Y=y},\ f \in F$, and $scope(f) \not\subseteq Y\}$.

Consistency. For most graphical models, the functions range has a special value "0" that is *absorbing* relative to the combination operator (e.g., multiplication). Combining anything with "0" yields a "0". The "0" value expresses the notion of inconsistent assignments. It is a primary concept in constraint networks but can also be defined relative to other graphical models that have a "0" element.

[1] The combination operator can also be defined axiomatically [29].

Definition 11 *(consistent partial assignment, solution).* Given a graphical model having a "0" element, a *partial assignment* is *consistent* if its cost is non-zero. A *solution* is a consistent assignment to all the variables.

Flat functions. Each function in a graphical model having a "0" element expresses implicitly a constraint. The *flat* constraint of function f_i is a constraint R_i over its scope that includes all and only the consistent tuples. In this paper, when we talk about a constraint network, we refer also to the flat constraint network that can be extracted from the general graphical model. When all the full assignments are consistent we say that the graphical model is *strictly positive*.

Unless otherwise noted, we assume that functions are expressed in a tabular explicit form, having an entry for every combination of values from the domains of their variables. Therefore, the specification of such functions is exponential in their scope size (the base of the exponent is the maximum domain size). Relations, or clauses, can be expressed as functions as well, associating a value of "0" or "1" for each tuple, depending on whether or not the tuple is in the relation (or satisfies a clause). The combination operator takes a set of functions and generates a new function whose scope is the union of the input functions scopes.

Definition 12 *(primal graph).* The *primal graph* of a graphical model is an undirected graph that has variables as its vertices and an edge connects any two variables that appear in the scope of the same function.

Reasoning problems, queries. There are various queries/tasks that can be posed over graphical models. We refer to all as *reasoning problems*. In general, a reasoning problem is a function from the graphical model to some set of elements, most commonly, the real numbers. We need one more functional operator, *marginalization*, to express most of the common queries.

Definition 13 *(reasoning problem).* A *reasoning problem* over a graphical model is defined by a marginalization operator and a set of subsets. It is therefore a triplet, $\mathcal{P} = \langle \mathcal{R}, \Downarrow_Y, \{Z_1, \ldots, Z_t\}\rangle$, where $\mathcal{R} = \langle X, D, F, \otimes\rangle$ is a graphical model and $Z = \{Z_1, \ldots, Z_t\}$ is a set of subsets of variables of X. If S is the scope of function f and $Y \subseteq X$, $\Downarrow_Y f \in \{\overset{max}{S-Y} f, \overset{min}{S-Y} f, \overset{\prod}{Y} f, \overset{\sum}{S-Y} f\}$, is a *marginalization* operator. $\mathcal{P}$ can be viewed as a vector function over the scopes $Z_1, \ldots, Z_t$. The reasoning problem is to compute $\mathcal{P}_{Z_1,\ldots,Z_t}(\mathcal{R})$:

$$\mathcal{P}_{Z_1,\ldots,Z_t}(\mathcal{R}) = \left(\Downarrow_{Z_1} \bigotimes_{i=1}^{r} f_i, \ldots, \Downarrow_{Z_t} \bigotimes_{i=1}^{r} f_i \right).$$

We will focus primarily on reasoning problems defined by $Z = \emptyset$. The marginalization operator is sometimes called an *elimination* operator because it removes some arguments from the input function's scope. Specifically, $\Downarrow_Y f$ is defined on Y. It therefore removes variables $S - Y$ from f's scope, S. Note that here $\prod$ is the relational projection operator and unlike the rest of the marginalization operators the convention is that it is defined by the set of variables that are *not* eliminated.

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

3.1. Constraint networks

Constraint networks is a framework for formulating real world problems, such as scheduling and design, planning and diagnosis, and many more as a set of constraints between variables. For example, one approach to formulating a scheduling problem as a constraint satisfaction problem (CSP) is to create a variable for each resource and time slice. Values of variables would be the tasks that need to be scheduled. Assigning a task to a particular variable (corresponding to a resource at some time slice) means that this resource starts executing the given task at the specified time. Various physical constraints (such as that a given job takes a certain amount of time to execute, or that a task can be executed at most once) can be modeled as constraints between variables. The *constraint satisfaction task* is to find an assignment of values to all the variables that does not violate any constraints, or else to conclude that the problem is inconsistent. Other tasks are finding all solutions and counting the solutions.

Definition 14 *(constraint network).* A *constraint network (CN)* is defined by a 4-tuple, $\langle X, D, C, \bowtie \rangle$, where X is a set of variables $X = \{X_1, \ldots, X_n\}$, associated with a set of discrete-valued domains, $D = \{D_1, \ldots, D_n\}$, and a set of constraints $C = \{C_1, \ldots, C_r\}$. 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 $S_i \subseteq X$. The relation denotes all compatible tuples of D_{S_i} allowed by the constraint. The combination operator, $\bowtie$, is join. The primal graph of a constraint network is called a *constraint graph*. A *solution* is an assignment of values to all the variables $x = (x_1, \ldots, x_n)$, $x_i \in D_i$, such that $\forall C_i \in C$, $x_{S_i} \in R_i$. The constraint network represents its set of solutions, $\bowtie_i C_i$.

Constraint satisfaction is a reasoning problem $\mathcal{P} = \langle \mathcal{R}, \Pi, Z = \emptyset \rangle$, where $\mathcal{R} = \langle X, D, C, \bowtie \rangle$ is a constraint network, and the marginalization operator is the projection operator Π. Namely, for constraint satisfaction $Z = \emptyset$, and $\Downarrow_Y$ is Π_Y. So the task is to find $\Downarrow_\emptyset \bigotimes_i f_i = \Pi_X \overset{\bowtie}{}_i f_i$ which corresponds to enumerating all solutions. When the combination operator is a product over the cost-based representation of the relations, and the marginalization operator is logical summation we get 1 if the constraint problem has a solution and "0" otherwise. For *counting*, the marginalization operator is summation and $Z = \emptyset$ too.

An immediate extension of constraint networks are *cost networks* where the set of functions are real-valued cost functions, and the primary task is optimization.

Definition 15 *(cost network, combinatorial optimization).* A *cost network* is defined by a 4-tuple, $\langle X, D, C, \sum \rangle$, where X is a set of variables $X = \{X_1, \ldots, X_n\}$, associated with a set of discrete-valued domains, $D = \{D_1, \ldots, D_n\}$, and a set of cost functions $C = \{C_1, \ldots, C_r\}$. Each C_i is a real-valued function defined on a subset of variables $S_i \subseteq X$. The combination operator, is $\sum$. The reasoning problem is to find a minimum or maximum cost solution which is expressed via the marginalization operator of maximization or minimization, and $Z = \emptyset$.

A task such as MAX-CSP: finding a solution that satisfies the maximum number of constraints (when the problem is inconsistent), can be defined by treating each relation as a cost function that assigns "0" to consistent tuples and "1" otherwise. Then the combination operator is summation and the marginalization operator is minimization. Namely, the task is to find $\Downarrow_\emptyset \bigotimes_i f_i = \min_X \sum_i f_i$.

3.2. Propositional satisfiability

A special case of a CSP is the *propositional satisfiability problem* (SAT). A formula φ in *conjunctive normal form* (CNF) is a conjunction of *clauses* $\alpha_1, \ldots, \alpha_t$ where a clause is a disjunction of *literals* (propositions or their negations). For example, $\alpha = (P \vee \neg Q \vee \neg R)$ is a clause, where P, Q and R are propositions, and P, $\neg Q$ and $\neg R$ are literals. The SAT problem requires deciding whether a given CNF theory has a *model*, i.e., a truth-assignment to its propositions that does not violate any clause.

Propositional satisfiability (SAT) can be defined as a CSP, where propositions correspond to variables, domains are $\{0, 1\}$, and constraints are represented by clauses, for example the clause $(\neg A \vee B)$ is the relation (or function) over its propositional variables that allows all tuples over (A, B) except $(A = 1, B = 0)$.

3.3. Belief networks

Belief networks [30] provide a formalism for reasoning about partial beliefs under conditions of uncertainty. They are defined by a directed acyclic graph over vertices representing random 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 16 (*belief networks*). A *belief network (BN)* is a graphical model $\mathcal{P} = \langle X, D, P_G, \prod \rangle$, where $X = \{X_1, \ldots, X_n\}$ is a set of variables over multi-valued domains $D = \{D_1, \ldots, D_n\}$. Given a directed acyclic graph G over X as nodes, $P_G = \{P_i\}$, where $P_i = \{P(X_i | pa(X_i))\}$ are conditional probability tables (CPTs for short) associated with each X_i, where $pa(X_i)$ are the parents of X_i in the acyclic graph G. A belief network represents a

probability distribution over X, $P(x_1,\ldots,x_n)=\prod_{i=1}^{n} P(x_i|x_{pa(X_i)})$. An evidence set e is an instantiated subset of variables.

When formulated as a graphical model, functions in F 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 $\bigotimes_j=\prod_j$. The primal graph of a belief network is called a moral graph. It connects any two variables appearing in the same CPT.

Definition 17 *(belief updating).* Given a belief network and evidence e, the *belief updating* task is to compute the posterior marginal probability of variable X_i, conditioned on the evidence. Namely,

$$Bel(X_i=x_i)=\alpha \sum_{\{(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)|E=e,X_i=x_i\}} \prod_{k=1}^{n} P(x_k,e|x_{pa_k}),$$

where α is a normalization constant. In this case, the marginalization operator is $\Downarrow_Y=\sum_{X-Y}$, and $Z_i=\{X_i\}$. Namely, $\forall X_i, \Downarrow_{X_i}\bigotimes_k f_k=\sum_{\{X-X_i|X_i=x_i\}}\prod_k P_k$. The query of finding the probability of the evidence is defined by $Z=\emptyset$.

Definition 18 *(most probable explanation).* 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 $(x_1^o,\ldots,x_n^o)$ such that

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

As a reasoning problem, an MPE task is to find $\Downarrow_\emptyset \bigotimes_i f_i=\max_X\prod_i P_i$. Namely, the marginalization operator is max and $Z=\emptyset$.

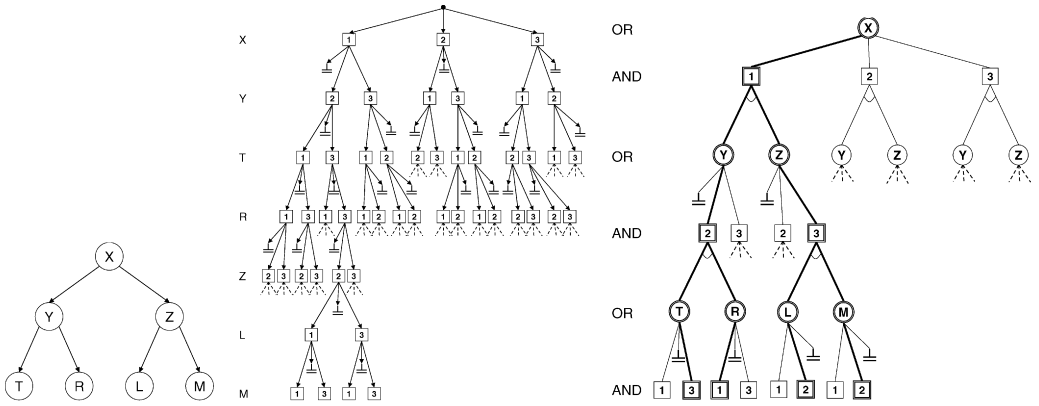
Markov networks are graphical models very similar to belief networks. The only difference is that the set of probabilistic functions P_i, called potentials, can be defined over any subset of variables. An important reasoning task for Markov networks is to find the partition function which is defined by the marginalization operator of summation, where $Z=\emptyset$.

4. AND/OR search trees for graphical models

We will next present the AND/OR search space for a general *graphical model* starting with an example of a constraint network.

Example 19. Consider the simple tree graphical model (i.e., the primal graph is a tree) in Fig. 1(a), over domains $\{1,2,3\}$, which represents a graph-coloring problem. Namely, each node should be assigned a value such that adjacent nodes have different values. Once variable X is assigned the value 1, the search space it roots can be decomposed into two independent subproblems, one that is rooted at Y and one that is rooted at Z, both of which need to be solved independently. Indeed, given $X=1$, the two search subspaces do not interact. The same decomposition can be associated with the other assignments to X, $\langle X,2\rangle$ and $\langle X,3\rangle$. Applying the decomposition along the tree (in Fig. 1(a) yields the AND/OR search tree in Fig. 1(c). In the AND/OR space a full assignment to all the variables is not a path but a subtree. For comparison, the traditional *OR* search tree is depicted in Fig. 1(b). Clearly, the size of the AND/OR search space is smaller than that of the regular OR space. The OR search space has $3\cdot 2^7$ nodes while the AND/OR has $3\cdot 2^5$ (compare 1(b) with 1(c)). If k is the domain size, a balanced binary tree with n nodes has an OR search tree of size $O(k^n)$. The AND/OR search tree, whose pseudo tree has depth $O(\log_2 n)$, has size $O((2k)^{\log_2 n})=O(n\cdot k^{\log_2 n})=O(n^{1+\log_2 k})$. When $k=2$, this becomes $O(n^2)$.

The AND/OR space is not restricted to tree graphical models. It only has to be guided by a *backbone* tree which spans the original primal graph of the graphical model in a particular way. We will define the AND/OR search space relative to a depth-first search tree (DFS tree) of the primal graph first, and will generalize to a broader class of backbone spanning trees subsequently. For completeness sake we define *DFS spanning tree*, next.



(a) A constraint tree (b) OR search tree (c) AND/OR search tree with one of its solution subtrees

Fig. 1. OR vs. AND/OR search trees; note the connector for AND arcs.

Definition 20 *(DFS spanning tree).* Given a DFS traversal ordering of an undirected graph $G = (V, E)$, $d = X_1, \ldots, X_n$, the *DFS spanning tree* $\mathcal{T}$ of G is defined as the tree rooted at the first node, X_1, which includes only the traversed arcs of G. Namely, $\mathcal{T} = (V, E')$, where $E' = \{(X_i, X_j) \mid X_j \text{traversed from } X_i\}$.

We are now ready to define the notion of AND/OR search tree for a graphical model.

Definition 21 *(AND/OR search tree).* Given a graphical model $\mathcal{R} = \langle X, D, F, \otimes \rangle$, its primal graph G and a backbone DFS 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 the value assignments in the domains of the variables. The structure of the AND/OR search tree is based on the underlying backbone tree $\mathcal{T}$. The root of the AND/OR search tree is an OR node labeled by 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 $\pi_n(x_i)$, respectively. The assignment sequence along path π_n, denoted $asgn(\pi_n)$ is the set of value assignments associated with the sequence of AND nodes along π_n:

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

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

(1) An OR node, n, labeled by X_i has a child AND node, m, labeled $\langle X_i, x_i \rangle$ iff $\langle X_i, x_i \rangle$ is consistent with the assignment $asgn(\pi_n)$. Consistency is defined relative to the flat constraints.
(2) An AND node m, labeled $\langle X_i, x_i \rangle$ has a child OR node r labeled Y, iff Y is a child of X in the backbone tree $\mathcal{T}$. Each OR arc, emanating from an OR to an AND node is associated with a weight to be defined shortly (see Definition 26).

Clearly, if a node n is labeled X_i (OR node) or x_i (AND node), $var(\pi_n)$ is the set of variables mentioned on the path from the root to X_i in the backbone tree, denoted by $path_{\mathcal{T}}(X_i)$.[2]

[2] When the AND/OR tree is extended to dynamic variable orderings the set of variables along different paths may vary.

A solution subtree is defined in the usual way:

Definition 22 *(solution subtree).* A *solution subtree* of an AND/OR search tree contains the root node. For every OR nodes it contains one of its child nodes and for each of its AND nodes it contains all its child nodes, and all its leaf nodes are consistent.

Example 23. In the example of Fig. 1(a), $\mathcal{T}$ is the DFS tree which is the tree rooted at X, and accordingly the root OR node of the AND/OR tree in 1(c) is X. Its child nodes are labeled $\langle X, 1\rangle$, $\langle X, 2\rangle$, $\langle X, 3\rangle$ (only the values are noted in the figure), which are AND nodes. From each of these AND nodes emanate two OR nodes, Y and Z, since these are the child nodes of X in the DFS tree of 1(a). The descendants of Y along the path from the root, ($\langle X, 1\rangle$), are $\langle Y, 2\rangle$ and $\langle Y, 3\rangle$ only, since $\langle Y, 1\rangle$ is inconsistent with $\langle X, 1\rangle$. In the next level, from each node $\langle Y, y\rangle$ emanate OR nodes labeled T and R and from $\langle Z, z\rangle$ emanate nodes labeled L and M as dictated by the DFS tree. In 1(c) a solution tree is highlighted.

4.1. Weights of OR-AND arcs

The arcs in AND/OR trees are associated with weights w that are defined based on the graphical model's functions and combination operator. The simplest case is that of constraint networks.

Definition 24 *(arc weight for constraint networks).* Given an AND/OR tree $S_{\mathcal{T}}(\mathcal{R})$ of a constraint network $\mathcal{R}$, each terminal node is assumed to have a single, dummy, outgoing arc. The outgoing arc of a terminal AND node always has the weight "1" (namely it is consistent and thus solved). An outgoing arc of a terminal OR node has weight "0", (there is no consistent value assignment). The weight of any internal OR to AND arc is "1". The arcs from AND to OR nodes have no weight.

We next define arc weights for any graphical model using the notion of buckets of functions.

Definition 25 *(buckets relative to a backbone tree).* Given a graphical model $\mathcal{R} = \langle X, D, F, \otimes\rangle$ and a backbone 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,

$$B_{\mathcal{T}}(X_i) = \big\{f \in F \mid X_i \in scope(f) \text{ and } scope(f) \subseteq path_{\mathcal{T}}(X_i)\big\}.$$

Definition 26 *(OR-to-AND weights).* Given an AND/OR tree $S_{\mathcal{T}}(\mathcal{R})$, of a graphical model $\mathcal{R}$, the weight $w_{(n,m)}(X_i, x_i)$ 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, $w_{(n,m)}(X_i, x_i) = \bigotimes_{f \in B_{\mathcal{T}}(X_i)} f(asgn(\pi_m)[scope(f)])$.

Definition 27 *(weight of a solution subtree).* Given a weighted AND/OR tree $S_{\mathcal{T}}(\mathcal{R})$, of a graphical model $\mathcal{R}$, and given a solution subtree t having OR-to-AND set of arcs $arcs(t)$, the weight of t is defined by $w(t) = \bigotimes_{e \in arcs(t)} w(e)$.

Example 28. Fig. 2 shows a belief network, a DFS tree that drives its weighted AND/OR search tree, and a portion of the AND/OR search tree with the appropriate weights on the arcs expressed symbolically. In this case the bucket of E contains the function $P(E|A, B)$, and the bucket of C contains two functions, $P(C|A)$ and $P(D|B, C)$. Note that $P(D|B, C)$ belongs neither to the bucket of B nor to the bucket of D, but it is contained in the bucket of C, which is the last variable in its scope to be instantiated in a path from the root of the tree. We see indeed that the weights on the arcs from the OR node E and any of its AND value assignments include only the instantiated function $P(E|A, B)$, while the weights on the arcs connecting C to its AND child nodes are the products of the two functions in its bucket instantiated appropriately. Fig. 3 shows a constraint network with four relations, a backbone DFS tree and a portion of the AND/OR search tree with weights on the arcs. Note that the complex weights would reduce to "0" and "1" in this case. However, since we use the convention that arcs appear in the search tree only if they represent a consistent extension of a partial solution, we will not see arcs having zero weights.

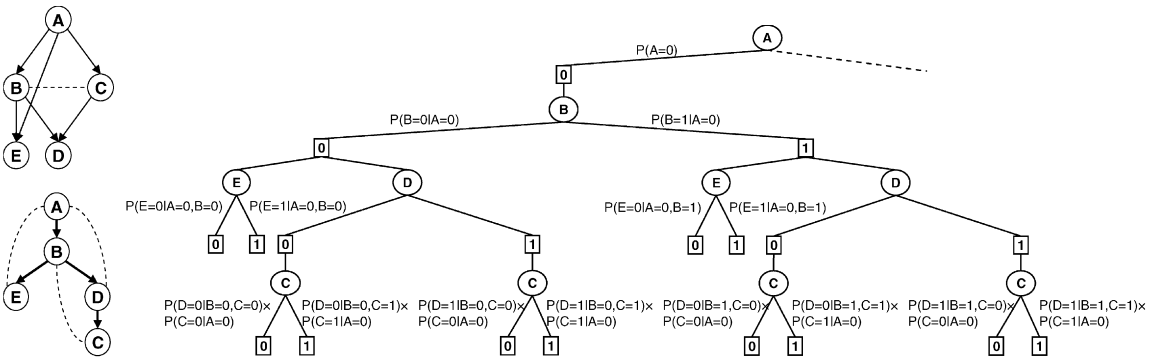


Fig. 2. Arc weights for probabilistic networks.

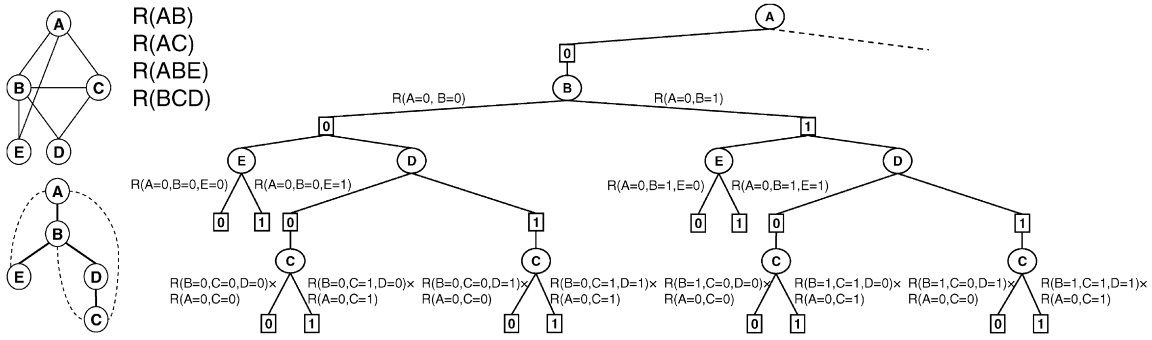


Fig. 3. Arc weights for constraint networks.

4.2. Properties of AND/OR search tree

Any DFS tree $\mathcal{T}$ of a graph G has the property that the arcs of G which are not in $\mathcal{T}$ are back-arcs. Namely, they connect a node to one of its ancestors in the backbone tree. This ensures that the scope of each function in F will be fully assigned on some path in $\mathcal{T}$, a property that is essential for the validity of the AND/OR search tree.

Theorem 29 *(correctness). Given a graphical model $\mathcal{R}$ having a primal graph G and a DFS spanning tree $\mathcal{T}$ of G, its weighted AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$ is sound and complete, namely*: 1) *there is a one-to-one correspondence between solution subtrees of $S_{\mathcal{T}}(\mathcal{R})$ and solutions of $\mathcal{R}$*; 2) *the weight of any solution tree equals the cost of the full solution it denotes*; *namely, if t is a solution tree of $S_{\mathcal{T}}(\mathcal{R})$ which denotes a solution $x = (x_1, \ldots, x_n)$ then $c(x) = w(t)$.*

The virtue of an AND/OR search tree representation is that its size may be far smaller than the traditional OR search tree. The size of an AND/OR search tree depends on the depth of its backbone DFS tree $\mathcal{T}$. Therefore, DFS trees of smaller depth should be preferred to drive the AND/OR search *tree*. An AND/OR search tree becomes an OR search tree when its DFS tree is a chain.

Theorem 30 *(size bounds of AND/OR search tree). Given a graphical model $\mathcal{R}$, with domains size bounded by k, and a DFS spanning tree $\mathcal{T}$ having depth m and l leaves, the size of its AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$ is* $O(l \cdot k^m)$ (*and therefore also* $O(nk^m)$ *and* $O((bk)^m)$ *when b bounds the branching degree of $\mathcal{T}$ and n is the number of variables*). *In contrast the size of its OR search tree along any ordering is* $O(k^n)$. *The above bounds are tight and realizable for fully consistent graphical models. Namely, one whose all full assignments are consistent.*

Table 1 demonstrates the size saving of AND/OR vs. OR search spaces for 5 random networks having 20 bivalued variables, 18 CPTs with 2 parents per child and 2 root nodes, when all the assignments are consistent (remember that this is the case when the probability distribution is strictly positive). The size of the OR space is the full binary tree

Table 1
OR vs. AND/OR search size, 20 nodes

Treewidth	Height	OR space		AND/OR space		
		Time (sec.)	Nodes	Time (sec.)	AND nodes	OR nodes
5	10	3.154	2,097,151	0.03	10,494	5,247
4	9	3.135	2,097,151	0.01	5,102	2,551
5	10	3.124	2,097,151	0.03	8,926	4,463
5	10	3.125	2,097,151	0.02	7,806	3,903
6	9	3.124	2,097,151	0.02	6,318	3,159

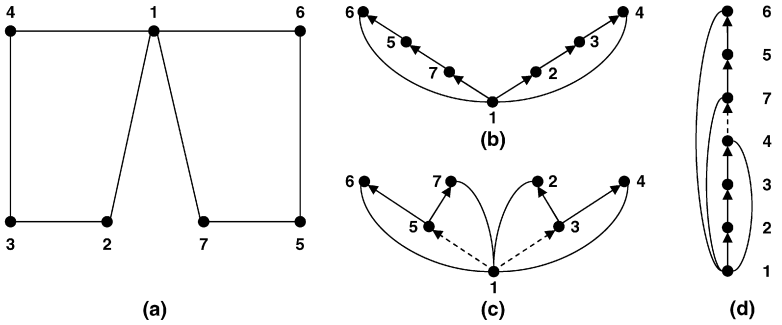


Fig. 4. (a) A graph; (b) a DFS tree $\mathcal{T}_1$; (c) a pseudo tree $\mathcal{T}_2$; (d) a chain pseudo tree $\mathcal{T}_3$.

of depth 20. The size of the full AND/OR space varies based on the backbone DFS tree. We can give a better analytic bound on the search space size by spelling out the depth m_i of each leaf node L_i in $\mathcal{T}$.

Proposition 31. *Given a graphical model $\mathcal{R}$, with domains size bounded by k, and a backbone spanning tree $\mathcal{T}$ having $L = \{L_1, \ldots, L_l\}$ leaves, where depth of leaf L_i is m_i, then the size of its full AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$ is $O(\sum_{k=1}^{l} k^{m_i})$. Alternatively, we can use the exact domain sizes for each variable yielding an even more accurate expression $O(\sum_{L_k \in L} \Pi_{\{X_j | X_j \in path_{\mathcal{T}}(L_k)\}} |D(X_j)|)$.*

4.3. From DFS trees to pseudo trees

There is a larger class of trees that can be used as backbones for AND/OR search trees, called *pseudo trees* [2]. They have the above mentioned back-arc property.

Definition 32 *(pseudo tree, extended graph).* Given an undirected graph $G = (V, E)$, a directed rooted tree $\mathcal{T} = (V, E')$ defined on all its nodes is a *pseudo tree* if any arc of G which is not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node in $\mathcal{T}$ to an ancestor in $\mathcal{T}$. The arcs in E' may not all be included in E. Given a pseudo tree $\mathcal{T}$ of G, the *extended graph* of G relative to $\mathcal{T}$ is defined as $G^{\mathcal{T}} = (V, E \cup E')$.

Clearly, any DFS tree and any chain of a graph are pseudo trees.

Example 33. Consider the graph G displayed in Fig. 4(a). Ordering $d_1 = (1, 2, 3, 4, 7, 5, 6)$ is a DFS ordering of a DFS tree $\mathcal{T}_1$ having the smallest DFS tree depth of 3 (Fig. 4(b)). The tree $\mathcal{T}_2$ in Fig. 4(c) is a pseudo tree and has a tree depth of 2 only. The two tree-arcs $(1, 3)$ and $(1, 5)$ are not in G. Tree $\mathcal{T}_3$ in Fig. 4(d), is a chain. The extended graphs $G^{\mathcal{T}_1}$, $G^{\mathcal{T}_2}$ and $G^{\mathcal{T}_3}$ are presented in Fig. 4(b), (c), (d) when we ignore directionality and include the dotted arcs.

It is easy to see that the weighted AND/OR search tree is well defined when the backbone trees is a pseudo tree. Namely, the properties of soundness and completeness hold and the size bounds are extendable.

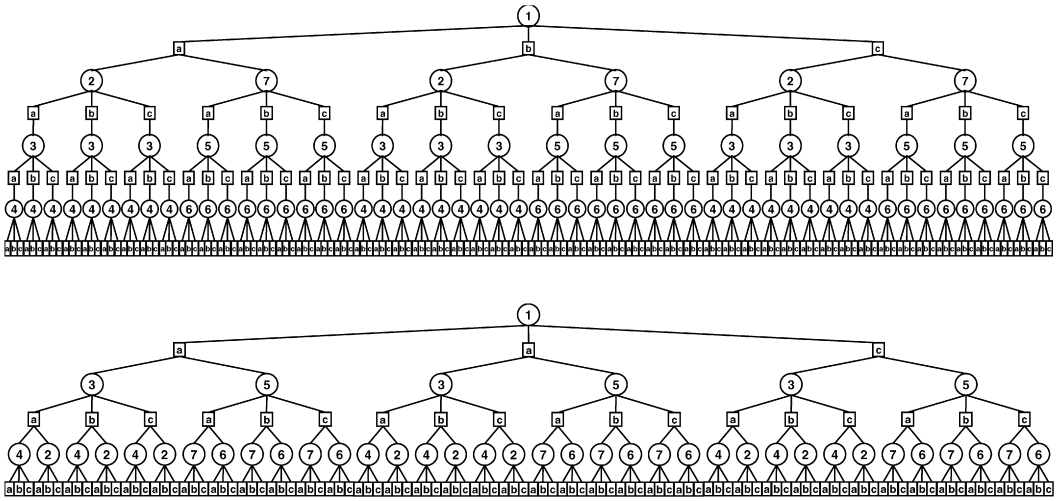


Fig. 5. AND/OR search tree along pseudo trees $\mathcal{T}_1$ and $\mathcal{T}_2$.

Theorem 34 *(properties of AND/OR search trees). Given a graphical model $\mathcal{R}$ and a backbone pseudo tree $\mathcal{T}$, its weighted 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.*

Example 35. Fig. 5 shows the AND/OR search trees along the pseudo trees $\mathcal{T}_1$ and $\mathcal{T}_2$ from Fig. 4. Here the domains of the variables are $\{a, b, c\}$ and the constraints are universal. The AND/OR search tree based on $\mathcal{T}_2$ is smaller, because $\mathcal{T}_2$ has a smaller depth than $\mathcal{T}_1$.

Finding good pseudo trees. Finding a pseudo tree or a DFS tree of minimal depth is known to be NP-complete. However various greedy heuristics are available. For example, pseudo trees can be obtained by generating a heuristically good induced graph along an ordering d and then traversing the induced graph depth-first, breaking ties in favor of earlier variables [8]. For more information see [31,32].

The definition of buckets relative to a backbone tree extends to pseudo trees as well, and this allows the definitions of weights for an AND/OR tree based on a pseudo tree. Next we define the notion of a *bucket tree* and show that it corresponds to a pseudo tree. This relationship will be used to make additional connections between various graph parameters.

Definition 36 *(bucket tree [33]).* Given a graphical model, its primal graph G and an ordering d, the *bucket tree* of G along d is defined as follows. Let G_d^* be the induced graph of G along d. Each variable X has an associated *bucket*, denoted by B_X, that contains X and its earlier neighbors in the induced graph G_d^* (similar to Definition 25). The nodes of the bucket tree are the n buckets. Each node B_X points to B_Y (B_Y is the parent of B_X) if Y is the latest earlier neighbor of X in G_d^*.

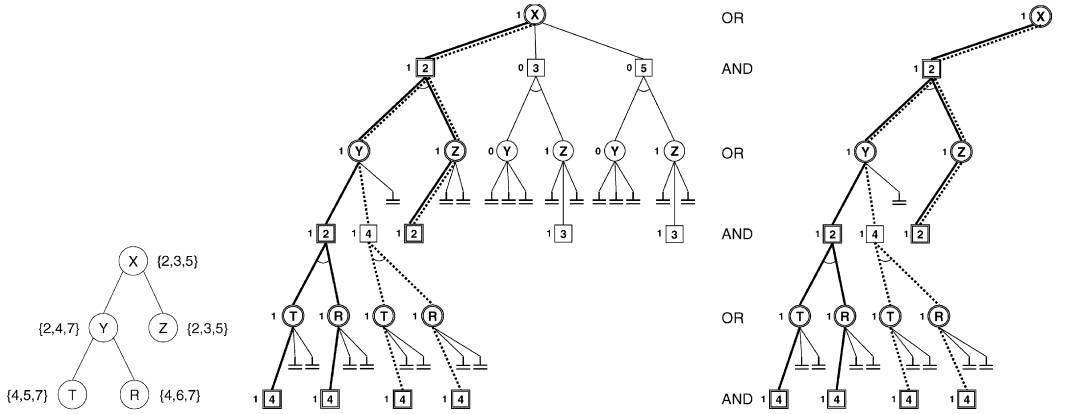
The following relationship between the treewidth and the depth of pseudo trees is known [8,26]. Given a *tree decomposition* of a primal graph G having n nodes, whose treewidth is w^*, there exists a pseudo tree $\mathcal{T}$ of G whose depth, m, satisfies: $m \leqslant w^* \cdot \log n$. It can also be shown that any bucket tree [33] yields a pseudo tree and that a min-depth bucket tree yields min-depth pseudo trees. The depth of a bucket tree was also called *elimination depth* in [26].

In summary,

Proposition 37. [8,26] *The minimal depth m over all pseudo trees satisfies $m \leqslant w^* \cdot \log n$, where w^* is the treewidth of the primal graph of the graphical model.*

Table 2
Average depth of pseudo trees vs. DFS trees; 100 instances of each random model

Model (DAG)	Width	Pseudo tree depth	DFS tree depth
$(N = 50, P = 2, C = 48)$	9.5	16.82	36.03
$(N = 50, P = 3, C = 47)$	16.1	23.34	40.60
$(N = 50, P = 4, C = 46)$	20.9	28.31	43.19
$(N = 100, P = 2, C = 98)$	18.3	27.59	72.36
$(N = 100, P = 3, C = 97)$	31.0	41.12	80.47
$(N = 100, P = 4, C = 96)$	40.3	50.53	86.54



(a) A constraint tree (b) Search tree (c) Backtrack-free search tree

Fig. 6. AND/OR search tree and backtrack-free tree.

Therefore,

Theorem 38. *A graphical model that has treewidth w^* has an AND/OR search tree whose size is* $O(n \cdot k^{(w^* \cdot \log n)})$, *where k bounds the domain size and n is the number of variables.*

For illustration, Table 2 shows the effect of DFS spanning trees against pseudo trees, both generated using brute-force heuristics over randomly generated graphs, where N is the number of variables, P is the number of variables in the scope of a function and C is the number of functions.

4.4. Pruning inconsistent subtrees for the flat constraint networks

Most advanced constraint processing algorithms incorporate no-good learning, and constraint propagation during search, or use variable elimination algorithms such as *adaptive-consistency* and *directional resolution* [34], generating all relevant no-goods, prior to search. Such schemes can be viewed as compiling a representation that would yield a *pruned* search tree. We next define the *backtrack-free* AND/OR search tree.

Definition 39 *(backtrack-free AND/OR search tree).* Given an AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$, the *backtrack-free AND/OR search tree* of $\mathcal{R}$ based on $\mathcal{T}$, denoted $BF_{\mathcal{T}}(\mathcal{R})$, is obtained by pruning from $S_{\mathcal{T}}(\mathcal{R})$ all inconsistent subtrees, namely all nodes that root no consistent partial solution.

Example 40. Consider 5 variables X, Y, Z, T, R over domains $\{2, 3, 5\}$, where the constraints are: X divides Y and Z, and Y divides T and R. The constraint graph and the AND/OR search tree relative to the DFS tree rooted at X, are given in Fig. 6(a). In 6(b) we present the $S_{\mathcal{T}}(\mathcal{R})$ search space whose nodes' consistency statuses (which will later will be referred to as *values*) are already evaluated having value "1" if consistent and "0" otherwise. We also highlight

two solution subtrees; one depicted by solid lines and one by dotted lines. Part (c) presents $BF_{\mathcal{T}}(\mathcal{R})$, where all nodes that do not root a consistent solution are pruned.

If we traverse the backtrack-free AND/OR search tree we can find a solution subtree without encountering any dead-ends. Some constraint networks specifications yield a backtrack-free search space. Others can be made backtrack-free by massaging their representation using *constraint propagation* algorithms before or during search. In particular, it is well known that variable elimination algorithms such as *adaptive-consistency* [35] and directional resolution [36], applied in a reversed order of d (where d is the DFS order of the pseudo tree), compile a constraint specification (resp., a Boolean CNF formula) that has a backtrack-free search space. Assuming that the reader is familiar with variable elimination algorithms [16], we define:

Definition 41 *(directional extension [35,36]).* Let $\mathcal{R}$ be a constraint problem and let d be a DFS traversal ordering of a backbone pseudo tree of its primal graph. We denote by $E_d(\mathcal{R})$ the constraint network (resp., the CNF formula) compiled by adaptive-consistency (resp., directional resolution) in reversed order of d.

Proposition 42. *Given a Constraint network $\mathcal{R}$, the AND/OR search tree of the directional extension $E_d(\mathcal{R})$, when d is a DFS ordering of $\mathcal{T}$, is identical to the backtrack-free AND/OR search tree of $\mathcal{R}$ based on $\mathcal{T}$. Namely $S_{\mathcal{T}}(E_d(\mathcal{R})) = BF_{\mathcal{T}}(\mathcal{R})$.*

Example 43. In Example 40, if we apply adaptive-consistency in reverse order of X, Y, T, R, Z, the algorithm will remove the values $3, 5$ from the domains of both X and Z yielding a tighter constraint network $\mathcal{R}'$. The AND/OR search tree in Fig. 6(c) is both $S_{\mathcal{T}}(\mathcal{R}')$ and $BF_{\mathcal{T}}(\mathcal{R})$.

Proposition 42 emphasizes the significance of no-good learning [37] for deciding inconsistency or for finding a single solution. These techniques are known as clause learning in SAT solvers, first introduced by [38] and are currently used in most advanced solvers [39]. Namely, when we apply no-good learning we explore the search space whose many inconsistent subtrees are pruned. For counting however, and for other relevant tasks, pruning inconsistent subtrees and searching the backtrack-free search tree yields a partial help only, as we elaborate later.

5. AND/OR search graphs

It is often the case that a search space that is a tree can become a graph if identical nodes are merged, because identical nodes root identical search subspaces, and correspond to identical reasoning subproblems. Any two nodes that root identical weighted subtrees can be *merged*, reducing the size search graph. For example, in Fig. 1(c), the search trees below any appearance of $\langle Y, 2\rangle$ are all identical, and therefore can be merged.

Sometimes, two nodes may not root identical subtrees, but they could still root search subspaces that correspond to equivalent subproblems. Nodes that root equivalent subproblems having the same universal model (see Definition 8) even though the weighted subtrees may not be identical, can be *unified*, yielding an even smaller search graph, as we will show.

We next formalize the notions of *merging* and *unifying* nodes and define the minimal AND/OR search graph.

5.1. Minimal AND/OR search graphs

An AND/OR search tree can also be viewed as a data structure that defines a *universal* graphical model (see Definition 8), defined by the weights of its set of solution subtrees (see Definition 22).

Definition 44 *(universal graphical model of AND/OR search trees).* Given a weighted AND/OR search tree $\mathcal{G}$ over a set of variables X and domains D, its *universal graphical model*, denoted by $U(\mathcal{G})$, is defined by its set of solutions as follows: if t is a solution subtree and $x = asgn(t)$ is the assignment tuple associated with t, then define $u(x) = w(t)$; otherwise $u(x) = 0$.

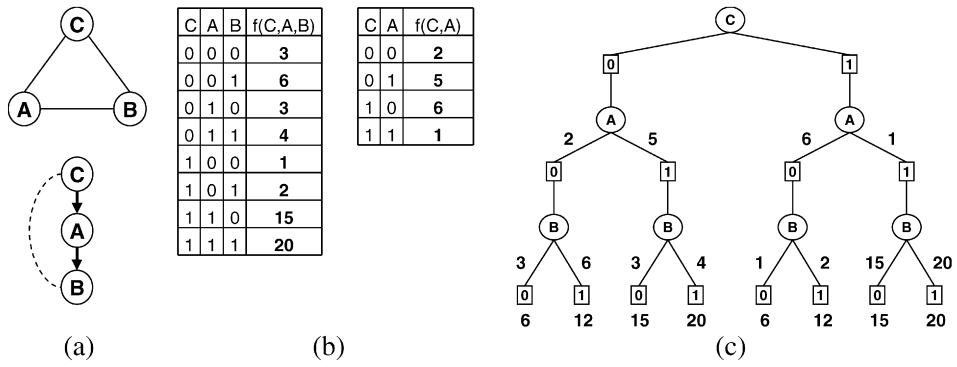


Fig. 7. Merge vs. unify operators.

A graphical model $\mathcal{R}$ is equivalent to its AND/OR search tree, $S_{\mathcal{T}}(\mathcal{R})$, which means that $u(\mathcal{R})$ is identical to $U(S_{\mathcal{T}}(\mathcal{R}))$. We will next define sound merge operations that transform AND/OR search trees into graphs that preserve equivalence.

Definition 45 *(merge).* Assume a given weighted AND/OR search graph $S'_{\mathcal{T}}(\mathcal{R})$ ($S'_{\mathcal{T}}(\mathcal{R})$ can be the AND/OR search tree $S_{\mathcal{T}}(\mathcal{R})$), and assume two paths $\pi_1 = \pi_{n_1}(x_i)$ and $\pi_2 = \pi_{n_2}(x_i)$ ending by AND nodes at level i having the same label x_i. Nodes n_1 and n_2 can be *merged* iff the weighted search subgraphs rooted at n_1 and n_2 are identical. The *merge* operator, $merge(n_1, n_2)$, redirects all the arcs going into n_2 into n_1 and removes n_2 and its subgraph. It thus transforms $S'_{\mathcal{T}}$ into a smaller graph. When we merge AND nodes only we call the operation AND-merge. The same reasoning can be applied to OR nodes, and we call the operation OR-merge.

We next define the semantic notion of *unifiable* nodes, as opposed to the syntactic definition of *merge*.

Definition 46 *(unify).* Given a weighted AND/OR search graph $\mathcal{G}$ for a graphical model $\mathcal{R}$ and given two paths π_{n_1} and π_{n_2} having the same label on nodes n_1 and n_2, then n_1 and n_2 are *unifiable*, iff they root equivalent conditioned subproblems (Definition 10). Namely, if $\mathcal{R}|_{asgn(\pi_1)} = \mathcal{R}|_{asgn(\pi_2)}$.

Example 47. Let's follow the example in Fig. 7 to clarify the difference between *merge* and *unify*. We have a graphical model defined by two functions (e.g., cost functions) over three variables. The search tree given in Fig. 7(c) cannot be reduced to a graph by *merge*, because of the different arc weights. However, the two OR nodes labeled A root equivalent conditioned subproblems (the cost of each individual solution is given at the leaves). Therefore, the nodes labeled A can be *unified*, but they cannot be recognized as identical by the *merge* operator.

Proposition 48 *(minimal graph). Given a weighted AND/OR search graph $\mathcal{G}$ based on pseudo tree $\mathcal{T}$*:

(1) *The* merge *operator has a unique fix point, called the* merge minimal *AND/OR search graph and denoted by* $M_{\mathcal{T}}^{merge}(\mathcal{G})$.
(2) *The* unify *operator has a unique fix point, called the* unify minimal *AND/OR search graph and denoted by* $M_{\mathcal{T}}^{unify}(\mathcal{G})$.
(3) *Any two nodes n_1 and n_2 of $\mathcal{G}$ that can be merged can also be unified.*

Definition 49 *(minimal AND/OR search graph).* The unify minimal AND/OR search graph of $\mathcal{R}$ relative to $\mathcal{T}$ will also be simply called the *minimal AND/OR search graph* and be denoted by $M_{\mathcal{T}}(\mathcal{R})$.

When $\mathcal{T}$ is a chain pseudo tree, the above definitions are applicable to the traditional OR search tree as well. However, we may not be able to reach the same compression as in some AND/OR cases, because of the linear structure imposed by the OR search tree.

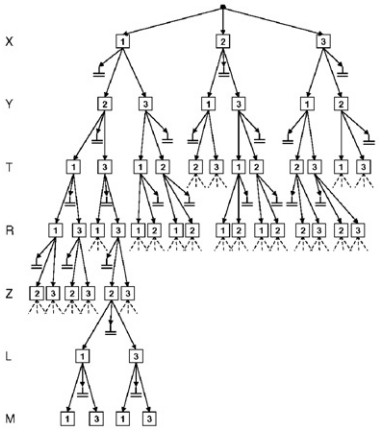


Fig. 8. OR search tree for the tree problem in Fig. 1(a).

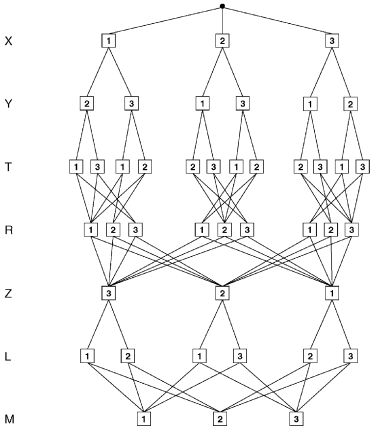


Fig. 9. The minimal OR search graph of the tree graphical model in Fig. 1(a).

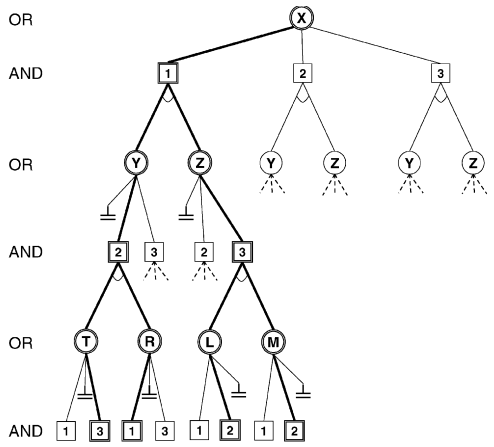


Fig. 10. AND/OR search tree for the tree problem in Fig. 1(a).

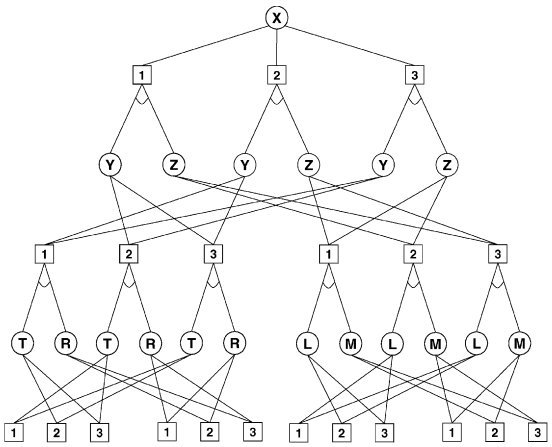


Fig. 11. The minimal AND/OR search graph of the tree graphical model in Fig. 1(a).

Example 50. The smallest OR search graph of the graph-coloring problem in Fig. 1(a) is given in Fig. 9 along the DFS order X, Y, T, R, Z, L, M. The smallest AND/OR graph of the same problem along the DFS tree is given in Fig. 11. We see that some variable-value pairs (AND nodes) must be repeated in Fig. 9 while in the AND/OR case they appear just once. In particular, the subgraph below the paths $(\langle X, 1\rangle, \langle Y, 2\rangle)$ and $(\langle X, 3\rangle, \langle Y, 2\rangle)$ in the OR tree cannot be merged at $\langle Y, 2\rangle$. You can now compare all four search space representations side by side in Figs. 8–11.

Note that in the case of constraint networks we can accommodate an even more general definition of merging of two AND nodes that are assigned *different* values from their domain, or two OR nodes labeled by different variables, as long as they root identical subgraphs. In that case the merged node should be labeled by the disjunction of the two assignments (this is similar to interchangeable values [23]).

5.2. Building AND/OR search graphs

In this subsection we will discuss practical algorithms for generating compact AND/OR search graphs of a given graphical model. In particular we will identify effective rules for recognizing unifiable nodes, aiming towards the minimal AND/OR search graph as much as computational resources allow. The rules allow generating a small AND/OR

graph called *the context minimal graph* without creating the whole search tree $S_{\mathcal{T}}$ first. We focus first on AND/OR search graphs of graphical models having no cycles, called *tree models* (i.e., the primal graph is a tree).

5.2.1. Building AND/OR search graphs for tree models and tree decompositions

Consider again the graph in Fig. 1(a) and its AND/OR search tree in Fig. 1(c) representing a constraint network. Observe that at level 3, node $\langle Y, 1\rangle$ appears twice, (and so are $\langle Y, 2\rangle$ and $\langle Y, 3\rangle$). Clearly however, the subtrees rooted at each of these two AND nodes are identical and we can reason that they can be merged because any specific assignment to Y uniquely determines its rooted subtree. Indeed, the AND/OR search graph in Fig. 11 is equivalent to the AND/OR search tree in Fig. 10 (same as Fig. 1(c)).

Definition 51 *(explicit AND/OR graphs for constraints tree models).* Given a tree model constraint network and the pseudo tree $\mathcal{T}$ identical to its primal graph, the *explicit AND/OR search graph* of the tree model relative to $\mathcal{T}$ is obtained from $S_{\mathcal{T}}$ by merging all AND nodes having the same label $\langle X, x\rangle$.

Proposition 52. *Given a rooted tree model* $\mathcal{T}$: (1) *Its explicit AND/OR search* graph *is equivalent to* $S_{\mathcal{T}}$. (2) *The size of the explicit AND/OR search graph is* $O(nk)$. (3) *For some tree models the explicit AND/OR search graph is minimal.*

The notion of explicit AND/OR search graph for a tree model is extendable to any general graphical models that are trees. The only difference is that the arcs have weights. Thus, we need to show that merged nodes via the rule in definition 51 root identical weighted AND/OR trees.

Proposition 53. *Given a general graphical model whose graph is a tree* $\mathcal{T}$*, its explicit AND/OR search* graph *is equivalent to* $S_{\mathcal{T}}$*, and its size is* $O(nk)$.

Next, the question is how to identify *efficiently* mergeable nodes for *general* non-tree graphical models. A guiding idea is to transform a graphical model into a tree decomposition first, and then apply the explicit AND/OR graph construction to the resulting tree decomposition. The next paragraph sketches this intuition.

A *tree decomposition* [33] (see Definition 5) of a graphical model partitions the functions into clusters. Each cluster corresponds to a subproblem that has a set of solutions and the clusters interact in a tree-like manner. Once we have a tree decomposition of a graphical model, it can be viewed as a regular (meta) tree model where each cluster is a node and its domain is the cross product of the domains of variables in the cluster. The constraint between two adjacent nodes in the tree decomposition is equality over the common variables. For more details about tree decompositions see [33]. For the meta-tree model the explicit AND/OR search graph is well defined: the OR nodes are the scopes of clusters in the tree decomposition and the AND nodes, are their possible value assignments. Since the graphical model is converted into a tree, its explicit AND/OR search graph is well defined and we can bound its size.

Theorem 54. *Given a* tree decomposition *of a graphical model, whose domain sizes are bounded by* k*, the* explicit AND/OR search graph *implied by the tree decomposition has a size of* $O(rk^{w^*})$*, where* r *is the number of clusters in the tree decomposition and* w^* *is the size of the largest cluster.*

The tree decomposition can guide an algorithm for generating an AND/OR search graph whose size is bounded exponentially by the induced width, which we will refer to in the next section as the *context minimal graph*.

While the idea of explicit AND/OR graph based on a tree decomposition can be extended to any graphical model, it is somewhat cumbersome. Instead, in the next section we propose a more direct approach for generating the context minimal graph.

5.2.2. The context based AND/OR graph

We will now present a generative rule for merging nodes in the AND/OR search graph that yields the size bound suggested above. We will need the notion of *induced width of a pseudo tree of G* for bounding the size of the AND/OR search *graphs*. We denote by $d_{DFS}(\mathcal{T})$ a linear DFS ordering of a tree $\mathcal{T}$.

Definition 55 *(induced width of a pseudo tree).* The induced width of G relative to the pseudo tree $\mathcal{T}$, $w_{\mathcal{T}}(G)$, is the induced width along the $d_{DFS}(\mathcal{T})$ ordering of the extended graph of G relative to $\mathcal{T}$, denoted $G^{\mathcal{T}}$.

Proposition 56. (1) *The minimal induced width of G over all pseudo trees is identical to the induced width* (*treewidth*)*, w^*, of G*. (2) *The minimal induced width restricted to chain pseudo trees is identical to its pathwidth, pw^*.*

Example 57. In Fig. 4(b), the induced graph of G relative to $\mathcal{T}_1$ contains also the induced arcs $(1,3)$ and $(1,5)$ and its induced width is 2. $G^{\mathcal{T}_2}$ is already triangulated (no need to add induced arcs) and its induced width is 2 as well. $G^{\mathcal{T}_3}$ has the added arc $(4,7)$ and when ordered it will have the additional induced arcs $(1,5)$ and $(1,3)$, yielding induced width 2 as well.

We will now provide definitions that will allow us to identify nodes that can be merged in an AND/OR graph. The idea is to find a minimal set of variable assignments from the current path that will always generate the same conditioned subproblem, regardless of the assignments that are not included in this minimal set. Since the current path for an OR node X_i and an AND node $\langle X_i, x_i\rangle$ differ by the assignment of X_i to x_i (Definition 2), the minimal set of assignments that we want to identify will be different for X_i and for $\langle X_i, x_i\rangle$. In the following two definitions ancestors and descendants are with respect to the pseudo tree $\mathcal{T}$, while connection is with respect to the primal graph G.

Definition 58 *(parents).* Given a primal graph G and a pseudo tree $\mathcal{T}$ of a reasoning problem $\mathcal{P}$, the *parents* of an OR node X_i, denoted by pa_i or pa_{X_i}, are the ancestors of X_i that have connections in G to X_i or to descendants of X_i.

Definition 59 *(parent-separators).* Given a primal graph G and a pseudo tree $\mathcal{T}$ of a reasoning problem $\mathcal{P}$, the *parent-separators* of X_i (or of $\langle X_i, x_i\rangle$), denoted by pas_i or pas_{X_i}, are formed by X_i and its ancestors that have connections in G to descendants of X_i.

It follows from these definitions that the parents of X_i, pa_i, separate in the primal graph G (and also in the extended graph $G^{\mathcal{T}}$ and in the induced extended graph $G^{\mathcal{T}^*}$) the ancestors (in $\mathcal{T}$) of X_i, from X_i and its descendants (in $\mathcal{T}$). Similarly, the parent-separators of X_i, pas_i, separate the ancestors of X_i from its descendants. It is also easy to see that each variable X_i and its parents pa_i form a clique in the induced graph $G^{\mathcal{T}^*}$. The following proposition establishes the relationship between pa_i and pas_i.

Proposition 60.

(1) *If Y is the single child of X in $\mathcal{T}$, then $pas_X = pa_Y$.*
(2) *If X has children $Y_1, \ldots, Y_k$ in $\mathcal{T}$, then $pas_X = \bigcup_{i=1}^{k} pa_{Y_i}$.*

Theorem 61 *(context based merge). Given $G^{\mathcal{T}^*}$, let π_{n_1} and π_{n_2} be any two partial paths in an AND/OR search graph, ending with two nodes, n_1 and n_2.*

(1) *If n_1 and n_2 are AND nodes annotated by $\langle X_i, x_i\rangle$ and*

$$asgn(\pi_{n_1})[pas_{X_i}] = asgn(\pi_{n_2})[pas_{X_i}] \tag{1}$$

then the AND/OR search subgraphs rooted by n_1 and n_2 are equivalent and n_1 and n_2 can be merged. $asgn(\pi_{n_i})[pas_{X_i}]$ is called the AND context *of n_i.*
(2) *If n_1 and n_2 are OR nodes annotated by X_i and*

$$asgn(\pi_{n_1})[pa_{X_i}] = asgn(\pi_{n_2})[pa_{X_i}] \tag{2}$$

then the AND/OR search subgraphs rooted by n_1 and n_2 are equivalent and n_1 and n_2 can be merged. $asgn(\pi_{n_i})[pa_{X_i}]$ is called the OR context of n_i.

Example 62. For the balanced tree in Fig. 1 consider the chain pseudo tree $d = (X, Y, T, R, Z, L, M)$. Namely the chain has arcs $\{(X,Y),(Y,T),(T,R),(R,Z),(Z,L),(L,M)\}$ and the extended graph also includes the arcs (Y,T), (Z,X) and (M,Z). The parent-separators of T along d are XYT (since the induced graph has the arc (T,X)), of R are XR, for Z they are Z and for M they are M. Indeed in the first 3 levels of the OR search graph in Fig. 9 there are

no merged nodes. In contrast, if we consider the AND/OR ordering along the DFS tree, the parent-separators of every node are the node itself, yielding a single appearance of each AND node having the same assignment annotation in the minimal AND/OR graph.

Definition 63 *(context minimal AND/OR search graph).* The AND/OR search graph of $\mathcal{R}$ based on the backbone tree $\mathcal{T}$ that is closed under context-based merge operator is called *context minimal* AND/OR search graph and is denoted by $C_{\mathcal{T}}(\mathcal{R})$.

We should note that we can in general merge nodes based both on AND and OR contexts. However, Proposition 60 shows that doing just one of them renders the other unnecessary (up to some small constant factor). In practice, we would recommend just the OR context based merging, because it has a slight (albeit by a small constant factor) space advantage. In the examples that we give in this paper, $C_{\mathcal{T}}(\mathcal{R})$ refers to an AND/OR search graph for which either the AND context based or OR context based merging was performed exhaustively.

Example 64. Consider the example given in Fig. 12(a). The OR context of each node in the pseudo tree is given in square brackets. The context minimal AND/OR search graph (based on OR merging) is given in Fig. 12(b).

Since the number of nodes in the context minimal AND/OR search graph cannot exceed the number of different contexts, we can bound the size of the context minimal graph.

Theorem 65. *Given a graphical model $\mathcal{R}$, its primal graph G, and a pseudo tree $\mathcal{T}$ having induced width $w = w_{\mathcal{T}}(G)$, the size of the context minimal AND/OR search graph based on $\mathcal{T}$, $C_{\mathcal{T}}(\mathcal{R})$, is* $O(n \cdot k^w)$*, when k bounds the domain size.*

Note that the criterion in Eqs. (1) and (2) is cautious. First, the real number of assignments over context variables includes only consistent assignments. Second, we have already seen (Example 7) that there exist nodes that can be *unified* but not *merged*. Here we give an example that shows that contexts can not identify all the nodes that can be *merged*. There could be paths whose contexts are not identical, yet they might root identical subgraphs.

Example 66. Let's return to the example of the Bayesian network given in Fig. 12(a), where $P(D|B, C)$ is given in the table, and the OR-context of each node in the pseudo tree is given in square brackets. Fig. 12(b) shows the context

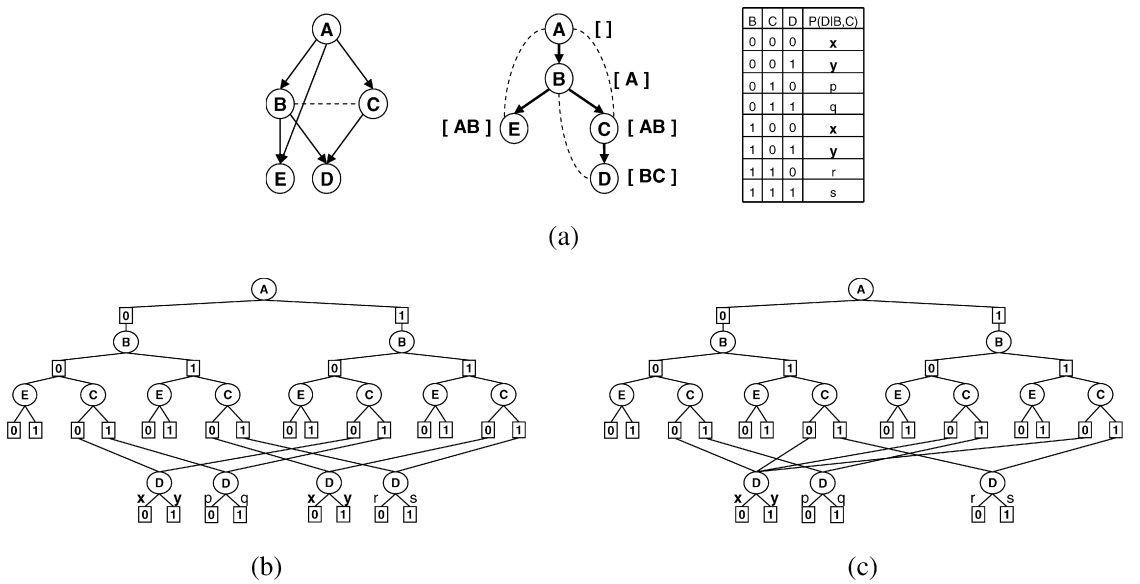


Fig. 12. Context minimal vs. minimal AND/OR graphs.

minimal graph. However, we can see that $P(D=0|B=0,C=0) = P(D=0|B=1,C=0) = x$ and $P(D=1|B=0,C=0) = P(D=1|B=1,C=0) = y$. This allows the *merging* of the corresponding OR nodes labeled with D, and Fig. 12(c) shows the merge minimal graph.

The context based merge offers a powerful way of bounding the search complexity:

Theorem 67. *The context minimal AND/OR search graph $C_{\mathcal{T}}$ of a graphical model having a backbone tree with bounded treewidth w can be generated in time and space* $O(nk^w)$.

Since the unify minimal AND/OR graph $M_{\mathcal{T}}^{unify}$ and the merge minimal AND/OR graph $M_{\mathcal{T}}^{merge}$ are subsets of $C_{\mathcal{T}}$, both are bounded by $O(n \cdot k^w)$, where $w = w_{\mathcal{T}}(G)$. Since $\min_{\mathcal{T}}\{w_{\mathcal{T}}(G)\}$ is equal to the treewidth w^* and since $\min_{\mathcal{T}\in chains}\{w_{\mathcal{T}}(G)\}$ is equal to the pathwidth pw^*, we get:

Corollary 68. *Given a graphical model $\mathcal{R}$, there exists a backbone tree $\mathcal{T}$ such that the unify minimal, merge minimal and context minimal AND/OR search graphs of $\mathcal{R}$ are bounded exponentially by the treewidth of the primal graph. The unify, merge and context minimal OR search graphs can be bounded exponentially by the pathwidth only.*

5.2.3. More on OR vs. AND/OR

It is well known [26] that for any graph $w^* \leqslant pw^* \leqslant w^* \cdot \log n$. It is easy to place m^* (the minimal depth over pseudo trees) in that relation yielding $w^* \leqslant pw^* \leqslant m^* \leqslant w^* \cdot \log n$. It is also possible to show that there exist primal graphs for which the upper bound on pathwidth is attained, that is $pw^* = O(w^* \cdot \log n)$.

Consider a complete binary tree of depth m. In this case, $w^* = 1$, $m^* = m$, and it is also known [40,41] that:

Theorem 69. [41] *If $\mathcal{T}$ is a binary tree of depth m then $pw^*(\mathcal{T}) \geqslant \frac{m}{2}$.*

Theorem 69 shows that for graphical models having a bounded tree width w, the minimal AND/OR graph is bounded by $O(nk^w)$ while the minimal OR graph is bounded by $O(nk^{w\cdot\log n})$. Therefore, even when caching, the use of an AND/OR vs. an OR search space can yield a substantial saving.

Remark 70. We have seen that AND/OR *trees* are characterized by the *depth* of the pseudo trees while minimal AND/OR *graphs* are characterized by their *induced width*. It turns out however that sometimes a pseudo tree that is optimal relative to w is far from optimal for m and vice versa. For example a primal graph model that is a chain has a pseudo tree having $m_1 = n$ and $w_1 = 1$ on one hand, and another pseudo tree that is balanced having $m_2 = \log n$ and $w_2 = \log n$. There is no single pseudo tree having both $w = 1$ and $m = \log n$ for a chain. Thus, if we plan to have linear space search we should pick one kind of a backbone pseudo tree, while if we plan to search a graph, and therefore cache some nodes, another pseudo tree should be used.

5.3. On the canonicity and generation of the minimal AND/OR graph

We showed that the merge minimal AND/OR graph is unique for a given graphical model, given a backbone pseudo tree (Proposition 48). In general, it subsumes the minimal AND/OR graph, and sometimes can be identical to it. For constraint networks we will now prove a more significant property of uniqueness relative to all equivalent graphical models given a backbone tree. We will prove this notion relative to *backtrack-free* search graphs which are captured by the notion of strongly minimal AND/OR graph. Remember that any graphical model can have an associated flat constraint network.

Definition 71 *(strongly minimal AND/OR graph[3]).* A *strongly minimal* AND/OR graph of $\mathcal{R}$ relative to a pseudo tree $\mathcal{T}$ is the minimal AND/OR graph, $M_{\mathcal{T}}(\mathcal{R})$, that is backtrack-free (i.e., any partial assignment in the graph leads to a solution), denoted by $M_{\mathcal{T}}^*(\mathcal{R})$. The strongly (backtrack-free) context minimal graph is denoted $C_{\mathcal{T}}^*(\mathcal{R})$.

[3] The minimal graph is built by lumping together "unifiable" nodes, which are those that root equivalent subproblems. Therefore, at each level (corresponding to one variable), all the nodes that root inconsistent subproblems will be unified. If we eliminate the redundant nodes, the minimal graph is already backtrack free.

5.3.1. Canonicity of strongly minimal AND/OR search graphs

We briefly discuss here the canonicity of the strongly minimal graph, focusing on constraint networks. Given two equivalent constraint networks representing the same set of solutions, where each may have a different constraint graph, are their strongly minimal AND/OR search graphs identical?

The above question is not well defined however, because an AND/OR graph for $\mathcal{R}$ is defined only with respect to a backbone pseudo tree. We can have two equivalent constraint networks having two different graphs where a pseudo tree for one graph may not be a pseudo tree for the other. Consider, for example a constraint network having three variables: X, Y and Z and equality constraints. The following networks, $\mathcal{R}_1 = \{R_{XY} = (X = Y), R_{YZ} = (Y = Z)\}$ and $\mathcal{R}_2 = \{R_{XZ} = (X = Z), R_{YZ} = (Y = Z)\}$ and $\mathcal{R}_3 = \{R_{XY} = (X = Y), R_{YZ} = (Y = Z), R_{XZ} = (X = Z)\}$ are equivalent. However, $\mathcal{T}_1 = (X \leftarrow Y \rightarrow Z)$ is a pseudo tree for $\mathcal{R}_1$, but not for $\mathcal{R}_2$ neither for $\mathcal{R}_3$. We ask therefore a different question: given two equivalent constraint networks and given a backbone tree that is a pseudo tree for both, is the strongly minimal AND/OR graph relative to $\mathcal{T}$ unique?

We will answer this question positively quite straightforwardly. We first show that equivalent networks that share a backbone tree have identical backtrack-free AND/OR search trees. Since the backtrack-free search trees uniquely determine their strongly minimal graph the claim follows.

Definition 72 *(shared pseudo trees).* Given a collection of graphs on the same set of nodes, we say that the graphs share a tree $\mathcal{T}$, if $\mathcal{T}$ is a pseudo tree of each of these graphs. A set of graphical models defined over the same set of variables share a tree $\mathcal{T}$, iff their respective primal graphs share $\mathcal{T}$.

Proposition 73. (1) *If $\mathcal{R}_1$ and $\mathcal{R}_2$ are two equivalent constraint networks that share $\mathcal{T}$, then $BF_{\mathcal{T}}(\mathcal{R}_1) = BF_{\mathcal{T}}(\mathcal{R}_2)$* (*see Definition* 39). (2) *If $\mathcal{R}_1$ and $\mathcal{R}_2$ are two equivalent graphical models* (*not necessarily constraint networks*) *that share $\mathcal{T}$, then $BF_{\mathcal{T}}(\mathcal{R}_1) = BF_{\mathcal{T}}(\mathcal{R}_2)$ as AND/OR search trees although their arcs may not have identical weights.*

Theorem 74. *If $\mathcal{R}_1$ and $\mathcal{R}_2$ are two equivalent constraint networks that share $\mathcal{T}$, then $M^*_{\mathcal{T}}(\mathcal{R}_1) = M^*_{\mathcal{T}}(\mathcal{R}_2)$.*

Theorem 74 implies that $M^*_{\mathcal{T}}$ is a canonical representation of a constraint network $\mathcal{R}$ relative to $\mathcal{T}$.

Generating the strongly minimal AND/OR graphs

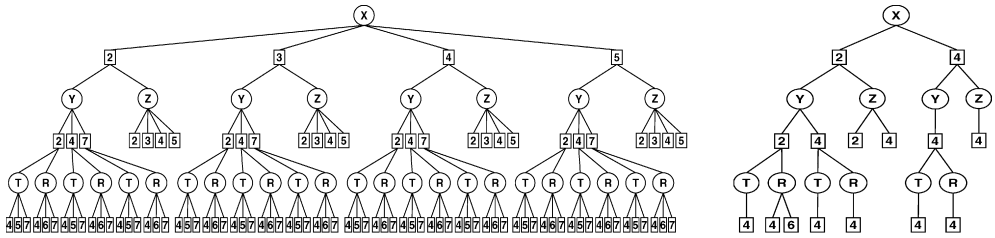
From the above discussion we see that several methods for generating the canonical AND/OR graph of a given graphical model, or a given AND/OR graph may emerge. The method we focused on in this paper is to generate the context minimal AND/OR graph first. Then we can process this graph from leaves to root, while computing the value of nodes, and additional nodes can be unified or pruned (if their value is "0").

There is another approach that is based on processing the functions in a variable elimination style, when viewing the pseudo tree as a bucket tree or a cluster tree. The original functions can be expressed as AND/OR graphs and they will be combined pairwise until an AND/OR graph is generated. This phase allows computing the value of each node (see Section 6) and therefore allows for unification. Subsequently a forward phase will allow generating the backtrack-free representation as well as allow computing the full values associated with each node. The full details of this approach are out of the scope of the current paper. For initial work restricted to constraint networks see [42].

5.4. Merging and pruning: Orthogonal concepts

Notice that the notion of minimality is orthogonal to that of pruning inconsistent subtrees (yielding the backtrack-free search space). We can merge two identical subtrees whose root value is "0" but still keep their common subtree. However, since our convention is that we don't keep inconsistent subtrees we should completely prune them, irrespective of them rooting identical or non-identical subtrees. Therefore, we can have a minimal search graph that is *not* backtrack-free as well as a non-minimal search graph (e.g., a tree) that is backtrack-free.

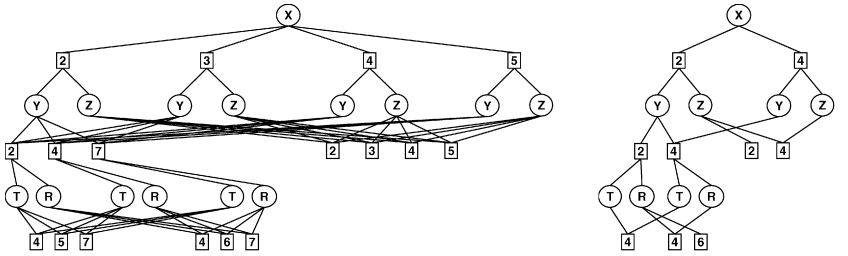
When the search space is backtrack-free and if we seek a single solution, the size of the minimal AND/OR search graph and its being OR vs. AND/OR are both irrelevant. It will, however, affect a traversal algorithm that counts all solutions or computes an optimal solution as was often observed [43]. For counting and for optimization tasks, even



(a) Full AND/OR tree

(b) Pruned backtrack-free AND/OR tree

Fig. 13. AND/OR trees.



(a) Context minimal unpruned AND/OR graph

(b) Context minimal pruned backtrack-free AND/OR graph

Fig. 14. AND/OR graphs.

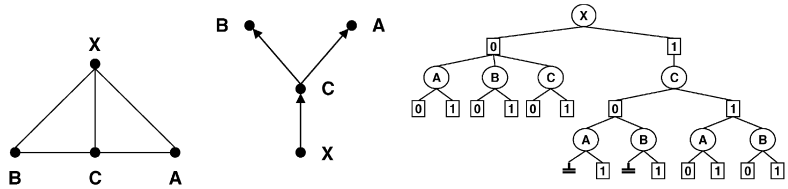


Fig. 15. (a) A constraint graph; (b) a spanning tree; (c) a dynamic AND/OR tree.

when we record all no-goods and cache all nodes by context, the impact of the AND/OR graph search vs. the OR graph search can still be significant.

Example 75. Consider the graph problem in Fig. 6(a) when we add the value 4 to the domains of X and Z. Fig. 13(a) gives the full AND/OR search tree and Fig. 13(b) gives the backtrack-free search tree. Fig. 14(a) gives the context minimal but unpruned search graph and Fig. 14(b) gives the minimal and pruned search graph.

5.5. Using dynamic variable ordering

The AND/OR search tree we defined uses a fixed variable ordering. It is known that exploring the search space in a dynamic variable ordering is highly beneficial. AND/OR search trees for graphical models can also be modified to allow dynamic variable ordering. A dynamic AND/OR tree that allows varied variable ordering has to satisfy that for every subtree rooted by the current path π, any arc of the primal graph that appears as a cross-arc (not a back-arc) in the subtree must be "inactive" conditioned on π.

Example 76. Consider the propositional formula $X \rightarrow A \vee C$ and $X \rightarrow B \vee C$. The constraint graph is given in Fig. 15(a) and a DFS tree in 15(b). However, the constraint subproblem conditioned on $\langle X, 0\rangle$, has no real constraint

between A, B, C, so the effective spanning tree below $\langle X, 0\rangle$ is $\{\langle X, 0\rangle \rightarrow A, \langle X, 0\rangle \rightarrow B, \langle X, 0\rangle \rightarrow C\}$, yielding the AND/OR search tree in Fig. 15(c). Note that while there is an arc between A and C in the constraint graph, the arc is *not* active when X is assigned the value 0.

Clearly, the constraint graph conditioned on any partial assignment can only be sparser than the original graph and therefore may yield a smaller AND/OR search tree than with fixed ordering. In practice, after each new value assignment, the conditional constraint graph can be assessed as follows. For any constraint over the current variable X, if the current assignment $\langle X, x\rangle$ does not make the constraint *active* then the corresponding arcs can be removed from the graph. Then, a pseudo tree of the resulting graph is generated, its first variable is selected, and search continues. A full investigation of dynamic orderings is outside the scope of the current paper.

6. Solving reasoning problems

6.1. Value functions of reasoning problems

As we described earlier, there are a variety of reasoning problems over weighted graphical models. For constraint networks, the most popular tasks are to decide if the problem is consistent, to find a single solution or to count solutions. If there is a cost function defined we may also seek an optimal solution. The primary tasks over probabilistic networks are belief updating, finding the probability of the evidence and finding the most likely tuple given the evidence. Each of these reasoning problems can be expressed as finding the *value* of some nodes in the weighted AND/OR search space where different tasks call for different value definitions. For example, for the task of finding a solution to a constraint network, the value of every node is either "1" or "0". The value "1" means that the subtree rooted at the node is consistent and "0" otherwise. Therefore, the value of the root node answers the consistency query. For solutions-counting the value function of each node is the number of solutions rooted at that node.

Definition 77 *(value function for consistency and counting).* Given a weighted AND/OR tree $S_{\mathcal{T}}(\mathcal{R})$ of a constraint network: The value of a node (AND or OR) for *deciding consistency* is "1" if it roots a consistent subproblem and "0" otherwise; The value of a node (AND or OR) for *counting solutions* is the number of solutions in its subtree.

It is easy to see that the value of nodes in the search graph can be computed recursively from leaves to root.

Proposition 78 *(recursive value computation).* (1) *For the consistency task the value of AND leaves is their labels and the value of OR leaves is "*0*" (they are inconsistent). An internal OR node is labeled "*1*" if one of its successor nodes is "*1*" and an internal AND node has value "*1*" iff all its successor OR nodes have value "*1*".*

(2) *The counting values of leaf AND nodes are "*1*" and of leaf OR nodes are "*0*". The counting value of an internal OR node is the sum of the counting values of all its child nodes. The counting value of an internal AND node is the product of the counting values of all its child nodes.*

We can now generalize to any reasoning problem, focusing on the simplified case when $Z = \emptyset$, namely when the marginalization has to be applied to all the variables. This special case captures most tasks of interest. We will start with the recursive definition.

Definition 79 *(recursive definition of values).* The value function of a reasoning problem $\mathcal{P} = \langle \mathcal{R}, \Downarrow_Y, Z\rangle$, where $\mathcal{R} = \langle X, D, F, \otimes\rangle$ and $Z = \emptyset$, is defined as follows: the value of leaf AND nodes is "1" and of leaf OR nodes is "0". The value of an internal OR node is obtained by *combining* the value of each AND child node with the weight (see Definition 26) on its incoming arc and then *marginalizing* over all AND children. The value of an AND node is the combination of the values of its OR children. Formally, if $children(n)$ denotes the children of node n in the AND/OR search graph, then:

$$v(n) = \bigotimes_{n' \in children(n)} v(n'), \quad \text{if } n = \langle Y, y\rangle \text{ is an AND node,}$$

$$v(n) = \Downarrow_{X-(\{Y\}\cup children(n))} \left(w_{(n,n')} \otimes v(n')\right)_{n' \in children(n)}, \quad \text{if } n = Y \text{ is an OR node.}$$

The following proposition states that given a reasoning task, computing the value of the root node solves the given reasoning problem.

Proposition 80. *Let $\mathcal{P} = \langle \mathcal{R}, \Downarrow_Y, Z \rangle$, where $\mathcal{R} = \langle X, D, F, \otimes \rangle$ and $Z = \emptyset$, and let X_1 be the root node in any AND/OR search graph $S'_{\mathcal{T}}(\mathcal{R})$. Then $v(X_1) = \Downarrow_\emptyset \bigotimes_{i=1}^{r} f_i$ when v is defined in Definition 79.*

Search algorithms that traverse the AND/OR search space can compute the value of the root node yielding the answer to the problem. The following section discusses such algorithms. Algorithms that traverse the weighted AND/OR search tree in a depth-first manner or a breadth-first manner are guaranteed to have a time bound exponential in the depth of the pseudo tree of the graphical model. Depth-first searches can be accomplished using either linear space only, or context based caching, bounded exponentially by the treewidth of the pseudo tree. Depth-first search is an anytime scheme and can, if terminated, provide an approximate solution for some tasks such as optimization. The next subsection presents typical depth-first algorithms that search AND/OR trees and graphs. We use *solution counting* as an example for a constraint query and the probability of evidence as an example for a probabilistic reasoning query. The algorithms compute the value of each node. For application of these ideas for combinatorial optimization tasks, such as MPE, see [31].

6.2. Algorithm AND/OR tree search and graph search

Algorithm 1 presents the basic depth-first traversal of the AND/OR search tree (or graph, if caching is used) for counting the number of solutions of a constraint network, AO-COUNTING (or for probability of evidence for belief networks, AO-BELIEF-UPDATING).

The context based caching is done based on tables. We exemplify with OR caching. For each variable X_i, a table is reserved in memory for each possible assignment to its parent set pa_i. Initially each entry has a predefined value, in our case "−1". The fringe of the search is maintained on a stack called OPEN. The current node is denoted by n, its parent by p, and the current path by π_n. The children of the current node are denoted by *successors*(n).

The algorithm is based on two mutually recursive steps: EXPAND and PROPAGATE, which call each other (or themselves) until the search terminates.

Since we only use OR caching, before expanding an OR node, its cache table is checked (line 6). If the same context was encountered before, it is retrieved from cache, and *successors*(n) is set to the empty set, which will trigger the PROPAGATE step.

If a node is not found in cache, it is expanded in the usual way, depending on whether it is an AND or OR node (lines 10–17). The only difference between counting and belief updating is line 12 vs. line 13. For counting, the value of a consistent AND node is initialized to 1 (line 12), while for belief updating, it is initialized to the bucket value for the current assignment (line 13). As long as the current node is not a dead-end and still has unevaluated successors, one of its successors is chosen (which is also the top node on OPEN), and the expansion step is repeated.

The bottom up propagation of values is triggered when a node has an empty set of successors (note that as each successor is evaluated, it is removed from the set of successors in line 31). This means that all its children have been evaluated, and its final value can now be computed. If the current node is the root, then the search terminates with its value (line 20). If it is an OR node, its value is saved in cache before propagating it up (line 22). If n is OR, then its parent p is AND and p updates its value by multiplication with the value of n (line 24). If the newly updated value of p is 0 (line 25), then p is a dead-end, and none of its other successors needs to be evaluated. An AND node n propagates its value to its parent p in a similar way, only by summation (line 30). Finally, the current node n is set to its parent p (line 32), because n was completely evaluated. The search continues either with a propagation step (if conditions are met) or with an expansion step.

6.3. General AND-OR search—AO(i)

General AND/OR algorithms for evaluating the value of a root node for any reasoning problem using tree or graph AND/OR search are identical to the above algorithms when product is replaced by the combination operator and

input : A constraint network $\mathcal{R} = \langle X, D, C\rangle$, or a belief network $\mathcal{P} = \langle X, D, P\rangle$; a pseudo tree $\mathcal{T}$ rooted at X_1; parents pa_i (OR-context) for every variable X_i; `caching` set to $true$ or $false$.
output : The number of solutions, or the updated belief, $v(X_1)$.

1 **if** `caching` $== true$ **then** // Initialize cache tables
2 Initialize cache tables with entries of "-1"
3 $v(X_1) \leftarrow 0$; `OPEN` $\leftarrow \{X_1\}$ // Initialize the stack `OPEN`
4 **while** `OPEN` $\neq \phi$ **do**
5 `n` $\leftarrow$ *top*(`OPEN`); remove `n` from `OPEN`
6 **if** `caching` $== true$ **and** `n` *is OR, labeled* X_i **and** $Cache(asgn(\pi_n)[pa_i]) \neq -1$ **then** // In cache
7 $v($`n`$) \leftarrow Cache(asgn(\pi_n)[pa_i])$ // Retrieve value
8 $successors($`n`$) \leftarrow \phi$ // No need to expand below
9 **else** // **EXPAND**
10 **if** `n` *is an OR node labeled* X_i **then** // OR-expand
11 $successors($`n`$) \leftarrow \{\langle X_i, x_i\rangle \mid \langle X_i, x_i\rangle$ is consistent with $\pi_n\}$
12 $v(\langle X_i, x_i\rangle) \leftarrow 1$, for all $\langle X_i, x_i\rangle \in successors($`n`$)$
$v(\langle X_i, x_i\rangle) \leftarrow \prod_{f \in B_{\mathcal{T}}(X_i)} f(asgn(\pi_n)[pa_i])$, for all $\langle X_i, x_i\rangle \in successors($`n`$)$ // AO-BU
13
14 **if** `n` *is an AND node labeled* $\langle X_i, x_i\rangle$ **then** // AND-expand
15 $successors($`n`$) \leftarrow children_{\mathcal{T}}(X_i)$
16 $v(X_i) \leftarrow 0$ for all $X_i \in successors($`n`$)$
17 Add $successors($`n`$)$ to top of `OPEN`
18 **while** $successors($`n`$) == \phi$ **do** // **PROPAGATE**
19 **if** `n` *is an OR node labeled* X_i **then**
20 **if** $X_i == X_1$ **then** // Search is complete
21 **return** $v($`n`$)$
22 **if** `caching` $== true$ **then**
23 $Cache(asgn(\pi_n)[pa_i]) \leftarrow v($`n`$)$ // Save in cache
24 $v($`p`$) \leftarrow v($`p`$) * v($`c`$)$
25 **if** $v($`p`$) == 0$ **then** // Check if `p` is dead-end
26 remove $successors($`p`$)$ from `OPEN`
27 $successors($`p`$) \leftarrow \phi$
28 **if** `n` *is an AND node labeled* $\langle X_i, x_i\rangle$ **then**
29 let `p` be the parent of `n`
30 $v($`p`$) \leftarrow v($`p`$) + v($`n`$)$;
31 remove `n` from $successors($`p`$)$
32 `n` $\leftarrow$ `p`

Algorithm 1. AO-counting/AO-belief-updating.

summation is replaced by the marginalization operator. We can view the AND/OR tree algorithm (which we will denote AOT) and the AND/OR graph algorithm (denoted AOG) as two extreme cases in a parameterized collection of algorithms that trade space for time via a controlling parameter i. We denote this class of algorithms as $AO(i)$ where i determines the size of contexts that the algorithm caches. Algorithm $AO(i)$ records nodes whose context size is i or smaller (the test in line 22 needs to be a bit more elaborate and check if the context size is smaller than i). Thus $AO(0)$ is identical to AOT, while $AO(w)$ is identical to AOG, where w is the induced width of the used backbone tree. For any intermediate i we get an intermediate level of caching, which is space exponential in i and whose execution time will increase as i decreases.

6.4. Complexity

From Theorems 34 and 38 we can conclude that:

Theorem 81. *For any reasoning problem,* AOT *runs in linear space and time* $O(nk^m)$*, when* m *is the depth of the pseudo tree of its graphical model and* k *is the maximum domain size. If the primal graph has a tree decomposition with treewidth* w^**, there exists a pseudo tree* $\mathcal{T}$ *for which AOT is* $O(nk^{w^* \cdot \log n})$.

Obviously, the algorithm for constraint satisfaction, that would terminate early with first solution, would potentially be much faster than the rest of the AOT algorithms, in practice.

Based on Theorem 65 we get complexity bounds for graph searching algorithms.

Theorem 82. *For any reasoning problem, the complexity of algorithm* AOG *is time and space* $O(nk^w)$*, where* w *is the induced width of the pseudo tree and* k *is the maximum domain size.*

Thus the complexity of AOG can be time and space exponential in the treewidth, while the complexity of any algorithm searching the OR space can be time and space exponential in its pathwidth. The space complexity can often be less than exponential in the treewidth. This is similar to the well known space complexity of tree decomposition schemes which can operate in space exponential only in the size of the cluster separators, rather than exponential in the cluster size. It is also similar to the *dead caches* concept presented in [12,32]. Intuitively, a node that has only one incoming arc will only be traversed once by search, and therefore its value does not need to be cached, because it will never be used again. For context based caching, such nodes can be recognized based only on the parents (or parent separators) sets.

Definition 83 *(dead cache).* If X is the parent of Y in $\mathcal{T}$, and $pa_X \subset pa_Y$, then pa_Y is a *dead cache*.

Given a pseudo tree $\mathcal{T}$, the induced graph along $\mathcal{T}$ can generate a tree decomposition based on the maximal cliques. The maximum separator size of the tree decomposition is the separator size of $\mathcal{T}$.

Proposition 84. *The space complexity of graph-caching algorithms can be reduced to being exponential in the separator's size only, while still being time exponential in the treewidth, if dead caches are not recorded.*

7. AND/OR search spaces and other schemes

7.1. Relationship with Variable Elimination

A comparison between Variable Elimination and memory intensive AND/OR search appears in [44]. That paper shows that Variable Elimination can be understood as bottom up layer by layer traversal of the context minimal AND/OR search graph. If the graphical model is strictly positive (has no determinism), then context based AND/OR search and Variable Elimination are essentially identical. When determinism is present, they may differ, because they traverse the AND/OR graph in different directions and encounter determinism (and can take advantage of it) differently. Therefore, for graphical models with no determinism, there is no principled difference between memory-intensive AND/OR search with fixed variable ordering and inference beyond: (1) different direction of exploring a common search space (top down for search vs. bottom up for inference); (2) different assumption of control strategy (depth-first for search and breadth-first for inference).

Another interesting observation is that many known advanced algorithms for constraint processing and satisfiability can be explained as traversing the AND/OR search tree, e.g. graph based backjumping [3,8,37]. For more details we refer the reader to [44].

7.2. Relationship with BTD (backtracking with tree-decomposition)

BTD [10] is a memory intensive method for solving constraint satisfaction problems, which combines search techniques with the notion of tree decomposition. This mixed approach can in fact be viewed as searching an AND/OR graph, whose backbone pseudo tree is defined by and structured along the tree decomposition. What is defined in [10] as *structural goods*, that is parts of the search space that would not be visited again as soon as their consistency is known, corresponds precisely to the decomposition of the AND/OR space at the level of AND nodes, which root independent subproblems. Not surprisingly, the time and space guarantees of BTD are the same as those of AND/OR graph search. An optimization version of the algorithm is presented in [11].

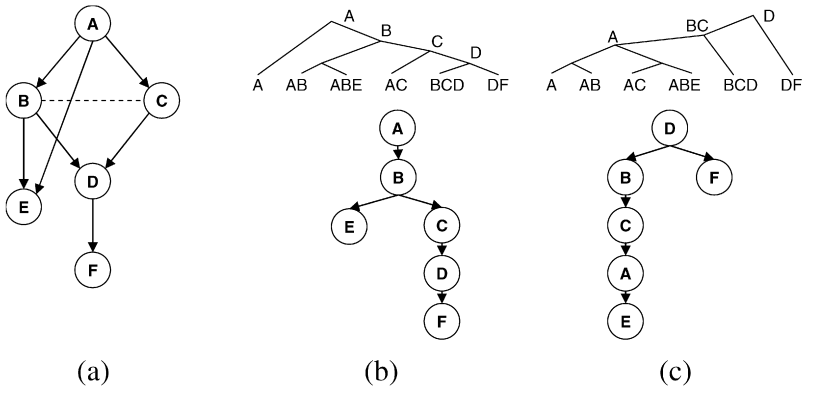


Fig. 16. Dtrees and AND/OR pseudo-trees.

7.3. Relationship with Recursive Conditioning

Recursive Conditioning (RC) [12] 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. Let's start with the example in Fig. 16, which shows: (a) a belief network; (b) and (c), two dtrees and the corresponding pseudo-trees for the AND/OR search. The dtrees also show the variables that are instantiated at some of the internal nodes. The pseudo-trees 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. It can also be shown that the context of the nodes in RC, as defined in [12] is identical to that in AND/OR.

7.4. Relationship with Value Elimination

Value Elimination [13] is a recently developed algorithm for Bayesian inference. It was already explained in [13] that, under static variable ordering, there is a strong relation between Value Elimination and Variable Elimination. From our paragraph on the relation between AND/OR search and Variable Elimination we can derive the connection between Value Elimination and AND/OR search, under static orderings. But we can also analyze the connection directly. Given a static ordering d for Value Elimination, we can show that it actually traverses an AND/OR space. The pseudo-tree underlying the AND/OR search graph traversal by Value Elimination can be constructed as the bucket tree in reversed d. However, the traversal of the AND/OR space will be controlled by d, advancing the frontier in a hybrid depth or breadth-first manner.

The most important part to analyze is the management of goods. When Value Elimination computes a factor at a leaf node, it backs up the value to the deepest node in the dependency set `Dset`. The `Dset` is identical to the context in the AND/OR space. For clarity reasons, we chose to have the AND/OR algorithm back up the value to its parent in the pseudo-tree, which may be different than the deepest variable in the context. We can however accommodate the propagation of the value like in Value Elimination, and maintain bookkeeping of the summation set `Sset`, and this would amount to a constant factor saving. Value Elimination continues by unionizing `Dsets` and `Ssets` whenever values are propagated, and this is identical to computing the context of the corresponding node in the AND/OR space (which is in fact the induced ancestor set of graph-based backjumping [45]).

In the presence of determinism, any backjumping strategy and nogood learning used by Value Elimination can also be performed in the AND/OR space. Context specific structure that can be used by Value Elimination, can also be used in AND/OR. Dynamic variable orderings can also be used in AND/OR spaces, but in this paper we limit the discussion to static orderings.

7.5. Relationship with case-factor diagrams

Case-Factor Diagrams (CFD) were introduced in [20] and represent a probabilistic formalism subsuming Markov random fields of bounded treewidth and probabilistic context free grammars. Case-factor diagrams are based on a variant of BDDs (binary decision diagram [17]) with both zero suppression and "factor nodes". Factor nodes are analogous to the AND nodes in an AND/OR search space. A case-factor diagram can be viewed as an AND/OR search space in which each outgoing arc from an OR node is explicitly labeled with an assignment of a value to a variable. Zero suppression is used to fix the value of variables not mentioned in a given solution. Zero suppression allows the formalism to concisely represent probabilistic context free grammars as functions from variable-value assignments to log probabilities (or energies).

7.6. AND/OR-search graphs and compilation

The authors have proposed in [42] the compilation of constraint networks into AND/OR Multi-Valued Decision Diagrams (AOMDDs). This is essentially the strongly minimal AND/OR graph representation of a constraint network with redundant variables removed for conciseness. An algorithm that achieves this is structurally similar to Variable Elimination. It uses a bottom up traversal of a bucket tree, and at each node an APPLY operator is used to combine all the AOMDDs of the bucket into another AOMDD. The APPLY is similar to the OBDD apply operator [17], but is adapted for AND/OR structures. Essentially, the AOMDD extends an OBDD (or a multi-valued decision diagram) with an AND/OR structure.

7.6.1. Relationship with d-DNNF

An AND/OR structure restricted to propositional theories is very similar to d-DNNF [18]. One can show a one-to-one linear translation from an AND/OR bi-valued tree of a propositional CNF theory into a d-DNNF. The AND/OR structure is more restrictive allowing disjunction only on the variable's value while in d-DNNF disjunction is allowed on more complex expressions; see [46] for implications of this distinction. The AND/OR search graph is built on top of a graphical model and can be viewed as a compiled scheme of a CNF into an AND/OR structure. Since an AND/OR search can be expressed as a d-DNNF, the construction via pseudo tree yields a scheme for d-DNNF compilation. In other words, given a CNF theory, the algorithm can be applied using a pseudo tree to yield an AND/OR graph, which can be transformed in linear time and space into a d-DNNF.

Conversely, given a d-DNNF that is specialized to variable-based disjunction for OR nodes, it is easy to create an AND/OR graph or a tree that is equivalent having a polynomially equivalent size. The AND/OR search graph for probabilistic networks is also closely related to algebraic circuits of probabilistic networks [19] which is an extension of d-DNNF to this domain.

7.6.2. Relationship with OBDDs

The notion of minimal OR search graphs is also similar to the known concept of *Ordered Binary Decision Diagrams* (*OBDD*) in the literature of hardware and software design and verification The properties of OBDDs were studied extensively in the past two decades [17,47].

It is well known that the size of the minimal OBDD is bounded exponentially by the *pathwidth* of the CNF's primal graph and that the OBDD is unique for a fixed variable ordering. Our notion of backtrack-free minimal AND/OR search graphs, if applied to CNFs, resembles *tree BDDs* [48]. Minimal AND/OR graphs are also related to Graph-driven BDDs (called G-FBDD) [49,50] in that they are based on a partial order expressed in a directed graph. Still, a G-FBDD has an OR structure, whose ordering is restricted to some partial orders, but not an AND/OR structure. For example, the OBDD based on a DFS ordering of a pseudo tree is a G-FBDD. Some other relationships between graphical model compilation and OBDDs were studied in [18].

In summary, putting OBDDs within our terminology, an OBDD representation of a CNF formula is a strongly minimal OR search graph where redundant nodes are removed.

7.6.3. Relationship with tree-driven automata

Fargier and Vilarem [21] proposed the compilation of CSPs into tree-driven automata, which have many similarities to the work in [42]. In particular, the compiled tree-automata proposed there is essentially the same as the AND/OR

multi-valued decision diagram. Their main focus is the transition from linear automata to tree automata (similar to that from OR to AND/OR), and the possible savings for tree-structured networks and hyper-trees of constraints due to decomposition. Their compilation approach is guided by a tree-decomposition while ours is guided by a variable-elimination based algorithms. And, it is well known that Variable Elimination and cluster-tree decomposition are, in principle, the same [24].

7.7. Relationship with disjoint support decomposition

The work on Disjoint Support Decompositions (DSD) [22] was proposed in the area of design automation [51], as an enhancement for BDDs aimed at exploiting function decomposition. The main common aspect of DSD and AOMDD [42] is that both approaches show how structure decomposition can be exploited in a BDD-like representation. DSD is focused on Boolean functions and can exploit more refined structural information that is inherent to Boolean functions. In contrast, AOMDDs assume only the structure conveyed in the constraint graph, and are therefore more broadly applicable to any constraint expression and also to graphical models in general. They allow a simpler and higher level exposition that yields graph-based bounds on the overall size of the generated AOMDD.

7.7.1. Relationship with semi-ring BDDs

In recent work [23] OBDDs were extended to semi-ring BDDs. The semi-ring treatment is restricted to the OR search spaces, but allows dynamic variable ordering. It is otherwise very similar in aim and scope to our strongly minimal AND/OR graphs. When restricting the strongly minimal AND/OR graphs to OR graphs only, the two are closely related, except that we express BDDs using the Shenoy–Shafer axiomatization that is centered on the two operation of combination and marginalization rather then on the semi-ring formulation. Minimality in the formulation in [23] is more general allowing merging nodes having different values and therefore can capture symmetries (called interchangeability).

8. Conclusions

The primary contribution of this paper is in viewing search for graphical models in the context of AND/OR search spaces rather than OR spaces. We introduced the AND/OR search tree, and showed that its size can be bounded exponentially by the depth of its pseudo tree over the graphical model. This implies exponential savings for any linear space algorithms traversing the AND/OR search tree. Specifically, if the graphical model has treewidth w^*, the depth of the pseudo tree is $O(w^* \cdot \log n)$.

The AND/OR search tree was extended into a graph by merging identical subtrees. We showed that the size of the minimal AND/OR search graph is exponential in the treewidth while the size of the minimal OR search graph is exponential in the pathwidth. Since for some graphs the difference between treewidth and pathwidth is substantial (e.g., balanced pseudo trees) the AND/OR representation implies substantial time and space savings for memory intensive algorithms traversing the AND/OR graph. Searching the AND/OR search *graph* can be implemented by goods caching during search, while no-good recording is interpreted as pruning portions of the search space independent of it being a tree or a graph, an OR or an AND/OR. For finding a single solution, pruning the search space is the most significant action. For counting and probabilistic inference, using AND/OR graphs can be of much help even on top of no-good recording.

We observe that many known advanced algorithms for constraint processing and satisfiability can be explained as traversing the AND/OR search tree (e.g., backjumping [3,8,37]). Also, recent algorithms in probabilistic reasoning such as Recursive Conditioning [12] and Value Elimination [13] can operate in linear space and can be viewed as searching the AND/OR search tree. In their memory intensive mode, these algorithms were noted to search the AND/OR graph, having similar time and space complexities. Also, as noted, recent work [10] proposes search guided by a tree decomposition either for constraint satisfaction or optimization, and is searching the AND/OR search *graph*, whose pseudo tree is constructed along the tree decomposition.

Acknowledgements

This work was supported in part by the NSF grant IIS-0412854 and by the MURI ONR award N00014-00-1-0617.

Appendix A. Proofs

Proof of Theorem 29 (correctness). 1) By definition, all the arcs of $S_{\mathcal{T}}(\mathcal{R})$ are consistent. Therefore, any solution tree of $S_{\mathcal{T}}(\mathcal{R})$ denotes a solution for $\mathcal{R}$ whose assignments are all the labels of the AND nodes in the solution tree. Also, by definition of the AND/OR tree, every solution of $\mathcal{R}$ must corresponds to a solution subtree in $S_{\mathcal{T}}(\mathcal{R})$. 2) By construction, the arcs in every solution tree have weights such that each function of F contributes to one and only one weight via the combination operator. Since the total weight of the tree is derived by combination, it yields the cost of a solution.

Proof of Theorem 30 (size bounds of AND/OR search tree). Let p be an arbitrary directed path in the DFS tree $\mathcal{T}$ that starts with the root and ends with a leaf. This path induces an OR search subtree which is included in the AND/OR search tree $S_{\mathcal{T}}$, and its size is $O(k^m)$, when m bounds the path length. The DFS tree $\mathcal{T}$ is covered by l such directed paths, whose lengths are bounded by m. The union of their individual search trees covers the whole AND/OR search tree $S_{\mathcal{T}}$, where every distinct full path in the AND/OR tree appears exactly once, and therefore, the size of the AND/OR search tree is bounded by $O(l \cdot k^m)$. Since $l \leqslant n$ and $l \leqslant b^m$, it concludes the proof.

Proof of Proposition 31. The proof is similar to that of Theorem 30, only each node contributes with its actual domain size rather than the maximal one, and each path to a leaf in $\mathcal{T}$ contributes with its actual depth, rather than the maximal one.

Proof of Theorem 34 (properties of AND/OR search trees). All the arguments in the proof of Theorem 29 carry immediately to AND/OR search spaces that are defined relative to a pseudo tree. Likewise, the bound size argument in the proof of Theorem 30 holds relative to the depth of the more general pseudo tree.

Proof of Proposition 42. First, we should note that if $\mathcal{T}$ is a pseudo tree of $\mathcal{R}$ and if d is a DFS ordering of $\mathcal{T}$, then $\mathcal{T}$ is also a pseudo tree of $E_d(\mathcal{R})$ and therefore $S_{\mathcal{T}}(E_d(\mathcal{R}))$ is a faithful representation of $E_d(\mathcal{R})$. $E_d(\mathcal{R})$ is equivalent to $\mathcal{R}$, therefore $S_{\mathcal{T}}(E_d(\mathcal{R}))$ is a supergraph of $BF_{\mathcal{T}}(\mathcal{R})$. We only need to show that $S_{\mathcal{T}}(E_d(\mathcal{R}))$ does not contain any dead-ends, in other words any consistent partial assignment must be extendable to a solution of $\mathcal{R}$. Adaptive consistency makes $E_d(\mathcal{R})$ strongly directional $w^*(d)$ consistent, where $w^*(d)$ is the induced width of $\mathcal{R}$ along ordering d [35]. It follows from this that either $\mathcal{R}$ is inconsistent, in which case the proposition is trivially satisfied, both trees being empty, or else any consistent partial assignment in $S_{\mathcal{T}}(E_d(\mathcal{R}))$ can be extended to the next variable in d, and therefore no dead-end is encountered.

Proof of Proposition 48 (minimal graph). (1) All we need to show is that the *merge* operator is not dependent on the order of applying the operator. Mergeable nodes can only appear at the same level in the AND/OR graph. Looking at the initial AND/OR graph, before the merge operator is applied, we can identify all the mergeable nodes per level. We prove the proposition by showing that if two nodes are initially mergeable, then they must end up merged after the operator is applied exhaustively to the graph. This can be shown by induction over the level where the nodes appear.

Base case: If the two nodes appear at the leaf level (level 0), then it is obvious that the exhaustive merge has to merge them at some point.

Inductive step: Suppose our claim is true for nodes up to level k and two nodes n_1 and n_2 at level $k+1$ are initially identified as mergeable. This implies that, initially, their corresponding children are identified as mergeable. These children are at level k, so it follows from the inductive hypothesis that the exhaustive merge has to merge the corresponding children. This in fact implies that nodes n_1 and n_2 will root the same subgraph when the exhaustive merge ends, so they have to end up merged. Since the graph only becomes smaller by merging, based on the above the process of merging has to stop at a fix point.

(2) Analogous to (1). (3) If the nodes can be merged, it follows that the subgraphs are identical, which implies that they define the same conditioned subproblems, and therefore the nodes can also be unified.

Proof of Proposition 52. Parts 1 and 2 follow from definitions. Regarding claim 3, for the graph coloring problem in Fig. 1(a), the minimal AND-OR search graph is identical to its explicit AND/OR search graph, $G_{\mathcal{T}}$ (see Fig. 11).

Proof of Proposition 53. In tree models, the functions are only over two variables. Therefore, after an assignment $\langle X, x\rangle$ is made and the appropriate weight is given to the arc from X to $\langle X, x\rangle$, the variable X and all its ancestors in the pseudo tree do not contribute to any arc weight below in the AND/OR search tree. Therefore, the conditioned subproblems rooted at any AND node labeled by $\langle X, x\rangle$ depend only on the assignment of X to x (and do not depend on any other assignment on the current path), so it follows that all the AND nodes labeled by $\langle X, x\rangle$ can be merged. Since the equivalence of AND/OR search spaces is preserved by merge, the explicit AND/OR search graph is equivalent to $S_{\mathcal{T}}$. At each AND level in the explicit graph there are at most k values, and therefore its size is $O(nk)$.

Proof of Theorem 54. The size of an explicit AND/OR *graph* of a tree model was shown to be $O(nk)$ (Proposition 52), yielding $O(r \cdot k^{w^*})$ size for the explicit AND/OR graph, because k is replaced by k^{w^*}, the number of possible assignments to a cluster of scope size w^*, and r replaces n.

Proof of Proposition 56. (1) The induced width of G relative to a given pseudo tree is always greater than w^*, by definition of w^*. It remains to show that there exists a pseudo tree $\mathcal{T}$ such that $w_{\mathcal{T}}(G) = w^*$. Consider an ordering d that gives the induced width w^*. The ordering d defines a bucket tree BT (see Definition 36), which can also be viewed as a pseudo tree for the AND/OR search, therefore $w_{BT}(G) = w^*$. (2) Analogous to (1).

Proof of Proposition 60. Both claims follow directly from Definitions 58 and 59.

Proof of Theorem 61 (context based merge). (1) The conditioned graphical models (Definition 10) at n_1 and n_2 are defined by the functions whose scopes are not fully assigned by π_{n_1} and π_{n_2}. Since n_1 and n_2 have the same labeling $\langle X_i, x_i\rangle$, it follows that $var(\pi_{n_1}) = var(\pi_{n_2})$, and therefore the two conditioned subproblems are based on the same set of functions, let's call it $F|_{var(\pi_{n_1})}$. The scopes of functions in $F|_{var(\pi_{n_1})}$ determine connections in the primal graph between ancestors of X_i and its descendants. Therefore, the only relevant variables that define the restricted subproblems are those in pas_i, and Eq. (1) ensures that they have identical assignments. It follows that the conditioned subproblems are identical, and n_1 and n_2 can be merged.

(2) Analogous to (1).

Proof of Theorem 65. The number of different nodes in the context minimal AND/OR search graph, $C_{\mathcal{T}}$, does not exceed the number of contexts. From Eqs. (1) and (2) we see that, for any variable, the number of contexts is bounded by the number of possible instantiations of the largest context in $G^{\mathcal{T}^*}$, which is bounded by $O(k^w)$. For all the n variables, the bound $O(n \cdot k^w)$ follows.

Proof of Theorem 67. We can generate $C_{\mathcal{T}}$ using depth-first or breadth-first search which caches all nodes via their contexts and avoids generating duplicate searches for the same contexts. Therefore, the generation of the search graph is linear in its size, which is exponential in w and linear in n.

Proof of Proposition 73. Let $B_1 = BF_{\mathcal{T}}(\mathcal{R}_1)$ and $B_2 = BF_{\mathcal{T}}(\mathcal{R}_2)$ be the corresponding backtrack-free AND/OR search trees of $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively. Namely, $BF_{\mathcal{T}}(\mathcal{R}_1) \subseteq S_{\mathcal{T}}(\mathcal{R}_1)$, $BF_{\mathcal{T}}(\mathcal{R}_2) \subseteq S_{\mathcal{T}}(\mathcal{R}_2)$. Clearly they are subtrees of the same full AND/OR tree. We claim that a path appears in B_1 iff it appears in B_2. If not, assume without loss of generality that there exists a path in B_1, π, which does not exists in B_2. Since this is a backtrack-free search tree, every path appears in some solution and therefore there is a solution subtree in B_1 that includes π which does not exist in B_2, contradicting the assumption that $\mathcal{R}_1$ and $\mathcal{R}_2$ have the same set of solutions. The second part has an identical proof based on flat functions (see introduction to Section 3).

Proof of Theorem 74. From Proposition 73 we know that $\mathcal{R}_1$ and $\mathcal{R}_2$ have the same backtrack-free AND/OR tree. Since the backtrack-free AND/OR search tree for a backbone tree $\mathcal{T}$ uniquely determines the strongly minimal AND/OR graph, the theorem follows.

Proof of Proposition 78 (recursive value computation). The proof is by induction over the number of levels in the AND/OR graph.

Basis step: If the graph has only two levels, one OR and one AND, then the claim is straightforward because the AND leaves are labeled by "1" if consistent and the OR node accumulates "1" or the sum of consistent values below, or "0" if there is no consistent value.

Inductive step: Assuming the proposition holds for k pairs of levels (one AND and one OR in each pair), proving it holds for $k+1$ pairs of levels is similar to the basis step, only the labeling of the top AND nodes is the sum of solutions below in the case of counting.

Proof of Proposition 80. The proof is again by induction, similar to the proof of Proposition 78. For simplicity of writing, the projection operator here takes as arguments the set of variables that are eliminated.

Basis step: If the model has only one variable, then the claim is obvious.

Inductive step: Let X be an OR node in the graph. Assume that the value of each OR node below it is the solution to the reasoning problem corresponding to the conditioned subproblem rooted by it. We need to prove that the value of X will be the solution to the reasoning problem of the conditioned subproblem rooted by X. Suppose X has children $Y_1, \ldots, Y_m$ in the pseudo tree. We have $v(Y_i) = \Downarrow_{Y_i \cup Desc(Y_i)} \bigotimes_{f \in F|_{\pi_{Y_i}}} f$, where $Desc(Y_i)$ are the descendants of Y_i, and the functions are restricted on the current path. Each AND node $\langle X, x\rangle$ will combine the values below. Because the sets $Y_i \cup Desc(Y_i)$ are pairwise disjoint, the marginalization operator commutes with the combination operator and we get:

$$v\big(\langle X, x\rangle\big) = \bigotimes_{i=1}^{m} \Downarrow_{Y_i \cup Desc(Y_i)} \bigotimes_{f \in F|_{\pi_{Y_i}}} f = \Downarrow_{\bigcup_{i=1}^{m}(Y_i \cup Desc(Y_i))} \bigotimes_{f \in F|_{\pi_X}} f.$$

The values $v(\langle X, x\rangle)$ are then combined with the values of the bucket of X, which are the weights $w_{(X,\langle X,x\rangle)}$. The functions that appear in the bucket of X do not contribute to any of the weights below Y_i, and therefore the marginalization over $\bigcup_{i=1}^{m}(Y_i \cup Desc(Y_i))$ can commute with the combination that we have just described:

$$w_{(X,\langle X,x\rangle)} \otimes v\big(\langle X, x\rangle\big) = \Downarrow_{\bigcup_{i=1}^{m}(Y_i \cup Desc(Y_i))} w_{(X,\langle X,x\rangle)} \otimes \Bigg(\bigotimes_{f \in F|_{\pi_X}} f \Bigg).$$

Finally, we get:

$$v(X) = \Downarrow_X w_{(X,\langle X,x\rangle)} \otimes v\big(\langle X, x\rangle\big) = \Downarrow_{X \cup Desc(X)} \bigotimes_{f \in F|_{\pi_X}} f.$$

Proof of Proposition 84. A bucket tree can be built by having a cluster for each variable X_i and its parents pa_i, and following the structure of the pseudo tree $\mathcal{T}$. Some of the clusters may not be maximal, and they have a one to one correspondence to the variables with dead caches. The parents pa_i that are not dead caches correspond to separators between maximal clusters in the bucket tree.

References

[1] N.J. Nilsson, Principles of Artificial Intelligence, Tioga, Palo Alto, CA, 1980.
[2] E.C. Freuder, M.J. Quinn, Taking advantage of stable sets of variables in constraint satisfaction problems, in: Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI'85), 1985, pp. 1076–1078.
[3] E.C. Freuder, M.J. Quinn, The use of lineal spanning trees to represent constraint satisfaction problems, Tech. Rep. 87-41, University of New Hampshire, Durham (1987).
[4] Z. Collin, R. Dechter, S. Katz, On the feasibility of distributed constraint satisfaction, in: Proceedings of the Twelfth International Conference of Artificial Intelligence (IJCAI'91), 1991, pp. 318–324.
[5] Z. Collin, R. Dechter, S. Katz, Self-stabilizing distributed constraint satisfaction, The Chicago Journal of Theoretical Computer Science 3 (4) (1999), special issue on self-stabilization.
[6] P.J. Modi, W. Shena, M. Tambea, M. Yokoo, Adopt: asynchronous distributed constraint optimization with quality guarantees, Artificial Intelligence 161 (2005) 149–180.
[7] R. Dechter, Constraint networks, in: Encyclopedia of Artificial Intelligence, 1992, pp. 276–285.

[8] R. Bayardo, D. Miranker, A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem, in: Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96), 1996, pp. 298–304.
[9] J. Larrosa, P. Meseguer, M. Sanchez, Pseudo-tree search with soft constraints, in: Proceedings of the European Conference on Artificial Intelligence (ECAI'02), 2002, pp. 131–135.
[10] C. Terrioux, P. Jégou, Hybrid backtracking bounded by tree-decomposition of constraint networks, Artificial Intelligence 146 (2003) 43–75.
[11] C. Terrioux, P. Jégou, Bounded backtracking for the valued constraint satisfaction problems, in: Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'03), 2003, pp. 709–723.
[12] A. Darwiche, Recursive conditioning, Artificial Intelligence 125 (1–2) (2001) 5–41.
[13] F. Bacchus, S. Dalmao, T. Pitassi, Value elimination: Bayesian inference via backtracking search, in: Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI'03), 2003, pp. 20–28.
[14] F. Bacchus, S. Dalmao, T. Pitassi, Algorithms and complexity results for #sat and bayesian inference, in: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03), 2003, pp. 340–351.
[15] T. Sang, F. Bacchus, P. Beam, H. Kautz, T. Pitassi, Combining component caching and clause learning for effective model counting, in: Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04), 2004.
[16] R. Dechter, Bucket elimination: A unifying framework for reasoning, Artificial Intelligence 113 (1999) 41–85.
[17] R.E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Transaction on Computers 35 (1986) 677–691.
[18] A. Darwiche, P. Marquis, A knowledge compilation map, Journal of Artificial Intelligence Research (JAIR) 17 (2002) 229–264.
[19] A. Darwiche, A differential approach to inference in Bayesian networks, Journal of the ACM 50 (3) (2003) 280–305.
[20] D. McAllester, M. Collins, F. Pereira, Case-factor diagrams for structured probabilistic modeling, in: Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04), 2004, pp. 382–391.
[21] H. Fargier, M. Vilarem, Compiling csps into tree-driven automata for interactive solving, Constraints 9 (4) (2004) 263–287.
[22] V. Bertacco, M. Damiani, The disjunctive decomposition of logic functions, in: ICCAD, International Conference on Computer-Aided Design, 1997, pp. 78–82.
[23] N. Wilson, Decision diagrams for the computation of semiring valuations, in: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05), 2005, pp. 331–336.
[24] R. Dechter, J. Pearl, Tree clustering for constraint networks, Artificial Intelligence 38 (1989) 353–366.
[25] S.A. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability—a survey, BIT 25 (1985) 2–23.
[26] H.L. Bodlaender, J.R. Gilbert, Approximating treewidth, pathwidth and minimum elimination tree-height, Tech. rep., Utrecht University (1991).
[27] H.L. Bodlaender, Treewidth: Algorithmic techniques and results, in: The Twenty Second International Symposium on Mathematical Foundations of Computer Science (MFCS'97), 1997, pp. 19–36.
[28] R. Dechter, A new perspective on algorithms for optimizing policies under uncertainty, in: International Conference on Artificial Intelligence Planning Systems (AIPS-2000), 2000, pp. 72–81.
[29] P. Shenoy, Valuation-based systems for bayesian decision analysis, Operations Research 40 (1992) 463–484.
[30] J. Pearl, Probabilistic Reasoning in Intelligent Systems, Morgan Kaufmann, 1988.
[31] R. Marinescu, R. Dechter, AND/OR branch-and-bound for graphical models, in: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05), 2005, pp. 224–229.
[32] D. Allen, A. Darwiche, New advances in inference by recursive conditioning, in: Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI'03), 2003, pp. 2–10.
[33] K. Kask, R. Dechter, J. Larrosa, A. Dechter, Unifying cluster-tree decompositions for reasoning in graphical models, Artificial Intelligence 166 (1–2) (2005) 165–193.
[34] R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[35] R. Dechter, J. Pearl, Network-based heuristics for constraint satisfaction problems, Artificial Intelligence 34 (1987) 1–38.
[36] I. Rish, R. Dechter, Resolution vs. search; two strategies for sat, Journal of Automated Reasoning 24 (1/2) (2000) 225–275.
[37] R. Dechter, Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition, Artificial Intelligence 41 (1990) 273–312.
[38] R.J. Bayardo, R.C. Schrag, Using csp look-back techniques to solve real world sat instances, in: Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97), 1997, pp. 203–208.
[39] J.P. Marques-Silva, K.A. Sakalla, Grasp: A search algorithm for propositional satisfiability, IEEE Transaction on Computers 48 (5) (1999) 506–521.
[40] N. Robertson, P. Seymour, Graph minors i. excluding a forest, J. Combin. Theory Ser. B 35 (1983) 39–61.
[41] D. Bienstock, N. Robertson, P. Seymour, R. Thomas, Quickly excluding a forest, J. Combin. Theory Ser. B 52 (1991) 274–283.
[42] R. Mateescu, R. Dechter, Compiling constraint networks into AND/OR multi-valued decision diagrams (AOMDDs), in: Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP'06), 2006, pp. 329–343.
[43] D.H. Frost, Algorithms and heuristics for constraint satisfaction problems, Tech. rep., Ph.D. thesis, Information and Computer Science, University of California, Irvine (1997).
[44] R. Mateescu, R. Dechter, The relationship between AND/OR search and variable elimination, in: Proceedings of the Twenty First Conference on Uncertainty in Artificial Intelligence (UAI'05), 2005, pp. 380–387.
[45] R. Dechter, D. Frost, Backjump-based backtracking for constraint satisfaction problems, Artificial Intelligence 136 (2) (2002) 147–188.
[46] J. Huang, A. Darwiche, Dpll with a trace: From sat to knowledge compilation, in: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05), 2005, pp. 156–162.
[47] K.L. McMillan, Symbolic Model Checking, Kluwer Academic, 1993.

[48] K.L. McMillan, Hierarchical representation of discrete functions with application to model checking, in: Computer Aided Verification, 1994, pp. 41–54.
[49] J. Gergov, C. Meinel, Efficient boolean manipulation with obdds can be extended to fbdds, IEEE Trans. Computers 43 (1994) 1197–1209.
[50] D. Sieling, I. Wegner, Graph driven BDDs—a new data structure for boolean functions, Theoretical Computer Science 141 (1994) 283–310.
[51] R. Brayton, C. McMullen, The decomposition and factorization of boolean expressions, in: ISCAS, Proceedings of the International Symposium on Circuits and Systems, 1982, pp. 49–54.