AND/OR Branch-and-Bound for Probabilistic Inference

Kustaa Kangas
Seminar: Heuristic Search
Autumn, 2013

Abstract

The Most Probable Explanation problem asks for the most probable assignment on random variables of a Bayesian network, when the values of some variables are known. We present the AND/OR search technique for solving the problem, combined with heuristic Branch-and-Bound pruning. We also present the mini-bucket elimination technique for bounded inference and show how it can be used to produce heuristic functions for AND/OR search.

1 INTRODUCTION

Bayesian networks are probabilistic models that represent conditional dependencies between random variables with a directed acyclic graph. A network defines the local distribution of each variable conditioned on its parents in the network, and the joint distribution of all variables factorizes into a product of the local distributions. This factorization admits efficient inference in the network, such as computing prior and posterior probabilities. In this report we focus on the Most Probable Explanation problem (MPE), which asks for the most probable assignment of the variables, given a set of evidence variables whose values are known. Although the problem is NP-hard, algorithms can significantly improve upon brute-force evaluation by exploiting the structure of the network. In this report, we present the AND/OR search technique for solving the MPE, formalized for graphical models in [3]. In addition to making use of structural independencies in the network, AND/OR search also admits a heuristic Branch-and-Bound technique to further narrow down the search space [5].

We begin by defining Bayesian networks and the MPE problem formally in Section 2. We present the basic AND/OR search in Section 3 and the Branch-and-Bound search in Section 4. In section 5 we discuss heuristic functions for the MPE problem, based on approximations of alternative inference algorithms such as variable elimination. We conclude the report in Section 6. Throughout, we use boldface to denote sets of random variables, e.g. $\mathbf{X}$, and their instantiations, $\mathbf{x}$. We occasionally write $P(x)$ short for $P(X = x)$.

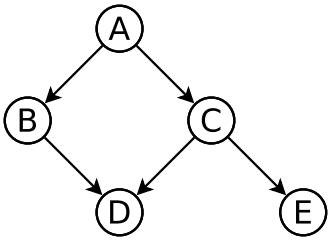

Figure 1: A simple Bayesian network structure, representing the joint distribution $P(A, B, C, D, E) = P(A) \cdot P(B|A) \cdot P(C|A) \cdot P(D|B, C) \cdot P(E|C)$.

2 PRELIMINARIES

2.1 Bayesian networks

A Bayesian network is a pair (G, P) where G is a directed acyclic graph on random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ with finite domains $\mathbf{D} = \{D_1, \ldots, D_n\}$ and P is the joint distribution of the variables. The key property of P is that it factorizes according to the network structure as

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|Pa_i) \qquad (1)$$

where Pa_i denotes the parents of X_i in G. For each X_i, the probabilities $P(X_i|Pa_i)$ are stored in a conditional probability table (CPT). A CPT has a single value for each joint assignment of X_i and Pa_i and thus has a size exponential only in $|Pa_i|$, as opposed to encoding the distribution in a single table. An example of Bayesian network is illustrated in Fig. 1.

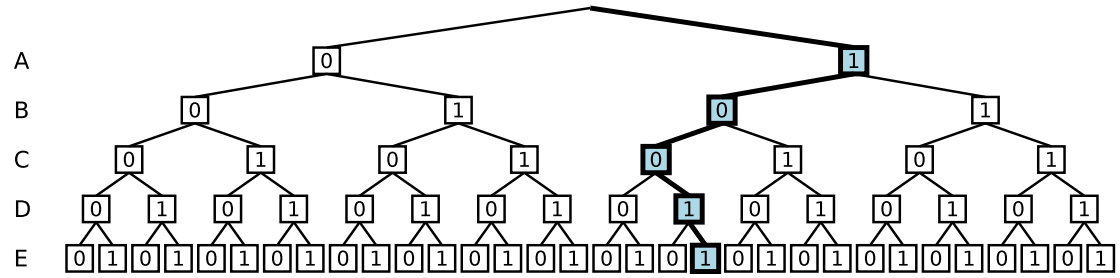

Figure 2: An OR search tree for binary variables A–E, each level corresponding to a variable instantiation. A solution path corresponding to the assignment $(A, B, C, D, E) = (1, 0, 0, 1, 1)$ is highlighted.

2.2 The MPE problem

Given a Bayesian network G on variables $\mathbf{X}$ and an assignment $\mathbf{e}$ of a subset $\mathbf{E}$ of the variables, the MPE problem asks for an assignment $\mathbf{m}$ of the remaining variables $\mathbf{M} = \mathbf{X} \setminus \mathbf{E}$, such that the joint probability $P(\mathbf{M} = \mathbf{m}|\mathbf{E} = \mathbf{e}) = P(\mathbf{M} = \mathbf{m}, \mathbf{E} = \mathbf{e})$ is maximized. To simplify the presentation of algorithms, we will assume for the remainder of the report that $\mathbf{E}$ is empty, i.e. we simply want to find the most probable assignment of $\mathbf{X}$. In the end, we will briefly describe a straightforward way to modify the algorithms to accommodate evidence variables.

A brute-force algorithm solves the MPE problem by evaluating $P(\mathbf{X})$ for every possible assignment of $\mathbf{X}$. A single assignment is evaluated using Equation 1 and the number of such assignments is equal to the product of the cardinalities of the variables, i.e. $\prod_i |\mathbf{D}_i|$. The brute-force search can be visualized as a search tree where traversing an edge corresponds to instantiating a single variable and a path from the root to a leaf corresponds to an assignment of all variables (Fig. 2).

A search of this kind is called OR search as each non-leaf node makes a decision between several mutually exclusive choices. OR search is not limited to algorithms that evaluate the entire search tree as it is often possible to prune out suboptimal subtrees, based on the best solution found so far. In many cases, however, it is more efficient to first reduce the search tree itself by making use of structural independencies in the Bayesian network. This leads to an extension of OR search trees known as AND/OR search trees, which we will present in the following section.

3 AND/OR SEARCH

In this section we present the AND/OR search method that can be seen as an extension of OR search. AND/OR search is applicable to any optimization problem that can be formalized as a so called graphical model [3]. We present the method in this general setting, using the MPE problem as a concrete example. We start with a formal definition of graphical models and AND/OR trees and then describe a simple algorithm for traversing them.

3.1 Graphical models

Definition 1 (Graphical model) *A* graphical model *is a 3-tuple (*$\mathbf{X}$*,* $\mathbf{F}$*,* $\bigotimes$*) where*

1. $\mathbf{X} = \{X_1, \ldots, X_n\}$ *is a set of variables with finite domains* $\mathbf{D} = \{D_1, \ldots, D_n\}$.

2. $\mathbf{F} = \{f_1, \ldots, f_m\}$ *is a collection of functions. Each* f_i *is defined on a subset* $\mathbf{S}_i$ *of* $\mathbf{X}$ *and maps each instantiation of* $\mathbf{S}_i$ *to a real number.*

3. *The operator* $\bigotimes$ *is either sum* $\sum$ *or product* $\prod$.

A graphical model represents the following problem: find an assignment $\mathbf{x}$ of the variables $\mathbf{X}$ such that the cost function $c(\mathbf{x}) = \bigotimes_{f \in \mathbf{F}} f(\mathbf{x})$ is maximized (or minimized). In other words, we wish to maximize either the sum or product of the functions $\mathbf{F}$ evaluated at $\mathbf{x}$. In the case of MPE, $\mathbf{X}$ is the set of random variables of a Bayesian network, $\mathbf{F}$ is the set of functions $f_i(X_i, Pa_i) = P(X_i|Pa_i)$ and $\bigotimes = \prod$. Thus, $c(\mathbf{x}) = \prod_{i=1}^{n} P(X_i|Pa_i) = P(\mathbf{X} = \mathbf{x})$, which is the probability we want to maximize. For an abstract model we shall call $\bigotimes$ the *combination* of functions.

Clearly, for any graphical model we could perform an OR search and evaluate the cost function at each leaf of the search tree. However, observe that we do not need to instantiate all variables to evaluate a single $f \in \mathbf{F}$, only the variables in the scope $\mathbf{S}_i$ of f. In many cases this fact allows us to split the search into subproblems that can be solved independently of each other. This idea leads to a search space that we call an AND/OR search tree.

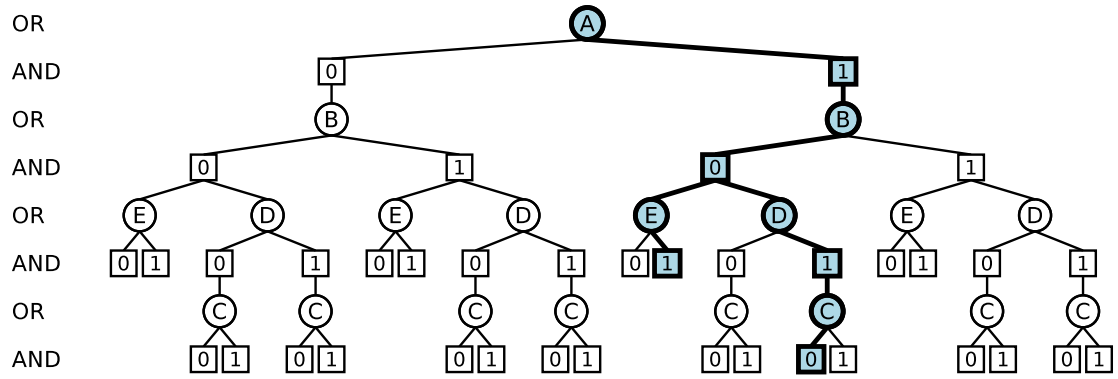

Figure 4: The complete AND/OR search tree induced by the pseudo tree in Fig. 3. A solution tree corresponding to the assignment $(A, B, C, D, E) = (1, 0, 0, 1, 1)$ is highlighted.

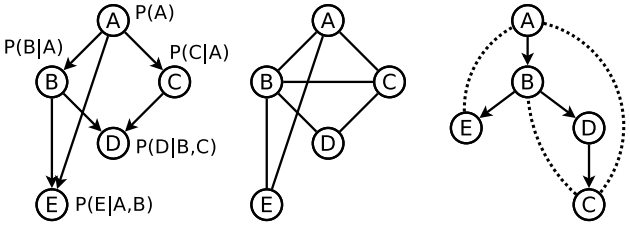

Figure 3: A Bayesian network on variables A–E with the respective CPTs (left), the primal graph of the network (center), and a pseudo tree on the primal graph (right). Dashed lines represent back-edges.

3.2 AND/OR search trees

In order to present AND/OR trees formally we first define the concept of *primal graphs* and *pseudo trees*.

Definition 2 (Primal graph) *The primal graph of a graphical model on variables* $\mathbf{X}$ *and functions* $\mathbf{F}$ *is an undirected graph* $(\mathbf{X}, E)$ *such that* E *contains an edge between a pair of variables if the variables appear together in the scope of some function in* $\mathbf{F}$.

The primal graph of a Bayesian network is also called a *moral graph*. A pair of variables are joined if they appear together in any CPT. Equivalently, a variable is connected to each variable appearing in its Markov blanket, that is, its parents, its children, and the parents of its children in the Bayesian network. Any primal graph has at least one *pseudo tree*.

Definition 3 (Pseudo tree) *Let* $G = (V, E)$ *be a primal graph and let* $\mathcal{T} = (V, E')$ *be a rooted tree on the vertices* V. *(The set* E' *does not need to be a subset of* E *but may contain edges between any pairs of vertices.) We say that* $\mathcal{T}$ *is a* pseudo tree *of* G *if every edge in* $E \setminus E'$ *is a back-edge, i.e., one end point of the edge is an ancestor of the other in* $\mathcal{T}$.

Fig. 3 illustrates the primal graph of a Bayesian network and one possible pseudo tree for it. The pseudo tree determines an order in which variables are instantiated and which variables are instantiated independently of others. For example, after assigning a value to A and B, the two subtrees E and $D \rightarrow C$ become independent subproblems as there are no functions that depend on variables in both subtrees. Note that this is equivalent to saying that all edges appearing in the primal graph but not in the pseudo tree are back-edges, as required by the definition. We may now define the AND/OR search tree induced by a pseudo tree.

Definition 4 (AND/OR search tree) *Let* $\mathcal{T}$ *be a pseudo tree of a graphical model on variables* $\mathbf{X}$. *The pseudo tree induces an AND/OR search tree, denoted* $S_{\mathcal{T}}$, *which contains alternating levels of OR and AND nodes. Each OR node is labeled by a variable* X_i *and each AND node by an instantiation* $X_i = x_i$ *(or* x_i *for short) of a variable. The root of* $S_{\mathcal{T}}$ *is an OR node labeled by the root of* $\mathcal{T}$. *Each OR node labeled* X_i *has a child AND node labeled* $X_i = x_i$ *for each* $x_i \in D_i$. *Each AND node labeled* $X_i = x_i$ *has a child OR node labeled* X_j *for each child* X_j *of* X_i *in* $\mathcal{T}$.

The AND/OR search tree induced by the pseudo tree in Fig. 3 is shown in Fig. 4. Semantically, OR nodes represent the choice between alternative solutions to the problem, i.e. alternative variable instantiations, while AND nodes represent decomposition into independent subproblems, conditioned on the assignments made at their ancestor nodes.

Note that if the pseudo tree is a chain, the induced search tree is equivalent to an OR tree. Conversely, shallow pseudo trees have much decomposition and induce smaller search spaces. Finding pseudo trees of minimum depth is an NP-complete problem, which we do not discuss further in this report. Some approximations can be found in [5].

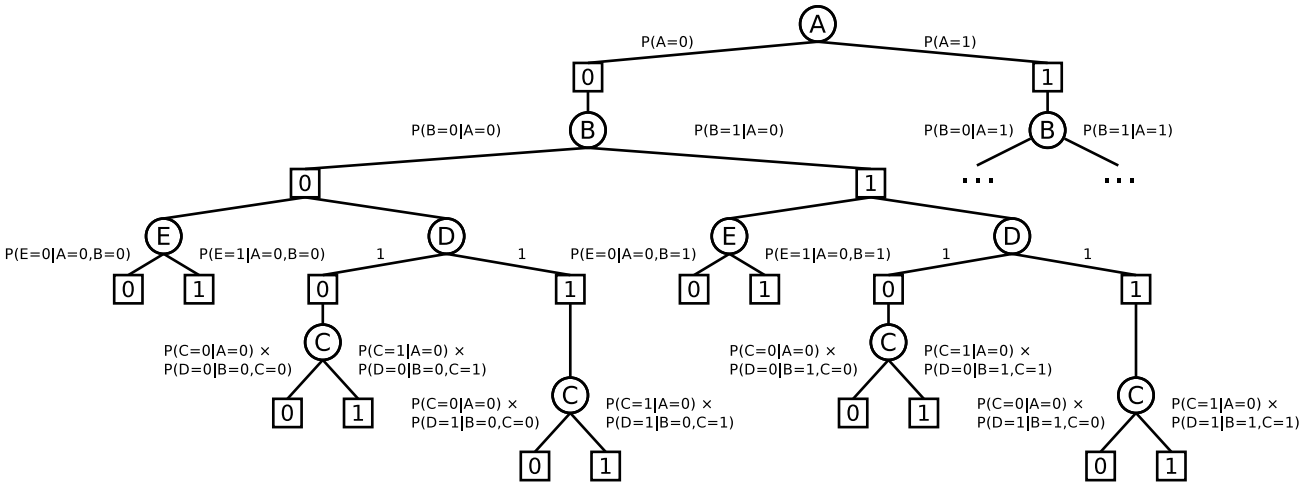

Figure 5: Part of the tree in Fig. 4, showing the weights of OR-to-AND edges.

Contrast to an OR tree where a solution is a path from the root to a leaf, a solution in an AND/OR tree is a *subtree* with exactly one assignment for every variable (Fig. 4). Formally:

Definition 5 (Solution tree). *A solution tree T of an AND/OR tree $S_{\mathcal{T}}$ is a subtree that contains 1) the root of $S_{\mathcal{T}}$, 2) exactly one AND child of every OR node in T, 3) all OR children of every AND node in T.*

Before presenting a simple AND/OR search algorithm, we show how to compute the cost $c(\mathbf{x})$ of a solution tree incrementally by evaluating functions in $\mathbf{F}$ as early in the search as possible. To this end, for each X_i we define a *bucket* $\mathbf{B}(X_i) \subseteq \mathbf{F}$ as follows: $\mathbf{B}(X_i)$ contains $f \in \mathbf{F}$ if X_i is in the scope of f and if the path from the root of pseudo tree $\mathcal{T}$ to X_i contains all variables in the scope of f. In other words, $\mathbf{B}(X_i)$ contains all functions that can be evaluated as soon as a search has instantiated X_i and no earlier. We define the *weight* of an OR-to-AND edge to be the combination of the functions evaluated by traversing it.

Definition 6 (OR-to-AND edge weight) *Let n be an OR node labeled X_i and m its child. Let $\mathbf{x}$ be the (partial) assignment of variables corresponding to the path from the root of $S_{\mathcal{T}}$ to m. The* weight *of the edge (n, m) is defined as*

$$w(n,m) = \bigotimes_{f \in \mathbf{B}(X_i)} f(\mathbf{x}).$$

For the pseudo tree illustrated in Fig. 3, the functions are $f_A = P(A)$, $f_B = P(B|A)$, $f_C = P(C|A)$, $f_D = P(D|B,C)$ and $f_E = P(E|A,B)$. The functions f_A, f_B, f_E are placed in the respective sets $\mathbf{B}(A)$, $\mathbf{B}(B)$, $\mathbf{B}(E)$, while f_C and f_D are both placed in $\mathbf{B}(C)$, since they can be evaluated only when the search has instantiated all variables on the path from the root to C. Fig. 5 shows a part of the induced AND/OR search tree with the respective edge weights. Note that since $\mathbf{B}(D)$ is empty, the weight of any edge instantiating D is the empty product, i.e. 1. Respectively, the empty sum will be 0 for problems involving summation. This also applies to the following definitions of solution tree costs.

Definition 7 (Cost of a solution tree) *Let T_n be the subtree of a solution tree rooted at node n. The cost of T_n, denoted $f(T_n)$, is defined recursively as follows.*

1. *If n is an OR node with a child m, then $f(T_n) = w(n,m) \otimes f(T_m)$.*
2. *If n is an AND node with a children $m_1, \ldots, m_k$, then $f(T_n) = \bigotimes_{i=1}^{k} f(T_{m_i})$.*

The cost of an optimal solution tree in an AND/OR search tree can also be defined recursively as follows.

Definition 8 (Optimal solution cost) *Let $S_{\mathcal{T}}$ be an AND/OR search tree. The cost of an optimal subtree rooted at node n is denoted $h^*(n)$ and defined recursively as follows.*

1. *If n is an OR node with children succ(n), then $h^*(n) = \max_{m \in succ(n)}(w(n,m) \otimes h^*(m))$*
2. *If n is an AND node with children succ(n), then $h^*(n) = \bigotimes_{m \in succ(n)} h^*(m)$.*

Clearly, the cost of an optimal complete solution tree is $h^*(s)$ where s is the root of $S_{\mathcal{T}}$. We will now describe a simple AND/OR search algorithm that finds an optimal solution tree by performing a straightforward depth-first evaluation of h^*.

Input: Graphical model $(\mathbf{X}, \mathbf{F}, \bigotimes)$, pseudo tree $\mathcal{T}$
Output: Problem solution

```
1  OPEN ← { OR node labeled by the root of T }
2  while OPEN ≠ ∅ do
3      n ← top(OPEN), remove n from OPEN
4      succ(n) ← ∅
5      if n is an OR node labeled X_i then
6          foreach x_i ∈ domain(X_i) do
7              m ← AND node labeled X_i = x_i
8              Add m to succ(n)
9              x ← assignment corresponding to π_m
10             w(n, m) ← ⊗_{f∈B(X_i)} f(x)
11     if n is an AND node labeled X_i = x_i then
12         foreach X_j ∈ children_T(X_i) do
13             Add an OR node labeled X_j to succ(n)
14     Add succ(n) to the top of OPEN
15     while succ(n) = ∅ do
16         if n is an OR node labeled X_i then
17             h*(n) ← max_{m∈succ(n)} h*(m)
18             if X_i is the root of T then
19                 return h*(n)
20         if n is an AND node labeled X_i = x_i then
21             h*(n) ← ⊗_{m∈succ(n)} h*(m)
22         p ← parent(n)
23         Remove n from succ(p)
24         n ← p
```

Algorithm 1: AND/OR search

3.3 Brute-force AND/OR search

Algorithm 1 presents a basic traversal of an AND/OR search tree induced by pseudo tree $\mathcal{T}$. We use $children_{\mathcal{T}}(X)$ to denote the children of X in $\mathcal{T}$. For a node n in $S_{\mathcal{T}}$, $succ(n)$ denotes its children, $parent(n)$ its parent, and π_n the path from the root of $S_{\mathcal{T}}$ to n.

The algorithm maintains an OPEN list of nodes, initially containing the root s of $S_{\mathcal{T}}$ (line 1), and iteratively *expands* the top node in the list. When a node n is expanded, the algorithm removes it from the OPEN list (3) and generates its children (4–13). If n is an OR node, the weights of its children are computed by evaluating the functions in the bucket of n, using the assignment corresponding to the path from s to the respective child (9-10). All children are then inserted at the *top* of the OPEN list (14), which guarantees that the search proceeds in depth-first order.

If the expanded node n has no children to generate, it initiates a propagation phase (15-24), which is completed before expanding the next node. In this phase, the algorithm first computes $h^*(n)$ (trivially either 0 or 1, depending on $\bigotimes$) and *propagates* the value to its parent p. If n is the last child to propagate its h^* value to p, the parent node may now compute its own value, $h^*(p)$, by taking either max (17) or $\bigotimes$ (21) over the h^* values of its children, according to the recurrence in Definition 8. It then propagates the value to its own parent node, which may then possibly compute its own value. The process is iterated (22–24) until it reaches a parent node with unexpanded children (15). In this case, one of those children is the next node in the OPEN list and is expanded next. The search is concluded when the final propagation phase ends, computing $h^*(s)$ (18-19).

As the algorithm computes the recurrence in Definition 8, it returns the cost of the optimal assignment of the variables $\mathbf{X}$. To obtain the actual assignment, one can trivially modify the algorithm to store not only h^* values of optimal subtrees but the subtrees themselves. Maximization chooses the best subtree while the $\bigotimes$ combines multiple subtrees into a larger subtree.

4 AND/OR BRANCH-AND-BOUND

The brute-force traversal of an AND/OR search tree finds an optimal solution by computing h^* for every node in the tree. In this section we augment the algorithm with a Branch-and-Bound technique that avoids the complete evaluation by pruning out subtrees that are guaranteed to be suboptimal. For the remainder of the section we omit the general treatment and focus solely on MPE. We trust the reader to generalize the following examples to other problems defined on graphical models. In particular, for minimization problems all references to upper bounds are replaced by lower bounds instead.

Consider a *partial solution tree* T of $S_{\mathcal{T}}$, which differs from a complete solution tree in that any node may have no children (Fig. 6). A partial solution tree can be extended to a complete solution tree in several ways. In particular, we are interested in finding an extension with an optimal cost, which can be defined as follows.

Definition 9 (Optimal evaluation of a partial solution tree) *Let T_n be a partial solution tree rooted at node n. Then, $f^*(T_n)$ is defined recursively as in Definition 7 with the following change: If T_n has no children, then $f^*(T_n) = h^*(n)$.*

In other words, $f^*(T)$ is the cost of the solution tree obtained by extending T with the optimal subtrees. Assume now that we are given a heuristic function h, called *node heuristic*, such that $h(n)$ is an upper bound for $h^*(n)$ (deriving such functions is discussed in the next section). Then, we can extend the definition of f on partial solution trees as follows.

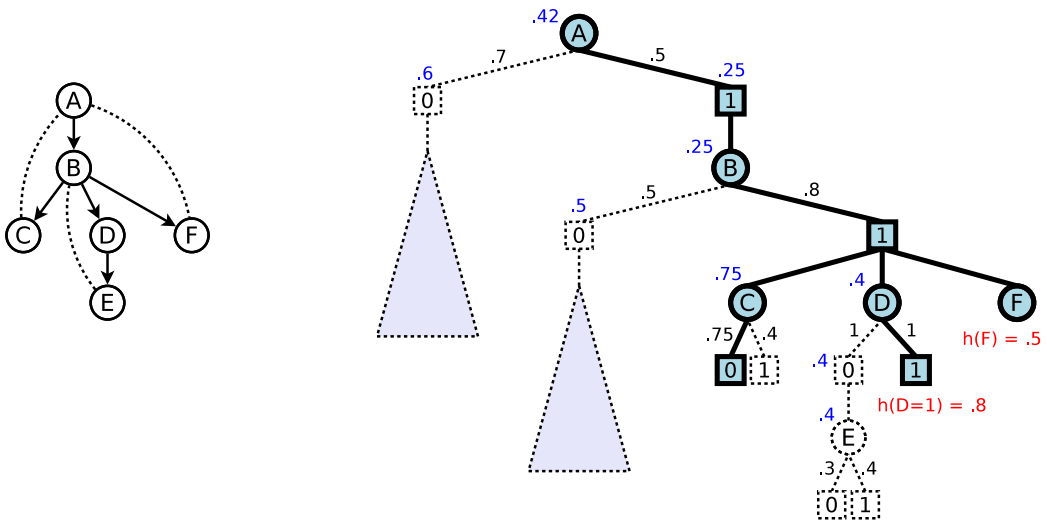

Figure 6: A pseudo tree (left) and an ongoing search in the induced AND/OR search tree (right). The current partial solution tree is highlighted and the subtrees already evaluated are shown in dashed lines. Edge weights are shown in black and the best subtree values found so far in blue. The heuristic function is shown in red.

Definition 10 (Heuristic evaluation of a partial solution tree) *Let T_n be a partial solution tree rooted at node n. Then, $f(T_n)$ is defined recursively as in Definition 7 with the following change: If T_n has no children, then $f(T_n) = h(n)$.*

Observe that the only difference between definitions 9 and 10 is that f^* uses the exact subtree cost h^* while f uses the heuristic h. Since $h^*(T_n) \leq h(T_n)$, it clearly follows that $f^*(T_n) \leq f(T_n)$. Therefore, f can be used to estimate the cost of the best solution that we can obtain by extending a partial solution tree and thus to prune out suboptimal extensions. As an example, consider Fig. 6 where the search has already evaluated h^* for the following subtrees:

$$\begin{array}{lcl} h^*(A=0) & = & 0.6 \\ h^*(B=0) & = & 0.5 \\ h^*(C) & = & 0.75 \\ h^*(D=0) & = & 0.4. \end{array}$$

Letting $best(X)$ denote the best solution cost found so far for the subtree rooted at OR node X, we have that

$$\begin{array}{lcl} best(A) & = & w(A,0) \cdot h^*(A,0) = 0.6 \cdot 0.7 = 0.42 \\ best(B) & = & w(B,0) \cdot h^*(B,0) = 0.5 \cdot 0.5 = 0.25 \\ best(D) & = & w(D,0) \cdot h^*(D,0) = 1 \cdot 0.4 = 0.4. \end{array}$$

The search is currently at the node $(D = 0)$ and the current partial solution tree contains the nodes A, $(A = 1)$, B, $(B = 1)$, C, $(C = 0)$, D, $(D = 1)$ and F. The node heuristic function h yields the upper bounds

$$\begin{array}{lcll} h^*(D=0) & \leq & h(D=1) & = & 0.8 \\ h^*(F) & \leq & h(F) & = & 0.5. \end{array}$$

The heuristic evaluation function f can now be computed for the partial solution tree according to Definition 10 as follows:

$$\begin{array}{lcll} f(T_{C=0}) & & & = & 1 \\ f(T_{D=1}) & = & h(D=1) & = & 0.8 \\ f(T_F) & = & h(F) & = & 0.5 \\ f(T_C) & = & w(C,0) \cdot f(T_{C=0}) = 0.75 \cdot 1 & = & 0.75 \\ f(T_D) & = & w(D,1) \cdot f(T_{D=1}) = 1 \cdot 0.8 & = & 0.8 \\ f(T_{B=1}) & = & f(T_C) \cdot f(T_D) \cdot f(T_F) & & \\ & = & 0.75 \cdot 0.8 \cdot 0.5 & = & 0.3 \\ f(T_B) & = & w(B,1) \cdot f(T_{B=1}) = 0.8 \cdot 0.3 & = & 0.24 \\ f(T_{A=1}) & = & f(T_A) = 0.3 & & \\ f(T_A) & = & w(A,1) \cdot f(T_{A=1}) = 0.5 \cdot 0.24 & = & 0.12 \end{array}$$

Observe that $f(T_A) < best(A)$. Since $f(T_A)$ is an upper bound for $f^*(T_a)$, expanding the current partial solution tree clearly cannot improve upon the current best solution. Therefore, we may safely prune away the subtree rooted at $(D = 0)$. However, note that $f(T_B) < best(B)$ holds as well, which is also a sufficient condition for pruning out $(D = 0)$. Thus, in this case it is not necessary to fully compute f. We may stop the evaluation as soon as we discover a value $f(X) < best(X)$ for any OR node X.

A straightforward modification to Algorithm 1 maintains $best(X)$ for every OR node X. Before expanding an AND node, the algorithm first computes f for every OR node X along the current path. If $f(X) < best(X)$ for any X, the evaluation is stopped and the AND node is pruned out. If the inequality doesn't hold for any X, then the AND node is expanded normally.

Figure 7: A primal graph (left) and a pseudo tree for it (right).

5 MINI-BUCKET HEURISTIC

The efficiency of Branch-and-Bound depends on the quality of the upper bounds provided by the node heuristic function h. In this section we describe a common method for deriving such bounds, based on approximations of alternative inference techniques. We first describe the exact bucket elimination algorithm, then present an approximation called mini-bucket elimination, and finally show how to derive a heuristic from it. A more formal treatment of these techniques can be found in [2], [4], [1] and [5].

5.1 Bucket elimination

Consider a Bayesian network on variables $\mathbf{X}$ and the following definitions.

Definition 11 (Operations on functions) *Let $f_1, \ldots, f_k$ be functions defined on subsets $S_1, \ldots, S_k$ of $\mathbf{X}$ and let $X \in \mathbf{X}$. We define the function $(\max_X f)$ on $\mathbf{X} \setminus \{X\}$ and the function $(\prod f_i)$ on the union of S_i as follows: $(\max_X f)(\mathbf{y}) = \max_x f(x, \mathbf{y})$ and $(\prod_i f_i)(\mathbf{x}) = \prod_i f_i(\mathbf{x})$.*

Using the definitions above, the MPE problem can be formulated as a task of evaluating the function

$$\max_{\mathbf{X}} \prod_i P(X_i|Pa_i).$$

Indeed, a direct evaluation would be equivalent to constructing the full joint probability table ($\prod$) and picking the assignment with the highest probability (max). For example, to solve the problem on the primal graph in Figure 7 (left), we would compute

$$\max_{a,b,c,d,e} P(a)P(b|a)P(c|a)P(d|b,c)P(e|c).$$

Obviously we don't want to evaluate this directly as computing the full joint probability table is both time and space exponential in the number of variables. Observing that $\max_X f_1 f_2 = f_1 \max_X f_2$ if X does not appear in the scope of f_1, we can instead rewrite the function as

$$\max_a P(a) \max_b P(b|a) \max_c P(c|a) \max_d P(d|b,c) \max_e P(e|c).$$

This can be evaluated from right to left as follows

$$\begin{aligned}
&= \max_a P(a) \max_b P(b|a) \max_c P(c|a)\lambda_E(c) \max_d P(d|b,c) \\
&= \max_a P(a) \max_b P(b|a) \max_c P(c|a)\lambda_E(c)\lambda_D(b,c) \\
&= \max_a P(a) \max_b P(b|a)\lambda_C(a,b) \\
&= \max_a P(a)\lambda_B(a) \\
&= \lambda_A.
\end{aligned}$$

Each λ_X denotes the function obtained by maximizing out variable X. For instance, maximizing C out of $P(c|a)\lambda_E(c)\lambda_D(b,c)$ produces $\lambda_C(a,b)$. If λ_X does not depend on the next variable(s) to be maximized out, it is taken outside. For example, $\max_d P(d|b,c)\lambda_E(c)$ is rewritten $\lambda_E(c) \max_d P(d|b,c)$. The evaluation ends at the constant function λ_A, which yields the desired maximum probability.

An evaluation like this is called bucket elimination: Each maximization *eliminates* a single variable and incorporates its "effect" in the resulting function. We think of each variable as having a *bucket*, which contains the functions that are multiplied before elimination. For example, the bucket of C contains the functions $P(c|a)$, $\lambda_E(c)$ and $\lambda_D(b,c)$. Observe that the largest functions evaluated in the example are $\lambda_{B,C}$ and $\lambda_{A,B}$. As these only depend on two variables, they are quite efficient to compute. In general, the order in which variables are eliminated significantly affects the size of the functions that must be computed. Thus, the efficiency of bucket elimination depends greatly on finding good orders, much like AND/OR search depends on finding good pseudo trees. In fact, variable elimination can be equivalently formalized as a specific traversal of an AND/OR search space [6].

5.2 Mini-bucket elimination

In some cases no order produces sufficiently small functions, which necessitates the use of approximations.

Assume, for example, that we are to compute

$$\max_e f_1(b,e)f_2(a,c,e)f_3(d,e)f_4(c,e),$$

which would result in a function $\lambda(a,b,c,d)$. Noting that $\max_x f(x)\cdot g(x) \leq \max_x f(x)\cdot\max_x g(x)$ for non-negative functions f, g, we can bound λ by partitioning the functions into groups and maximizing over each group separately. For example,

$$\begin{aligned}\max_e f_2(a,c,e)f_4(c,e) &= \lambda_1(a,c)\\ \max_e f_1(b,e)f_3(d,e) &= \lambda_2(b,d)\end{aligned}$$

gives the bound

$$\lambda(a,b,c,d) \leq \lambda_1(a,c)\cdot\lambda_2(b,d).$$

Mini-bucket elimination with a given i-parameter operates as bucket elimination, but replaces all functions with scopes larger than i with upper bounds as above. These are more efficient to compute and yield an upper bound on the exact solution to MPE.

5.3 Mini-bucket heuristic for AND/OR search

Consider an AND/OR search on the example in Figure 7 (left), using the pseudo tree $\mathcal{T}$ (right). We say that $\mathcal{T}$ is *compatible* with an ordering $X_1,\ldots,X_n$ of the variables if the fact that X_i is an ancestor of X_j in $\mathcal{T}$ implies $i<j$. For example, the alphabetical order is clearly compatible with $\mathcal{T}$.

Assume for a while that we were given the λ functions produced by performing bucket elimination on the alphabetical order (as presented in Section 5.1). We note that they represent the probabilities

$$\begin{aligned}\lambda_E(c) &= \max_e P(e|c)\\ \lambda_D(b,c) &= \max_d P(d|b,c)\\ \lambda_C(a,b) &= \max_{c,d,e} P(c|a)P(e|c)P(d|b,c)\\ \lambda_B(a) &= \max_{b,c,d,e} P(b|a)P(c|a)P(e|c)P(d|b,c).\end{aligned}$$

Consider now an AND node n corresponding to the instantiation $(A=a,B=b)$. By the definition of $h^*(n)$, we see that $h^*(n)=\lambda_C(a,b)$. Letting n denote $(A=a,B=b,C=c)$ instead, we see again by the definition of $h^*(n)$ that $h^*(n)=\max_{d,e}P(d|b,c)P(e|c)=\lambda_E(c)\lambda_D(b,c)$. More generally, let $h(n)$ be the product of intermediate functions λ such that the scope of λ is instantiated by n but not by any ancestor of n. Then, assuming the order of bucket elimination is compatible with $\mathcal{T}$, it is fairly easy to see that $h(n)=h^*(n)$.

Although an h like this would be an ideal heuristic function, it's not practical since we would require the λ functions to compute it. On other hand, if we did have the λ functions, it would be faster to simply do bucket elimination instead. However, the above definition of h still works if we compute the λ functions using mini-bucket elimination instead. This is feasible if a suitably small i-parameter is used. As argued earlier, this produces an upper bound for the actual probability, which is exactly what we require of a heuristic function. Experiments regarding the mini-bucket heuristic and a suitable i-parameter can be found in [5]. A variant of the heuristic that employs dynamic variable orderings is also discussed.

6 CONCLUSION

We have presented the AND/OR search technique for graphical models and described a specific application to solving the MPE problem in Bayesian networks. We have described both a brute-force traversal of the search space and a heuristic approach employing Branch-and-Bound with suitable upper bounds. Finally, we have derived such bounds for MPE, based on approximate variable elimination.

In our treatment, MPE asks for the most probable assignment for *all* variables. To accommodate evidence variables whose values are known, conflicting values are removed from all relevant CPTs and the remaining probabilities are normalized. This simplifies the problem and may result in more shallow pseudo trees.

References

[1] Rina Dechter. Mini-buckets: A general scheme for generating approximations in automated reasoning. In *IJCAI*, pages 1297–1303, 1997.

[2] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2):41–85, 1999.

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

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

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

[6] Robert Mateescu and Rina Dechter. The relationship between and/or search and variable elimination. In *UAI*, pages 380–387, 2005.