Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Look-Ahead with Mini-Bucket Heuristics for MPE

Rina Dechter, Kalev Kask, William Lam
University of California, Irvine
Irvine, California, USA

Javier Larrosa
UPC Barcelona Tech
Barcelona, Spain

Abstract

The paper investigates the potential of look-ahead in the context of AND/OR search in graphical models using the Mini-Bucket heuristic for combinatorial optimization tasks (e.g., MAP/MPE or weighted CSPs). We present and analyze the complexity of computing the residual (a.k.a. Bellman update) of the Mini-Bucket heuristic and show how this can be used to identify which parts of the search space are more likely to benefit from look-ahead and how to bound its overhead. We also rephrase the look-ahead computation as a graphical model, to facilitate structure exploiting inference schemes. We demonstrate empirically that augmenting Mini-Bucket heuristics by look-ahead is a cost-effective way of increasing the power of Branch-And-Bound search.

Introduction

Look-ahead is known to be useful and even essential in the context of online search algorithms (e.g., game playing schemes such as alpha-beta, planning under uncertainty (Geffner and Bonet 2013; Vidal 2004) where the search space is enormous and traversing it to completion is out of the question. Look-ahead can improve the heuristic function h of a node by expanding the search tree below it to a certain depth d, evaluate the static h at tip nodes and roll back this information to n (known as a Bellman update).

Here we investigate the potential of look-ahead (Bu et al. 2014) in the different context of complete search. We consider the min-sum problem over a Graphical model (which includes, among others, the *most probable explanation* task of Markov Networks (Pearl 1988)). This problem is usually solved by depth-first search *Branch-and-Bound* (DFBnB). The search is guided by a heuristic function that provides an optimistic bound on the *cost of the best extension* of any partial assignment. It is often observed that these types of algorithms spend a significant time proving the optimality of their output (where the only use of h is pruning).

Our framework is AND/OR Branch-And-Bound (AOBB) guided by the static Mini-Bucket Elimination heuristic (MBE), which, assisted by variational cost-shifting methods, is one of the best current approaches for the min-sum problem (Marinescu and Dechter 2009; L. Otten and Dechter 2012). However, this heuristic can be weak, (e.g., mainly for problems having a high induced width). A natural approach is to improve it via dynamic look-ahead during search itself.

In the context of depth-first BnB a better heuristic may i) allow finding an optimal solution early with less search (and thus the best upper bound for pruning), and ii) once the optimal bound is reached, it will help proving optimality by better pruning of the search space.

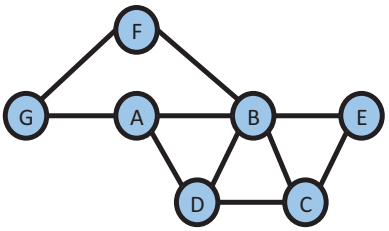
What we show in this paper is that in the context of graphical models and when using the mini-bucket heuristic, we can bound the computation of d-level look-ahead sufficiently, making it cost effective. This is accomplished by i) exploiting a relationship between the residual of the heuristic and a new concept of bucket error that can be pre-compiled, to facilitate a selective look-ahead process combined with effective pruning of the d-level lookahead tree whenever it is applied, and ii) using an inference-based message-passing algorithm to compute the look-ahead, thus exploiting the graphical model structure at each node.

In Section 2, we provide background. Section 3 introduces the notions of residual and local bucket-error of the mini-bucket heuristic that captures the mini-bucket approximation error. Section 4 presents the experiments and discussion. Section 5 concludes the paper.

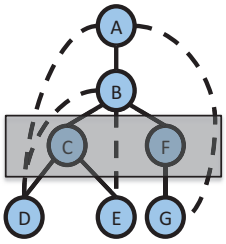
Background

Graphical Models A *graphical model* is a tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, where $\mathbf{X} = \{X_i : i \in V\}$ is a set of variables indexed by a set V and $\mathbf{D} = \{D_i : i \in V\}$ is the set of finite domains of values for each X_i, $\mathbf{F}$ is a set of discrete functions. In this paper we focus on the *min-sum problem*, $C^* = \min_x \sum_{f \in F} f(x)$ wich is applicable in a wide range of reasoning problems (e.g, MPE/MAP). The *scope* of f (i.e. the subset of $\mathbf{X}$ relevant to f) is noted $scope(f)$. We will use lower case y to denote the assignment of a set of variables $Y \subseteq \mathbf{X}$. Thus, $f(x, Y)$ is a function with scope $X \cup Y$ partially assigned by x. In an abuse of notation we will sometimes write $f(y)$ with $scope(f) \subset Y$ assuming that the irrelevant parts of the assignment will be ignored.

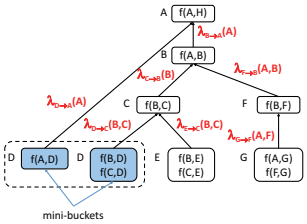
Primal graph, pseudo-tree. The *primal graph* G of a graphical model $\mathcal{M}$ has each variable X_i in a node and an edge (X_i, X_j) is in G iff the pair of variables appears in the scope of any $f \in \mathbf{F}$. A *pseudo-tree* $T = (V, E')$ of the



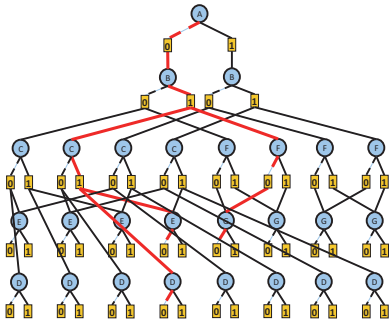
(a) A primal graph of a graphical model over 8 variables. We assume pair-wise functions.



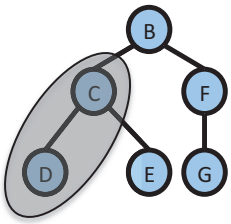
(b) A pseudo-tree T. Grayed area is $T_{B,1}$



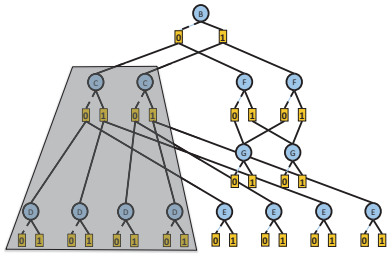
(c) Mini-bucket tree with i-bound of 3.



(d) The AND/OR context minimal search graph.



(e) Depth 2, look-ahead tree from B. Minimal pruned tree in grayed area.



(f) Depth 2, look-ahead search space from B. Minimal pruned search space in grayed area.

Figure 1

primal graph $G = (V, E)$ is a rooted tree over the same set of vertices V such that every edge in $E - E'$ connects a node to one of its ancestors in T.

Notation. Let T be a pseudo-tree. $\bar{X}_p$ denotes X_p and its *ancestors* in T. $ch(X_p)$ denote the children of X_p in T. The subtrees rooted by each $ch(X_p)$ is denoted T_p. The same subtrees of T_p pruned below depth d is denoted $T_{p,d}$. We will often abuse notation and will consider a tree as the set of its nodes, or its variables.

Example. Figure 1a displays the primal graph of a graphical model with 10 binary cost functions, each one corresponds to an edge in the graph. Figure 1b displays a pseudo-tree. The ancestors of C are A and B. Its children are D and E. The grayed area corresponds to $T_{B,1}$.

AND/OR search. A state-of-the-art method to solve reasoning tasks on graphical models is to search their AND/OR graph (Marinescu and Dechter 2009). The AND/OR search tree, which is guided by a *pseudo-tree* T of the primal graph G, consists of alternating levels of OR and AND nodes. Each OR node is labeled with a variable $X_p \in \mathbf{X}$. Its children are AND nodes, each labeled with an instantiation x_p of X_p. The weight of the edge, $w(X_p, x_p)$, is the sum of costs of all the functions $f \in F$ that are completely instantiated at x_p but are not at its parent X_p. Children of AND nodes are OR nodes, labeled with the children of X_p in T. Each child represents a conditionally independent subproblems given assignments to their ancestors. Those edges are not weighted. The root of the AND/OR search *tree* is the root of T. The path from an AND node labeled x_p to the root corresponds to a partial assignment to $\bar{X}_p$ denoted $\bar{x}_p$.

A *solution tree* in the AND/OR tree is a subtree that (1) contains its root , (2) if an OR node is in the solution subtree, then exactly one of its children is in the solution tree, (3) if an AND node is in the solution tree, then all its children are. A solution tree corresponds to a complete assignment and its cost is the sum-cost of its arc weights.

The AND/OR search tree can be converted into a *graph* by merging nodes that root identical subproblems. It was shown that identical subproblems can be identified by their *context* (a partial instantiation that decomposes the subproblem from the rest of the problem graph), yielding an *AND/OR search graph* (Marinescu and Dechter 2009).

Example. Figure 1d displays the AND/OR search graph of the running example. A solution tree, corresponding to the assignment $(A = 0, B = 1, C = 1, D = 0, E = 0, F = 0, G = 0)$, is highlighted.

AND/OR Branch-And-Bound (AOBB) is a state-of-the-art algorithm for solving combinatorial optimization problems over graphical models (Marinescu and Dechter 2009). It explores the set of (partial) assignments by traversing the weighted AND/OR context-minimal search graph in a depth-first manner, while keeping track of the current best solution (upper bound, ub) of the subtree rooted at each node $\bar{x}_p$, denoted $ub(\bar{x}_p)$. Each AND node has an associated heuristic value $h(\bar{x}_p)$ which is a lower bound on the best cost

extension of $\bar{x}_p$ to its descendents in T_p. Then whenever $h(\bar{x}_p) \geq ub(\bar{x}_p)$, search below $\bar{x}_p$ can be safely pruned.

The heuristic is also used to guide the order in which OR nodes and AND nodes are expanded (namely, in which order independent subproblems are considered and in which order values for the current variable are considered).

Mini-Bucket Elimination Heuristic. Current implementations of AOBB (Kask and Dechter 2001; Marinescu and Dechter 2009) are guided by the mini-bucket heuristic. This heuristic is based on *Mini-Bucket Elimination* and is called $MBE(i)$ where i, called *i-bound*, allows trading off pre-processing time and space for heuristic accuracy with actual search. It belongs to the class of *static heuristics*, meaning that it is pre-processed before AOBB starts. $MBE(i)$ works relative to the same pseudo-tree T which defines the AND/OR search graph. Each node X_p of T is associated with a bucket B_p that includes a set of functions. The *scope of a bucket* is the union of scopes of all its functions before it is processed as described next. First, each function f of the graphical model is placed in a bucket B_p if X_p is the deepest variable in T s.t. $X_p \in scope(f)$. Then $MBE(i)$ processes the buckets of the pseudo-tree from leaves towards the root. Processing a bucket may require partitioning the bucket's functions into *mini-buckets* $B_p = \cup_r B_p^r$, where each B_p^r includes no more than i variables. Then, processing each mini-bucket separately, all its functions are combined (by sum in our case) and the bucket's variable (X_p) is eliminated (by min in our case). Each resulting new function, also called *a message*, is placed in the closest ancestor bucket whose variable is contained in its scope. MBE(i) is time and space exponential in the i-bound.

Bucket Elimination (BE). In the special case in which the i-bound is high enough so there is no partitioning into mini-buckets, the algorithm is the exact bucket-elimination (BE) scheme (Dechter 2013). The time and space complexity of BE is exponential in the size of the largest bucket scope encountered which is called the *induced width* and is denoted w^*, for that ordering.

Notation. In the following, f_p denotes an original function placed in Bucket B_p (if there is more than one, they are all summed into a single function), and $\lambda_{j\to p}$ denotes a message created at bucket B_j and placed in bucket B_p. Processing bucket B_p produces messages $\lambda_{p\to i}$ for some ancestors X_i of X_p (i.e, $X_i \in \bar{X}_p - X_p$).

Example. Figure 1c illustrates the execution of MBE(3) in our running example by means of the so-called mini-bucket tree (nodes are buckets and tree-edges show message exchanges). In this example, bucket B_D is the only one that needs to be partitioned into mini-buckets. Each mini-bucket generates a message. In the figure, messages are displayed along an edge to emphasize the bucket where they are generated and the bucket where they are placed. For instance, $\lambda_{D\to C}$ is generated in bucket D (as $\lambda_{D\to C} = \min_D\{f(B,D) + f(B,D)\}$) and placed in bucket C.

Extracting the mini-bucket heuristic (Kask and Dechter 1999). The (static) mini-bucket heuristic used by AOBB requires a MBE(i) execution prior to search keeping all the message functions. Let $\bar{x}_p$ be a partial assignment. Λ_j denotes the sum of the messages sent from bucket B_j to all of the instantiated ancestor variables,

$$\Lambda_j(\bar{x}_p) = \sum_{X_q \in \bar{X}_p} \lambda_{j\to q}(\bar{x}_p) \quad (1)$$

The heuristic value at node $\bar{x}_p$ is,

$$h(\bar{x}_p) = \sum_{X_j \in T_p} \Lambda_j(\bar{x}_p) \quad (2)$$

Example. In our example, the heuristic function of partial assignment $(A = 0, B = 1)$ is $h(A = 0, B = 1) = \lambda_{D\to A}(A = 0) + \lambda_{C\to B}(B = 1) + \lambda_{F\to B}(A = 0, B = 1)$

AND/OR Look-Ahead

OR Look-Ahead In the field of *heuristic search* the general task is to find the least cost path in a weighted directed graph from one initial node to one of the possibly many goal nodes. The search is guided by an admissible heuristic function $h(n)$ which underestimates the least cost path from n to a goal state. More accurate $h(n)$ decreases the expanded search space (Nilsson 1980). If h is not accurate enough it can be improved by means of a look-ahead, a technique useful in game playing and in planning with uncertainty, especially for anytime solvers (Geffner and Bonet 2013; Vidal 2004).

DEFINITION **1 (look-ahead, residual).** *Let $ch(n)$ be the children of node n in the search graph and h be a heuristic function. The d look-ahead of n is,*

$$h^d(n) = \min_{n_i \in ch(n)} \{w(n, n_i) + h^{d-1}(n_i)\}$$

where $w(n, n_i)$ is the weight of the arc, and $h^0 = h$. The look-ahead residual*, defined by,*

$$res(n) = h^1(n) - h(n)$$

measures the error of h with respect to the next level.

AND/OR Look-Ahead. Focusing now on AND/OR search for graphical models, let n be an AND node labeled by x_p terminating a partial assignment $\bar{x}_p$. Extending the definition of look-ahead to the next level of AND nodes (i.e, two levels in the AND/OR graph) we get:

$$h^d(\bar{x}_p) = \sum_{X_q \in ch(X_p)} \min_{x_q} \{w(X_q, x_q) + h^{d-1}(\bar{x}_q)\} \quad (3)$$

Specializing to the MBE heuristic we can show

PROPOSITION **1.** *If h denotes the mini-bucket heuristic, $\bar{x}_p$ is a partial assignment and $x^{\downarrow}_{p,d}$ is an extension of the assignment to all the variables in $T_{p,d}$*

$$h^d(\bar{x}_p) = h(\bar{x}_p) + L^d(\bar{x}_p) - \sum_{X_k \in T_{p,d}} \Lambda_k(\bar{x}_p) \quad (4)$$

where,

$$L^d(\bar{x}_p) = \min_{x^{\downarrow}_{p,d}} \{ \sum_{X_k \in T_{p,d}} [f_k(\bar{x}_p, x^{\downarrow}_{p,d}) + \sum_{X_j \in T_p - T_{p,d}} \lambda_{j\to k}(\bar{x}_p, x^{\downarrow}_{p,d})]\} \quad (5)$$

Proof. (sketch) The reformulation is obtained after expanding the recursive definition of lookahead (eq. 3) and replacing the generic heuristic by the MBE heuristic (eq. 2 and 1) □

$L^d(\bar{x}_p)$ is a min-sum problem over a graphical model, as defined next.

DEFINITION **2 (look-ahead graphical model).** *Given a graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$ and messages generated by MBE(i) along pseudo-tree T, $\mathcal{M}(\bar{x}_p, d)$ is the depth-d look-ahead graphical model at $\bar{x}_p$. It has as variables the set $\{X_k |\ X_k \in T_{p,d}\}$ and has functions $\{f_k(X, \bar{x}_p) |\ X_k \in T_{p,d}\} \cup \{\lambda_{j\to k}(X, \bar{x}_p) |\ X_j \in T_p - T_{p,d}, X_k \in T_{p,d}\}$.*

Clearly, $T_{p,d}$ is a valid pseudo-tree of $\mathcal{M}(\bar{x}_p, d)$, that we call the look-ahead tree. The induced width of $\mathcal{M}(\bar{x}_p, d)$ is noted $lw_{p,d}$.

Example. In Figure 1, the depth 1 look-ahead graphical model relative to assignment $(A = 0, B = 1)$ contains variables $\{C, F\}$, and the functions, $\{f(B = 1, C), f(B = 1, F), \lambda_{D\to C}(B = 1, C), \lambda_{E\to C}(B = 1, C), \lambda_{G\to F}(A = 0, F)\}$, whose induced width is 1. Figure 1e displays the depth 2 look-ahead tree at variable B.

Clearly, $lw_d \leq \min\{d, w^*\}$ and the bound is tight. In practice, only shallow look-aheads may be feasible because the complexity of solving the look-ahead graphical model is exponential in lw_d. *A main source for improving the efficiency in our look-ahead is using effective variable elimination algorithms to compute the look-ahead at each node.* We next focus on the main method for bounding the look-ahead overhead using the notion of bucket error.

Bounding the Look-Ahead Overhead

Bucket Error

The residual measures the error of one level look-ahead. For the mini-buckets heursitic the error is originated by the mini-bucket partitioning. In the following we will compare the message a bucket would have computed without partitioning (called exact bucket message and noted μ^*_k) against the messages computed by mini-buckets of the bucket (called combined mini-bucket message and noted μ_k). The difference between the two captures the error introduced by partitioning the bucket.

DEFINITION **3 (bucket and mini-bucket messages at B_k).** *Given a mini-bucket partition $B_k = \cup_k B^r_k$, we define the combined mini-bucket message at B_k,*

$$\mu_k = \sum_r (min_{X_k} \sum_{f \in B^r_k} f) \quad (6)$$

In contrast, the exact message that would have been generated with no partitioning at B_k is,

$$\mu^*_k = min_{X_k} \sum_{f \in B_k} f \quad (7)$$

Notice that while μ^*_k is exact for B_k it may contain partitioning errors introduced in earlier buckets.

DEFINITION **4 (local bucket-error at B_k).** *Given a run of MBE, the local bucket error function at B_k denoted Err_k is,*

$$Err_k = \mu^*_k - \mu_k \quad (8)$$

The scope of Err_k is a subset of $\bar{X}_k - X_k$

Example. In an MBE(3) execution as in Figure 1c the bucket error of D is $Err_D = \min_D[f(A, D) + f(B, D) + f(C, D)] - (\lambda_{D\to A}(A) + \lambda_{D\to C}(B, C))$.

Algorithm 1 (BEE) computes the bucket errors. Following MBE(i), it executes a second pass from leaves to root along the pseudo-tree. When considering bucket B_k, it computes μ_k, μ^*_k and Err_k. The complexity is exponential in the scope of the bucket after the $MBE(i)$ execution. The total cost is dominated by the largest scope. This number is captured by a new parameter that we call *pseudo-width*.

Algorithm 1: Bucket Error Evaluation (BEE)

Input: A Graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, a pseudo-tree T, i-bound
Output: The error function Err_k for each bucket
Initialization: Run $MBE(i)$ w.r.t. T.
for *each* $B_k, X_k \in \mathbf{X}$ **do**
  Let $B_k = \cup_k B^r_k$ be the partition used by $MBE(i)$
  $\mu_k = \sum_r (min_{X_k} \sum_{f \in B^r_k} f)$
  $\mu^*_k = min_{X_k} \sum_{f \in B_k} f$
  $Err_k \leftarrow \mu^*_k - \mu_k$
return Err functions

DEFINITION **5 (pseudo-width(i)).** *Given a a run of MBE(i) along pseudo-tree T, the pseudo-width of B_j, $psw^{(i)}_j$ is the number of variables in the bucket at the moment of being processed. The pseudo-width of T relative to MBE(i) is denoted $psw(i) = \max_j psw^{(i)}_j$.*

THEOREM **1 (complexity of BEE).** *The complexity of BEE is $O(n \cdot k^{psw(i)})$, where n is the number of variables, k bounds the domain size and $psw(i)$ is the pseudo-width along T relative to MBE(i).*

The pseudo-width lies between the width and the induced width w^* of the ordering, and it grows with the i-bound. When the i-bound of MBE is large, computing the error exactly may not be possible.

We next show (Theorem 2) the relationship between the residual of the mini-bucket heuristic and the bucket errors. To prove it we need the following lemmas which relates μ_k (Eq. 6) to the MBE heuristic of its parent X_p, and relates μ^* (Eq. 7) to the combinatorial part of the look-ahead computation (Eq. 5).

Lemma 1. *If X_k is a child of a variable X_p. Then, $\Lambda_k(\bar{X}_p) = \mu_k(\bar{X}_p)$*

Proof. $\Lambda_k(\bar{X}_p)$ (see Eq. 1) is the sum of messages that $MBE(i)$ sends from B_k to the buckets of variables in $\bar{X}_p$.

Since X_p is the parent of X_k, $\Lambda_k(\bar{X}_p)$ is the sum of all the messages departing from X_k, which is the definition of $\mu_k(\bar{X}_p)$ □

Lemma 2. *If X_k is a child of a variable $X_p \in T$. Then, $L^1(\bar{x}_p) = \sum_{X_k \in ch(X_p)} \mu^*_k(\bar{x}_p)$*

Proof. (sketch) In the expression of $L^1(\bar{x}_p)$ (see Eq. 5) it is possible to push the minimization into the summation. Thus,

$$L^1(\bar{x}_p) = \sum_{X_k \in ch(X_p)} \min_{x_k} \{f_k(\bar{x}_p, x_k) + \sum_{X_j \in T_p - ch(X_p)} \lambda_{j\to k}(\bar{x}_p, x_k)\}$$

The set of functions inside of each $\min_{x_k}$ are, by definition, the set of functions in B_k placed there either originally or are messages received from its descendent, yielding (Eq. 7)

$$= \sum_{X_k \in ch(X_p)} \min_{x_k} \sum_{f \in B_k} f = \sum_{X_k \in ch(X_p)} \mu^*_k$$

□

THEOREM **2 (residual and bucket-error).** *Assume an execution of $MBE(i)$ along T yielding heuristic h Then, for every $\bar{x}_p$*

$$res(\bar{x}_p) = \sum_{X_k \in ch(X_p)} Err_k(\bar{x}_p) \quad (9)$$

Proof. (sketch) From the definition of residual and Prop 1,

$$res(\bar{x}_p) = L^1(\bar{x}_p) - (\sum_{X_k \in ch(X_p)} \Lambda_k(\bar{x}_p))$$

From the Lemma 1 and Lemma 2,

$$res(\bar{x}_p) = \sum_{X_k \in ch(X_p)} \mu^*_k(\bar{x}_p) - \sum_{X_k \in ch(X_p)} \mu_k(\bar{x}_p)$$

Grouping together expressions that refer to the same child proves the theorem. □

Corollary 1. *When a bucket is not partitioned into mini-buckets, its bucket error is 0, and therefore it contributes zero to the residual and the look-ahead of its parent.*

Pruned Look-Ahead Trees

Given the error functions, we can now identify variables that are *irrelevant* for lookahead (that is, their look-ahead does not produce any improved heuristic). Once those are identified, for each variable and look-ahead depth d, we will prune its look-ahead subtree to include only paths that touch variables that are relevant. If the bucket error function is zero for a variable, the variable is clearly irrelevant for 1-level look-ahead. However to allow more flexibility we will identify those buckets whose *average relative bucket error* is below a given threshold ϵ. Below, $dom(B_j)$ denotes all the assignments to variables in the scope of bucket B_j.

DEFINITION **6 (average relative bucket error).** *The average relative bucket error of X_j given a run of MBE(i) is*

$$\tilde{E}_j = \frac{1}{|dom(B_j)|} \sum_{\bar{x}_j} \frac{Err_j(\bar{x}_j)}{\mu^*(\bar{x}_j)}$$

DEFINITION **7 (irrelevant variable).** *A variable X_j is ϵ-irrelevant iff $\tilde{E}_j \leq \epsilon$.*

DEFINITION **8 (minimal pruned d-level subtree).** *Given a threshold ϵ, a minimal pruned look-ahead subtree $T^*_{p,d}$ for variable X_p is obtained from $T_{p,d}$ by removing ϵ-irrelevant leaves X_j (recursively until quiescence).*

Example. Corollary 1 identifies trivial 0-irrelevant variables. For instance, in the MBE execution displayed in Figure 1c, all variables but D are 0-irrelevant. It may happen that variable D is also 0-irrelevant or ϵ-irrelevant for some small ϵ, but we cannot tell from the symbolic execution displayed in the figure. It can be detected using bucket errors.

Figure 1e shows a depth 2 look-ahead tree for B and (grayed) the minimal pruned subtree assuming $\epsilon = 0$ and only irrelevant variables as for Corollary 1.

Algorithm 2: Minimal Pruned Look-Ahead Subtree

Input: A Graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, a pseudo-tree T, i-bound, threshold ϵ, depth d
Output: Minimal pruned look-ahead subtrees
Initialization: Run $MBE(i)$ w.r.t. T.
$X' \leftarrow \mathbf{X}$
Run $BEE(\mathcal{M}, T, i)$
for *each* $B_j, X_j \in \mathbf{X}$ **do**
  $\tilde{E}_j \leftarrow \frac{1}{|dom(B_j)|} \sum_{\bar{x}_j} \frac{Err_j(\bar{x}_j)}{\mu^*(\bar{x}_j)}$
  if $\tilde{E}_j < \epsilon$ **then** $X' \leftarrow X' - \{X_j\}$;
for *each* X_p *in* $\mathbf{X}$ **do**
  Initialize $T^*_{p,d}$ to $T_{p,d}$
  while $T^*_{p,d}$ *has leaves in* X' **do**
    Remove leaves $X_j \notin X'$ from $T^*_{p,d}$
return $T^*_{p,d}$ for each X_p

Algorithm 2 describes the generation of pruned look-ahead trees for each variable.

Experimental Evaluation

Algorithm Setup. We experimented with running AND/OR branch-and-bound (AOBB) guided by $MBE(i)$ with moment-matching (Ihler et al. 2012). To ensure a fixed search space for all algorithms we used pre-computed variable orderings. We tried 2-3 different i-bounds for each problem instance, aiming at a diversity of heuristic accuracy.

We run Algorithm 2 for pre-processing, yielding a minimal pruned look-ahead subtree for each variable, When the BEE computation (and its table) gets too large (e.g. over 10^6) we sample (e.g. 10^5 assignments). Within each i-bound setting, we varied the look-ahead depth from 0 to 6.

Benchmarks. Includes instances from genetic linkage analysis (`pedigree, largeFam`) and medical diagnosis (`promedas`) (see **Table 2**). In total, we evaluated 59 problem instances with induced widths ranging from 19 to 120.

instances (n,w,h,k,fns,ar)	Lookahead depth	i-bound ϵ / rv / nel / ptime / **time** / spd / **nodes** / red	 ϵ / rv / nel / ptime / **time** / spd / **nodes** / red	 ϵ / rv / nel / ptime / **time** / spd / **nodes** / red
Pedigree networks				
		i=6	i=10	i=20
	LH(0)	- / - / - / 0 / **8628** / 1.00 / **4764** / 1.00	- / - / - / 0 / **311** / 1.00 / **175** / 1.00	- / - / - / 48 / **55** / 1.00 / **3** / 1.00
pedigree7	LH(1)	1.0 / 111 / 88 / 0 / **6895** / 1.25 / **3637** / 1.31	1.0 / 76 / 67 / 1 / **257** / 1.21 / **135** / 1.30	0.01 / 52 / 51 / 52 / **59** / 0.93 / **3** / 1.16
(867,32,90,4,1069,4)	LH(3)	1.0 / 111 / 137 / 0 / **5761** / 1.50 / **2020** / 2.36	1.0 / 57 / 89 / 1 / **233** / 1.33 / **96** / 1.82	0.01 / 52 / 83 / 52 / **59** / 0.93 / **2** / 1.80
	LH(6)	1.0 / 111 / 164 / 0 / **4681** / 1.84 / **519** / 9.17	1.0 / 57 / 116 / 1 / **481** / 0.65 / **53** / 3.30	0.01 / 52 / 94 / 52 / **65** / 0.85 / <**1** / 3.88
		i=5	i=8	
	LH(0)	- / - / - / 0 / **1262** / 1.00 / **826** / 1.00	- / - / - / 0 / **35** / 1.00 / **23** / 1.00	- / - / - / - / - / - / - / - / -
pedigree18	LH(1)	0.01 / 151 / 134 / 0 / **912** / 1.38 / **564** / 1.47	0.01 / 92 / 82 / 0 / **20** / 1.75 / **13** / 1.76	- / - / - / - / - / - / - / - / -
(931,19,102,5,1185,5)	LH(3)	0.01 / 151 / 203 / 0 / **691** / 1.83 / **311** / 2.66	0.01 / 92 / 132 / 0 / **12** / 2.92 / **6** / 4.00	- / - / - / - / - / - / - / - / -
	LH(6)	0.01 / 93 / 192 / 0 / **300** / 4.21 / **66** / 12.47	0.01 / 92 / 152 / 0 / **13** / 2.69 / **2** / 11.06	- / - / - / - / - / - / - / - / -
LargeFam linkage networks				
		i=17	i=18	
	LH(0)	- / - / - / 33 / **10427** / 1.00 / **6730** / 1.00	- / - / - / 34 / **4349** / 1.00 / **2809** / 1.00	- / - / - / - / - / - / - / - / -
lf3_11_53	LH(1)	0.01 / 78 / 73 / 36 / **8611** / 1.21 / **4875** / 1.38	0.01 / 75 / 70 / 36 / **3653** / 1.19 / **2116** / 1.33	- / - / - / - / - / - / - / - / -
(1094,39,71,3,1567,4)	LH(3)	0.01 / 78 / 96 / 35 / **5481** / 1.90 / **1674** / 4.02	0.01 / 75 / 91 / 36 / **2750** / 1.58 / **901** / 3.12	- / - / - / - / - / - / - / - / -
	LH(6)	0.01 / 78 / 101 / 36 / **20147** / 0.52 / **583** / 11.55	0.01 / 75 / 98 / 36 / **10918** / 0.40 / **323** / 8.68	- / - / - / - / - / - / - / - / -
		i=10	i=12	i=14
	LH(0)	- / - / - / 0 / **1507** / 1.00 / **926** / 1.00	- / - / - / 0 / **39** / 1.00 / **26** / 1.00	- / - / - / 0 / **18** / 1.00 / **12** / 1.00
lf3_15_53	LH(1)	0.01 / 80 / 71 / 1 / **1149** / 1.31 / **671** / 1.38	0.01 / 80 / 77 / 2 / **37** / 1.05 / **22** / 1.20	0.01 / 80 / 76 / 3 / **18** / 1.00 / **10** / 1.20
(1480,32,71,3,2171,4)	LH(3)	0.01 / 80 / 112 / 1 / **887** / 1.70 / **413** / 2.25	0.01 / 80 / 116 / 2 / **31** / 1.26 / **14** / 1.90	0.01 / 80 / 114 / 3 / **13** / 1.38 / **5** / 2.17
	LH(6)	0.01 / 80 / 134 / 1 / **1401** / 1.08 / **198** / 4.68	0.01 / 80 / 138 / 2 / **65** / 0.60 / **8** / 3.28	0.01 / 80 / 129 / 3 / **28** / 0.64 / **3** / 3.41
Promedas networks				
		i=20	i=23	
	LH(0)	- / - / - / 7 / **10645** / 1.00 / **3528** / 1.00	- / - / - / 43 / **9228** / 1.00 / **3048** / 1.00	- / - / - / - / - / - / - / - / -
or_chain_151	LH(1)	0.01 / 106 / 98 / 15 / **9398** / 1.13 / **2930** / 1.20	0.01 / 103 / 96 / 53 / **21603** / 0.43 / **3948** / 0.77	- / - / - / - / - / - / - / - / -
(1911,94,165,2,1928,3)	LH(3)	0.01 / 106 / 148 / 15 / **8278** / 1.29 / **2176** / 1.62	0.01 / 103 / 136 / 53 / **5895** / 1.57 / **1559** / 1.96	- / - / - / - / - / - / - / - / -
	LH(6)	0.01 / 106 / 172 / 14 / **9402** / 1.13 / **1123** / 3.14	0.01 / 103 / 160 / 52 / **5934** / 1.56 / **757** / 4.03	- / - / - / - / - / - / - / - / -
		i=15	i=20	i=24
	LH(0)	- / - / - / 0 / **2958** / 1.00 / **1328** / 1.00	- / - / - / 4 / **917** / 1.00 / **403** / 1.00	- / - / - / 64 / **542** / 1.00 / **210** / 1.00
or_chain_230	LH(1)	0.01 / 82 / 78 / 3 / **2590** / 1.14 / **1086** / 1.22	0.01 / 68 / 63 / 8 / **863** / 1.06 / **364** / 1.11	0.01 / 64 / 62 / 72 / **506** / 1.07 / **184** / 1.14
(1338,61,109,2,1357,3)	LH(3)	0.01 / 82 / 111 / 3 / **2169** / 1.36 / **739** / 1.80	0.01 / 60 / 95 / 8 / **800** / 1.15 / **290** / 1.39	0.01 / 64 / 95 / 70 / **435** / 1.25 / **130** / 1.61
	LH(6)	0.01 / 82 / 123 / 3 / **3552** / 0.83 / **379** / 3.50	0.01 / 40 / 91 / 8 / **1180** / 0.78 / **202** / 2.00	0.01 / 64 / 107 / 70 / **615** / 0.88 / **83** / 2.52

Table 1: Statistics on pedigree, largefam, promedas instances for exact solutions. We selected 2-3 i-bounds for each instance, aiming to vary magnitudes of search, ranging from significant to minimal (dictated by our memory limit of 4Gb for constructing the MBE heuristic). The time and nodes are emphasized in **bold**. For each i-bound, we box the best particular time across the different look-ahead depths.

Benchmark	# inst	n	k	w	h	$\|F\|$	a
Pedigree	17	387 1015	3 7	19 39	58 143	438 1290	4 5
LargeFam	14	950 1530	3 3	32 40	66 95	1383 2352	4 4
Promedas	28	735 2113	2 2	41 120	77 180	749 2134	3 3

Table 2: Benchmark statistics. # inst - n. of instances, n - n. of variables, w - induced width, h - pseudotree height, k - max. domain size, $|F|$ - n. of functions, a - max. arity. The top value is the min. and the bottom value is the max. for that statistic.

Results

In **Table 1**, we report detailed statistics across a representative set of instances from our benchmarks. The main three columns cover the different i-bounds. Each tuple reports the ϵ, and the number of ϵ-relevant variables (rv), as well as the number of variables where look-ahead was applied (i.e., whose pruned trees were not empty) (nel). We report preprocessing time (in seconds) (ptime), total CPU time (in seconds) (time), its associated speedup (spd), the number of nodes generated (in millions of nodes), and its associated ratio of node reduction (red). We report a subset of depths for space reasons.

Deeper look-ahead yields smaller search space. As expected, we see universally across all our experiments that more look-ahead results in smaller search space . However, this is not always cost effective.

Look-ahead benefits weaker heuristics more. For example, on pedigree7 the best speedup at i=6 is 1.84 while the best speedup at i=10 is 1.33; on pedigree18 the best speedup at i=5 is 4.21 while the best speedup at i=8 is 2.92.

For weaker heuristics, deeper look-ahead is more effective. For example, we observe that on pedigree7 the best look-ahead at i=6 is d=6 while the best look-ahead at i=10 is d=3; on pedigree18 the best look-ahead at i=5 is d=6 while the best look-ahead at i=8 is d=3.

Weaker heuristics will invite more look-ahead for the same ϵ. We observe how the threshold of 0.01 yields a fraction of ϵ-relevant variables leading to a small subset of variables where look-ahead is applied (lhn). These numbers also differ depending on the heuristic strength.

Figure 2 presents best speedups using scatter diagrams as well as avg/min/max best speedups per benchmark set.

Figure 3 provides a comparison of average speedups between the full look-ahead tree vs the minimal pruned tree, as a function of the look-ahead level d, using a fixed threshold of 0.01. The full look-ahead tree corresponds to applying

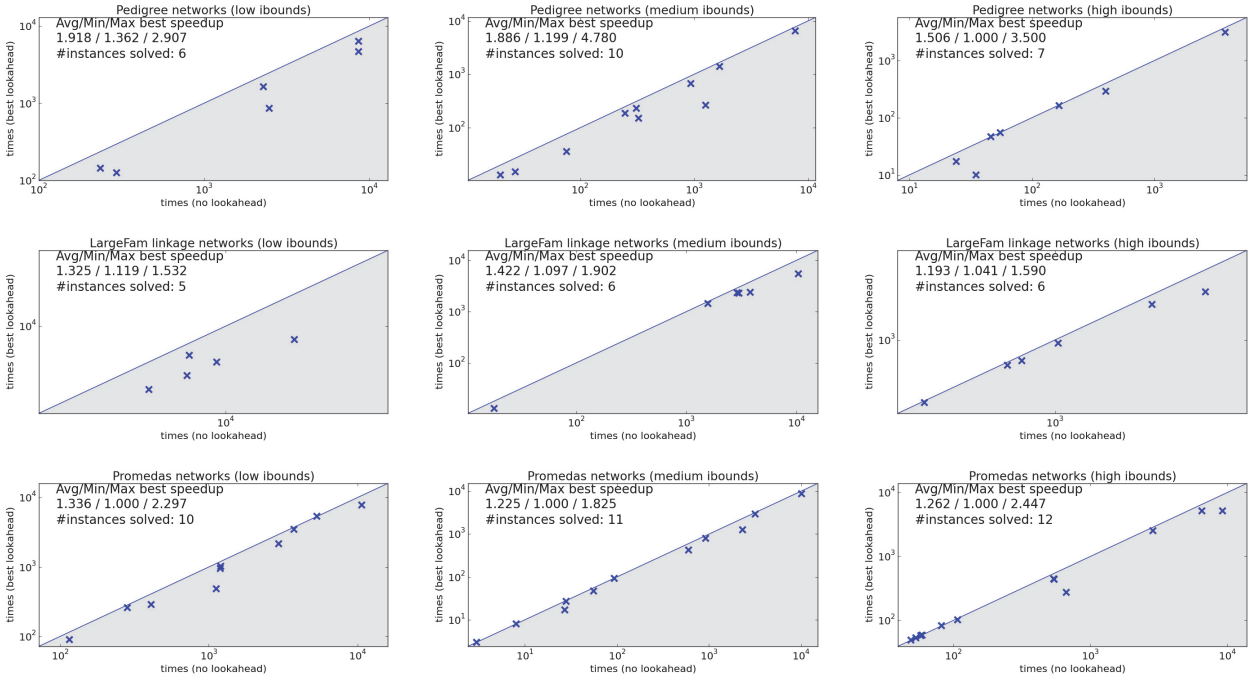


Figure 2: Total CPU times for using no look-ahead plotted against best total CPU times for any level of look-ahead (in log scale). Each point in the gray region represents an instance where the look-ahead heuristic has better time performance for some setting of the depth. We also report the number of instances which were actually solved and the average speedup across these instances.

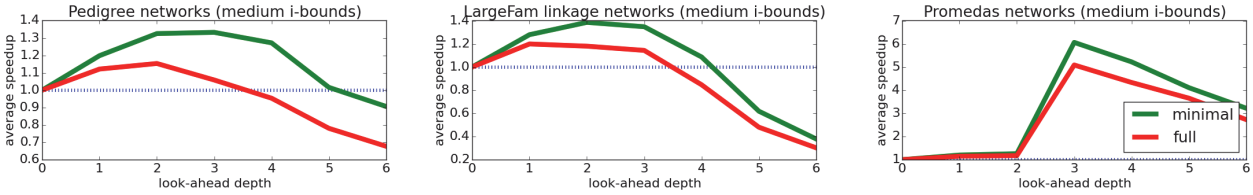


Figure 3: Average speedups relative to using no look-ahead per depth for the minimal and full look-ahead subtrees.

lookahead naively. We observe here the advantage of using minimal pruned trees. For example, on the `pedigree` instances, there is almost no speedup on average when using the full look-ahead subtree by a depth of 3, but using the minimal pruned tree allows look-ahead to remain cost-effective.

Conclusion

This paper opens up an investigation on the use of look-ahead in graphical models. We address the challenge of finding an effective balance between the look-ahead overhead and its punning power, by exploiting the graphical structure. We have showed a general relationship between the residual of the mini-bucket heuristic and a local bucket-error due to the mini-bucket algorithm. This relationship shows that the residual (and consequently depth-1 look-ahead) can be pre-compiled using a message-passing like computation. As such, we showed that the bucket-errors can assist in controlling the look-ahead at any depth. We showed empirically that even this simple mechanism of identifying irrelevant variables can be instrumental in making deep-level look-head cost-effective. In addition, for the mini-bucket heuristic, we have characterized look-ahead as an inference task, which allows a full inference solving approach. In the future, we plan to automate parameter selection for optimizing look-ahead, and explore the potential for anytime combinatorial optimization.

Acknowledgements

We thank the reviewers for their valuable feedback. This work was suppported in part by NSF grants IIS-1065618, IIS-1526842, and IIS-1254071, the US Air Force under Contract No. FA8750-14-C-0011 under the DARPA PPAML program, and MINECO under projects TIN2013-45732-C4-3-P and TIN2015-69175-C4-3-R.

References

Bu, Z.; Stern, R.; Felner, A.; and Holte, R. C. 2014. A* with lookahead re-evaluated. In Edelkamp, S., and Barták, R., eds., *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014*. AAAI Press.

Dechter, R. 2013. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Ihler, A.; Flerova, N.; Dechter, R.; and Otten, L. 2012. Join-graph based cost-shifting schemes. In *Uncertainty in Artificial Intelligence (UAI)*. Corvallis, Oregon: AUAI Press. 397–406.

Kask, K., and Dechter, R. 1999. Branch and bound with mini-bucket heuristics. *Proc. IJCAI-99*.

Kask, K., and Dechter, R. 2001. A general scheme for automatic search heuristics from specification dependencies. *Artificial Intelligence* 91–131.

L. Otten, A. Ihler, K. K., and Dechter, R. 2012. Winning the pascal 2011 map challenge with enhanced and/or branch-and-bound. In *Workshop on DISCML 2012 (a workshop of NIPS 2012)*.

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

Nilsson, N. J. 1980. *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA.

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

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 150–160.