Branch and Bound with Mini-Bucket Heuristics

Kalev Kask and Rina Dechter

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

Abstract

The paper describes a branch and bound scheme that uses heuristics generated mechanically by the mini-bucket approximation. This scheme is presented and evaluated for optimization tasks such as finding the Most Probable Explanation (*MPE*) in Bayesian networks. The mini-bucket scheme yields monotonic heuristics of varying strengths which cause different amounts of pruning, allowing a controlled tradeoff between preprocessing and search. The resulting Branch and Bound with Mini-Bucket heuristic (BBMB), is evaluated using random networks, probabilistic decoding and medical diagnosis networks. Results show that the BBMB scheme overcomes the memory explosion of bucket-elimination allowing a gradual tradeoff of space for time, and of time for accuracy.

1 Introduction

This paper proposes a new scheme for augmenting branch-and-bound search with heuristics generated automatically by the *Mini-Bucket* algorithms. Mini-bucket is a class of parameterized approximation algorithms based on the recently proposed bucket-elimination framework. The approximation uses a controlling parameter which allows adjustable levels of accuracy and efficiency [Dechter and Rish, 1997]. The algorithms were presented and analyzed for deterministic optimization tasks and probabilistic tasks, such as finding the most probable explanation (MPE), belief updating, and finding the maximum a posteriori hypothesis. Encouraging empirical results were reported for MPE on randomly generated noisy-or networks, on medical-diagnosis CPCS networks, and on coding problems [Rish *et al.*, 1998]. In some cases, however, the approximation was largely suboptimal, even when using the highest feasible accuracy level.

One way of improving the mini-bucket scheme is by embedding it in a general search algorithm. The intermediate functions created by the mini-bucket scheme can be interpreted as a heuristic evaluation function and used by any heuristic search algorithm. For instance, an upper bound on the probability of the best possible extension of any partial assignment in an MPE task can be derived. The tightness of these bounds can be controlled by the accuracy parameter of the mini-bucket scheme.

In this paper we evaluate this idea using Branch-and-Bound search which searches the space of partial assignments in a depth-first manner. It expands a partial assignment only if its upper-bounding heuristic estimate is larger than the currently known best lower bound. The virtue of branch-and-bound compared to best-first search, is that it requires a limited amount of memory and can be used as an anytime scheme - when interrupted, Branch-and-Bound outputs the best solution found so far. In [Kask and Dechter, 1999a] we apply this approach to Best-First search and compare the two schemes.

The resulting search algorithm, BBMB (Branch and Bound with Mini-Bucket heuristics) is evaluated and compared against other algorithms (such as bucket elimination, the mini-bucket scheme and iterative belief propagation), on a number of test problems, including coding networks, random networks, and CPCS networks. We show that the BBMB scheme is effective for a larger range of problems because of its gradual trade-off between preprocessing and search, and time and accuracy. Unlike bucket elimination, BBMB does not suffer from memory explosion and is often quicker to find an optimal solution. We investigated this approach for the optimization task of finding the *Most Probable Explanation* (MPE).

Section 3 presents an overview of the relevant algorithms. In Section 4 we describe our branch-and-bound scheme and its guiding heuristic function. Section 5 presents empirical evaluations, while Section 6 provides discussion and conclusions. For space reasons we omit all proofs. For more details see [Kask and Dechter, 1999a].

1.1 Related work

MPE appears in applications such as medical diagnosis, circuit diagnosis, natural language understanding and probabilistic decoding. For example, given data on clinical findings, MPE can postulate on a patient's probable affliction. In decoding, the task is to identify the most likely input message transmitted over a noisy channel

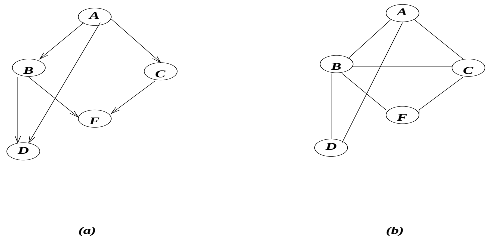


Figure 1: belief network $P(f, d, c, b, a) = P(f|c, b)P(d|b, a)\ P(b|a)P(c|a)$

given the observed output. Researchers in natural language consider the understanding of text to consist of finding the most likely facts (in internal representation) that explains the existence of the given text. In computer vision and image understanding, researchers formulate the problem in terms of finding the most likely set of objects that explains the image. Scientific theories are models that attempt to fit the given observations, and so on.

It is known that solving the MPE task is NP-hard. Complete algorithms for MPE use either the *cycle cutset* technique or the *join-tree-clustering* [Pearl, 1988] and the bucket-elimination scheme [Dechter, 1996]. However, these methods work well only if the network is sparse enough to allow small cutsets or small clusters. Following Pearl's stochastic simulation algorithms for the MPE task [Pearl, 1988], the suitability of Stochastic Local Search (SLS) algorithms for MPE was studied in the context of Medical diagnosis applications [Peng and Reggia, 1986], [Peng and Reggia, 1989] and more recently in [Kask and Dechter, 1999b]. Best first search algorithms were also proposed [Shimony and Charniak, 1991] as well as algorithms based on linear programming [Santos, 1991].

2 Background

2.1 Notation and definitions

Belief Networks provide a formalism for reasoning about partial beliefs under conditions of uncertainty. They are defined by a directed acyclic graph over nodes representing random variables of interest.

DEFINITION **2.1 (Belief Networks)** *Given a set, $X = \{X_1, \ldots, X_n\}$ of random variables over multivalued domains $D_1, \ldots, D_n$, a belief network is a pair (G, P) where G is a directed acyclic graph and $P = \{P_i\}$. $P_i = \{P(X_i \,|\, pa\,(X_i)\,)\}$ are conditional probability matrices associated with X_i. The set $pa(X_i)$ is called the parent set of X_i. An assignment $(X_1 = x_1, ..., X_n = x_n)$ can be abbreviated to $x = (x_1, ..., x_n)$. The BN represents a probability distribution $P(x_1, ..., x_n) = \Pi_{i=1}^n P(x_i|x_{pa(X_i)})$, where, x_S is the projection of x over a subset S. An evidence set e is an instantiated subset of variables. The argument set of a function h is denoted $S(h)$.*

DEFINITION **2.2 (Most Probable Explanation)** *Given a belief network and evidence e, the Most Probable Explanation (MPE) task is to find an assignment $(x_1^o, \ldots, x_n^o)$ such that*

$$P(x_1^o, \ldots, x_n^o) = max_{X_1,...,X_n} \prod_{k=1}^{n} P(X_k \,|\, pa\,(X_k),\, e)$$

DEFINITION **2.3 (graph concepts)** *An* ordered graph *is a pair (G, d) where G is an undirected graph and $d = X_1, ..., X_n$ is an ordering of the nodes. The* width *of a* node *in an ordered graph is the number of its earlier neighbors. The* width w(d) *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 by processing the nodes recursively, from last to first; when node X is processed, all its earlier neighbors are connected. The* moral graph *of a directed graph G is the undirected graph obtained by connecting the parents of all the nodes in G and then removing the arrows. An example of a belief network is given in Figure 1a, and its moral graph in Figure 1b.*

2.2 Bucket and mini-bucket algorithms

Bucket elimination is a unifying algorithmic framework for dynamic-programming algorithms applicable to probabilistic and deterministic reasoning [Dechter, 1996]. The input to a bucket-elimination algorithm consists of a collection of functions or relations (e.g., clauses for propositional satisfiability, constraints, or conditional probability matrices for belief networks). Given a variable ordering, the algorithm partitions the functions into buckets, each placed in the bucket of its latest argument in the ordering. The algorithm has two phases. During the first, top-down phase, it processes each bucket, from the last variable to the first by a variable elimination procedure that computes a new function, placed in a lower bucket. For MPE, this procedure computes a product of all probability matrices in the bucket and maximizes over the bucket's variable. During the second, bottom-up phase, the algorithm constructs a solution by assigning a value to each variable along the ordering, consulting the functions created during the top-down phase.

THEOREM **2.1** *[Dechter, 1996] The time and space complexity of Elim-MPE, the bucket elimination algorithm for MPE, are exponential in the induced width $w^*(d)$ of the network's ordered moral graph along ordering d.* □

The *Mini-bucket elimination* is an approximation scheme designed to avoid the space and time problems of full bucket elimination. In each bucket, all the functions are first partitioned into smaller subsets called mini-buckets which are then processed independently. Here is the rationale. Let $h_1, ..., h_j$ be the functions in $bucket_p$. When *Elim-MPE* processes $bucket_p$, it computes the function h^p: $h^p = max_{X_p} \Pi_{i=1}^j h_i$. The mini-bucket algorithm, on the other hand, creates a partitioning $Q' = \{Q_1, ..., Q_r\}$ where the mini-bucket Q_l contains the functions $h_{l_1}, ..., h_{l_k}$. The approximation will compute $g^p = \Pi_{l=1}^r max_{X_p} \Pi_{l_i} h_{l_i}$. Clearly, $h^p \leq g^p$. Thus,

Algorithm Approx-MPE(i) (MB(i))
Input: A belief network $BN = \{P_1, ..., P_n\}$; ordering d;
Output: An upper bound on the MPE, an assignment and the set of ordered augmented buckets.
1. **Initialize:** Partition matrices into buckets. Let $S_1, ..., S_j$ be the subset of variables in $bucket_p$ on which matrices (old or new) are defined.
2. **(Backward)** For $p \leftarrow n$ downto 1, do
• **If** $bucket_p$ contains $X_p = x_p$, assign $X_p = x_p$ to each h_i and put each in appropriate bucket.
• **else,** for $h_1, h_2, ..., h_j$ in $bucket_p$, generate an (i)-partitioning, $Q' = \{Q_1, ..., Q_r\}$. For each $Q_l \in Q'$ containing $h_{l_1}, ...h_{l_t}$ generate function h^l, $h^l = max_{X_p} \Pi_{i=1}^{t} h_{l_i}$. Add h^l to the bucket of the largest-index variable in $U_l \leftarrow \bigcup_{i=1}^{j} S(h_{l_i}) - \{X_p\}$.
3. **(Forward)** For $i = 1$ to n do, given $x_1, ..., x_{p-1}$ choose a value x_p of X_p that maximizes the product of all the functions in X_p's bucket.
4. Output the ordered set of augmented buckets, an upper bound and a lower bound assignment.

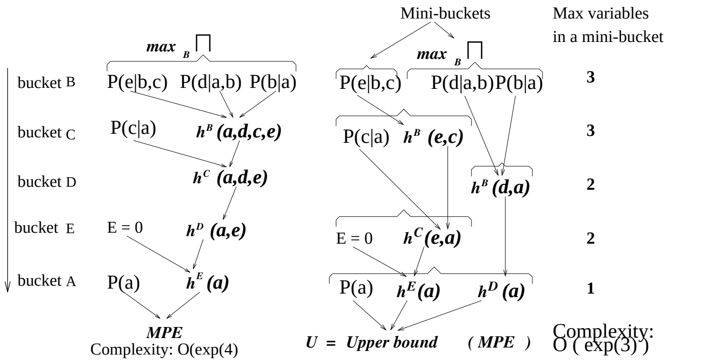
Figure 2: algorithm *Approx-MPE(i)*

the algorithm computes an upper bound on the probability of the MPE assignment. Subsequently, in its second phase, the algorithm computes an assignment that provides a lower bound. The quality of the upper bound depends on the degree of the partitioning into mini-buckets. Given a bound parameter i, the algorithm creates an i-partitioning, where each mini-bucket includes no more than i variables. Algorithm *Approx-MPE(i)* (sometimes called MB(i)) is described in Figure 2. The algorithm outputs not only an upper bound on the MPE and an assignment (whose probability yields a lower bound), but also the collection of augmented buckets. By comparing the upper bound to the lower bound we can always have a bound on the error for the given instance.

THEOREM **2.2** *[Dechter and Rish, 1997] Algorithm Approx-MPE(i) generates an upper bound on the exact MPE and its time and space complexity is $O(exp(i))$.*

When the bound i is large enough (i.e. when $i \geq w^*$), the algorithm coincides with full bucket elimination.

Example 2.3 *Figure 3(b) illustrates how algorithms* Elim-MPE *and* Approx-MPE(i) *for $i = 3$ process the network in Figure 1(a) along the ordering (A, E, D, C, B).* Elim-MPE *records new functions $h^B(a, d, c, e)$, $h^C(a, d, e)$, $h^D(a, e)$, and $h^E(a)$ during its backwards phase. Then, in the bucket of A, it computes MPE $= \max_a P(a)h^E(a)$. Subsequently, an MPE assignment ($A = a'$, $B = b'$, $C = c'$, $D = d'$, $E = 0$) ($E = 0$ is an evidence) is computed by maximizing the product functions in each bucket. Namely, $a' = \arg\max_a P(a)h^E(a)$, $e' = 0$, $d' = \arg\max_d h^C(a', d, e = 0)$, and so on. The approximation* Approx-MPE(3) *splits bucket B into two mini-buckets each containing no more than 3 variables, and generates $h^B(e, c)$ and $h^B(d, a)$. An upper bound U on the MPE value is computed in A's bucket, $U = max_a P(a) \cdot h^E(a) \cdot h^D(a)$. A suboptimal MPE tuple is computed by assigning a value to each variable that maximizes the product of functions in the corresponding bucket, given the values of preceding variables.*



(a) A trace of *Elim-MPE* (b) A trace of *approx-mpe(3)*.

Figure 3: (a) Algorithm *approx-mpe(i,m)* and (b) a trace of the algorithm's execution.

3 Heuristic Search with Mini-Bucket

3.1 Notation

Given an ordered set of augmented buckets generated by the mini-bucket algorithm along $d = X_1, ..., X_n$, we use the following convention

- P_{p_j} denotes an input conditional probability matrices placed in bucket p, (namely, its highest-ordered variable is X_p).
- h_{p_j} denotes an arbitrary function in bucket p generated by the mini-bucket algorithm.
- h_j^p denotes a function created by the j-th mini-bucket in bucket p.
- λ_{p_j} denotes an arbitrary function in bucket p.

We denote by $buckets(1..p)$ the union of all functions in the bucket of X_1 through the bucket of X_p. Remember that $S(f)$ denotes the set of arguments of function f.

3.2 The idea

We will now show that the new functions recorded by the mini-bucket algorithm can be used to express upper bounds on the most probable extension of any partial assignment. Therefore, they can serve as heuristics in an evaluation function which guides either a *best-first search* or a *branch-and-bound search.*

DEFINITION **3.1 (Exact Evaluation Function)** *Let $\bar{x} = \bar{x}^p = (x_1, ..., x_p)$. The probability of the most probable extension of $\bar{x}^p$, denoted $f^*(\bar{x}^p)$ is:*

$$max_{\{X_{p+1},...,X_n | X_i = x_i, \ \forall \ i, \ 1 \leq i \leq p\}} \prod_{k=1}^{n} P(X_k \mid pa(X_k), e)$$

The above product defining f^* can be divided into two smaller products expressed by the functions in the ordered augmented buckets. In the first product all the arguments are instantiated, and therefore the maximization operation is applied to the second product only. Denoting

$$g(\bar{x}) = \prod_{P_i \in \; buckets(1..p)} P_i(\bar{x}_{S(P_i)})$$

and

$$H^*(\bar{x}) = max_{\{X_{p+1},...,X_n | X_i = x_i, \; \forall \; 1 \leq i \leq p\}} \prod_{P_i \in buckets(p+1..n)} P_i,$$

we get

$$f^*(\bar{x}) = g(\bar{x}) \cdot H^*(\bar{x})$$

During search, the g function can be evaluated over the partial assignment $\bar{x}^p$, while H^* can be estimated by a heuristic function. Our first proposal is to estimate $H^*(\bar{x}^p)$ by a function $H_1 = \prod_j h_{p_j}$ which is the product of all h functions generated by the mini-bucket algorithm which reside in bucket p. It can be shown, using properties of mini-bucket scheme, that this product indeed provides an upper bound on H^*, yielding an *admissible* heuristic.

However, the heuristic function H_1 is non-monotonic. In fact, even if provided with augmented buckets that are generated by exact full bucket elimination, this heuristic may still be nonmonotonic. A heuristic function is *monotonic* if the evaluation function along any path in the search tree is not increasing.

We next modify H_1 to make it monotone and more accurate. The *modified* heuristic function H_2 adds to the product of H_1 all those h functions in buckets(1..p-1) that were generated in buckets processed before bucket p. The rationale is that some conditional probability matrices in H^* are approximated by h functions that are placed below bucket p (because they are not defined over X_p).

DEFINITION **3.2** *Given an ordered set of augmented buckets, the heuristic function $H_2(\bar{x}^p)$, is the product of all the h functions that satisfy the following two properties: 1) They are generated in buckets $(p+1, ..., n)$, and 2) They reside in buckets 1 through p. Namely, $H_2(\bar{x}^p) = \prod_{i=1}^{p} \prod_{h_j^k \in bucket_i} h_j^k$, where $k > p$, (i.e. h_j^k is generated by a bucket processed before bucket p.)*

THEOREM **3.1 (Mini-Bucket Heuristic)** *For every partial assignment $\bar{x} = \bar{x}^p = (x_1, ..., x_p)$, of the first p variables, the evaluation function $f(\bar{x}^p) = g(\bar{x}^p) \cdot H_2(\bar{x}^p)$ is: 1) Admissible - it never underestimates the probability of the best extension of $\bar{x}^p$. 2) Monotonic, namely $f(\bar{x}^{p+1})/f(\bar{x}^p) \leq 1$.* □

The following proposition shows how $g(\bar{x}^{p+1})$ and $H(\bar{x}^{p+1})$ can be updated recursively based on $g(\bar{x}^p)$ and $H(\bar{x}^p)$ and functions residing in bucket $p+1$. (From now on we will use H to mean H_2.)

Proposition 1 *Given a partial assignment $\bar{x}^p = (x_1, \ldots x_p)$, both $g(\bar{x}^p)$ and $H(\bar{x}^p)$ can be computed recursively by*

$$g(\bar{x}^p) = g(\bar{x}^{p-1}) \cdot \Pi_j P_{p_j}(\bar{x}^p_{S(P_{p_j})}) \qquad (1)$$

$$H(\bar{x}^p) = H(\bar{x}^{p-1}) \cdot \Pi_k h_{p_k} / \Pi_j h_j^p \qquad (2)$$

Algorithm BBMB(i)
Input: A belief network $BN = \{P_1, ..., P_n\}$; ordering d.
Output: An MPE assignment, or a lower bound and an upper-bound on the MPE.
1. **Initialize:** Run MB(i) algorithm which generates a set of ordered augmented buckets and an upper-bound on MPE. Set lower bound L to 0. Set current variable index p to 0.
2. **Search:** Execute the following procedure until variable X_1 has no legal values left.
• **Expand:** Given a partial instantiation $\bar{x}^p$, compute all partial assignments $\bar{x}^{p+1} = (\bar{x}^p, v)$ for each value v of X_{p+1}. For each node $\bar{x}^{p+1}$ compute its heuristic value $f(\bar{x}^{p+1}) = g(\bar{x}^{p+1}) \cdot H(\bar{x}^{p+1})$ using $g(\bar{x}^{p+1}) = g(\bar{x}^p) \cdot \Pi_j P_{p+1_j}$ and $H(\bar{x}^{p+1}) = H(\bar{x}^p) \cdot \Pi_k h_{p+1_k} / \Pi_j h_j^{p+1}$.
Prune those assignments $\bar{x}^{p+1}$ for which $f(\bar{x}^{p+1})$ is smaller than the lower bound L.
• **Forward:** If X_{p+1} has no legal values left, goto Backtrack. Otherwise let $\bar{x}^{p+1} = (\bar{x}^p, v)$ be the best extension to $\bar{x}^p$ according to f. If $p+1 = n$, then set $L = f(\bar{x}^{p+1})$ and goto Backtrack. Otherwise remove v from the list of legal values. Set $p = p+1$ and goto Expand.
• **Backtrack:** If p = 1, Exit. Otherwise set $p = p-1$ and repeat the Forward step.

Figure 4: Algorithm *BBMB(i)*

3.3 Search with Mini-Bucket Heuristics

The tightness of the upper bound generated by the mini-bucket approximation depends on its i-bound. Larger values of i yield better upper-bounds, but require more computation. Therefore, Branch-and-Bound search, if parameterized by i, allows a controllable tradeoff between preprocessing and search, or between heuristic strength and its overhead.

In Figure 4 we present algorithm BBMB(i). This algorithm is initialized by running the mini-bucket algorithm, producing the set of ordered augmented buckets. It then traverses the search space in a depth-first manner, instantiating variables from first to last. Throughout the search, the algorithm maintains a lower bound on the probability of the MPE assignment, which corresponds to the probability of the best full variable instantiation found thus far. It uses the heuristic evaluation function $f(\bar{x}^p)$ to prune the search space. Search terminates when it reaches a time-bound or when the first variable has no values left. In the latter case, the algorithm has found an optimal solution.

The heuristic function generated by the mini-bucket approximation can also be used to guide any A* type algorithm. For a description of *Best-First* search with mini-bucket heuristics see [Kask and Dechter, 1999a].

4 Experimental Methodology

We tested the performance of our scheme on sev-

N C,P,K	Elim MPE #[time] w^*	*opt*	MB / BBMB i=2 #[time]	MB / BBMB i=4 #[time]	MB / BBMB i=6 #[time]	MB / BBMB i=8 #[time]	MB / BBMB i=10 #[time]	MB / BBMB i=12 #[time]	MB / BBMB i=14 #[time]
256 100,2,2	91[4.91] 14.6	>0.95	14[0.03] 89[6.30]	30[0.03] 100[1.04]	44[0.04] 100[0.21]	**50[0.07]** **100[0.13]**	63[0.15] 100[0.18]	68[0.40] 100[0.45]	81[1.20] 100[1.28]
256 105,2,2	69[7.11] 15.8	>0.95	6[0.03] 83[8.86]	20[0.03] 99[1.38]	29[0.04] 100[0.69]	**42[0.08]** **100[0.19]**	47[0.17] 100[0.26]	67[0.49] 100[0.52]	75[1.54] 100[1.63]
256 110,2,2	41[9.06] 17.5	>0.95	8[0.04] 77[14.1]	15[0.03] 100[3.68]	23[0.04] 99[1.07]	33[0.08] 100[0.41]	**47[0.18]** **100[0.35]**	55[0.57] 100[0.70]	67[1.85] 100[1.98]
256 115,2,2	17[12.3] 19.1	>0.95	6[0.03] 71[17.5]	10[0.03] 98[5.74]	16[0.05] 100[1.78]	22[0.09] 100[0.85]	**43[0.21]** **100[0.71]**	37[0.66] 100[0.84]	51[2.17] 100[2.34]
256 120,2,2	11[9.99] 20.3	>0.95	4[0.04] 57[20.4]	10[0.04] 95[6.50]	10[0.05] 97[2.60]	29[0.08] 100[1.96]	**30[0.22]** **100[0.97]**	39[0.66] 100[1.00]	53[2.10] 100[2.35]
256 125,2,2	2[21.1] 22.4	>0.95	2[0.03] 49[21.5]	8[0.04] 86[9.44]	9[0.05] 95[4.79]	21[0.09] 96[2.17]	28[0.25] 100[1.62]	**35[0.78]** **100[1.52]**	35[2.50] 100[3.34]

Table 1: Random MPE. Time bound 30 sec. 100 samples.

eral types of networks. On each problem instance we ran bucket elimination (Elim-MPE), mini-bucket approximation with various i-bounds (MB(i)) and branch and bound with some levels of mini-bucket heuristics (BBMB(i)).

The main measure of performance of the approximation algorithm given a fixed time bound, is the accuracy ratio $opt = P_{alg}/P_{MPE}$ between the probability of the solution found by the test algorithm (P_{alg}) and the probability of the optimal solution (P_{MPE}), whenever P_{MPE} is available. We also record the running time of each algorithm.

We report the distribution of problems with respect to 5 predefined ranges of accuracy : $opt \geq 0.95$, $opt \geq 0.5$, $opt \geq 0.2$, $opt \geq 0.01$ and $opt < 0.01$. We recorded the number of problems that Elim-MPE solved as well as the average induced width w^* of the test problems. Because of space restrictions in most cases we report only the number of problems that fall in the highest accuracy range $opt \geq 0.95$. However, since most problems were solved by BBMB optimally, we lose only minimal information.

In addition, during the execution of BBMB we also stored the current lower bound L at regular time intervals. Comparing the lower bound against the optimal solution, allows reporting the accuracy of BBMB as a function of time.

4.1 Random Bayesian Networks and Noisy-OR Networks

Random Bayesian networks and Noisy-OR networks were randomly generated using parameters (N, K, C, P), where N is the number of variables, K is their domain size, C is the number of conditional probability matrices and P is the number of parents in each conditional probability matrix.

The structure of each test problem is created by randomly picking C variables out of N and for each, randomly selecting P parents from preceding variables, relative to some ordering. For random Bayesian networks, each probability table is generated randomly. For Noisy-OR networks, each probability table represents an OR-function with a given noise and leak probabilities : $P(X = 0|Y_1, ..., Y_P) = P_{leak} \times \Pi_{Y_i=1} P_{noise}$.

Tables 1 and 2 present results of experiments with random Bayesian networks and Noisy-OR networks respectively. In each table, parameters N, K and P are fixed, while C, controlling network's sparseness, is changing. For each C, we generate 100 problem instances. Each entry in the table reports the number of instances that fall in a specific range of accuracy, as well as the average running time of each algorithm (note that BBMB time includes the preprocessing time by MB).

For example, Table 1 reports the results with random problems having N=256, K=2, P=2. There are 5 horizontal blocks, each corresponding to a different value of C. Each block has two rows, one for MB and one for BBMB, reporting the number of problems in accuracy range of 0.95 and the average running time.

The second column reports the results of Elim-MPE, namely the number of instances it solved, their average w^* and running time. The rest of the columns report results of MB and BBMB for various levels of i. Looking at the first line in Table 1 we see that in the best accuracy range, $opt \geq 0.95$, MB with $i = 2$ solved only 14 problems using 0.03 seconds on the average. BBMB with $i = 2$ solved 89 instances in this range while using 6.30 seconds on the average. Note that Elim-MPE on the other hand solved 91 instances using much more time than BBMB with any i-bound.

Each row demonstrates a tradeoff between preprocessing by MB and subsequent search by BBMB. As expected, MB can solve more instances as i increases while its average time increases. At the same time, the search time of BBMB decreases as i increases. We observe that the total BBMB time improves when i increases until a threshold point and then worsens. We have highlighted the best performance point in each row. Below this threshold, the heuristic function is weak, resulting in long search. Above the threshold, the extra preprocessing is not cost effective. As problems become harder BBMB achieves its best performance at higher thresholds. For example, when C is 100 and 105, the threshold is $i = 8$, when C is 110, 115 and 120, it is $i = 10$, and when C is 125, it is $i = 12$.

Table 2 reporting on Noisy-OR shows similar results (based on $opt \geq 0.95$ range). On this class BBMB is very effective and solved all problems exactly, while Elim-

N C,P	Elim MPE #[time] w^*	*opt*	MB / BBMB i=2 #[time]	MB / BBMB i=6 #[time]	MB / BBMB i=10 #[time]	MB / BBMB i=14 #[time]
128 90,2	6[10.9] 20.4	>0.95	31[0.04] 100[0.54]	**42[0.05]** **100[0.11]**	49[0.22] 100[0.25]	71[2.22] 100[2.22]
128 95,2	1[7.67] 21.5	>0.95	26[0.02] 99[0.78]	**36[0.04]** **100[0.11]**	44[0.22] 100[0.27]	59[2.45] 100[2.46]
128 100,2	0[-] 23.6	>0.95	10[0.05] 98[1.01]	**26[0.06]** **100[0.25]**	44[0.27] 100[0.33]	57[2.77] 100[2.79]
128 105,2	0[-] 24.7	>0.95	16[0.03] 99[2.18]	29[0.04] 100[0.45]	**38[0.26]** **100[0.37]**	48[3.17] 100[3.20]

Table 2: Noisy-OR MPE. Noise 0.2, Leak 0.01. Time bound 30 sec. 100 samples. 10 evidence.

MPE solved almost none. For $i \geq 6$ BBMB can solve all 100 instances, with threshold $i = 6$.

In Figures 5 and 6 we provide an alternative view of the performance of BBMB(i). Let $F_{BBMB(i)}(t)$ be the fraction of the problems solved completely by BBMB(i) by time t. Each graph in Figures 5 and 6 plots $F_{BBMB(i)}(t)$ for some specific value i. Figure 5 shows the distribution of $F_{BBMB(i)}(t)$ for random Bayesian networks when N=256, C=125, K=2 and P=2 (corresponding to the last row in Table 1), whereas Figure 6 shows the distribution of $F_{BBMB(i)}(t)$ for Noisy-OR networks when N=128, C=95 and P=2 (corresponding to the second row in Table 2).

Figures 5 and 6 display a trade-off between preprocessing and search. Clearly, if $F_{BBMB(i)}(t) > F_{BBMB(j)}(t)$, then $F_{BBMB(i)}(t)$ completely dominates $F_{BBMB(j)}(t)$. For example, in Figure 5 BBMB(10) completely dominates BBMB(6). When $F_{BBMB(i)}(t)$ and $F_{BBMB(j)}(t)$ intersect, they display a trade-off as a function of time. For example, if we have only few seconds, BBMB(6) is better than BBMB(14). However, when sufficient time is allowed, BBMB(14) is superior to BBMB(6).

The same pattern appears in Figure 6. BBMB(6) completely dominates BBMB(2), while there is a trade-off between BBMB(2) and BBMB(14) depending on the amount of time allowed.

4.2 Random Coding Networks

Our random coding networks fall within the class of *linear block codes*. They can be represented as four-layer belief networks (Figure 7). The second and third layers correspond to input information bits and parity check bits respectively. Each parity check bit represents an XOR function of input bits u_i. Input and parity check nodes are binary while the output nodes are real-valued. In our experiments each layer has the same number of nodes because we use code rate of R=K/N=1/2, where K is the number of input bits and N is the number of transmitted bits.

Given a number of input bits K, number of parents P for each XOR bit and channel noise variance σ^2, a coding network structure is generated by randomly picking parents for each XOR node. Then we simulate an input signal by assuming a uniform random distribution of information bits, compute the corresponding values of the parity check bits, and generate an assignment to

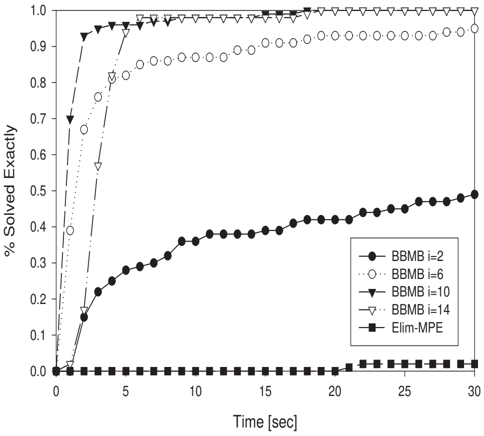


Figure 5: Random Bayesian. N=256, C=125, K=2, P=2. 100 samples.

N=100 K=50 σ	*opt*	MB / BBMB i=6	MB / BBMB i=10	MB / BBMB i=14	IBP
0.22	>0.95	**1727**[0.06] **1998**[0.13]	1830[0.40] 2000[0.42]	1936[4.54] 2000[4.55]	2000 [0.08]
0.28	>0.95	1081[0.08] 1977[0.75]	**1306[0.38]** **1996[0.69]**	**1627**[4.55] **1999**[4.63]	1996 [0.08]
0.32	>0.95	660[0.08] 1908[1.77]	**899**[0.39] **1972[1.28]**	**1247**[4.54] **1990**[4.82]	1993 [0.08]
0.40	>0.95	172[0.06] 1522[6.19]	**272**[0.37] **1779**[4.24]	**450**[4.46] **1900**[6.51]	1741 [0.08]
0.51	>0.95	21[0.06] 811[17.0]	35[0.37] 1175[13.4]	**84**[4.41] **1435**[11.9]	616 [0.08]

Table 3: Random coding. Time 30 sec. 2000 samples.

the output nodes by adding Gaussian noise to each information and parity check bit. The decoding algorithm takes as input the coding network and the observed real-valued output assignment and recovers the original input bitvector by computing or approximating an MPE assignment.

We tested two sets of random coding networks - K=50 and K=100 input bits. Table 3 reports results on random coding networks having K=50. In addition to Elim-MPE and BBMB, we also ran Iterative Belief Propagation (IBP) which was recently observed as the best performing decoding algorithm. It is identical to iterative application of Pearl's belief updating on tree-like networks [Pearl, 1988]. For each σ we generated and tested 2000 samples divided into 200 different networks each simulated with 10 different input bit vectors.

In Table 3 we observe a familiar pattern of preprocessing-search tradeoff. For each level of noise σ there is an empirical optimal i balancing preprocessing cost and search. For $\sigma = 0.22$ the threshold is 6. As noise increases, the threshold increases. Also, as noise increases, the role of search becomes more significant. Although IBP is superior (because it is faster) for small

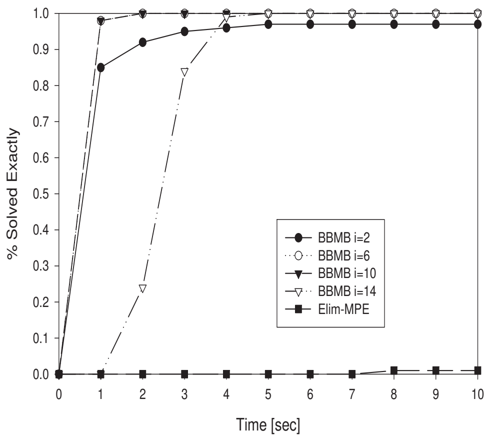


Figure 6: Noisy-OR. N=128, C=95, P=2. 100 samples.

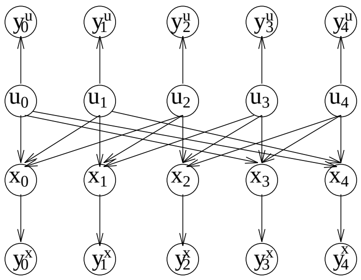


Figure 7: Belief network for structured (10,5) block code with parent set size P=3

noise, it has a shortcoming for high noise levels since its performance cannot be improved with extra time while BBMB can.

Table 4 reports the Bit Error Rate (BER) for MB/BBMB(14) and IBP. BER is a standard measure used in the coding literature denoting the fraction of input bits that were decoded incorrectly. When compared using BER, IBP is about the same as BBMB(14) when the noise is small, but slightly better when the noise is large.

Figure 8 shows the distribution of $F_{BBMB(i)}(t)$ for Random Coding networks when K=100 and $\sigma = 0.28$. We observe that no algorithm completely dominates any other algorithm. However, we can see a trade-off depending on the time available. When the time allowed is small (less then 10 seconds), BBMB(i) with i less than 14 is better. However, when more time is allowed, BBMB(14) is clearly superior.

σ	MB / BBMB i = 14	IBP
0.22	0.0012/0.00024	0.00024
0.28	0.010/0.0015	0.0015
0.32	0.025/0.0041	0.0034
0.40	0.090/0.021	0.016
0.51	0.184/0.102	0.084

Table 4: Random coding BER.

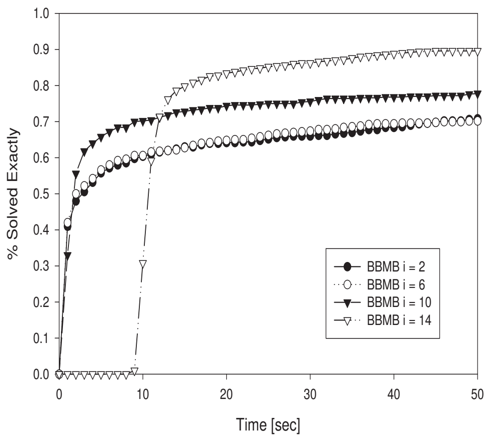


Figure 8: BBMB Random Coding. K=100, $\sigma = 0.28$. 500 samples.

4.3 CPCS Networks

As another realistic domain, we used the CPCS networks derived from the Computer-Based Patient Care Simulation system, and based on INTERNIST-1 and Quick Medical Reference expert systems [Pradhan *et al.*, 1994]. The nodes in CPCS networks correspond to diseases and findings. Representing it as a belief network requires some simplifying assumptions, 1) conditional independence of findings given diseases, 2) noisy-OR dependencies between diseases and findings, and 3) marginal independencies of diseases. For details see [Pradhan *et al.*, 1994].

In Table 5 we have results of experiments with two binary CPCS networks, cpcs360b (N = 360, C = 335) and cpcs422b (N = 422, C = 348), with 1000 and 150 instances respectively. Each instance had 10 evidence nodes picked randomly.

Since cpcs360b network is solved quite effectively by MB we see that BBMB added search time is small, serving primarily to prove the optimality of the MB solution. On the other hand, on cpcs422b MB can solve a third of the instances accurately when i is small, and more as i increases. BBMB can solve all instances accurately for $i \geq 12$ when search takes little additional time.

5 Discussion and Conclusion

Our experiments demonstrate that combining branch-and-bound with mini-bucket heuristics (BBMB) provides a powerful method of attacking optimization problems such as the MPE.

CPCS360b 1000 samples	MB / BBMB i=4	MB / BBMB i=8	MB / BBMB i=12	MB / BBMB i=16
>0.95	**939[0.94]** **1000[0.96]**	950[0.96] 1000[0.97]	983[2.04] 1000[2.04]	991[16.4] 1000[16.4]
>0.50	13[0.94] 0[-]	8[0.96] 0[-]	6[2.01] 0[-]	6[16.1] 0[-]
>0.20	44[0.95] 0[-]	38[0.97] 0[-]	11[2.03] 0[-]	3[16.6] 0[-]
>0.01	4[0.94] 0[-]	4[0.96] 0[-]	0[-] 0[-]	0[-] 0[-]
<0.01	0[-] 0[-]	0[-] 0[-]	0[-] 0[-]	0[-] 0[-]
CPCS422b 150 samples	MB / BBMB i=4	MB / BBMB i=8	MB / BBMB i=12	MB / BBMB i=16
>0.95	56[23.1] 144[25.5]	67[23.1] 148[24.5]	**78[23.2]** **150[23.4]**	98[39.9] 150[40.0]
>0.50	14[22.9] 1[45.0]	8[22.9] 0[-]	19[23.4] 0[-]	22[40.1] 0[-]
>0.20	12[23.0] 0[-]	16[22.9] 0[-]	23[23.4] 0[-]	17[40.1] 0[-]
>0.01	34[23.1] 2[45.0]	27[23.1] 2[45.0]	15[23.2] 0[-]	8[40.0] 0[-]
<0.01	34[23.1] 3[45.0]	32[22.9] 0[-]	15[23.0] 0[-]	5[40.0] 0[-]

Table 5: CPCS networks. Time 30 and 45 resp.

On the one hand it avoids the storage bottleneck of full bucket elimination while frequently solving problems optimally. On the other hand, it improves output quality substantially over the mini-bucket approximation, especially when the former is highly suboptimal. However, the most important feature of BBMB is the ability to control the balance between preprocessing (for heuristic generation) and search, using its bounding parameter i. Pure search (low i) and pure bucket-elimination (high i) are two extremes ends of that spectrum.

In all our experiments, including random Bayesian networks, Noisy-OR networks, coding networks and medical diagnosis CPCS networks, we observed that optimal performance (measured by time within a certain accuracy range) occurred at an intermediate threshold point of i. Weaker heuristics, lying below the threshold, and stronger heuristics, lying above the threshold, were less cost-effective. We also observed that as problems grew harder, stronger heuristics became more cost-effective. The control of the gradual change of space-time-accuracy tradeoff of BBMB now makes a larger set of problem solvable.

Although the best threshold point may not be predictable for every problem instance, a preliminary empirical analysis can be informative when given a class of problems that is not too heterogeneous.

We have also tested the mini-bucket heuristic function with Best-First search. We found that if we are interested in the optimal solution only, BFMB(i) is often significantly faster than BBMB(i) when given sufficient time and space. However, unlike BBMB, BFMB is not an anytime algorithm [Kask and Dechter, 1999a].

References

[Dechter and Rish, 1997] R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In *Proceedings of Uncertainty in Artificial Intelligence (UAI97)*, pages 132–141, 1997.

[Dechter, 1996] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference algorithms. In *Uncertainty in Artificial Intelligence (UAI-96)*, pages 211–219, 1996.

[Kask and Dechter, 1999a] K. Kask and R. Dechter. On the power of mini-bucket heuristics for improved search. *UCI Technical report*, 1999.

[Kask and Dechter, 1999b] K. Kask and R. Dechter. Stochastic local search for bayesian networks. In *Workshop on AI and Statistics (AISTAT99)*, 1999.

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

[Peng and Reggia, 1986] Y. Peng and J.A. Reggia. Plausability of diagnostic hypothesis. In *National Conference on Artificial Intelligence (AAAI86)*, pages 140–145, 1986.

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

[Pradhan *et al.*, 1994] M. Pradhan, G. Provan, B. Middleton, and M. Henrion. Knowledge engineering for large belief networks. In *Proc. Tenth Conf. on Uncertainty in Artificial Intelligence*, 1994.

[Rish *et al.*, 1998] I. Rish, K. Kask, and R. Dechter. Approximation algorithms for probabilistic decoding. In *Uncertainty in Artificial Intelligence (UAI-98)*, 1998.

[Santos, 1991] E. Santos. On the generation of alternative explanations with implications for belief revision. In *Uncertainty in Artificial Intelligence (UAI-91)*, pages 339–347, 1991.

[Shimony and Charniak, 1991] S.E. Shimony and E. Charniak. A new algorithm for finding map assignments to belief networks. In *P. Bonissone, M. Henrion, L. Kanal, and J. Lemmer Eds. Uncertainty in Artificial Intelligence*, volume 6, pages 185–193, 1991.