# Mini-Buckets: A General Scheme for Bounded Inference

RINA DECHTER

*University of California, Irvine, Irvine, California*

AND

IRINA RISH

*IBM T. J. Watson Research Center, Hawthorne, New York*

Abstract. This article presents a class of approximation algorithms that extend the idea of bounded-complexity inference, inspired by successful constraint propagation algorithms, to probabilistic inference and combinatorial optimization. The idea is to bound the dimensionality of dependencies created by inference algorithms. This yields a parameterized scheme, called *mini-buckets*, that offers adjustable trade-off between accuracy and efficiency. The mini-bucket approach to optimization problems, such as finding the most probable explanation (MPE) in Bayesian networks, generates both an approximate solution and bounds on the solution quality. We present empirical results demonstrating successful performance of the proposed approximation scheme for the MPE task, both on randomly generated problems and on realistic domains such as medical diagnosis and probabilistic decoding.

Categories and Subject Descriptors: F.2.3 [**Analysis of Algorithms and Problem Complexity**]: Tradeoffs between Complexity Measures; G.1.6 [**Numerical Analysis**]: Optimization; I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Accuracy/complexity trade-off; approximation algorithms, Bayesian networks, combinatorial optimization, probabilistic inference.

---

The work of R. Dechter was partially supported by the National Science Foundation (NSF) grant IRI-9157636 and by Air Force Office of Scientific Research grant, AFOSR 900136, Rockwell International and Amada of America.

This work was supported in part by NSF grant IIS-0086529 and by MURI ONR award N00014-00-1-0617.

This work was done while I. Rish was completing her Ph.D. at the University of California, Irvine, Information and Computer Science Department.

Authors'addresses: R. Dechter, Department of Information and Computer Science, University of California, Irvine, CA, e-mail: dechter@ics.uci.edu; I. Rish, IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, e-mail: rish@us.ibm.com.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2003 ACM 0004-5411/03/0300-0107 $5.00

## 1. *Introduction*

Automated reasoning tasks such as constraint satisfaction and optimization, probabilistic inference, decision-making and planning are generally hard (NP-hard). One way to cope with this computational complexity is to identify tractable problem classes. Another way is to design algorithms that compute approximate rather than exact solutions.

Although approximation within given error bounds is also known to be NP-hard [Dagum and Luby 1993; Roth 1996], there are approximation strategies that work well in practice. One approach advocates *anytime algorithms*. These algorithms can be interrupted at any time, producing the best solution found thus far [Horvitz 1987, 1990, Dean and Boddy 1988; Boddy and Dean 1989]. Another approach is to identify classes of problems for which some solution quality can be guaranteed, thus applying the idea of tractability to approximation. Yet another approach is to use approximation algorithms that output a bound on their accuracy for each problem instance.

In this article, we present approximation algorithms that accommodate some of these properties. The class of *mini-bucket* approximation algorithms applies a *local inference* approach to probabilistic reasoning and combinatorial optimization using the *bucket-elimination* framework. Bucket elimination is a unifying algorithmic scheme that generalizes nonserial dynamic programming and variable elimination, to enable complex problem-solving and reasoning activities. Among the algorithms that can be expressed as bucket elimination are *directional-resolution* for propositional satisfiability [Dechter and Rish 1994; Rish and Dechter 2000], *adaptive-consistency* for constraint satisfaction [Dechter and Pearl 1987], *Fourier* and *Gaussian elimination* for linear inequalities [Lassez and Mahler 1992], *dynamic-programming* for combinatorial optimization [Bertele and Brioschi 1972], as well as many algorithms for probabilistic inference [Dechter 1999].

In all these areas, problems are represented by a set of functions or dependencies over subsets of variables (e.g., constraints, cost functions, or probabilities). The dependencies can be represented by an *interaction* graph. The algorithms infer and record new dependencies which amounts to adding new edges to the graph. Generally, representing a dependency among $r$ variables ($r$ is called the *arity* of a dependency) requires enumerating $O(\exp(r))$ tuples. As a result, the complexity of inference is time and space exponential in the arity (the number of arguments) of the largest dependency recorded by these algorithms, which is captured by a graph parameter known as *induced width*, or *treewidth* [Arnborg 1985; Dechter 1992].

In constraint networks, the computational complexity of inference can be bounded using constraint propagation or local consistency algorithms, such as $i$*-consistency* algorithms [Freuder 1982; Dechter 1992] that restrict the arity of recorded dependencies to $i$. Known special cases are *arc-consistency* ($i = 2$) and *path-consistency* ($i = 3$) [Mackworth 1977; Freuder 1978; Dechter 1992]. In general, *constraint-propagation* algorithms transform a given constraint problem into an equivalent but more explicit representation, by inferring new constraints that are added to the problem. Intuitively, an $i$-consistency algorithm will make any solution of any subproblem of size $i$ extensible (if possible) to some surrounding variables and constraints. These algorithms are interesting because they are polynomial, and yet they are often sufficient for discovering inconsistency. Indeed, the recent success of constraint-processing algorithms can be attributed primarily

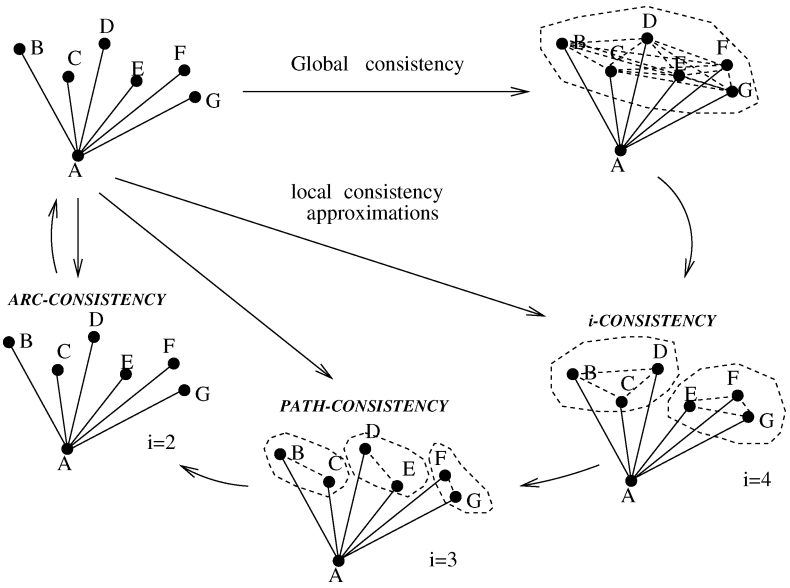



FIG. 1. From global to local consistency: Graph aspects of algorithm *i*-consistency and two particular cases, path-consistency (i = 3) and arc-consistency (i = 2).

to this class of algorithms, either used as stand-alone, incomplete algorithms, or incorporated within backtracking search [Dechter 1998; Dechter and Frost 1997]. The idea, visualized in Figure 1, shows that while exact algorithms may record arbitrarily large constraints (depicted by large cliques), *i*-consistency algorithms enforce consistency over smaller subproblems, recording constraints of size *i* or less. The *i*-consistency enforcing algorithm is an iterative procedure that goes over all subsets of *i* variables and filters out all those assignment that are not extensible; the procedure terminates when it converges to a fixed point, or when an inconsistent subproblem is discovered.

The mini-bucket approximation presented in the this article is inspired by the success of local consistency propagation, extending the idea to probabilistic reasoning and combinatorial optimization. This yields a parameterized scheme controlled by a bound on the size of functions that are recorded. Higher bounds result in more accurate solutions but require more computation. The mini-bucket algorithms generate both an approximate solution and a bound on the solution quality. We identify regions of completeness and present empirical results for the task of finding the *most probable explanation (MPE)* in Bayesian belief networks (see Section 2).

As we show, the proposed mini-bucket algorithms demonstrate good performance both on randomly generated problems and on realistic domains such as medical diagnosis and probabilistic decoding. One of the most important aspects of this scheme is that its parameterization yields an adaptive scheme that allows the user to tailor the algorithm to a particular problem domain and to the available time and space resources.

The article is organized as follows: Section 2 provides necessary definitions and preliminaries. The next three sections present and analyze the mini-bucket approximation for the probabilistic inference tasks of finding a *most probable explanation*

*(MPE), belief updating (BEL)* and finding a *most probable a posteriori hypothesis (MAP)*. Section 6 presents the mini-bucket algorithm for optimization problems. Section 7 identifies cases of completeness and Section 8 discusses extensions to anytime algorithms. In Section 9, empirical evaluation is carried out for the MPE task on randomly generated noisy-OR networks, on the CPCS networks for medical diagnosis [Pradhan et al. 1994], and on classes of probabilistic decoding problems. Section 10 discusses related work, while Section 11 provides concluding remarks and discusses future work.

## 2. *Background*

*Belief networks* provide a formalism for reasoning under uncertainty. A belief network is defined by a directed acyclic graph over nodes representing random variables of interest (e.g., the temperature of a device, the gender of a patient, a feature of an object, an event occurrence). The arcs signify the existence of direct causal influences between the linked variables, and the strength of these influences are quantified by conditional probabilities. A belief network relies on the notion of a directed graph.

*Definition* 1 (*Graphs*). A *directed graph* is a pair $G = (V, E)$, where $V = \{X_1, \ldots, X_n\}$ is a set of nodes and $E = \{(X_i, X_j) | X_i, X_j \in V, i \neq j\}$ is a set of edges. Two nodes $X_i$ and $X_j$ are called *neighbors* if there is an edge between them (either $(X_i, X_j)$ or $(X_j, X_i)$). We say that $X_i$ *points* to $X_j$ if $(X_i, X_j) \in E$; $X_i$ is called a *parent* of $X_j$, while $X_j$ is called a *child* of $X_i$. The set of parent nodes of $X_i$ is denoted $pa_{X_i}$, or $pa_i$, while the set of child nodes of $X_i$ is denoted $ch_{X_i}$, or $ch_i$. We call a node and its parents a *family*. A directed graph is *acyclic* if it has no directed cycles. In an *undirected graph*, the directions of the edges are ignored: $(X_i, X_j)$ and $(X_j, X_i)$ are identical. A directed graph is *singly-connected* (also known as a *polytree*), if its underlying undirected graph (called *skeleton graph*) has no (undirected) cycles. Otherwise, it is called *multiply-connected*.

*Definition* 2 (*Belief Networks*). Let $X = \{X_1, \ldots, X_n\}$ be a set of random variables, each having a finite number of possible *states*, or *values*. The set of values $D_i$ is the *domain* of $X_i$. A *belief network* is a pair $(G, \mathcal{P})$, where $G = (X, E)$ is a directed acyclic graph over nodes, denoting the variables, and $\mathcal{P} = \{P(X_i | pa_i) | i = 1, \ldots, n\}$ is the set of conditional probability tables (CPTs) defined for each variable $X_i$ and its parents $pa_i$ in $G$. A belief network represents a joint probability distribution over $X$ having the product form

$$P(\bar{x}) = P(x_1, \ldots, x_n) = \Pi_{i=1}^{n} P(x_i | \bar{x}_{pa_i}), \tag{1}$$

where $\bar{x} = (x_1, \ldots, x_n)$, an abbreviation for $\bar{x} = (X_1 = x_1, \ldots, X_n = x_n)$, denotes an assignment to all the variables from their respective domains, and $\bar{x}_{pa_i}$ denotes the assignment to the parents of $X_i$.

The following notation will be used in this article. We denote variables by uppercase letters, and use lower-case letters for the corresponding domain values. We use vector notation (e.g., $\bar{x}$) for assignments to subsets of variables. For example, $A = \bar{a}$ denotes value assignment $\bar{a}$ to a subset of variables $A$ from their respective domains. If $\bar{a}$ is a partial assignment over $A$ and $S$ is another subset of variables,

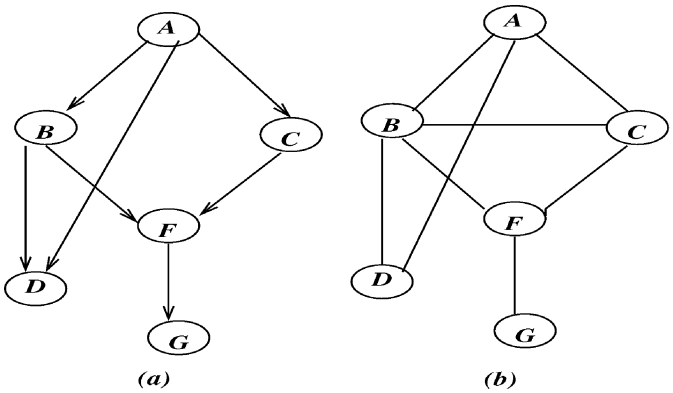



FIG. 2. (a) A belief network representing the joint probability distribution $P(g,\ f,\ d,\ c,\ b, a) = P(g|f)P(f|c, b)P(d|b, a)P(b|a)P(c|a)P(a)$, and (b) its moral graph.

then $\bar{a}_S$ denotes the projection of $\bar{a}$ over $S$, namely, the partial assignment from $\bar{a}$ restricted to variables in $S$. An *evidence* $\bar{e}$ is an assignment to a subset of *evidence variables*, that is, the variables that are observed.

The set of arguments of a function $f$ is called the *scope* of $f$. Thus, the scope of a CPT is its family. The *moral graph* $G^M$ of a belief network $(G,\ P)$ is obtained by connecting ("marrying") all the parents of each node and removing the directionality of edges. Thus, each CPT, associated with a family in a belief network, corresponds to a clique (complete subgraph) in the moral graph.

*Example* 1. Consider the belief network that represents the joint probability distribution

$$P(g,\ f, d, c, b, a) = P(g|f)P(f|c, b)P(d|b, a)P(b|a)P(c|a)P(a).$$

Its acyclic directed graph is shown in Figure 2(a), and the corresponding moral graph is shown in Figure 2(b). In this case, $pa(F) = \{B, C\}$, $pa(B) = \{A\}$, $pa(A) = \emptyset$, $ch(A) = \{B, D, C\}$.

2.1. PROBABILISTIC TASKS. The main benefit of having a joint-probability representation is that it allows answering a range of queries. The primary probabilistic reasoning queries are of two types: conditional probability queries, known as belief updating queries, and most probable assignment queries. *Belief updating* is the task of finding the posterior probability $P(Y|\bar{e})$ of *query* nodes $Y \subset X$ given evidence over a subset of variables $E$, $E = \bar{e}$. Applications that require answering such queries are numerous, including medical and fault diagnosis, genetic inheritance over family trees, reasoning about images and so on.

The *most probable assignment* queries require computing the most likely assignment to some subset of unobserved variables given observed evidence. Finding the Most Probable Explanation to *all* unobserved variables (the *MPE task*) is an important special case. Let $Y = X - E$, and let $\bar{y}$ be an assignment to $Y$. The MPE task is to find

$$MPE(\bar{y}^{mpe}|e) = \max_{\bar{y}} P(\bar{y}|e). \tag{2}$$

Note that there may be more than one maximizing assignment $\bar{y}^{mpe}$. The more general query, called *maximum a posteriori hypothesis (MAP)*, requires finding a maximum probability assignment to a *subset of hypothesis* variables, given the evidence. Namely, the set of variables $Y$ can be a strict subset of $X - E$.

Both tasks arise in a wide variety of applications, such as probabilistic error-correcting coding, speech recognition, medical diagnosis, airplane maintenance, monitoring and diagnosis in complex distributed computer systems, and so on. MPE queries are often used as ways of "completing" unknown information. For example, in probabilistic decoding, the task is to reconstruct a message (e.g., a vector of bits) sent through a noisy channel, given the channel output; in speech recognition and image understanding, the objective is to find a sequence of objects (letters, images) that is most likely to produce the observed sequence such as phonemes or pixel intensities; yet another example is diagnosis, where the task is to reconstruct the hidden state of nature (e.g., a set of possible diseases and unobserved symptoms the patient may have, or a set of failed nodes in a computer network) given observations of the test outcomes (e.g., symptoms, medical tests, or network transactions results).

The general MAP queries are more applicable, used in cases such as medical diagnosis, when we observe part of the symptoms, and can accommodate some of the tests, and still wish to find the most likely assignments to the diseases only, rather than to both diseases and all unobserved variables. Although the MAP query is more general, MPE is an important special case because it is computationally simpler and thus should be applied when appropriate. It often serves as a "surrogate" task for MAP due to computational reasons. Since all the above problems can be posed as MPE or MAP queries, finding efficient algorithms clearly has a great practical value.

All the above probabilistic tasks are known to be NP-hard [Cooper 1990][1]. However, there exists a polynomial propagation algorithm for singly connected networks [Pearl 1988]. The two main approaches to extending this algorithm to multiply connected networks are the *cycle-cutset* approach, also called *conditioning*, and the *tree-clustering* approach [Pearl 1988; Lauritzen and Spiegelhalter 1988; Shachter 1986], which is also closely related to *variable-elimination* techniques [D'Ambrosio 1994; Zhang and Poole 1996; Dechter 1996]. We present probabilistic approximation algorithms based on the variable-elimination scheme called *bucket elimination* [Dechter 1996, 1999].

2.2. THE BUCKET ELIMINATION SCHEME. This section provides a brief overview of bucket elimination algorithms for some probabilistic reasoning tasks. Given a belief network $(G, P)$ and a variable ordering $o = (X_1, \ldots, X_n)$, the belief $P(x_1|\bar{e})$ is defined as

$$P(x_1|\bar{e}) = \frac{P(x_1, \bar{e})}{P(\bar{e})} = \alpha P(x_1, \bar{e}) = \alpha \sum_{x_2} \cdots \sum_{x_n} \prod_i P(x_i|\bar{x}_{pa_i}), \qquad (3)$$

[1] In fact, recent results show that while the decision problems associated with belief updating and MPE are NP-complete, MAP is harder and is not in NP [Park 2002].

**Algorithm elim-bel**

**Input:** A belief network $BN = (G, \mathcal{P})$, ordering $o$, evidence $\bar{e}$.
**Output:** $P(x_1|\bar{e})$, the belief in every value $x_1$ of $X_1$ given evidence $\bar{e}$.
(1) **Initialize:** Partition $P = \{P_1, ..., P_n\}$ into buckets $bucket_1, \ldots, bucket_n$, where $bucket_p$ contains all CPTs $h_1, h_2, ..., h_t$ whose highest-index variable is $X_p$.
(2) **Backward:** for $p = n$ to 2 do
Let $h_1, h_2, ..., h_t$ be the CPTs in $bucket_p$ over scopes $S_1, ..., S_t$.
—**If** $X_p$ is observed ($X_p = a$), assign $X_p = a$ in each $h_j$
and put the result in its highest-variable bucket
—**Else** compute $h^p = \sum_{X_p} \Pi_{h_j \in bucket_p} h_j$
and place $h^p$ in its highest-variable bucket
(put constants in $bucket_1$).
(3) **Return** $Bel(x_1) = \alpha \Pi_{h_j \in bucket_1} h_j$,
where $\alpha$ is a normalizing constant over $X_1$'s values.

FIG. 3. Algorithm *elim-bel* for belief updating in belief networks.

where $\alpha$ is a normalizing constant. By the distributivity law,

$$\sum_{x_2} \cdots \sum_{x_n} \prod_i P(x_i|\bar{x}_{pa_i}) = F_1 \sum_{x_2} F_2 \ldots \sum_{x_n} F_n, \tag{4}$$

where each $F_i = \prod_x P(x|\bar{x}_{pa_i})$ is the product of all the probabilistic components (functions) defined on $X_i$ and *not* defined on any variable $X_j$ for $j > i$.

Algorithm *elim-bel* [Dechter 1996] shown in Figure 3 computes the sum in Eq. (4) from right to left, sequentially eliminating variables from $X_n$ to $X_1$. Initially, all functions (CPTs) $P(X|pa_i)$ participating in the product $F_i = \prod_x P(x|\bar{x}_{pa_i})$ are placed in the *bucket* of $X_i$ (denoted $bucket_i$). For each $X_i$, from $i = n$ to $i = 2$, the algorithm multiplies the functions in $bucket_i$, then sums over $X_i$, and places the resulting function in the bucket corresponding to the highest-index variable in the function's scope (clearly, this is one of the "lower" buckets). If $X_i$ is observed (e.g., $X_i = a$), then $X_i$ is assigned the value $a$ in each of the bucket's functions, and each resulting function is placed in its highest-variable bucket. This simplifies computation, and graphically corresponds to removing the evidence node from the moral graph. Note that constants (the results of eliminating a variable that is the only argument of a function) are placed into the first (lowest) bucket. Finally, the algorithm processes the lowest bucket, $bucket_1$. The algorithm returns the normalized product of functions in $bucket_1$ which yields the updated belief $P(X_1|\bar{e})$. Note that only multiplication (no summation over $X_1$) need to be performed in this bucket.

The following example illustrates *elim-bel* on the network in Figure 4(a).

*Example* 2. Given the belief network in Figure 4(a), the ordering $o = (A, E, D, C, B)$, and evidence $e = \{E = 0\}$, $Bel(a) = P(a|E = 0) = \alpha P(a, E = 0)$ is computed by bucket elimination as follows: First, all the CPT's are partitioned into the ordered buckets as shown in Figure 5. At first, only the CPT functions (shown in nonbold style in Figure 4(a)) are included. (Note that upper-case letters denote nodes, and lower-case letters denote their values). Carrying out the computation from right to left using the bucket data-structure, and denoting by $h^X$ the function computed in bucket $X$, we get new functions as demonstrated in Figure 5.

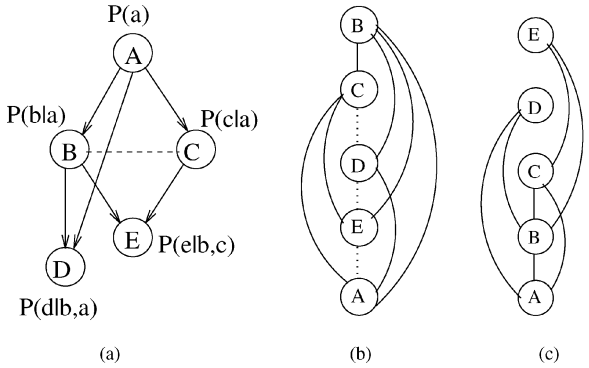



FIG. 4. (a) A belief network. The dotted arc $(B, C)$ is added by the moral graph. (b) The induced graph along $o = (A, E, D, C, B)$, and (c) the induced graph along $o = (A, B, C, D, E)$.

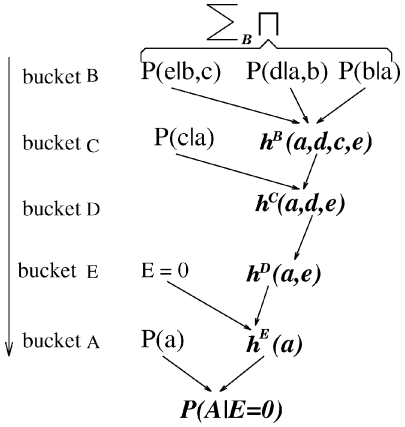



FIG. 5. An example of elim-bel's execution.

The computation performed in each bucket is given by:

(1) bucket B: $h^B(a, d, c, e) = \sum_b P(e|b, c)P(d|a, b)P(b|a)$
(2) bucket C: $h^C(a, d, e) = \sum_c P(c|a)h^B(a, d, c, e)$
(3) bucket D: $h^D(a, e) = \sum_d h^C(a, d, e)$
(4) bucket E: $h^E(a) = h^D(a, E = 0)$
(5) bucket A: $Bel(a) = P(a|E = 0) = \alpha P(a)h^E(a)$,

where $\alpha$ is a normalizing constant.

Similar bucket elimination algorithms were derived for the tasks of finding MPE, MAP, and for finding the maximum expected utility (MEU) [Dechter 1996, 1999]. Given a belief network $(G, P)$, a variable ordering $o = (X_1, \ldots, X_n)$, and an evidence $\bar{e}$, the MPE task is to find the most-probable assignment to the variables, namely, to find

$$\bar{x}^{MPE} = (x_1^{MPE}, \ldots, x_n^{MPE}) = \arg \max_{x_1,\ldots,x_n} P(x_1, \ldots, x_n|\bar{e}) \tag{5}$$

and the corresponding value $\max_{x_1,\ldots,x_n} P(x_1, \ldots, x_n|\bar{e})$. However, from algorithmic prospective, it is more convenient to maximize the joint probability $P(\bar{x}, \bar{e})$

**Algorithm elim-mpe**
**Input:** A belief network $BN = (G, P)$, ordering $o$, evidence $\bar{e}$.
**Output:** An MPE assignment $\bar{x}^{MPE}$ and its probability.

(1) **Initialize:** Partition $P = \{P_1, ..., P_n\}$ into buckets $bucket_1, \ldots, bucket_n$, where $bucket_p$ contains all CPTs $h_1, h_2, ..., h_t$ whose highest-index variable is $X_p$.
(2) **Backward:** for $p = n$ to 2 do
—**If** $X_p$ is observed ($X_p = a$), replace $X_p$ by $a$ in each $h_j$
and put the result in its highest-variable bucket.
—**Else** compute
$h^p = \max_{x_p} \Pi_{h_j \in bucket_p} h_j$ and $x_p^{mpe} = \arg\max_{x_p} \Pi_{h_j \in bucket_p} h_j$;
place $h^p$ in its highest-variable bucket and $x_p^{MPE}$ in $bucket_p$;
(put constants in $bucket_1$).
(3) **Forward: for** $p = 1$ to $n$,
given $(x_1^{mpe}, ..., x_{p-1}^{mpe})$, assign $x_p^{mpe} = \arg\max_{x_p} \Pi_{h_j \in bucket_p} h_j$.
(4) **Return** the assignment $\bar{x}^{mpe} = (x_1^{mpe}, ..., x_n^{mpe})$ and the value
$MPE = \max_{x_1} \Pi_{h_j \in bucket_1} h_j$.

FIG. 6. Algorithm *elim-mpe* for finding Most Probable Explanation in belief networks.

rather than the conditional probability $P(\bar{x}|\bar{e})$. Clearly, both probabilities achieve the maximum at the same point $\bar{x}^{MPE}$ since $P(\bar{x}, \bar{e}) = P(\bar{x}|\bar{e}) \cdot P(\bar{e})$, where $P(\bar{e})$ is independent of $X$. Thus, instead of computing $\max_{\bar{x}} P(\bar{x}|\bar{e})$, we compute $\max_{\bar{x}} P(\bar{x}, \bar{e})$ and call it the *MPE probability*, *MPE value*, or simply *MPE* (clearly, $P(\bar{x}|\bar{e}) = MPE/P(\bar{e})$; however, computing $P(\bar{e})$ may not be an easy task as it requires belief updating over a subset of evidence variables).

The MPE task can be solved by algorithm *elim-mpe* (see Figure 6), which is similar to *elim-bel* except that summation in *elim-bel* is replaced by maximization in *elim-mpe*. The main difference is that the "backward" phase is followed by a "forward" phase that computes an MPE assignment as follows: given an assignment to the first $i - 1$ variables, the assignment to the $i$th variable is computed by maximizing the product of all functions in the bucket of $X_i$. The algorithm returns the joint probability of the most-likely assignment, $MPE = \max_x P(x, e)$.

The bucket elimination scheme can be generalized using the notion of elimination operator applied to the functions in each bucket. Some elimination operators are defined below:

*Definition* 3 (*Elimination Functions*). Given a function $h$ defined over subset of variables $S$, where $X \in S$, the functions $(\min_X h)$, $(\max_X h)$, $(\sum_X h)$, and $(mean_X h)$ are defined over $U = S - \{X\}$ as follows: For every $\bar{u} \in U$, we define operators $(\min_X h)(\bar{u}) = \min_x h(\bar{u}, x)$, $(\max_X h)(\bar{u}) = \max_x h(\bar{u}, x)$, $(\sum_X h)(\bar{u}) = \sum_x h(\bar{u}, x)$, and $(mean_X h)(\bar{u}) = \sum_x \frac{h(\bar{u},x)}{|X|}$, where $(\bar{u}, x)$ is the extension of tuple $\bar{u}$ by assignment $X = x$, and where $|X|$ is the cardinality of $X$'s domain. Given a set of functions $h_1, \ldots, h_j$ defined over the subsets $S_1, \ldots, S_j$, the product function $(\Pi_j h_j)$ and $\sum_j h_j$ are defined over $U = \cup_j S_j$. For every $\bar{u} \in U$, $(\Pi_j h_j)(\bar{u}) = \Pi_j h_j(\bar{u}_{S_j})$, and $(\sum_j h_j)(\bar{u}) = \sum_j h_j(\bar{u}_{S_j})$.

An important property of bucket elimination algorithms is that their complexity can be predicted using a graph parameter called *induced width* [Dechter and Pearl 1987] (also known as *tree-width* [Arnborg 1985]), which describes the largest clique

created in the graph by bucket elimination, and which corresponds to the largest scope of function recorded by the algorithm.

*Definition* 4 (*Induced Width*). Given an undirected graph $G$, the *width* of $X_i$ along ordering $o$ is the number of $X_i$'s neighbors preceding $X_i$ in $o$. The *width of the graph* along $o$, denoted $w_o$, is the maximum width over all variables along $o$. The *induced graph* of $G$ along $o$ is obtained by recursively connecting the preceding neighbors of each $X_i$, going from $i = n$ to $i = 1$. The induced width along $o$, denoted $w_o^*$, is the width of the induced graph along $o$, while the induced width $w^*$ is the minimum induced width along any ordering.

*Example* 3. Figures 4(b) and 4(c) depict the induced graphs (induced edges are shown as dashed lines) of the moral graph in Figure 4(a) along the orderings $o = (A, E, D, C, B)$ and $o' = (A, B, C, D, E)$, respectively. Clearly, $w_o^* = 4$ and $w_{o'}^* = 2$.

It can be shown that

THEOREM 1 [DECHTER 1999]. *The time and space complexity of bucket elimination algorithms is $O(n \cdot d^{w_o^*+1})$, where $n$ is the number of variables, $d$ bounds the variables' domain size and $w_o^*$ is the induced width of the moral graph along ordering $o$, after all evidence nodes and their adjacent edges are removed.*

The induced width will vary depending on the variable ordering. Although finding a minimum-$w^*$ ordering is NP-hard [Arnborg 1985], heuristic algorithms are investigated [Bertele and Brioschi 1972; Dechter 1992; Kjaerulff 1990, 1992; Robertson and Seymour 1995; Bodlaender 1997; Bodlaender et al. 2001]. For more details on bucket elimination and induced width, see Dechter [1999].

## 3. *Mini-Bucket Approximation for MPE*

We introduce the idea of mini-bucket approximation using the combinatorial optimization task of finding the most probable explanation, MPE.

Since the MPE task is NP-hard and since complete algorithms (such as the *cycle cutset* technique, *join-tree-clustering* [Pearl 1988] and bucket elimination [Dechter 1996]) work well only on relatively sparse networks, approximation methods are necessary. Researchers investigated several approaches for finding MPE. The suitability of Stochastic Local Search (SLS) algorithms for MPE was studied in the context of medical diagnosis applications [Peng and Reggia 1989] and, more recently, in [Kask and Dechter 1999b]. Best-First search algorithms were proposed [Shimony and Charniack 1991] as well as algorithms based on linear programming [Santos 1991].

In this article, we propose approximation algorithms based on bucket elimination. Consider the bucket-elimination algorithm *elim-mpe*. Since the complexity of processing a bucket depends on the number of arguments (arity) of the functions being recorded, we propose to approximate these functions by a collection of smaller-arity functions. Let $h_1, \ldots, h_t$ be the functions in the bucket of $X_p$, and let $S_1, \ldots, S_t$ be their scopes. When *elim-mpe* processes bucket($X_p$), the function $h^p = max_{X_p} \Pi_{i=1}^t h_i$ is computed. A simple approximation idea is to compute an upper bound on $h^p$ by "migrating" the maximization inside the multiplication. Since, in general, for any two non-negative functions $Z(x)$ and $Y(x)$, $\max_x Z(x) \cdot Y(x) \leq \max_x Z(x) \cdot \max_x Y(x)$, this approximation will compute an upper bound on $h^p$.

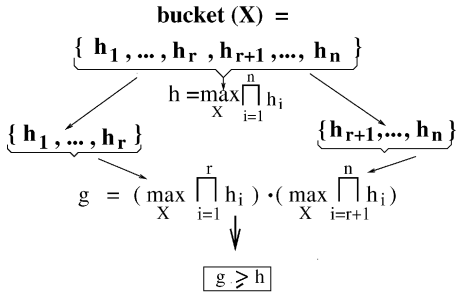



FIG. 7. The idea of mini-bucket approximation.

For example, we can compute a new function $g^p = \Pi_{i=1}^{t} \max_{X_p} h_i$, that is an upper bound on $h^p$. Procedurally, it means that maximization is applied separately to each function, requiring less computation.

The idea is demonstrated in Figure 7, where the bucket of variable $X$ having $n$ functions is split into two mini-buckets of size $r$ and $(n - r)$, $r \leq n$, and it can be generalized to any partitioning of a set of functions $h_1, \ldots, h_t$ into subsets called *mini-buckets*. Let $Q = \{Q_1, \ldots, Q_r\}$ be a partitioning into mini-buckets of the functions $h_1, \ldots, h_t$ in $X_p$'s bucket, where the mini-bucket $Q_l$ contains the functions $h_{l_1}, \ldots, h_{l_r}$. The complete algorithm *elim-mpe* computes $h^p = \max_{X_p} \Pi_{i=1}^{t} h_i$, which can be rewritten as $h^p = \max_{X_p} \Pi_{l=1}^{r} \Pi_{l_i} h_{l_i}$. By migrating maximization into each mini-bucket we can compute: $g_Q^p = \Pi_{l=1}^{r} \max_{X_p} \Pi_{l_i} h_{l_i}$. The new functions $\max_{X_p} \Pi_{l_i} h_{l_i}$ are placed separately into the bucket of the highest variable in their scope and the algorithm proceeds with the next variable. Functions without arguments (i.e., constants) are placed in the lowest bucket. The maximized product generated in the first bucket is an upper bound on the MPE probability. A lower bound can also be computed as the probability of a (suboptimal) assignment found in the forward step of the algorithm. Clearly, as the mini-buckets get smaller, both complexity and accuracy decrease.

*Definition* 5. Given two partitionings $Q'$ and $Q''$ over the same set of elements, $Q'$ is a refinement of $Q''$ if and only if, for every set $A \in Q'$, there exists a set $B \in Q''$ such that $A \subseteq B$.

PROPOSITION 1. *If $Q''$ is a refinement of $Q'$ in bucket$_p$, then $h^p \leq g_{Q'}^p \leq g_{Q''}^p$.*

PROOF. Based on the above discussion it is easy to see that for any partitioning $Q$ (be it $Q'$ or $Q''$) we have $h^p \leq g_Q^p$.

By definition, given a refinement $Q'' = \{Q_1'', \ldots, Q_k''\}$ of a partitioning $Q' = \{Q_1', \ldots, Q_m'\}$, each mini-bucket $i \in \{1, \ldots, k\}$ of $Q''$ belongs to some mini-bucket $j \in \{1, \ldots, m\}$ of $Q'$. In other words, each mini-bucket $j$ of $Q'$ is further partitioned into the corresponding mini-buckets of $Q''$, $Q_j' = \{Q_{j_1}'', \ldots, Q_{j_l}''\}$. Therefore,

$$g_{Q''}^p = \prod_{i=1}^{k} \left( \max_{X_p} \Pi_{l \in Q_i''} h_l \right) = \prod_{j=1}^{m} \prod_{Q_i'' \subseteq Q_j'} \left( \max_{X_p} \Pi_{l \in Q_i''} h_l \right)$$

$$\geq \prod_{j=1}^{m} \left( \max_{X_p} \Pi_{l \in Q_j'} h_l \right) = g_{Q'}^p. \qquad \square$$

**Algorithm mbe-mpe(i,m)**
**Input:** A belief network $BN = (G, P)$, an ordering $o$, evidence $\bar{e}$.
**Output:** An upper bound $U$ and a lower bound $L$ on the $MPE = \max_{\bar{x}} P(\bar{x}, \bar{e})$, and a suboptimal solution $\bar{x}^a$ that provides $L = P(\bar{x}^a)$.
1. **Initialize:** Partition $P = \{P_1, ..., P_n\}$ into buckets $bucket_1, \ldots, bucket_n$, where $bucket_p$ contains all CPTs $h_1, h_2, ..., h_t$ whose highest-index variable is $X_p$.
2. **Backward: for** $p = n$ to 2 **do**
   - **If** $X_p$ is observed ($X_p = a$), assign $X_p = a$ in each $h_j$ and put the result in its highest-variable bucket (put constants in $bucket_1$).
   - **Else** for $h_1, h_2, ..., h_t$ in $bucket_p$ do
     Generate an $(i, m)$-mini-bucket-partitioning, $Q' = \{Q_1, ..., Q_r\}$.
     **for each** $Q_l \in Q'$ containing $h_{l_1}, ...h_{l_t}$, **do**
       compute $h^l = max_{X_p} \Pi_{j=1}^{t} h_{l_j}$ and place it in the bucket of the highest-index variable in $U_l \leftarrow \bigcup_{j=1}^{t} S_{l_j} - \{X_p\}$, where $S_{l_j}$ is the scope of $h_{l_j}$ (put constants in $bucket_1$).
3. **Forward: for** $p = 1$ to $n$, given $x_1^a, ..., x_{p-1}^a$, **do**
   assign a value $x_p^a$ to $X_p$ that maximizes the product of all functions in $bucket_p$.
4. **Return** the assignment $\bar{x}^a = (x_1^a, ..., x_n^a)$, a lower bound $L = P(\bar{x}^a)$, and an upper bound $U = max_{x_1} \prod_{h_j \in bucket_1} h^j$ on the $MPE = \max_{\bar{x}} P(\bar{x}, \bar{e})$.

FIG. 8. Algorithm *mbe-mpe(i, m)*.

The mini-bucket elimination (*mbe*) algorithm for finding MPE, *mbe-mpe(i, m)*, is described in Figure 8. It has two input parameters that control the mini-bucket partitioning.

*Definition* 6 (*(i, m)-partitioning*). Let $H$ be a collection of functions $h_1, \ldots, h_t$ defined on scopes $S_1, \ldots, S_t$, respectively. We say that a function $f$ is *subsumed* by a function $h$ if any argument of $f$ is also an argument of $h$. A partitioning of $h_1, \ldots, h_t$ is *canonical* if any function $f$ subsumed by another function is placed into the bucket of one of those subsuming functions. A partitioning $Q$ into mini-buckets is an $(i, m)$-partitioning if and only if (1) it is canonical, (2) at most $m$ non-subsumed functions are included in each mini-bucket, (3) the total number of variables in a mini-bucket does not exceed $i$, and (4) the partitioning is *refinement-maximal*, namely, there is no other $(i, m)$-partitioning that it refines.

The parameters $i$ (number of variables) and $m$ (number of functions allowed per mini-bucket) are not independent, and some combinations of $i$ and $m$ do not allow an $(i, m)$-partitioning. However,

PROPOSITION 2. *If the bound $i$ on the number of variables in a mini-bucket is not smaller than the maximum family size, then, for any value of $m > 0$, there exists an $(i, m)$-partitioning of each bucket.*

PROOF. For $m = 1$, each mini-bucket contains one family. The arity of the recorded functions will only decrease and thus in each bucket an $(i, 1)$-partitioning always exists. Any $(i, m)$-partitioning that satisfies conditions 1–3 (but not necessarily condition 4), always includes all $(i, 1)$-partitionings satisfying conditions 1–3. Therefore, the set of $(i, m)$-partitionings satisfying conditions 1–3 is never empty, and there exists an $(i, m)$-partitioning satisfying conditions 1–4. □

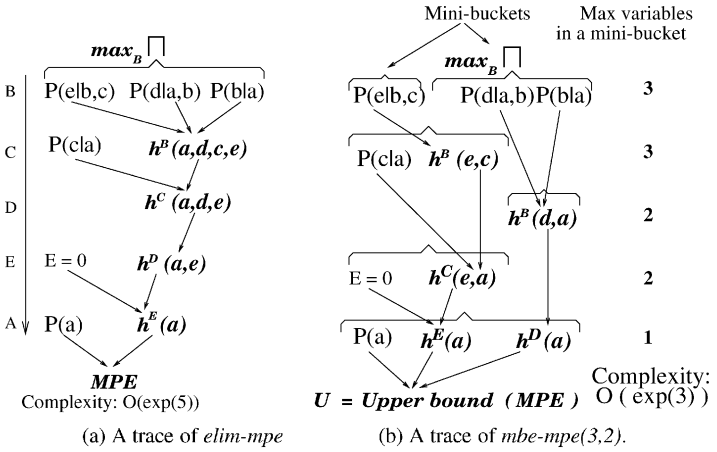



FIG. 9. Comparison between (a) *elim-mpe* and (b) *mbe-mpe*(3, 2).

Although the two parameters $i$ and $m$ are not independent, they do allow a flexible control of the mini-bucket scheme. The properties of the mini-bucket algorithms are summarized in the following theorem.

THEOREM 2. *Algorithm mbe-mpe($i$, $m$) computes an upper bound on the MPE. its time and space complexity is $O(n \cdot exp(i))$ where $i \leq n$.*

We will prove the theorem later (Section 7) in a more general setting, common to all mini-bucket elimination algorithms.

In general, as $m$ and $i$ increase, we get more accurate approximations. Note, however, a monotonic increase in accuracy as a function of $i$ can be guaranteed only for refinements of a given partitioning.

*Example* 4. Figure 9 compares algorithms *elim-mpe* and *mbe-mpe(i,m)* where $i = 3$ and $m = 2$ over the network in Figure 4(a) along the ordering $o = (A, E, D, C, B)$. The exact algorithm *elim-mpe* sequentially records the new functions (shown in boldface) $h^B(a, d, c, e)$, $h^C(a, d, e)$, $h^D(a, e)$, and $h^E(a)$. Then, in the bucket of $A$, it computes $M = \max_a P(a)h^E(a)$. Subsequently, an MPE assignment $(A = a', B = b', C = c', D = d', E = e')$ where $e' = 0$ is the evidence, is computed along $o$ by selecting a value that maximizes the product of functions in the corresponding buckets conditioned on the previously assigned values. 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.

On the other hand, since bucket($B$) includes five variables, *mbe-mpe*(3, 2) splits it into two mini-buckets $\{P(e|b, c)\}$ and $\{P(d|a, b), P(b|a)\}$, each containing no more than three variables, as shown in Figure 9(b) (the (3, 2)-partitioning is selected arbitrarily). The new functions $h^B(e, c)$ and $h^B(d, a)$ are generated in different mini-buckets and are placed independently in lower buckets. In each of the remaining lower buckets that still need to be processed, the number of variables is not larger than 3 and therefore no further partitioning occurs. An upper bound on the MPE value is computed by maximizing over A the product of functions in $A$'s bucket: $U = max_a P(a)h^E(a)h^D(a)$. Once all the buckets are processed, a suboptimal MPE tuple is computed by assigning a value to each variable that maximizes the product of functions in the corresponding bucket. By design, *mbe-mpe*(3, 2)

does not produce functions on more than two variables, while the exact algorithm *elim-mpe* records a function on four variables.

In summary, algorithm *mbe-mpe(i, m)* computes an interval $[L, U]$ containing the MPE value where $U$ is the upper bound computed by the backward phase and $L$ is the probability of the returned assignment.

Remember that *mbe-mpe* computes the bounds on $MPE = \max_{\bar{x}} P(\bar{x}, \bar{e})$, rather than on $M = \max_{\bar{x}} P(\bar{x}|\bar{e}) = MPE/P(\bar{e})$. Thus

$$\frac{L}{P(\bar{e})} \leq M \leq \frac{U}{P(\bar{e})}.$$

Clearly, the bounds $U$ and $L$ for *MPE* are very close to zero when the evidence $\bar{e}$ is unlikely: however, the ratio between the upper and the lower bound is not dependent on $P(\bar{e})$. As we will see next, approximating conditional probabilities using bounds on joint probabilities is more problematic for belief updating.

## 4. *Mini-Bucket Approximation for Belief Updating*

As shown in Section 2, the bucket elimination algorithm *elim-bel* for belief assessment is similar to *elim-mpe* except that maximization is replaced by summation and no value assignment is generated. Algorithm *elim-bel* finds $P(x_1, \bar{e})$ and then computes $P(x_1|\bar{e}) = \alpha P(x_1, \bar{e})$ where $\alpha$ is the normalization constant (see Figure 3).

The mini-bucket idea used for approximating MPE can be applied to belief updating in a similar way. Let $Q' = \{Q_1, \ldots, Q_r\}$ be a partitioning of the functions $h_1, \ldots h_t$ (defined over scopes $S_1, \ldots, S_t$, respectively) in $X_p$'s bucket. Algorithm *elim-bel* computes $h^p : U_p \rightarrow \Re$, where $h^p = \sum_{X_p} \Pi_{i=1}^t h_i$, and $U_p = \cup_i S_i - \{X_p\}$. Note that $h^p = \sum_{X_p} \Pi_{i=1}^t h_i$, can be rewritten as $h^p = \sum_{X_p} \Pi_{l=1}^r \Pi_{l_i} h_{l_i}$. If we follow the MPE approximation precisely and migrate the summation operator into each mini-bucket, we will compute $f_{Q'}^p = \Pi_{l=1}^r \sum_{X_p} \Pi_{l_i} h_{l_i}$. This, however, is an unnecessarily large upper bound of $h^p$ in which each $\Pi_{l_i} h_{l_i}$ is bounded by $\sum_{X_p} \Pi_{l_i} h_{l_i}$. Instead, we rewrite $h^p = \sum_{X_p} (\Pi_{1_i} h_{1_i}) \cdot (\Pi_{l=2}^r \Pi_{l_i} h_{l_i})$. Subsequently, instead of bounding a function of $X$ by its sum over $X$, we can bound ($i > 1$), by its maximum over $X$, which yields $g_{Q'}^p = (\sum_{X_p} \Pi_{1_i} h_{1_i}) \cdot (\Pi_{l=2}^r \max_{X_p} \Pi_{l_i} h_{l_i})$. In summary, an upper bound $g^p$ of $h^p$ can be obtained by processing one of $X_p$'s mini-buckets by summation and the rest by maximization. Clearly,

PROPOSITION 3. *For every partitioning $Q$, $h^p \leq g_Q^p \leq f_Q^p$. Also, if $Q''$ is a refinement partitioning of $Q'$, then $h^p \leq g_{Q'}^p \leq g_{Q''}^p$.*

A lower bound on the belief, or its mean value, can be obtained in a similar way. Algorithm *mbe-bel-max(i, m)* that uses the *max* elimination operator is described in Figure 10. Algorithms *mbe-bel-min* and *mbe-bel-mean* can be obtained by replacing the operator *max* by *min* and by *mean*, respectively.

4.1. NORMALIZATION. Note that *aprox-bel-max* computes an upper bound on $P(x_1, \bar{e})$ but not on $P(x_1|\bar{e})$. If an exact value of $P(\bar{e})$ is not available, deriving a bound on $P(x_1|\bar{e})$ from a bound on $P(x_1, \bar{e})$ is not easy, because $g(x_1)/\sum_{x_1} g(x_1)$, where $g(x)$ is the upper bound on $P(x_1, \bar{e})$, is not necessarily an upper bound

**Algorithm mbe-bel-max(i,m)**
**Input:** A belief network $BN = (G, P)$, an ordering $o$, and evidence $\bar{e}$.
**Output:** an upper bound on $P(x_1, \bar{e})$.
1. **Initialize:** Partition $P = \{P_1, ..., P_n\}$ into buckets $bucket_1, \ldots, bucket_n$, where $bucket_k$ contains all CPTs $h_1, h_2, ..., h_t$ whose highest-index variable is $X_k$.
2. **Backward:** for $k = n$ to 2 do
   - **If** $X_p$ is observed ($X_k = a$), assign $X_k \leftarrow a$ in each $h_j$ and put the result in the highest-variable bucket of its scope (put constants in $bucket_1$).
   - **Else** for $h_1, h_2, ..., h_t$ in $bucket_k$ do
     Generate an $(i, m)$-mini-bucket-partitioning, $Q' = \{Q_1, ..., Q_r\}$.
     **For each** $Q_l \in Q'$, containing $h_{l_1}, ...h_{l_t}$, do
       **If** $l = 1$ compute $h^l = \sum_{X_k} \Pi_{j=1}^t h_{1_j}$
       **Else** compute $h^l = max_{X_k} \Pi_{j=1}^t h_{l_j}$
       Add $h^l$ to the bucket of the highest-index variable in $U_l \leftarrow \bigcup_{j=1}^t S_{l_j} - \{X_k\}$, (put constant functions in $bucket_1$).
3. **Return** the product of functions in the bucket of $X_1$, which is an upper bound on $P(x_1, \bar{e})$ (denoted $g(x_1)$).

FIG. 10. Algorithm *mbe-bel-max(i, m)*.

on $P(x_1|\bar{e})$. In principle, we can derive a lower bound, $f$, on $P(\bar{e})$ using *mbe-bel-min* (in this case, the observed variables initiate the ordering), and then compute $g(x_1)/f$ as an upper bound on $P(x_1|\bar{e})$. This however is likely to make the bound quite weak due to compounded error. In many practical scenarios, however, we are interested in the ratio between the belief in two competing values of $X_1$. Since $P(x_i, e)/P(x_j, e) = P(x_i|e)/P(x_j|e)$, the ratio between the upper bounds of the respective join probabilities can serve as a good comparative measure between the conditional probabilities as well.

Alternatively, let $U_i$ and $L_i$ be the upper bound and lower bounding functions on $P(X_1 = x_i, \bar{e})$ obtained by *mbe-bel-max* and *mbe-bel-min*, respectively. Then,

$$\frac{L_i}{P(\bar{e})} \leq P(x_i|\bar{e}) \leq \frac{U_i}{P(\bar{e})}.$$

Therefore, although $P(\bar{e})$ is not known, the ratio of upper to lower bounds remains the same. Yet, the difference between the upper and the lower bounds can grow substantially, especially in cases of rare evidence. Note that if $P(\bar{e}) \leq U_i$, we get $L_i/P(\bar{e}) \leq P(X_1|\bar{e}) \leq 1$, so that the upper bound is trivial. Finally, note there is no bound for $g_{mean}(x_i)$, and therefore, the approximation of $g_{mean}(x_i)/\sum_{x_1} g_{mean}(x_1)$ can also be a lower or an upper bound of the exact belief. Interestingly, the computation of $g_{mean}(X_1 = x_i)/\sum_{x_1} g_{mean}(x_1)$ is achieved when processing all mini-buckets by summations, and subsequently normalizing.

## 5. *Mini-Bucket Elimination for MAP*

Algorithm *elim-map* for computing the MAP is a combination of *elim-mpe* and *elim-bel*; some of the variables are eliminated by summation, while the others by maximization. The MAP task is generally more difficult than MPE and belief updating [Park 2002]. From variable elimination perspective it restricts the

possible variable orderings and therefore may require higher $w^*$ which implies higher complexity.

Given a belief network, a subset of hypothesis variables $A = \{A_1, \ldots, A_k\}$, and evidence $\bar{e}$, the problem is to find an assignment to the hypothesized variables that maximizes their probability conditioned on $\bar{e}$. Formally, we wish to find

$$\bar{a}_k^{map} = \arg\max_{a_k} P(\bar{a}_k|\bar{e}) = \arg\max_{a_k} \frac{\sum_{\bar{x}_{k+1}^n} \Pi_{i=1}^n P(x_i, \bar{e}|\bar{x_{pa_i}})}{P(\bar{e})}. \tag{6}$$

where $\bar{x} = (a_1, \ldots, a_k, x_{k+1}, \ldots, x_n)$ denotes an assignment to all variables, while $\bar{a}_k = (a_1, \ldots, a_k)$ and $\bar{x}_{k+1}^n = (x_{k+1}, \ldots, x_n)$ denote assignments to the hypothesis and nonhypothesis variables, respectively. Since $P(\bar{e})$ is a normalization constant, the maximum of $P(\bar{a}_k|\bar{e})$ is achieved at the same point as the maximum of $P(\bar{a}_k, \bar{e})$. Namely, as before, we have $P(\bar{a}_k|\bar{e}) = P(\bar{a}_k, \bar{e})/P(\bar{e})$. Thus, we define $MAP = P(\bar{a}_k, \bar{e})$ and derive an approximation to this quantity which is easier than approximating $P(\bar{a}_k|\bar{e})$.

The bucket-elimination algorithm for finding the exact MAP, *elim-map* [Dechter 1996, 1999], assumes only orderings in which the hypothesized variables appear first and thus are processed last by the algorithm (this restricted ordering implies increased complexity as remarked above). The algorithm has a backward phase as usual but its forward phase is relative to the hypothesis variables only. The application of the mini-bucket scheme to *elim-map* for deriving an upper bound is a straightforward extension of the algorithms *mbe-mpe* and *mbe-bel-max*. We partition each bucket into mini-buckets as before. If the bucket's variable is eliminated by summation, we apply the rule we have in *mbe-bel-max* in which one mini-bucket is approximated by summation and the rest by maximization. When the algorithm reaches the hypothesis buckets, their processing is identical to that of *mbe-mpe*. Algorithm *mbe-map(i, m)* is described in Figure 11.

Deriving a lower bound on the MAP is no longer a simple extension of *mbe-map* as we observed for MPE. Once *mbe-map* terminates, we have an upper bound and we can compute an assignment to the hypothesis variables. While the probability of this assignment is a lower bound for the MAP, obtaining this probability is no longer possible by a simple forward step over the generated buckets. It requires an exact inference, or a lower bound approximation. We cannot use the functions generated by *mbe-bel-max* in the buckets of summation variables since those serve as upper bounds. One possibility is, once an assignment is obtained, to rerun the mini-bucket algorithm over the non-hypothesis variables using the min operator (as in *mbe-bel-min*, and then compute a lower bound on the assigned tuple in another forward step over the first $k$ buckets that take into account the original functions and only those computed by *mbe-bel-min*.

*Example* 5. We will next demonstrate the mini-bucket approximation for MAP on an example inspired by *probabilistic decoding* [MacKay and Neal 1996; Frey 1998].[2] Consider a belief network which describes the decoding of a *linear block code*, shown in Figure 12. In this network, $U_i$ are *information bits* and $X_j$ are *code bits*, which are functionally dependent on $U_i$. The vector $(U, X)$, called the channel input, is transmitted through a noisy channel which adds Gaussian

[2] Probabilistic decoding is discussed in more details in Section 9.5.

**Algorithm mbe-map(i,m)**
**Input:** A belief network $BN = (G, P)$, a subset of variables $A = \{A_1, ..., A_k\}$, an ordering of the variables, $o$, in which the $A$'s appear first, and evidence $\bar{e}$.
**Output:** An upper bound $U$ on the $MAP$ and a suboptimal solution $A = \bar{a}^a_k$.
1. **Initialize:** Partition $P = \{P_1, ..., P_n\}$ into buckets $bucket_1, \ldots, bucket_n$ where $bucket_P$ contains all CPTs, $h_1, ..., h_t$ whose highest index variable is $X_p$.
2. **Backward:** for $p = n$ to 2 do
- **If** $X_p$ is observed ($X_p = a$), assign $X_p = a$ in each $h_i$ and put the result in its highest-variable bucket (put constants in $bucket_1$).
- **Else** for $h_1, h_2, ..., h_j$ in $bucket_p$ do
  Generate an $(i, m)$-partitioning, $Q^{'}$ of the matrices $h_i$ into mini-buckets $Q_1, ..., Q_r$.
  - **If** $X_P \notin A$ /* not a hypothesis variable */
    **for each** $Q_l \in Q^{'}$, containing $h_{l_1}, ...h_{l_t}$, do
    **If** $l = 1$, compute $h^l = \sum_{X_p} \Pi^t_{i=1} h_{1_i}$
    **Else** compute $h^l = max_{X_p} \Pi^t_{i=1} h_{l_i}$
    Add $h^l$ to the bucket of the highest-index variable in $U_l \leftarrow \bigcup^t_{i=1} S_{l_i} - \{X_p\}$, (put constants in $bucket_1$).
  - **Else** ($X_p \in A$) /* a hypothesis variable */
    **for each** $Q_l \in Q^{'}$ containing $h_{l_1}, ...h_{l_t}$ compute $h^l = max_{X_p} \Pi^t_{i=1} h_{l_i}$ and place it in the bucket of the highest-index variable in $U_l \leftarrow \bigcup^t_{i=1} S_{l_i} - \{X_p\}$, (put constants in $bucket_1$).
3. **Forward: for** $p = 1$ to $k$, given $A_1 = a^a_1, ..., A_{p-1} = a^a_{p-1}$, assign a value $a^a_p$ to $A_p$ that maximizes the product of all functions in $bucket_p$.
4. **Return** An upper bound $U = max_{a_1} \prod_{h_i \in bucket_1} h_i$ on $MAP$, computed in the first bucket. and the assignment $\bar{a}^a_k = (a^a_1, ..., a^a_k)$.

FIG. 11. Algorithm *mbe-map(i, m)*.

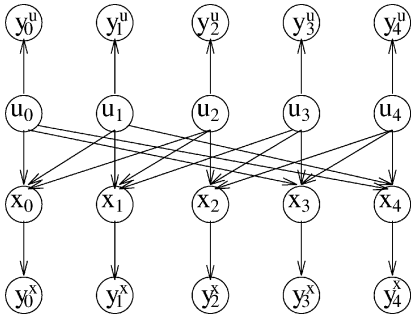



FIG. 12. Belief network for a linear block code.

noise and results in the channel output vector $Y = (Y^u, Y^x)$ . The decoding task is to assess the most likely values for the $U$'s given the observed values $Y = (\bar{y}^u, \bar{y}^x)$, which is the MAP task where $U$ is the set of hypothesis variables, and $Y = (\bar{y}^u, \bar{y}^x)$ is the After processing the observed buckets we get the following bucket configuration (lower case $y$'s are observed values):

*bucket*$(X_0) = P(y^x_0|X_0)$, $P(X_0|U_0, U_1, U_2)$,
*bucket*$(X_1) = P(y^x_1|X_1)$, $P(X_1|U_1, U_2, U_3)$,
*bucket*$(X_2) = P(y^x_2|X_2)$, $P(X_2|U_2, U_3, U_4)$,
*bucket*$(X_3) = P(y^x_3|X_3)$, $P(X_3|U_3, U_4, U_0)$,
*bucket*$(X_4) = P(y^x_4|X_4)$, $P(X_4|U_4, U_0, U_1)$,

$bucket(U_0) = P(U_0),\ P(y_0^u|U_0),$
$bucket(U_1) = P(U_1),\ P(y_1^u|U_1),$
$bucket(U_2) = P(U_2),\ P(y_2^u|U_2),$
$bucket(U_3) = P(U_3),\ P(y_3^u|U_3),$
$bucket(U_4) = P(U_4),\ P(y_4^u|U_4).$

Processing by *mbe-map*$(4, 1)$ of the first top five buckets by summation and the rest by maximization, results in the following mini-bucket partitionings and function generation:

$bucket(X_0) = \{P(y_0^x|X_0),\ P(X_0|U_0, U_1, U_2)\},$
$bucket(X_1) = \{P(y_1^x|X_1),\ P(X_1|U_1, U_2, U_3)\},$
$bucket(X_2) = \{P(y_2^x|X_2),\ P(X_2|U_2, U_3, U_4)\},$
$bucket(X_3) = \{P(y_3^x|X_3),\ P(X_3|U_3, U_4, U_0)\},$
$bucket(X_4) = \{P(y_4^x|X_4),\ P(X_4|U_4, U_0, U_1)\},$
$bucket(U_0) = \{P(U_0),\ P(y_0^u|U_0),\ h^{X_0}(U_0, U_1, U_2)\},\ \{h^{X_3}(U_3, U_4, U_0)\},$
$\{h^{X_4}(U_4, U_0, U_1)\},$
$bucket(U_1) = \{P(U_1),\ P(y_1^u|U_1),\ h^{X_1}(U_1, U_2, U_3),\ h^{U_0}(U_1, U_2)\},\ \{h^{U_0}(U_4, U_1)\},$
$bucket(U_2) = \{P(U_2),\ P(y_2^u|U_2),\ h^{X_2}(U_2, U_3, U_4),\ h^{U_1}(U_2, U_3)\},$
$bucket(U_3) = \{P(U_3),\ P(y_3^u|U_3),\ h^{U_0}(U_3, U_4),\ h^{U_1}(U_3, U_4),\ h^{U_2}(U_3, U_4)\},$
$bucket(U_4) = \{P(U_4),\ P(y_4^u|U_4),\ h^{U_1}(U_4),\ h^{U_3}(U_4)\}.$

The first five buckets are not partitioned at all and are processed as full buckets, since in this case a full bucket is a $(4, 1)$-partitioning. This processing generates five new functions, three are placed in bucket $U_0$, one in bucket $U_1$ and one in bucket $U_2$. Then, bucket $U_0$ is partitioned into three mini-buckets processed by maximization, creating two functions placed in bucket $U_1$ and one function placed in bucket $U_3$. Bucket $U_1$ is partitioned into two mini-buckets, generating functions placed in bucket $U_2$ and bucket $U_3$. Subsequent buckets are processed as full buckets. Note that the scope of recorded functions is bounded by 3.

In the bucket of $U_4$ we get an upper bound $U$ satisfying $U \geq MAP = P(U, \bar{y}^u, \bar{y}^x)$ where $\bar{y}^u$ and , $\bar{y}^x$ are the observed outputs for the $U$'s and the $X$'s bits transmitted. In order to bound $P(U|\bar{e})$, where $\bar{e} = (\bar{y}^u, \bar{y}^x)$, we need $P(\bar{e})$, which is not available. Yet, again, in most cases we are interested in the ratio $P(U = \bar{u_1}|\bar{e})/P(U = \bar{u_2}|\bar{e})$ for competing hypotheses $U = \bar{u_1}$ and $U = \bar{u_2}$ rather than in the absolute values. Since $P(U|\bar{e}) = P(U, \bar{e})/P(\bar{e})$ and the probability of the evidence is just a constant factor independent of $U$, the ratio is equal to $P(U_1, \bar{e})/P(U_2, \bar{e})$.

## 6. *Mini-Buckets for Discrete Optimization*

The mini-bucket principle can also be applied to deterministic discrete optimization problems which can be defined over *cost networks*, yielding approximation to dynamic programming for discrete optimization [Bertele and Brioschi 1972]. Cost networks is a general model encompassing constraint-satisfaction, and constraint-optimization in general. In fact, the MPE task is a special case of combinatorial optimization and its approximation via mini-buckets can be straightforwardly extended to the general case. For an explicit treatment, see [Dechter 1997b]. For

**Algorithm mbe-opt(i,m)**
**Input:** A cost network $(X, D, C)$, $C = \{C_1, ..., C_l\}$; ordering $o$, a set of assignments $e$.
**Output:** A lower and an upper bound on the optimal cost.
1. **Initialize:** Partition $C$ and $e$ into $bucket_1, \ldots, bucket_n$, where $bucket_p$ contains all components $h_1, h_2, ..., h_t$ whose highest-index variable is $X_p$.
2. **Backward:** for $p = n$ to 2 do
   - **If** $X_p$ is observed ($X_p = a$), replace $X_p$ by $a$ in each $h_i$ and put the result in its highest-variable bucket (put constants in $bucket_1$).
   - **Else** for $h_1, h_2, ..., h_t$ in $bucket_p$ do
     Generate an $(i, m)$-mini-bucket-partitioning, $Q' = \{Q_1, ..., Q_r\}$.
     **For each** $Q_l \in Q'$ containing $h_{l_1}, ...h_{l_t}$, compute $h^l = min_{X_p} \sum_{i=1}^{t} h_{l_i}$ and add it to the bucket of the highest-index variable in $U_l \leftarrow \bigcup_{i=1}^{t} S_{l_i} - \{X_p\}$, where $S_{l_i}$ is the set of arguments of $h_{l_i}$ (put constants in $bucket_1$).
3. **Forward: for** $p = 1$ to $n$, given $X_1 = x_1^{opt}, ..., X_{p-1} = x_{p-1}^{opt}$, assign a value $x_p^{opt}$ to $X_p$ that minimizes the sum of all functions in $bucket_p$.
4. **Return** the assignment $x^{opt} = (x_1^{opt}, ..., x_n^{opt})$, an upper bound $U = C(x^{opt})$, and a lower bound $L = min_{x_1} \sum_{h^i \in bucket_1} h^i$ on the optimal cost.

FIG. 13. Algorithm *mbe-opt(i, m)*.

completeness sake, we present the algorithm explicitly within the framework of cost networks.

A *cost network* is a triplet $(X, D, C)$, where $X$ is a set of discrete variables, $X = \{X_1, \ldots, X_n\}$, over domains $D = \{D_1, \ldots, D_n\}$, and $C$ is a set of real-valued cost functions $C_1, \ldots, C_l$, also called *cost components*. Each function $C_i$ is defined over a scope $S_i = \{X_{i_1}, \ldots, X_{i_r}\} \subseteq X$, $C_i : \times_{j=1}^{r} D_{ij} \rightarrow R^+$. The *cost graph* of a cost network has a node for each variable and edges connecting variables included in the same scope. The *cost function* is defined by $C(X) = \sum_{i=1}^{l} C_i$. The optimization (minimization) problem is to find an assignment $x^{opt} = (x_1{}^{opt}, \ldots, x_n{}^{opt})$ such that $C(x^{opt}) = \min_{x=(x_1,\ldots,x_n)} C(x)$.

Algorithm *mbe-opt* is described in Figure 13. Step 2 (backward step) computes a lower bound on the cost function while Step 3 (forward step) generates a suboptimal solution which provides an upper bound on the cost function.

## 7. *Complexity and Tractability*

7.1. THE CASE OF LOW INDUCED WIDTH. All mini-bucket algorithms have similar worst-case complexity bounds and completeness conditions. We denote by *mini-bucket-elimination(i, m)*, or simply *mbe(i, m)*, a generic mini-bucket scheme with parameters $i$ and $m$, without specifying the particular task it solves, which can be either one of probabilistic inference tasks defined above or a general discrete optimization problem. Theorem 2 applies to all mini-bucket algorithms:

THEOREM 3. *Algorithm mbe(i, m) takes $O(r \cdot d^i)$ time and space, where $r$ is the number of input functions,*[3] *$d$ is a bound on the domain size of the variables.*

[3] Note that $r = n$ for Bayesian networks, but can be higher or lower for general constraint optimization tasks.

*For $m = 1$, the algorithm is time and space $O(r \cdot d^{|F|})$, where $|F|$ is the maximum scope of any input function, $|F| \leq i \leq n$.*

PROOF. We can associate a bucket-elimination or a mini-bucket elimination algorithm with a *computation tree* where leaf nodes correspond to the original input functions (CPTs or cost functions), and each internal node $v$ corresponds to the result of applying an elimination operator (e.g., product followed by summation) to the set of node's children, $ch(v)$ (children correspond to all functions in the corresponding bucket or mini-bucket). We can compress the computation tree so that each node having a single child will be merged into one node with its parent, so that the branching degree in the resulting tree is not less than 2. Computing an internal node that is a compressed sequence of single-child nodes takes $O(d^i)$ time and space since it only requires a sequence of elimination operations over a single function whose scope size is bounded by $i$. Indeed, given a function over $i$ variables, an elimination of a single variable takes $O(d^i)$ time and space. Thus, elimination of $1 \leq k \leq i$ variables takes time and space $\sum_{j=i-k+1}^{i} O(d^j) = O(\sum_{j=i-k+1}^{i} d^j) = O(d^i)$. The cost of computing any other internal node $v$ is $O(|ch(v)| \cdot d^i)$ where $|ch(v)| \leq m$ and $i$ bounds the resulting scope size of generated functions. Since the number of leaf nodes is bounded by $r$, the number of internal nodes in the computation tree is bounded by $r$ as well (since the branching factor of each internal node is at least 2). Thus the total amount of computation over all internal nodes in the computation tree is time and space $O(r \cdot d^i)$ in general, which becomes to $O(n \cdot d^i)$ for belief networks. □

The above proof, suggested in [Larrosa 2000], refines the original proof given in [Dechter and Rish 1997].

We next identify cases for which the mini-bucket scheme coincides with the exact algorithm, and is therefore complete.

THEOREM 4. *Given an ordering of the variables, $o$, algorithm* mbe$(i, n)$ *applied along $o$ is complete for networks having $w_o^* \leq i$.*

PROOF. The claim trivially follows from the observation that each full bucket satisfies the condition of being an $(i, n)$-partitioning and it is the only one which is refinement-maximal. □

7.2. THE CASE OF MINI-BUCKET $(n, 1)$. Another case is *mbe*$(n, 1)$, which allows only one nonsubsumed function in a mini-bucket. It is easy to see that *mini-bucket*$(n, 1)$ is complete for polytrees if applied along some *legal* orderings. A *legal ordering* of a polytree (see Figure 14) is one where (1) all evidence nodes appear last in the ordering, and (2) among the remaining variables, each child node appears before its parents and all the parents of the same family are consecutive in the ordering. Such an ordering is always feasible for polytrees, and using this ordering, each bucket contains only one nonsubsumed function. Clearly, algorithm *mini-bucket*$(n, 1)$ is complete along such orderings and is therefore complete for polytrees.

In summary,

THEOREM 5. *Given a polytree, there exists an ordering $o$ such that algorithm mbe$(n, 1)$ finds an exact solution in time and space $O(n \cdot exp(|F|))$, where $|F|$ is the largest scope size of any input function.*

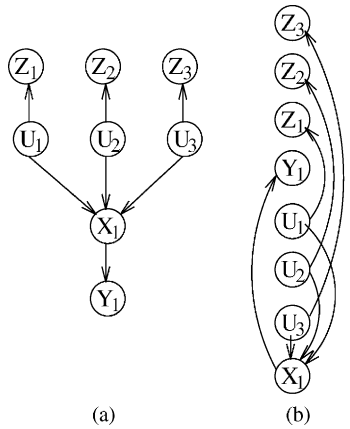



FIG. 14. (a) A polytree and (b) a legal ordering, assuming that nodes $Z_1$, $Z_2$, $Z_3$ and $Y_1$ are observed.

*Example* 6. Consider a legal ordering $o = (X_1, U_3, U_2, U_1, Y_1, Z_1, Z_2, Z_3)$ of the polytree in Figure 14(a), where the last four variables $Y_1,\ Z_1,\ Z_2,\ Z_3$ in the ordering are observed. Processing the buckets from last to first, after the last four buckets were already processed as observation buckets, we get (observed values shown in low-case):

*bucket*$(U_1) = P(U_1),\ P(X_1|U_1, U_2, U_3),\ P(z_1|U_1)$,
*bucket*$(U_2) = P(U_2),\ P(z_2|U_2)$,
*bucket*$(U_3) = P(U_3),\ P(z_3|U_3)$
*bucket*$(X_1) = P(y_1|X_1)$.

It is easy to see that the only legal partitionings correspond to full buckets.

Note also that on polytrees, *mbe*$(n, 1)$ is similar to Pearl's well-known propagation algorithm. One difference, however, is that Pearl's algorithm records only functions defined on a single variable, while mini-bucket$(n, 1)$ may record functions whose scope is at most the size of a family.

## 8. *Anytime Inference*

An important property of the mini-bucket scheme is that it provides an adjustable trade-off between accuracy of solution and computational complexity. Both the accuracy and the complexity increase with increasing parameters $i$ and $m$. While, in general, it may not be easy to predict the algorithm's performance for a particular parameter setting, it is possible to use this scheme within the *anytime* framework.

*Anytime algorithms* can be interrupted at any time producing the best solution found thus far [Horvitz 1987, 1990; Dean and Boddy 1988; Boddy and Dean 1989]. As more time is available, better solutions are guaranteed by such algorithms. In the context of Bayesian networks, anytime algorithms were first considered by the name of *flexible computation* under computational resource constraints [Horvitz 1987, 1988, 1990]. One of the first probabilistic anytime inference algorithms was *bounded conditioning* algorithm [Horvitz et al. 1989] that works by conditioning on a small, high probability cutset instances, including more of the instances as more computational resources become available.

**Algorithm anytime-mpe($\epsilon$)**
**Input:** Initial values of $i$ and $m$, $i_0$ and $m_0$; increments $i_{step}$ and $m_{step}$, and desired approximation error $\epsilon$.
**Output:** $U$ and $L$
1. **Initialize:** $i = i_0, m = m_0$.
2. **do**
3. run *mbe-mpe(i,m)*
4. $U \leftarrow$ upper bound of *mbe-mpe(i,m)*
5. $L \leftarrow$ lower bound of *mbe-mpe(i,m)*
6. Retain best bounds $U, L$, and best solution found so far
7. **if** $1 \leq U/L \leq 1 + \epsilon$, return solution
8. **else** increase $i$ and $m$: $i \leftarrow i + i_{step}$ and $m \leftarrow m + m_{step}$
9. **while** computational resources are available
10. **Return** the largest $L$
and the smallest $U$ found so far.

FIG. 15. Algorithm *anytime-mpe($\epsilon$)*.

In general, any inference algorithm that adapts to limited computational resources by ignoring some information about the problem, and is able to recover that information incrementally as more resources become available, can be called an anytime inference algorithm [Guo and Hsu 2002; Wellman and Liu 1994]. Many approximation schemes can be used by anytime methods since they are based on exploiting only partial information about the problem, for example, ignoring a subset of "weak" edges [Kjaerulff 1994; van Engelen 1997], using partial variable assignments [Poole 1996; Santos and Shimony 1998] (including partial cutset assignments [Horvitz et al. 1989]), using only a subset of nodes [Draper 1995], or a subset of (relatively high) probability entries in CPTs [Jensen and Andersen 1990]. In particular, our mini-bucket scheme exploits partial (bounded) dependencies among the variables. Clearly, an iterative application of such schemes with less restrictions on the amount of information they use, results in anytime inference algorithms that eventually become exact, if sufficient computational resources are available.

Our idea of extending the mini-bucket scheme to an anytime algorithm is to run a sequence of mini-bucket algorithms with increasing values of $i$ and $m$ until either a desired level of accuracy is obtained, or until the computational resources are exhausted. The anytime algorithm *anytime-mpe($\epsilon$)* for MPE is presented in Figure 15. The parameter $\epsilon$ is the desired accuracy level. The algorithm uses initial parameter settings, $i_0$ and $m_0$, and increments $i_{step}$ and $m_{step}$. Starting with $i = i_0$ and $m = m_0$, *mbe-mpe(i, m)* computes a suboptimal MPE solution and the corresponding lower bound $L$, and an upper bound ($U$) for increasing values of $i$ and $m$. The algorithm terminates when either $1 \leq U/L \leq 1 + \epsilon$, or when the computational resources are exhausted, returning the largest lower bound and the smallest upper bound found so far, as well as the current best suboptimal solution. Note that the algorithm is complete when $\epsilon = 0$.

Another anytime extension of the mini-bucket, is to embed it within a complete anytime heuristic search algorithm such as *branch-and-bound*. Since, the mini-bucket approximations computes bounds (upper or lower) of the exact quantities, these bounds can be used as heuristic functions to guide search algorithms as

well as for pruning the search space. In other words, rather than stopping with the first solution found, as it is done in the forward step of *mbe-mpe*, we can continue searching for better solutions, while using the mini-bucket functions to guide and prune the search. This approach was explored recently and demonstrated great promise both for probabilistic optimization tasks such as MPE as well as for constraint satisfaction problems [Kask et al. 2001].

## 9. *Empirical Evaluation*

9.1. METHODOLOGY. Our empirical study is focused on approximating the MPE. We investigate the impact of the parameters $i$ and $m$ on the performance of *mbe-mpe(i, m)* by varying one parameter at a time. *Mbe-mpe(m)* denotes the algorithm with an unrestricted $i$ and a varying $m$, while *mbe-mpe(i)* assumes an unrestricted $m$ and a varying $i$. Both algorithms use the following brute-force strategy for selecting a mini-bucket partitioning. First, a canonical partitioning is found, i.e. all subsumed functions are placed into mini-buckets of one of their subsuming functions. Then, for *mbe-mpe(m)*, each group of $m$ successive mini-buckets is combined into one mini-bucket. For *mbe-mpe(i)*, we merge the successive canonical mini-buckets into a new one until the total number of variables in that mini-bucket exceeds $i$. Then the process is repeated for the next group of canonical mini-buckets, and so on. Also, in our implementation, we use a slightly different interpretation of the parameter $i$. We allow $i < |F|$, where $|F|$ is maximum family size, and bound the number of variables in a mini-bucket by $\max\{i, |F|\}$ rather than by $i$.

The accuracy of an approximation is measured by the error ratios $MPE/L$ and $U/MPE$, where $U$ and $L$ are, respectively, the upper and the lower bound on *MPE* found by *mbe-mpe*, where *MPE* is the probability of the exact MPE solution found by *elim-mpe*. When computing the exact MPE assignment is infeasible, we report only the ratio $U/L$ (note that $U/L$ is an upper bound on the error ratios $MPE/L$ and $U/MPE$). The efficiency gain is represented by the *time ratio* $TR = T_e/T_a$, where $T_e$ is the running time for *elim-mpe* and $T_a$ is the running time for *mbe-mpe*. We also report the width, $w_o$, and the induced width, $w_o^*$, of the network's graph along the *min-degree*[4] ordering $o$ [Bertele and Brioschi 1972; El Fattah and Dechter 1995] used by the algorithms. When there is no confusion, we omit the explicit specification of the ordering and use notation $w$ and $w^*$ instead of $w_o$ and $w_o^*$, respectively. Remember that for $i > w^*$, *mbe-mpe(i)* coincides with *elim-mpe*. For diagnostic purposes, we also report the maximum number of mini-buckets created in a single bucket, *max mb* (we report averages where *max mb* is rounded to the nearest integer).

We present empirical results for randomly generated networks and for applications such as medical diagnosis (*CPCS* networks [Pradhan et al. 1994]) and probabilistic decoding [McEliece et al. 1997; Frey 1998; MacKay and Neal 1996; Frey and MacKay 1998]. Our objective is to assess the effectiveness of the mini-bucket algorithms for different problem classes and to understand how structural domain

[4] The min-degree ordering procedure works as follows: Given a moralized belief network with $n$ nodes, a node with minimum degree is assigned index $n$ (placed last in the ordering). Then the node's neighbors are connected, the node is deleted from the graph, and the procedure repeats, selecting the $n-1$th node, and so on.

TABLE I. AVERAGE PERFORMANCE OF *mbe-mpe* ON 200 INSTANCES OF BINARY-VALUED UNIFORM RANDOM NETWORKS

| **mbe-mpe($m$) for** $m = 1, 2, 3$ | | | | | |
|---|---|---|---|---|---|
| m | MPE/L | U/MPE | TR | $T_a$ | max mb |
| **30 nodes, 80 edges** ($w = 8$, $w^* = 11$) | | | | | |
| 1 | 43.2 | 46.2 | 296.1 | 0.1 | 4 |
| 2 | 4.0 | 3.3 | 25.0 | 2.2 | 2 |
| 3 | 1.3 | 1.1 | 1.4 | 26.4 | 1 |
| **60 nodes, 90 edges** ($w = 4$, $w^* = 11$) | | | | | |
| 1 | 9.9 | 21.7 | 131.5 | 0.1 | 3 |
| 2 | 1.8 | 2.8 | 27.9 | 0.6 | 2 |
| 3 | 1.0 | 1.1 | 1.3 | 11.9 | 1 |

(a) mbe-mpe($m$)

| **mbe-mpe($i$) for** $i = 5, 8, 11$ | | | | | |
|---|---|---|---|---|---|
| i | MPE/L | U/MPE | TR | $T_a$ | max mb |
| **30 nodes, 80 edges** ($w = 8$, $w^* = 11$) | | | | | |
| 5 | 29.2 | 20.7 | 254.6 | 0.1 | 3 |
| 8 | 17.3 | 7.5 | 151.0 | 0.2 | 3 |
| 11 | 5.0 | 3.0 | 45.3 | 0.6 | 2 |
| **60 nodes, 90 edges** ($w = 4$, $w^* = 11$) | | | | | |
| 5 | 2.8 | 6.1 | 112.8 | 0.1 | 2 |
| 8 | 1.9 | 2.8 | 71.7 | 0.2 | 2 |
| 11 | 1.4 | 1.6 | 24.2 | 0.5 | 2 |

(b) mbe-mpe($i$)

properties affect its performance. In the following four sections, we present the results for randomly generated networks (uniform random networks and noisy-OR networks), for CPCS networks, and for coding networks.

### 9.2. UNIFORM RANDOM PROBLEMS.

9.2.1. *Random Problem Generators.* Our uniform random problem generator takes as an input the number of nodes, $n$, the number of edges, $e$, and the number of values per variable, $v$. An acyclic directed graph is generated by randomly picking $e$ edges and subsequently removing possible directed cycles, parallel edges, and self-loops. Then, for each node $x_i$ and its parents $x_{pa_i}$, the conditional probability tables (CPTs) $P(x_i|x_{pa_i})$ are generated from the uniform distribution over [0, 1]. Namely, each $P(x_i|x_{pa_i})$ is replaced by $P(x_i|x_{pa_i})/\sum_{x_i} P(x_i|x_{pa_i})$.

9.2.2. *Results.* In Tables I and II, we present the results obtained when running *mbe-mpe(m)* and *mbe-mpe(i)* on 200 instances of uniform random networks having 30 nodes and 80 edges (referred to as "dense" networks), and on 200 instances of networks having 60 nodes and 90 edges (referred to as "sparse" networks). We computed the MPE solution and its approximations assuming no evidence nodes.

Tables I(a) and I(b) show the mean values of $MPE/L$, $U/MPE$, *TR*, $T_a$, *max mb*, $w$, and $w^*$, for $m = 1, 2, 3$, and for $i = 5, 8, 11$. Tables II(a) and II(b) present the approximation errors $MPE/L$ and $U/MPE$, showing the percentage of instances whose error ratio belongs to one of the intervals $[r, r + 1]$, where $r = 1, 2, 3$, or to the interval $[4, \infty]$. For each interval, we also show the mean time ratio *TR* computed on the corresponding instances. Table III presents the results on larger networks where the exact inference was often infeasible, so that we report only $U/L$.

The main observation is that the approximation algorithm solves many problem instances quite accurately ($MPE/L \in [1, 2]$ and $U/MPE \in [1, 2]$), spending 1–2 orders of magnitude less time than the complete algorithm. For instance, in Table I(a), *mbe-mpe*($m = 2$) on sparse instances solved all problems with mean $MPE/L$ ratio less than 2 and with speedup of almost 28. When controlled by $i$, the performance was even better. An accuracy $MPE/L < 2$ was achieved with speedup of almost 72 ($i = 8$) on sparse networks.

As expected, the average errors $MPE/L$ and $U/MPE$ decrease with increasing $m$ and $i$, approaching 1, while the time complexity of the approximation algorithms, $T_a$, increases, approaching the runtime of the exact algorithm, $T_e$. Table II

TABLE II. APPROXIMATION RESULTS FOR UNIFORM RANDOM NETWORKS WITH 30 NODES, 80 EDGES, AND WITH 60 NODES, 90 EDGES (200 INSTANCES PER EACH NETWORK CLASS)

**mbe-mpe($m$)**

| 30 nodes, 80 edges | | | | | |
|---|---|---|---|---|---|
| range | m | MPE/L % | Mean TR | U/MPE % | Mean TR |
| [1, 2] | 2 | 48 | 20.8 | 29.5 | 10.9 |
| (2, 3] | 2 | 16 | 25.7 | 27.5 | 22.2 |
| (3, 4] | 2 | 7.5 | 53.1 | 17 | 22.1 |
| (4, ∞) | 2 | 29.5 | 25.3 | 26 | 46.0 |
| [1, 2] | 3 | 92 | 1.4 | 97 | 1.4 |
| (2, 3] | 3 | 5 | 2.0 | 3 | 4.9 |
| (3, 4] | 3 | 1 | 1.2 | 1 | 1.3 |
| (4, ∞) | 3 | 3 | 1.6 | 0 | 0.0 |
| **60 nodes, 90 edges** | | | | | |
| range | m | MPE/L % | Mean TR | U/MPE % | Mean TR |
| [1, 2] | 1 | 26.5 | 172.8 | 0 | 0.0 |
| (2, 3] | 1 | 16 | 64.3 | 0 | 0.0 |
| (3, 4] | 1 | 9 | 43.5 | 1 | 17.4 |
| (4, ∞) | 1 | 48.5 | 147.5 | 99 | 132.7 |
| [1, 2] | 2 | 79.5 | 26.1 | 41 | 21.2 |
| (2, 3] | 2 | 10 | 28.0 | 31 | 32.6 |
| (3, 4] | 2 | 5.5 | 42.4 | 14 | 24.4 |
| (4, ∞) | 2 | 5 | 40.5 | 14 | 40.3 |
| [1, 2] | 3 | 100 | 1.3 | 100 | 1.3 |
| (2, 3] | 3 | 0 | 1.0 | 1 | 1.0 |
| (3, 4] | 3 | 0 | 0.0 | 0 | 0.0 |
| (4, ∞) | 3 | 0 | 0.0 | 0 | 0.0 |

(a) mbe-mpe($m$)

**mbe-mpe($i$)**

| 30 nodes, 80 edges | | | | | |
|---|---|---|---|---|---|
| range | i | MPE/L % | Mean TR | U/MPE % | Mean TR |
| [1, 2] | 8 | 31 | 150.1 | 2.5 | 33.0 |
| (2, 3] | 8 | 10 | 100.5 | 7 | 101.1 |
| (3, 4] | 8 | 10.5 | 114.7 | 12.5 | 132.9 |
| (4, ∞) | 8 | 48.5 | 169.8 | 78 | 162.1 |
| [1, 2] | 11 | 51 | 41.3 | 29 | 27.0 |
| (2, 3] | 11 | 15 | 41.3 | 32 | 50.5 |
| (3, 4] | 11 | 11 | 69.2 | 17 | 45.4 |
| (4, ∞) | 11 | 23 | 44.5 | 22 | 60.6 |
| **60 nodes, 90 edges** | | | | | |
| range | i | MPE/L % | Mean TR | U/MPE % | Mean TR |
| [1, 2] | 5 | 57.5 | 91.4 | 3 | 28.5 |
| (2, 3] | 5 | 15 | 158.3 | 15.5 | 71.0 |
| (3, 4] | 5 | 9 | 82.3 | 17.5 | 57.2 |
| (4, ∞) | 5 | 18.5 | 157.2 | 64 | 142.0 |
| [1, 2] | 8 | 80 | 64.9 | 38.5 | 36.9 |
| (2, 3] | 8 | 11.5 | 88.9 | 25 | 72.0 |
| (3, 4] | 8 | 3 | 27.4 | 21 | 96.3 |
| (4, ∞) | 8 | 5.5 | 158.4 | 15.5 | 124.5 |
| [1, 2] | 11 | 85.5 | 24.4 | 81 | 23.5 |
| (2, 3] | 11 | 11.5 | 29.7 | 13.5 | 29.1 |
| (3, 4] | 11 | 0.5 | 11.4 | 5 | 37.3 |
| (4, ∞) | 11 | 2.5 | 21.1 | 0.5 | 14.0 |

(b) mbe-mpe($i$)

TABLE III. RESULTS ON RANDOM NETWORKS OF TWO TYPES: (A) *mbe-mpe(m)* AND (B) *mbe-mpe(i)* ON 100 INSTANCES OF NETWORKS WITH 100 NODES AND 130 EDGES (WIDTH 4), AND (C) *mbe-mpe(i)* FOR $i = 2$ TO 20 ON 100 INSTANCES OF NETWORKS WITH 100 NODES AND 200 EDGES (WIDTH $w = 6$)

**mbe-mpe($m$)**

| m | U/L | $T_a$ | max mb |
|---|---|---|---|
| 1 | 781.1 | 0.1 | 3 |
| 2 | 10.4 | 3.4 | 2 |
| 3 | 1.2 | 132.5 | 1 |
| 4 | 1.0 | 209.6 | 1 |

(a) 100 nodes, 130 edges

**mbe-mpe($i$)**

| i | U/L | $T_a$ | max mb |
|---|---|---|---|
| 2 | 475.8 | 0.1 | 3 |
| 5 | 36.3 | 0.2 | 2 |
| 8 | 14.6 | 0.3 | 2 |
| 11 | 7.1 | 0.8 | 2 |
| 14 | 3.0 | 3.7 | 2 |
| 17 | 1.7 | 24.8 | 1 |

(b) 100 nodes, 130 edges

**mbe-mpe($i$)**

| i | U/L | $T_a$ | max mb |
|---|---|---|---|
| 2 | 1350427.6 | 0.2 | 4 |
| 5 | 234561.7 | 0.3 | 3 |
| 8 | 9054.4 | 0.5 | 3 |
| 11 | 2598.9 | 1.8 | 3 |
| 14 | 724.1 | 10.5 | 3 |
| 17 | 401.8 | 75.3 | 3 |
| 20 | 99.5 | 550.2 | 2 |

(c) 100 nodes, 200 edges

demonstrates how the distribution of error changes with increasing $m$ and $i$. Namely, a larger percentage of problems can be solved with a lower error.

On sparse random problems, we observe a substantial time speedup accompanied with low error. For example, *mbe-mpe*($i = 8$) and *mbe-mpe*($i = 11$) achieve a relatively small error ($MPE/L < 2$ and $U/MPE < 2$) in approximately 80% of cases, while being up to 2 orders of magnitude more efficient than *elim-mpe* (see the data for the 60-node networks in Table II(b)). Clearly, for dense random problems

the approximation quality is worse since they tend to have larger induced width. However, even for dense networks we observe that for $i = 11$ and for $m = 2$, approximately 50% of the instances were solved with an accuracy $MPE/L < 2$ accompanied with an order of magnitude speedup over the exact algorithm (see Tables II(a) and II(b) for 30-node networks).

We observe that the lower bound provided by the (suboptimal) solution tuple is closer to *MPE* than the upper bound. Namely, $MPE/L \in [1, 2]$ in a larger percent of cases than $U/MPE \in [1, 2]$ for both problem classes and for all $i$ and $m$ (see Table II).

The parameter $i$, bounding the number of variables in a mini-bucket, allows a better control over the approximation quality than $m$, the number of (nonsubsumed) functions in a mini-bucket. This results in a more accurate approximation scheme and in a larger time speedup. For example, as we can see from Table II, *mbe-mpe*($m = 2$) solved about 80% of the instances with accuracy $MPE/L \in [1, 2]$ and average speedup $TR = 26$, while *mbe-mpe*($i = 8$) solved 80% of the sparse instances in the same $MPE/L$ range with time speedup $TR = 65$.

The results in Tables III(a) and III(b) demonstrate further the advantages of more refined control allowed by the parameter $i$. For example, in Table III(a), $U/L \leq 10.4$ is achieved in 3.4 seconds by *mbe-mpe*($m = 2$), while $U/L \leq 7.1$ requires 0.8 seconds on the average by *mbe-mpe*($i = 11$) (Table III(b)). On denser problems having 100 nodes and 200 edges *mbe-mpe(m)* is inaccurate for $m = 1$ and $m = 2$, and already infeasible space-wise for $m = 3$ and $m = 4$. Therefore, we report only the results for *mbe-mpe(i)*, which allowed smaller error in a reasonable amount of time. However, its accuracy was still not acceptable ($U/L \approx 100$), even when the number of mini-buckets was bounded just by 2 (Table III(c)). Note, however, that $U/L$ may be a very loose upper bound on the actual accuracy.

9.3. RANDOM NOISY-OR PROBLEMS. The second set of experiments was performed on randomly generated *noisy-OR* networks.[5] A noisy-OR conditional probability table (CPT) is defined on binary-valued nodes as follows: given a child node $x$, and its parents $y_1, \ldots, y_n$, each $y_i$ is associated with a *noise parameter* $q_i = P(x = 0|y_i = 1, y_k = 0)$, $k \neq i$. The conditional probabilities are defined as follows [Pearl 1988]:

$$P(x|y_1, \ldots, y_n) = \begin{cases} \prod_{y_i=1} q_i & \text{if } x = 0, \\ 1 - \prod_{y_i=1} q_i & \text{if } x = 1. \end{cases} \tag{7}$$

Obviously, when all $q_i = 0$, we have a logical OR-gate. The parameter $1 - q_i = P(x = 1|y_i = 1, y_k = 0)$ for $k \neq i$ is also called *link probability*.

We generated random noisy-OR networks using the random graph generator described earlier, and randomly selecting the conditional probabilities for each CPT from the interval $[0, q]$, where $q$ was the bound on the noise parameter.

Table IV presents the results of evaluating algorithms *mbe-mpe(m)* and *mbe-mpe(i)*. We observe a good approximation quality ($MPE/L < 1.5$) accompanied by one or two orders of magnitude efficiency improvement for *mbe-mpe(i)*, using $i = 11$ and $i = 14$. In these cases the number of mini-buckets in a single bucket was at most two. The parameter $m$ was too coarse to allow accurate approximation

[5] Noisy-OR is an example of *causal independence* [Heckerman and Breese 1995], which implies that several causes (parent nodes) contribute independently to a common effect (child node).

TABLE IV. AVERAGE RESULTS FOR *elim-mpe* VERSUS *mbe-mpe(m)* AND *mbe-mpe(i)* ON 200 RANDOM NOISY-OR NETWORK INSTANCES WITH 30 NODES, 90 EDGES ($w = 9$, $w^* = 12$), ONE EVIDENCE NODE $X_1 = 1$, AND MAXIMUM NOISE LEVEL $q = 1$

| *mbe-mpe(m)* | | | | | | *mbe-mpe(i)* | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| m | MPE/L | U/MPE | TR | $T_a$ | max mb | i | MPE/L | U/MPE | TR | $T_a$ | max mb |
| 1 | 1285.8 | 59.0 | 441.2 | 0.0 | 4 | 5 | 48585.6 | 47.7 | 578.3 | 0.0 | 4 |
| 2 | 179.9 | 5.5 | 30.8 | 1.0 | 2 | 7 | 126.2 | 22.2 | 347.6 | 0.1 | 3 |
| 3 | 1.3 | 1.2 | 1.2 | 15.5 | 1 | 11 | 1.3 | 16.4 | 105.1 | 0.2 | 2 |
| 4 | 1.1 | 1.1 | 1.0 | 18.0 | 1 | 14 | 1.2 | 1.5 | 19.2 | 1.2 | 2 |

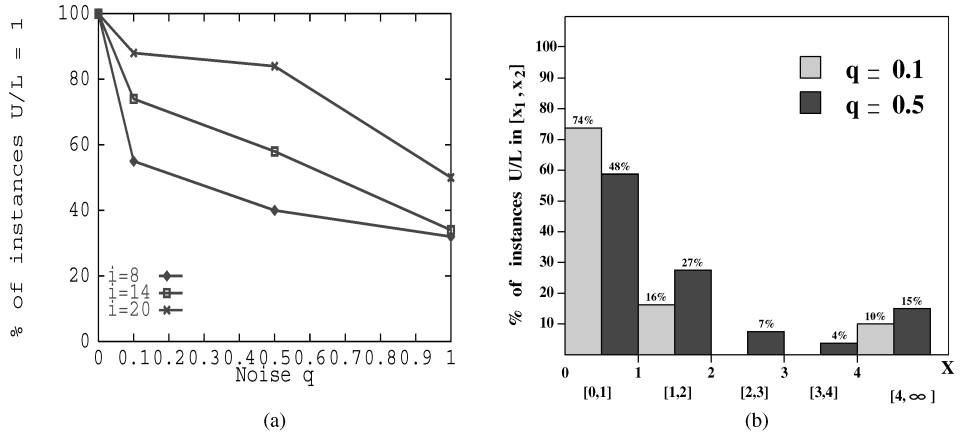



FIG. 16. Results on 200 random noisy-OR networks, each having 50 nodes, 150 edges, and 10 evidence nodes: (a) frequency of problems solved exactly (U/L=1) versus noise $q$ for different values of $i$; (b) a histogram of U/L for $q = 0.1$ and $q = 0.5$.

that also yields a good performance. For example, the case of m = 2 was fast but very inaccurate ($MPE/L \approx 180$), while for m = 3 the mini-bucket approximation often coincided with the exact algorithm since the average *max mb* (rounded to the nearest integer) equals 1.

Subsequently, we present results for *mbe-mpe(i)* on larger networks (Figure 16). Algorithm *elim-mpe* was intractable on these problems. The most apparent phenomenon here is that the approximation improves with decreasing noise $q$, that is, $U/L \to 1$ for $q \to 0$. In Figure 16(a), the percentage of instances for which $U/L = 1$ is plotted against $q$ for *mbe-mpe*(8), *mbe-mpe*(14), and *mbe-mpe*(20). When $q = 0$ (deterministic dependence $x \Leftrightarrow y_1 \vee \cdots \vee y_k$ between a child $x$ and its parents $y_i$, $i = 1, \ldots, k$), we observe almost 100% accuracy, which then decreases with increasing $q$ for all values of $i = 8, 14, 20$. One possible explanation is that, in the absence of noise, we get loosely connected constraint-satisfaction problems that can be easily solved by local constraint propagation techniques coinciding in this case with the mini-bucket scheme.

Notice that the behavior of *mbe-mpe(i)* is "extreme": it is either very accurate, or very inaccurate (Figure 16(b)). This phenomenon is more noticeable for small $q$.

9.4. CPCS NETWORKS. To evaluate the mini-bucket approach on realistic benchmarks, we used the CPCS networks derived from the Computer-based Patient Case Simulation system [Parker and Miller 1987; Pradhan et al. 1994]. CPCS network representation is based on INTERNIST-1 [Miller et al. 1982] and Quick

Medical Reference (QMR) [Miller et al. 1986] expert systems. The nodes of CPCS networks correspond to diseases and findings. In the original knowledge base, the probabilistic dependencies between the nodes are represented by *frequency weights* that specify the increase in the probability of a finding (child node) given a certain disease (parent node). This representation was later converted into a belief network using several simplifying assumptions: (1) conditional independence of findings given diseases, (2) noisy-OR dependencies between diseases and findings, and (3) marginal independence of diseases [Shwe et al. 1991].

In CPCS networks, the noisy-OR CPTs may also include *leak probabilities* not specified in Eq. 7. Namely, given a child node $x$ and its parents $y_1, \ldots, y_n$, the *leak probability* is defined as $leak = P(x = 1|y_1 = 0, \ldots, y_n = 0)$. The definition of a noisy-OR CPT is then modified as follows:

$$P(x = 0|y_1, \ldots, y_n) = (1 - leak) \prod_{y_i=1}^{n} q_i, \tag{8}$$

where $q_i$ are noise parameters defined earlier. Some CPCS networks include multivalued variables and *noisy-MAX* CPTs, which generalize noisy-OR by allowing $k$ values per node, as follows:

$$l_i = P(x = i\,|y_1 = 0, \ldots, y_n = 0), i = 1, \ldots k-1, \quad \text{and}$$

$$P(x = i\,|y_1, \ldots, y_n) = l_i \prod_{j=1}^{n} q_j^{y_j}, i = 0, \ldots k-2,$$

$$P(x = k-1|y_1, \ldots, y_n) = 1 - \sum_{i=0}^{i=k-2} l_i \prod_{j=1}^{n} q_j^{y_j}, \tag{9}$$

where $q_j^{y_j}$ is a noise coefficient for parent $j$ and the parent's value $y_j$. This definition coincides with the one given by Eq. (8) for $k = 2$, assuming $l_0 = 1 - leak$, $q_j^0 = 1$, and $q_j^1 = q_j$.

We experimented with both binary (noisy-OR) and non-binary (noisy-MAX) CPCS networks. The noisy-MAX network *cpcs179* (179 nodes, 305 edges) has 2 to 4 values per node, while the noisy-OR networks *cpcs360b* (360 nodes, 729 edges) and *cpcs422b* (422 nodes, 867 edges) have binary nodes (the letter 'b' in the network's name stands for "binary"). Since our implementation used standard conditional probability tables the non-binary versions of the larger CPCS networks with 360 and 422 nodes did not fit into memory. Each CPCS network was tested for different sets of evidence nodes.

9.4.1. *Experiments Without Evidence.* In Table V we present the results obtained on *cpcs179*, *cpcs360b*, and *cpcs422b* networks assuming no evidence nodes (i.e., there is only one network instance in each case) and using a min-degree elimination ordering, as before. Note that *mbe-mpe($i = w^* + 1$) is equivalent to exact elim-mpe*. Each row contains the usual parameters and measures as well as an additional parameter called the *effective* induced width, $w_a^*$, defined as the size of largest mini-bucket minus one. This parameter does not exceed $i - 1$ nor the largest family size.

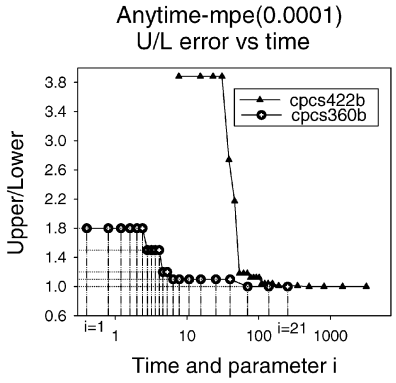
TABLE V. *mbe-mpe(i)* ON CPCS NETWORKS IN CASE OF NO EVIDENCE

| i | M | U | L | U/MPE | MPE/L | $T_a$ | $T_e$ | $T_e/T_a$ | max mb | $w_a^*$ |
|---|---|---|---|---|---|---|---|---|---|---|
| **cpcs179 network,** $w = 8$, $w^* = 8$ | | | | | | | | | | |
| Greedy value = 6.9e-3, MPE/greedy = 1.0, greedy time = 0.0 | | | | | | | | | | |
| 1 | 6.9e-3 | 9.1e-3 | 6.9e-3 | 1.3 | 1.0 | 0.3 | 1.0 | 3.3 | 4 | 8 |
| 4 | 6.9e-3 | 9.1e-3 | 6.9e-3 | 1.3 | 1.0 | 0.3 | 1.0 | 3.3 | 3 | 8 |
| 6 | 6.9e-3 | 6.9e-3 | 6.9e-3 | 1.0 | 1.0 | 0.4 | 1.0 | 2.5 | 2 | 8 |
| 8 | 6.9e-3 | 6.9e-3 | 6.9e-3 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1 | 8 |
| **cpcs360b network,** $w = 18$, $w^* = 20$ | | | | | | | | | | |
| Greedy value = 2.0e-7, MPE/greedy = 1.0, greedy time = 0.0 | | | | | | | | | | |
| 1 | 2.0e-7 | 3.7e-7 | 2.0e-7 | 1.8 | 1.0 | 0.4 | 115.8 | 289.5 | 8 | 11 |
| 6 | 2.0e-7 | 4.0e-7 | 2.0e-7 | 2.0 | 1.0 | 0.4 | 115.8 | 289.5 | 9 | 11 |
| 11 | 2.0e-7 | 2.4e-7 | 2.0e-7 | 1.2 | 1.0 | 0.5 | 115.8 | 231.6 | 6 | 11 |
| 16 | 2.0e-7 | 2.2e-7 | 2.0e-7 | 1.1 | 1.0 | 4.9 | 115.8 | 23.6 | 3 | 15 |
| 19 | 2.0e-7 | 2.0e-7 | 2.0e-7 | 1.0 | 1.0 | 30.4 | 115.8 | 3.8 | 3 | 18 |
| **cpcs422b network,** $w = 22$, $w^* = 23$ | | | | | | | | | | |
| Greedy value = 1.19e-4, MPE/greedy = 41.2, greedy time = 0.1 | | | | | | | | | | |
| 1 | 0.0049 | 0.1913 | 0.0049 | 3.88 | 1.00 | 7.7 | 1697.6 | 220.5 | 12 | 17 |
| 6 | 0.0049 | 0.0107 | 0.0049 | 2.17 | 1.00 | 7.7 | 1697.6 | 220.5 | 10 | 17 |
| 11 | 0.0049 | 0.0058 | 0.0049 | 1.17 | 1.00 | 7.8 | 1697.6 | 217.6 | 9 | 17 |
| 18 | 0.0049 | 0.0050 | 0.0049 | 1.01 | 1.00 | 34.6 | 1697.6 | 49.1 | 3 | 17 |
| 20 | 0.0049 | 0.0069 | 0.0049 | 1.40 | 1.00 | 98.8 | 1697.6 | 17.2 | 3 | 19 |
| 21 | 0.0049 | 0.0049 | 0.0049 | 1.00 | 1.00 | 156.1 | 1697.6 | 10.9 | 2 | 20 |

In addition, we compute a lower bound on *MPE* (called *Greedy* here) using a simple *greedy strategy* as follows. Before applying a mini-bucket algorithm we generate a tuple (forward step) using only the original functions in each bucket. Namely, for each variable $X_i$ along the given ordering we assign to $X_i$ a value $x' = \arg\max_x \prod_j f_j$, where $f_j$ is a function in $bucket_i$. The probability of the generated tuple is another lower bound on the *MPE*. For each network, we report the *Greedy* lower bound and the ratio *MPE*/*Greedy* in a separate row.

We observe that for the above three instances, the lower bound computed by *mbe-mpe*($i = 1$) already provides the probability of the exact MPE solution. For *cpcs179* and *cpcs360b*, even the greedy solution coincides with the MPE. The upper bound converges slowly and reaches the *MPE* probability at higher values of $i$, such as $i = 6$ for *cpcs179*, $i = 19$ for *cpcs360b*, and $i = 21$ for *cpcs422b*. Still, those values are smaller than $w^* + 1$, which is 9 for *cpcs179*, 21 for *cpcs360b*, and 24 for *cpcs422b*. Therefore, the exact solution is found before the approximate algorithm coincides with the exact one (we see that there are still two or three mini-buckets in a bucket). Note the nonmonotonic convergence of the upper bound. For example, on the *cpcs360b* network, the upper bound $U$ equals 3.7e-7 for $i = 1$, but then jumps to 4.0e-7 for $i = 6$. Similarly, on *cpcs422b* network, $U = 0.0050$ for $i = 18$, but $U = 0.0069$ for $i = 20$. We also observe that the greedy approximation for *cpcs422b* is an order of magnitude less accurate than the lower bound found by *mbe-mpe*($i = 1$) (see Table V), demonstrating that the mini-bucket scheme with $i = 1$ can accomplish a nontrivial task very efficiently.

The results in Table V are reorganized in Figure 17 from the perspective of algorithm *anytime-mpe(*$\epsilon$*)*. *Anytime-mpe(*$\epsilon$*)* runs *mbe-mpe(i)* for increasing $i$ until $U/L < 1 + \epsilon$. We started with $i = 1$ and were incrementing it by 1. We present the results for $\epsilon = 0.0001$ in Figure 17. The table compares the time of *anytime-mpe*(0.0001) and of *anytime-mpe*(0.1) against the time of the exact




| Algorithm | Time (sec) | |
|---|---|---|
| | *cpcs360* | *cpcs422* |
| **elim-mpe** | 115.8 | 1697.6 |
| **anytime-mpe**($\epsilon$), $\epsilon = 0.0001$ | 70.3 | 505.2 |
| **anytime-mpe**($\epsilon$), $\epsilon = 0.1$ | 70.3 | 110.5 |

FIG. 17. *anytime-mpe*(0.0001) on *cpcs360b* and *cpcs422b* networks for the case of no evidence.

algorithm. We see that the anytime approximation can be an order of magnitude faster.

9.4.2. *Likely and Random Evidence.* Subsequently, we experimented with *likely evidence* and *random evidence*. A random evidence set of size $k$ is generated by randomly selecting $k$ nodes and assigning value 1 to all of them. This approach usually produces a highly unlikely evidence set that results in low *MPE* probability. Alternatively, likely evidence is generated via *ancestral simulation* (*forward sampling* as follows: Starting with the root nodes and following an ancestral ordering where parents precede children, we simulate a value of each node in accordance with its conditional probability table. A given number of evidence nodes is then selected randomly. Ancestral simulation results in relatively high-probability tuples, which produce higher values of *MPE* than those for random evidence. As we demonstrate below, this has a dramatic impact on the quality of the approximation.

In Table VI, we show the results for *cpcs360b*. We generated 1000 instances of likely evidence and 1000 instances of random evidence, each of size 10. We first show the results for a single "typical" instance (one per each type of evidence), and then the average results over the complete sample set. We see that the probability of MPE solution decreases dramatically when switching from likely to random evidence. We observe that *mbe-mpe*($i = 1$) and the simple greedy approximation compute the exact MPE solution, while the upper bound converges to *MPE* only for larger values of $i$. The average $MPE/L$ ratio, however, is greater than 1 for $i \leq 5$ (e.g., $MPE/L = 2.1$ for $i = 1$), which tells us that the lower bound differs from *MPE* probability on some instances. The average $MPE/Greedy = 17.2$ is significantly larger. For random evidence the approximation error increases. The average lower bound is strictly less than the $M$ for $i < 17$, and the average $MPE/L = 25$ for $i = 1$. The results for *cpcs422b* (not shown here) were similar to those for *cpcs360b*. Note that $U/L$ is an order of magnitude lower for likely evidence than for random evidence (especially when $i < 8$), but is still about an order of magnitude higher than in the case of no evidence.

TABLE VI. *mbe-mpe(i)* ON *cpcs360b* (360 NODES, 729 EDGES, $w = 18$, $w^* = 20$)

| **ONE SAMPLE of each evidence type** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| i | M | U | L | U/L | U/MPE | MPE/L | $T_a$ | $T_e$ | $T_e/T_a$ | max mb | $w_a^*$ |
| LIKELY evidence: 10 nodes | | | | | | | | | | | |
| Greedy = 4.5e-10, MPE/greedy = 1.0, greedy time = 0.0 | | | | | | | | | | | |
| 1 | 4.5e-10 | 4.1e-9 | 4.5e-10 | 9.0 | 9.0 | 1.0 | 0.4 | 115.8 | 289.5 | 8 | 11 |
| 7 | 4.5e-10 | 3.3e-9 | 4.5e-10 | 7.3 | 7.3 | 1.0 | 0.4 | 115.8 | 289.5 | 9 | 11 |
| 11 | 4.5e-10 | 1.9e-9 | 4.5e-10 | 4.1 | 4.1 | 1.0 | 0.5 | 115.8 | 231.6 | 6 | 11 |
| 17 | 4.5e-10 | 4.7e-10 | 4.5e-10 | 1.1 | 1.1 | 1.0 | 9.8 | 115.8 | 11.9 | 3 | 16 |
| RANDOM evidence: 10 nodes | | | | | | | | | | | |
| Greedy = 1.1e-29, MPE/greedy = 7.0, greedy time = 0.0 | | | | | | | | | | | |
| 1 | 7.7e-29 | 8.9e-27 | 7.7e-29 | 115.4 | 115.4 | 1.0 | 0.4 | 116.0 | 290 | 7 | 11 |
| 7 | 7.7e-29 | 2.2e-26 | 7.7e-29 | 284.4 | 284.4 | 1.0 | 0.4 | 116.0 | 290 | 7 | 11 |
| 11 | 7.7e-29 | 4.0e-28 | 7.7e-29 | 5.2 | 5.2 | 1.0 | 0.5 | 116.0 | 232 | 5 | 11 |
| 17 | 7.7e-29 | 8.3e-29 | 7.7e-29 | 1.1 | 1.1 | 1.0 | 8.0 | 116.0 | 14.5 | 3 | 16 |
| **AVERAGES on 1000 instances** | | | | | | | | | | | |
| i | M | U | L | U/L | U/MPE | MPE/L | $T_a$ | $T_e$ | $T_e/T_a$ | max mb | $w_a^*$ |
| LIKELY evidence: 10 nodes | | | | | | | | | | | |
| Greedy = 1.18e-7, MPE/Greedy = 17.2, greedy time = 0.0 | | | | | | | | | | | |
| 1 | 1.2e-7 | 2.4e-7 | 1.2e-7 | 82 | 11 | 2.1 | 0.40 | 41.44 | 104.16 | 7.98 | 11 |
| 7 | 1.2e-7 | 2.1e-7 | 1.2e-7 | 8.6 | 8.2 | 1.0 | 0.40 | 41.44 | 103.53 | 8.94 | 11 |
| 11 | 1.2e-7 | 1.5e-7 | 1.2e-7 | 3.5 | 3.3 | 1.0 | 0.51 | 41.44 | 80.93 | 5.86 | 11 |
| 17 | 1.2e-7 | 1.3e-7 | 1.2e-7 | 1.3 | 1.3 | 1.0 | 9.59 | 41.44 | 4.35 | 3.04 | 16 |
| RANDOM evidence: 10 nodes | | | | | | | | | | | |
| Greedy = 5.01e-21, MPE/Greedy = 2620, greedy time = 0.0 | | | | | | | | | | | |
| 1 | 6.1e-21 | 2.3e-17 | 2.4e-21 | 2.5e+6 | 2.8e+5 | 25 | 0.40 | 40.96 | 102.88 | 7.95 | 11 |
| 7 | 6.1e-21 | 7.3e-17 | 5.9e-21 | 1.3e+5 | 1.2e+5 | 1.5 | 0.40 | 40.96 | 102.41 | 8.89 | 11 |
| 11 | 6.1e-21 | 2.4e-18 | 6.0e-21 | 2.4e+4 | 2.1e+3 | 3.1 | 0.51 | 40.96 | 80.02 | 5.81 | 11 |
| 17 | 6.1e-21 | 1.8e-20 | 6.1e-21 | 15 | 15 | 1.0 | 9.53 | 40.96 | 4.31 | 3.03 | 16 |

Since the variance of $U/L$ is high, we also present histograms of $log(U/L)$ in Figures 18 and 19, which summarize and highlight our main observations. The accuracy increases for larger values of $i$ (histograms in Figure 18 shift to the left with increasing $i$). As before, we see a dramatic increase in accuracy in case of likely evidence (Figure 18), and observe that the lower bound is often much closer to $M$ than the upper bound: $MPE/L$ is closer to 1 than $U/L$ (see Figure 19).

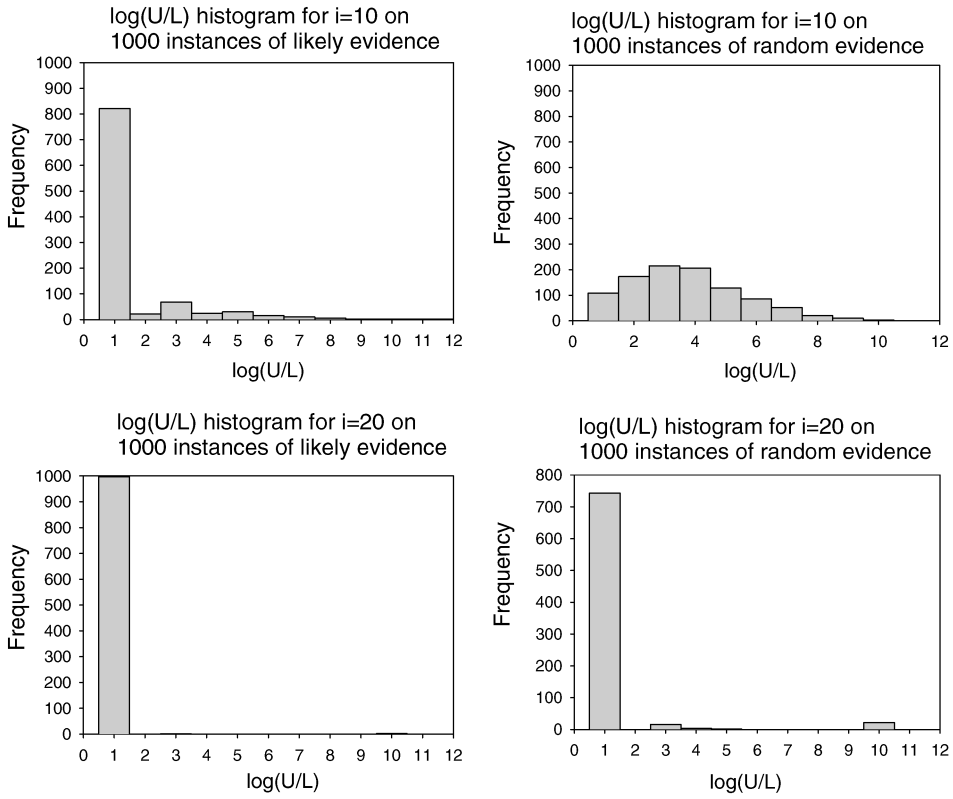
In summary, on CPCS networks

(1) *mbe-mpe(i)* computed accurate solutions for relatively small $i$. At the same time, the algorithm was sometimes orders of magnitude faster than *elim-mpe*.

(2) As expected, both the upper and the lower bounds converge to the exact *MPE* as $i$ increases. The lower bound is much closer to $M$, meaning that *mbe-mpe(i)* can find a good (suboptimal) MPE assignment before it is confirmed by the upper bound. In other words, we can find a good solution much faster than we can find a tight bound on the quality of the approximation.

(3) The preprocessing done by *mbe-mpe(i)* is necessary. A simple greedy assignment often provided a much less accurate lower bound.

(4) The approximation is significantly more accurate for likely than for unlikely evidence.

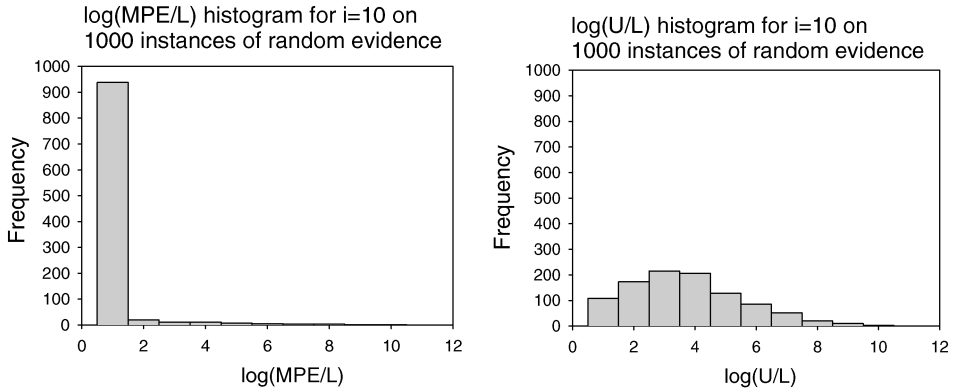



(a) Likely evidence (b) Random evidence

FIG. 18. Histograms of $U/L$ for $i = 10, 20$ on the *cpcs360b* network with 1000 sets of likely and random evidence, each of size 10.

9.5. PROBABILISTIC DECODING. In this section, we evaluate the quality of the mini-bucket algorithm *mbe-mpe* for the task of probabilistic decoding. We compare it to the exact elimination algorithms (*elim-mpe*, *elim-map* and *elim-bel*) and to the state-of-the-art approximate decoding algorithm, *iterative belief propagation* (*IBP*), on several classes of *linear block codes*, such as *Hamming codes*, *randomly generated block codes*, and *structured low-induced-width block codes*.

9.5.1. *Channel Coding.* The purpose of *channel coding* is to provide reliable communication through a noisy channel. Transmission errors can be reduced by adding redundancy to the information source. For example, a *systematic error-correcting code* [McEliece et al. 1997] maps a vector of $K$ *information bits* $u = (u_1, \ldots, u_K)$, $u_i \in \{0, 1\}$, into an $N$-bit *codeword* $c = (u, x)$, adding $N - K$ *code bits* $x = (x_1, \ldots, x_{N-K})$, $x_j \in \{0, 1\}$. The *code rate* $R = K/N$ is the fraction of the information bits relative to the total number of transmitted bits. A broad class of systematic codes includes linear block codes. Given a binary-valued *generator matrix* $G$, an *(N, K) linear block code* is defined so that the codeword $c = (u, x)$ satisfies $c = uG$, assuming summation modulo 2. The bits $x_i$ are also called the




(a) MPE/L (b) U/L

FIG. 19. Histograms of $U/L$ and $MPE/L$ for $i = 10$ and RANDOM evidence on the *cpcs360b* network. Each histogram is obtained on 1000 randomly generated evidence sets, each of size 10.

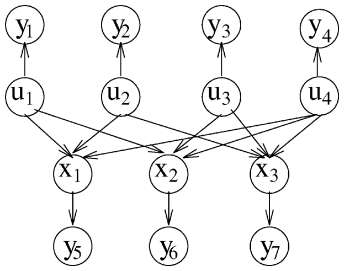



FIG. 20. Belief network for a (7, 4) Hamming code.

*parity-check* bits. For example, the generator matrix

$$G = \begin{matrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

defines a (7, 4) *Hamming code*.

The codeword $c = (u, x)$, also called the *channel input*, is transmitted through a noisy channel. A commonly used Additive White Gaussian Noise (AWGN) channel model assumes that independent Gaussian noise with variance $\sigma^2$ is added to each transmitted bit, producing a *real-valued channel output* $y$. Given $y$, the decoding task is to restore the input information vector $u$ [Frey 1998; McEliece et al. 1997; MacKay and Neal 1996].

It was observed that the decoding problem can be formulated as a probabilistic inference task over a belief network [McEliece et al. 1997]. For example, a (7, 4) Hamming code can be represented by the belief network in Figure 20, where the bits of $u$, $x$, and $y$ vectors correspond to the nodes, the parent set for each node $x_i$ is defined by non-zero entries in the $(K + i)$th column of $G$, and the (deterministic) conditional probability function $P(x_i|pa_i)$ equals 1 if $x_i = u_{j_1} \oplus \cdots \oplus u_{j_p}$ and 0 otherwise, where $\oplus$ is the summation modulo 2 (also, XOR, or parity-check

operation). Each output bit $y_j$ has exactly one parent, the corresponding channel input bit. The conditional density function $P(y_j|c_j)$ is a Gaussian (normal) distribution $N(c_j;\sigma)$, where the mean equals the value of the transmitted bit, and $\sigma^2$ is the noise variance.

The probabilistic decoding task can be formulated in two ways. Given the observed output $y$, the task of *bit-wise* probabilistic decoding is to find the most probable value of each *information bit*, namely:

$$u_k^* = \arg \max_{u_k \in \{0,1\}} P(u_k|y), \quad \textit{for } 1 \le k \le K.$$

The *block-wise* decoding task is to find a maximum a posteriori (maximum-likelihood) *information sequence*

$$u' = \arg \max_u P(u|y).$$

Block-wise decoding is sometimes formulated as finding a most probable explanation (MPE) assignment $(u', x')$ to the codeword bits, namely, finding

$$(u', x') = \arg \max_{(u,x)} P(u, x|y).$$

Accordingly, bit-wise decoding, which requests the posterior probabilities for each information bit, can be solved by belief updating algorithms, while the block-wise decoding translates to the MAP or MPE tasks, respectively.

In the coding community, decoding error is measured by the *bit error rate (BER)*, defined as the average percentage of incorrectly decoded bits over multiple transmitted words (blocks). It was proven by Shannon [1948] that, given the noise variance $\sigma^2$, and a fixed code rate $R = K/N$, there is a theoretical limit (called *Shannon's limit*) on the smallest achievable BER, no matter which code is used. Unfortunately, Shannon's proof is nonconstructive, leaving open the problem of finding an optimal code that achieves this limit. In addition, it is known that low-error (i.e., high-performance) codes tend to be long [Gallager 1965], and thus intractable for exact (optimal) decoding algorithms [McEliece et al. 1997]. Therefore, finding low-error codes is not enough; good codes must be also accompanied by efficient *approximate* decoding algorithms.

Recently, several high-performance coding schemes have been proposed (*turbo codes* [Berrou et al. 1993], *low-density generator matrix codes* [Cheng 1997], *low-density parity-check* codes [MacKay and Neal 1996]), that outperform by far the best up-to-date existing codes and get quite close to Shannon's limit. This is considered "the most exciting and potentially important development in coding theory in many years" [McEliece et al. 1997]. Surprisingly, it was observed that the decoding algorithm employed by those codes is equivalent to an iterative application of Pearl's *belief propagation* algorithm [Pearl 1988] that is designed for polytrees and, therefore, performs only local computations. This successful performance of iterative belief propagation (IBP) on multiply-connected coding networks suggests that approximations by local computations may be suitable for this domain. In the following section, we discuss iterative belief propagation in more detail.

9.5.2. *Iterative Belief Propagation.* Iterative belief propagation (IBP) computes an approximate belief for each variable in the network. It applies Pearl's belief propagation algorithm [Pearl 1988], developed for singly-connected networks, to multiply-connected networks, as if there are no cycles. The algorithm works by

**Iterative belief propagation (IBP)**
**Input:** A Belief Network $BN = \{P_1, ..., P_n\}$, evidence $e$, an activation schedule $A$, the number of iterations $I$.
**Output:** $Bel(X_i) = P(X_i|e)$ for all $i = 1, ..., n$.
1. **Initialize** $\lambda$ and $\pi$.
For evidence node $x_i = j$, set $\lambda_{x_i}(k)$ to 1 for $j = k$ and to 0 for $j \neq k$. For node $x$ having no parents set $\pi_x$ to prior $P(x)$. Otherwise, set $\lambda_x$ and $\pi_x$ to $(1, ..., 1)$.
2. **For iterations** = 1 to $I$:
For each node $x$ along $A$, do:
• For each $x = j$, compute
$\lambda_{x,u_i}(j) = \alpha \sum_j \lambda_x(j) \sum_{u_l : l \neq i} P(x = j \mid u_1, ..., u_m) \prod_{l \neq i} \pi_{u_l,x}$
/* $\alpha$ is a normalization constant */
• For each $x = j$, compute
$\pi_{x,y_i}(j) = \alpha \prod_{k \neq i} \lambda_{y_k,x}(j) \sum_{u_l} P(x = j \mid u_1, ..., u_m) \prod_l \pi_{u_l,x}$.
3. **Belief update:** For each $x$ along $A$
• For each $x = j$, compute $\lambda_x(j) = \prod_i \lambda_{y_i,x}(j)$, where $y_i$ are $x$'s children.
• For each $x = k$, compute $\pi_x(j) = \sum_{u_1,...,u_m} P(x = j \mid u_1, ..., u_m) \prod_i \pi_{u_i,x}$, where $u_i$ are $x$'s parents.
• Compute $BEL(x) = \alpha \lambda_x \pi_x$.

FIG. 21. Iterative belief propagation (IBP) algorithm.

sending messages between the nodes: each parent $u_i$ of a node $x$ sends a *causal* support message $\pi_{u_i,x}$ to $x$, while each of $x$'s children, $y_j$, sends to $x$ a *diagnostic* support message $\lambda_{y_j,x}$. The causal support from all parents and the diagnostic support from all children are combined into vectors $\pi_x$ and $\lambda_x$, respectively.

Nodes are processed (activated) in accordance with a variable ordering called an *activation schedule*. Processing all nodes along the given ordering yields one iteration of belief propagation. Subsequent iterations update the messages computed during previous iterations. Algorithm IBP(I) stops after $I$ iterations. If applied to polytrees, two iterations of the algorithm are guaranteed to converge to the correct a posteriori beliefs [Pearl 1988]. For multiply-connected networks, however, the algorithm may not even converge, or it may converge to incorrect beliefs. Some analysis of iterative belief propagation on networks with cycles is presented in [Weiss 1997] for the case of a single cycle.

For the sake of completeness, algorithm IBP(I) is shown in Figure 21. In our implementation, we assumed an activation schedule that first updates the input variables of the coding network and then updates the parity-check variables. Clearly, evidence variables are not updated. *Bel*($x$) computed for each node can be viewed as an approximation to the posterior beliefs. The tuple generated by selecting the most probable value for each node is the output of the decoding algorithm.

9.5.3. *Experimental Methodology.* We experimented with several types of ($N$, $K$) linear block codes, which include (7, 4) and (15, 11) Hamming codes, randomly generated codes, and structured codes with relatively low induced width. The code rate was $R = 1/2$, that is, $N = 2K$. As described above, linear block codes can be represented by four-layer belief networks having $K$ nodes in each layer (see Figure 12). The two outer layers represent the channel output $y = (y^u, y^x)$, where $y^u$ and $y^x$ result from transmitting the input vectors $u$ and $x$, respectively. The input nodes are binary (0/1), while the output nodes are real-valued.

Random codes are generated as follows: For each parity-check bit $x_j$, $P$ parents are selected randomly out of the $K$ information bits. Random codes are similar to the *low-density generator matrix* codes [Cheng 1997], which randomly select a given number $C$ of *children* nodes for each information bit.

Structured codes are generated as follows. For each parity bit $x_i$, $P$ sequential parents $\{u_{(i+j) mod K}, 0 \leq j < P\}$ are selected. Figure 12 shows a belief network of the structured code with $K = 5$ and $P = 3$. Note that the induced width of the network is 3, given the order $x_0, \ldots, x_4, u_0, \ldots, u_4$. In general, a structured $(N, K)$ block code with $P$ parents per each code bit has induced width $P$, no matter how large $K$ and $N$ are.

Given $K$, $P$, and the channel noise variance $\sigma^2$, a coding network *instance* is generated as follows: First, the appropriate belief network structure is created. Then, an input signal is simulated, assuming uniform prior distribution of information bits. The parity-check bits are computed accordingly and the codeword is "transmitted" through the channel. As a result, Gaussian noise with variance $\sigma^2$ is added to each information and parity-check bit yielding the channel output $y$, namely, a real-valued assignment to the $y_i^u$ and $y_j^x$ nodes.[6] The decoding algorithm takes as an input the coding network and the observation (evidence) $y$ and returns the recovered information sequence $u'$.

We experimented with iterative belief propagation *IBP*($I$), with the exact elimination algorithms for belief updating (*elim-bel*), for finding MAP (*elim-map*), and for finding MPE (*elim-mpe*), and with the mini-bucket approximation algorithms *mbe-mpe(i)*. In our experiments, *elim-map* always achieved the same bit error rate (BER) as *elim-mpe*, and, therefore, only the latter is reported.

For each Hamming code network and for each structured code network, we simulate 10,000 and 1,000 input signals, respectively, and report the corresponding BERs associated with the algorithms. For the random code networks, the BER is computed over 1,000 random networks while using only one randomly generated signal per network.

Note that in our previous experiments we compared actual probabilities (*MPE* and its bounds), while in the coding domain we use a secondary error measure, the BER. The bit error rate is plotted as a function of the channel noise and is compared to the Shannon limit and to the performance of a high-quality turbo-code reported in Frey [1998] and used as a reference here (this code has a very large block size $K = 65,536$, code rate 1/2, and was decoded using 18 iterations of IBP until convergence [Frey 1998]). In the coding community, the channel noise is commonly measured in units of decibels (dB), $10\ log_{10} E_b/N_o$, where $E_b/N_o$ is called signal-to-noise-ratio and defined as

$$E_b/N_o = \frac{P}{2\sigma^2 R}.$$

$P$ is the transmitter power, that is, bit 0 is transmitted as $-\sqrt{P}$, and bit 1 is transmitted as $\sqrt{P}$; $R$ is the code rate, and $\sigma^2$ is the variance of Gaussian noise. We use 0/1 signaling (equivalent to $-\frac{1}{2}/+\frac{1}{2}$ signaling ), so that $P = \frac{1}{4}$.

[6] Note that simulation of the channel output is akin to the simulation of likely evidence in a general Bayesian network (i.e., forward sampling, or ancestral simulation). As observed in the previous sections, $mbe - mpe$ is more accurate, on average, when evidence is likely. Not surprisingly, similar results were observed on coding networks.

TABLE VII. BER OF EXACT DECODING ALGORITHMS *elim-bel* (DENOTED *bel*) AND *elim-mpe* (DENOTED *mpe*) ON SEVERAL BLOCK CODES (AVERAGE ON 1000 RANDOMLY GENERATED INPUT SIGNALS)

| $\sigma$ | Hamming code | | | | Random code | | | | | | Structured code | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | (7,4) | | (15,11) | | K = 10, N = 20, P = 3 | | K = 10, N = 20, P = 5 | | K = 10, N = 20, P = 7 | | K = 25, P = 4 | |
| | bel | mpe | bel | mpe | bel | mpe | bel | mpe | bel | mpe | bel | mpe |
| 0.3 | 6.7e-3 | 6.8e-3 | 1.6e-2 | 1.6e-2 | 1.8e-3 | 1.7e-3 | 5.0e-4 | 5.0e-4 | 2.1e-3 | 2.1e-3 | 6.4e-4 | 6.4e-4 |
| 0.5 | 9.8e-2 | 1.0e-1 | 1.5e-1 | 1.6e-1 | 8.2e-2 | 8.5e-2 | 8.0e-2 | 8.1e-2 | 8.7e-2 | 9.1e-2 | 3.9e-2 | 4.1e-2 |

9.5.4. *Results.*

9.5.4.1. EXACT MPE VERSUS EXACT BELIEF-UPDATE DECODING. Before experimenting with the approximation algorithms for bit-wise and for block-wise decoding (namely, for belief updating and for MPE), we tested whether there is a significant difference between the corresponding exact decoding algorithms. We compared the exact *elim-mpe* algorithm against the exact *elim-bel* algorithm on several types of networks, including two Hamming code networks, randomly generated networks with different number of parents, and structured code. The BER on 1000 input signals, generated randomly for each network, are presented in Table VII. When the noise is relatively low ($\sigma = 0.3$), both algorithms have practically the same decoding error, while for larger noise ($\sigma = 0.5$) the bit-wise decoding (*elim-bel*) gives a slightly smaller BER than the block-wise decoding (*elim-mpe*). Consequently, comparing an approximation algorithm for belief updating (IBP(I)) to an approximation algorithm for MPE (*mbe-mpe(i)*) makes sense in the coding domain.

9.5.4.2. STRUCTURED LINEAR BLOCK CODES. In Figure 22, we compare the algorithms on the structured linear block code networks with $N = 50$ and 100, $K = 25$ and 50, and $P = 4$ and $P = 7$. The figures also displays the Shannon limit and the performance of IBP(18) on a state-of-the-art turbo-code having input block size $K = 65{,}536$ and rate $1/2$ (the results are copied from Frey [1998]). Clearly, our codes are far from being optimal: even the exact *elim-mpe* decoding yields a much higher error than the turbo-code. However, the emphasis of our preliminary experiments was not on improving the state-of-the-art decoder but rather on evaluating the performance of *mbe-mpe* and comparing it to the performance of IBP($i$) and *mbe-mpe* on different types of networks.

We observe that:

(1) As expected, the exact *elim-mpe* decoder always gives the smallest error among the algorithms we tested;
(2) IBP(10) is more accurate on average than IBP(1);
(3) *mbe-mpe(i)*, even for $i = 1$, is close to *elim-mpe*, due to the low induced width of the structured code networks ($w^* = 6$ for $P = 4$, and $w^* = 12$ for $P = 7$), and it outperforms IBP on all structured networks;
(4) Increasing the parents set size from $P = 4$ (Figures 22(a) and 22(b)) to $P = 7$ (Figures 22(c) and 22(d)), makes the difference between IBP and *mbe-mpe(i)* become even more pronounced. On networks with $P = 7$ both mbe-mpe(1) and mbe-mpe(7) achieve an order of magnitude smaller error than IBP(10).

Next, we consider the results for each algorithm separately, while varying the number of parents from $P = 4$ to $P = 7$. We see that the error of IBP(1) remains practically the same, the error of the exact elim-mpe changes only slightly, while the error of IBP(10) and *mbe-mpe(i)* increases. However, the BER of IBP(10)

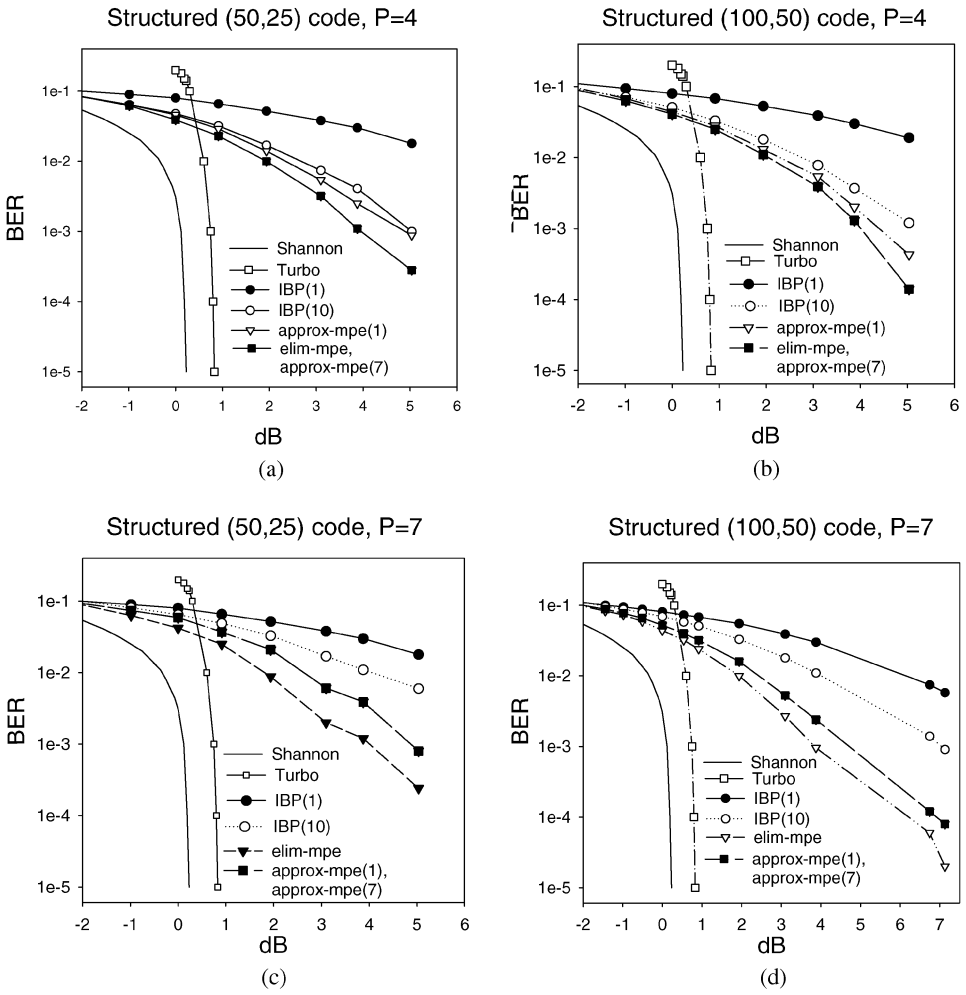



FIG. 22. The average performance of *elim-mpe*, *mbe-mpe*(i), and IBP(I) on rate 1/2 *structured block codes* and 1000 randomly generated input signals. The induced width of the networks is: (a), (b) $w^* = 6$; (c), (d) $w^* = 12$. The bit error rate (BER) is plotted versus the channel noise measured in decibels (dB), and compared to the Shannon limit and to the performance of IBP(18) on a high-quality code reported [Frey 1998] (a turbo-code having input block size $K = 65{,}536$ and rate 1/2. Notice that *mbe-mpe*(7) coincides with *elim-mpe* in (a) and (b), while in (c) and (d) it coincides with *mbe-mpe*(1).

increased more dramatically with increased parent set. Note that the induced width of the network increases with the increase in parent set size. In the case of $P = 4$ (induced width 6), *mbe-mpe*(7) coincides with *elim-mpe*; in the case of $P = 7$ (induced width 12), the approximation algorithms do not coincide with *elim-mpe*. Still, they are better than IBP.

9.5.4.3. RANDOM LINEAR BLOCK CODE. On randomly generated linear block networks (Figure 23(a)), the picture was reversed: *mbe-mpe(i)* for both $i = 1$ and $i = 7$ was worse than IBP(10), although as good as IBP(1). *Elim-mpe* always ran out of memory on these networks (the induced width exceeded 30). The results are not surprising since *mbe-mpe(i)* can be inaccurate if $i$ is much lower than the

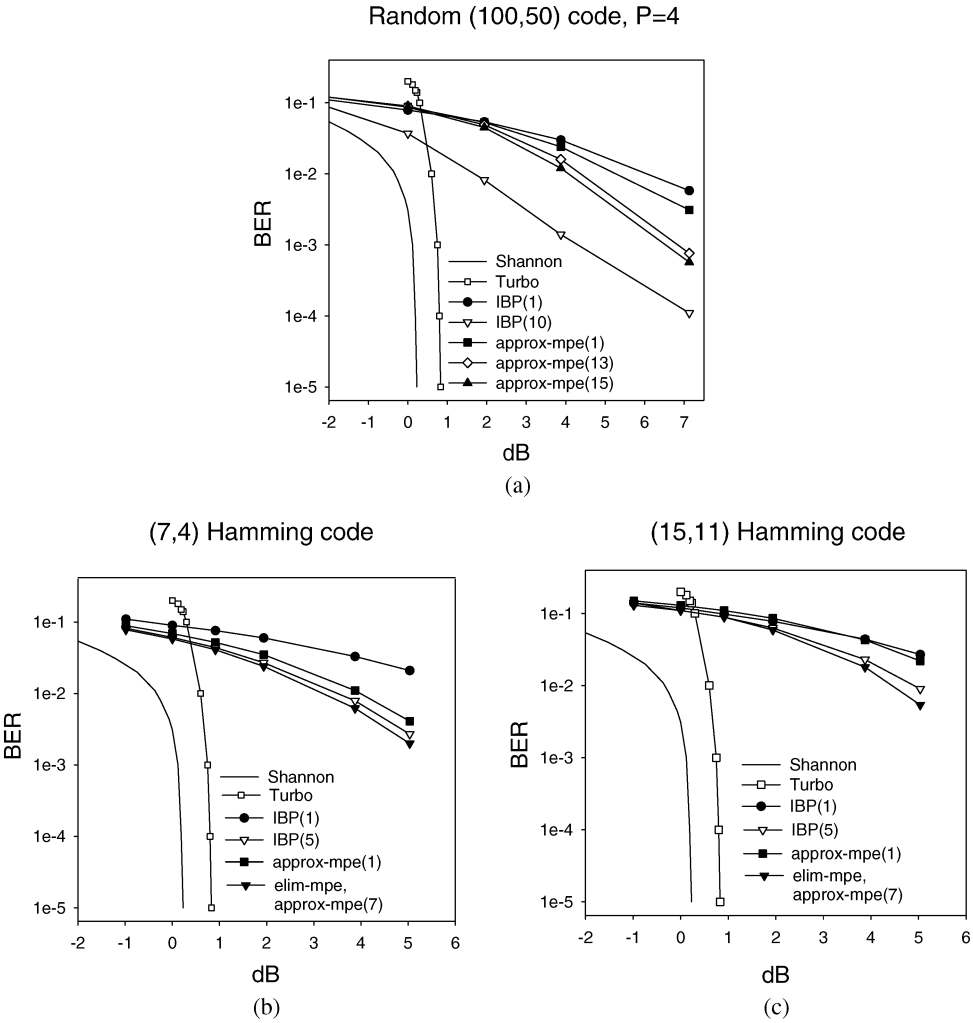



FIG. 23. The average performance of *elim-mpe*, *mbe-mpe*(i), and IBP(I) on (a) 1000 instances of rate 1/2 *random block codes*, one signal instance per code; and on (b) (7, 4) and (c) (15, 11) *Hamming codes*, 1000 signal instances per each code. The induced width of the networks is: (a) $30 \leq w^* \leq 45$; (b) $w^* = 3$; (c) $w^* = 9$. The bit error rate (BER) is plotted versus the channel noise measured in decibels (dB), and compared to the Shannon limit and to the performance of IBP(18) on a high-quality code reported [Frey 1998] (a turbo-code having input block size $K = 65{,}536$ and rate 1/2. Notice that *mbe-mpe*(7) coincides with *elim-mpe* in (a) and (b), while in (c) and (d) it coincides with *mbe-mpe*(1).

induced width. However, it is not clear why IBP is better in this case. Also, note that the experiments on random networks differ from those described above. Rather than simulating many input signals for one network, we average results over many random networks with one random signal per each network. To be more conclusive, one needs to compare our algorithms on other random code generators such as the recently proposed low-density generator matrix codes [Cheng 1997] and low-density parity-check codes [MacKay and Neal 1996]. However, such experiments are outside the scope of the current paper.

9.5.4.4. HAMMING CODES. We tested the belief propagation and the mini-bucket approximation algorithms on two Hamming code networks, one with $N = 7$, $K = 4$, and the other one with $N = 15, K = 11$. The results are shown in Figures 23(b) and 23(c). Again, the most accurate decoder was the exact *elim-mpe*. Since the induced width of the (7, 4) Hamming network is only 3, *mbe-mpe*(7) coincides with the exact algorithm. IBP(1) is much worse than the rest of the algorithms, while IBP(5) is very close to *elim-mpe*. Algorithm *mbe-mpe*(1) is slightly worse than IBP(5). On the larger Hamming code network, the results are similar, except that both *mbe-mpe*(1) and *mbe-mpe*(7) are significantly inferior to IBP(5). Since the networks were quite small, the runtime of all the algorithms was less than a second, and the time of IBP(5) was comparable to the time of exact elim-mpe.

9.5.5. *Channel Coding: Summary.* We showed that

(1) On a class of structured codes having low induced width the mini-bucket approximation *mbe-mpe* outperforms *IBP*;
(2) On a class of random networks having large induced width and on some Hamming codes *IBP* outperforms *mbe-mpe*;
(3) As expected, the exact MPE decoding (*elim-mpe*) outperforms approximate decoding. However, on random networks, finding exact MPE was not feasible due to the large induced width.
(4) On some classes of problems, the *exact* maximum-likelihood decoding using *elim-mpe* and the *exact* belief update decoding using *elim-bel* have comparable error for relatively low channel noise; for higher noise, belief-update decoding gives a slightly smaller (by $\approx 0.1\%$) bit error rate than the MPE decoding.

As dictated by theory, we observed a correlation between the network's induced width and the quality of the mini-bucket's approach. In summary, our results on structured codes demonstrate that the mini-bucket scheme may be a better decoder than IBP on coding networks having relatively low induced width. Additional experiments are required in order to generalize this result for practical codes having large block size (e.g., $N \approx 10^4$).

Our experiments were restricted to networks having small parent sets since the mini-bucket and the belief propagation approaches are, in general, time and space exponential in the parent set. This limitation can be removed by using the specific structure of *deterministic* CPTs in the coding networks, which is a special case of *causal independence* [Heckerman and Breese 1995; Zhang and Poole 1996]. Such networks can be transformed into networks having families of size three only. Indeed, in coding practice, the belief propagation algorithm exploits the special structure of the CPTs and is linear in the family size.

## 10. *Related Work*

The basic idea for approximating dynamic-programming type algorithms appears in the early work of Montanari [1972] who proposed to approximate a discrete function of high arity by a sum of functions having lower arity. Montanari uses the mean square error in choosing the low-arity representation.

The mini-bucket approximation is the first attempt to approximate all bucket elimination algorithms within a single principled framework. The bucket elimination

framework [Dechter 1999] provides a convenient and succinct language for expressing elimination algorithms in many areas. In addition to dynamic programming [Bertele and Brioschi 1972], constraint satisfaction [Dechter and Pearl 1987], and Fourier elimination [Lassez and Mahler 1992], there are variations on these ideas and algorithms for probabilistic inference [Cannings et al. 1978; Tatman and Shachter 1990; Zhang and Poole 1996].

Our approach is inspired by *adaptive-consistency*, a full bucket elimination algorithm for constraint satisfaction whose approximation, *directional i-consistency* and its relational variant *directional-relational-consistency(i, m)* ($DRC_{(i,m)}$), enforce bounded levels of consistency [Dechter and van Beek 1997]. For example, directional relational arc-consistency, $DRC_1$, is similar to *mini-bucket*($m = 1$); directional path-consistency, $DRC_2$, corresponds to *mini-bucket*($m = 2$); and so on.

Note that mini-bucket approximations can be used as heuristics for subsequent search, similar to pre-processing by local consistency prior to backtrack search for constraint solving. Indeed, this direction was recently pursued quite successfully embedding mini-bucket heuristics in branch and bound search [Kask and Dechter 1999, 2001; Kask 2000]. In propositional satisfiability, *bounded-directional-resolution* with bound $b$ [Dechter and Rish 1994; Rish and Dechter 2000] corresponds to *mini-bucket*($i = b$). It bounds the original resolution-based Davis–Putnam algorithm [Davis and Putnam 1960].

Further comparisons of the mini-bucket scheme with search algorithms (for MPE task), as well as combinations of mini-bucket preprocessing with search were reported in [Kask and Dechter 1999b]. Empirical results show that on some classes of random problems the mini-bucket algorithm *mbe-mpe* outperforms greedy local search, stochastic local search (SLS), a combination of greedy with SLS, simulated annealing, and iterative belief propagation (unlike in coding networks). The overall winner is the combination of greedy search with SLS applied on top of the mini-bucket scheme (i.e., using the mini-bucket solution as an initial assignment).

Another idea related to bounding dependencies is that of removing *weak dependencies* in a join-tree clustering scheme presented in [Kjaerulff 1994]. This work suggests the use of Kullback–Leibler distance (KL-distance, or relative entropy) in deciding which dependencies to remove. Both the KL-distance measure and the mean square error can be used to improve the mini-bucket partitioning scheme. A similar approximation idea based on ignoring some dependencies was proposed by [Boyen and Koller 1998] for stochastic processes, and can be perceived as similar to mini-bucket($n$, 1).

The mini-bucket scheme is closely related to other local approximations, such as *iterative belief propagation (IBP), generalized belief propagation (GBP)* algorithms [Yedidia et al. 2001] and in particular the recently proposed *Iterative Join-Graph Propagation* [Dechter et al. 2002] that were successfully used in practical applications such as probabilistic error-correcting coding [Frey and MacKay 1998]. The mini-bucket algorithms can be viewed as a noniterative version of all those approaches.

The mini-bucket scheme was recently extended to tree-elimination algorithms such as bucket tree elimination and join-tree clustering schemes and is called *mini-bucket tree-elimination (MBTE)*. This extension allows approximating updated

beliefs of all the variables at once, as well as computing other quantities of interest, using single additional function generation pass in the reverse direction. This approach was tested empirically for updating beliefs and showed highly competitive results in many cases, compared with Gibbs sampling and iterative belief propagation [Mateescu et al. 2002].

A collection of approximation algorithms for sigmoid belief networks was recently presented [Jaakkola and Jordan 1996] in the context of a recursive algorithm similar to bucket elimination. Upper and lower bounds are derived by approximating sigmoid functions by Gaussian functions. This approximation can be viewed as a singleton mini-bucket algorithm ($m = 1$) where Gaussian functions replace the *min* or *max* operations applied in each mini-bucket. Approximations for sigmoid belief networks belong to a wider class of *variational methods* for approximate inference [Jordan et al. 1998]. Finally, there is a large family of approximation techniques somewhat orthogonal to local propagation, namely, sampling techniques (Markov-Chain Monte-Carlo, or MCMC mehtods) often applied to approximate inference in Bayesian networks [Pearl 1988; MacKay 1998]. However, sometimes slow convergence of these methods calls for hybrid algorithms that combine sampling with local propagation and other approaches, thus exploiting different kinds of structure present in joint probability distributions.

## 11. *Conclusions*

The article describes a new approximate inference scheme, called mini-buckets, that trades accuracy for efficiency when computational resources are bounded. The scheme bounds the dimensionality of dependencies created by inference algorithms. The mini-bucket scheme is based on the bucket-elimination framework [Dechter 1997a] that is applicable across a wide variety of areas.

We presented and analyzed the mini-bucket approximation algorithms for the probabilistic tasks of belief updating, finding the most probable explanation (MPE), and finding the maximum a posteriori hypothesis (MAP) as well as for general combinatorial optimization tasks. We identified regions of completeness and demonstrated promising empirical results obtained both on randomly generated networks and on realistic domains such as medical diagnosis and probabilistic decoding. The complexity bounds of the mini-bucket algorithms provide some guidelines for selecting algorithm's parameters based both on memory considerations and on the problem's graph structure. An anytime version of the algorithm iteratively increments the controlling parameters and guarantees an increase in accuracy as long as computational resources are available.

Our experimental work focused on evaluating *mbe-mpe*, the mini-bucket approximation algorithm for MPE. We demonstrated that the algorithm provides a good approximation accompanied by a substantial speedup in many cases. Specifically, we observed that

(a) As expected, the algorithm's performance (when $i$ is fixed) decreases with increasing network density.

(b) The algorithm works significantly better for structured rather than for uniformly distributed CPTs. For example, for noisy-OR random networks and for CPCS

networks (noisy-OR and noisy-MAX CPTs), we often computed an accurate solution in cases when the exact algorithm was much slower, or infeasible.

(c) For probabilistic decoding we demonstrated that the mini-bucket scheme outperforms the state-of-the-art iterative belief propagation decoding algorithm on problems having low $w^*$. However, on randomly generated high-$w^*$ codes, the mini-bucket approximation was inferior to IBP.

(d) The lower bound computed by *mbe-mpe* was often closer to the MPE than the corresponding upper bound, indicating that good solutions are found long before this can be verified by the generated upper bound.

(e) The approximation accuracy is significantly better for problems having high MPE (e.g., likely evidence), as observed on CPCS networks and coding problems. Also, the algorithm's performance improves considerably with decreasing noise in noisy-OR CPTs, yielding an exact solution in practically all zero-noise cases while using a relatively low bound $i$. We believe that these results can also be attributed to high MPE values that normally accompany noisy-OR problems having low noise.

The mini-bucket scheme can be improved along the following lines. Instead of using the brute-force procedure for partitioning a bucket into mini-buckets we can improve the decomposition by minimizing a distance metric between the exact function and its approximation. Candidate metrics are relative entropy (KL-distance) [Kjaerulff 1994] and the min-square error [Montanari 1972]. The approximation may be further improved for the special cases of noisy-OR, noisy-MAX, and noisy-XOR (also known as *causal independence* [Heckerman and Breese 1995; Zhang and Poole 1996; Rish and Dechter 1998]) structures that are often present in real-life domains (e.g., CPCS and coding networks). Also, it was shown that combining the mini-bucket scheme with heuristic search has a great potential [Kask and Dechter 1999].

Finally, theoretical explanation of our empirical results (e.g., on relatively high-MPE and low-noise problems) and prediction of mini-bucket accuracy still remain open questions. Initial theoretical analysis and certain optimality conditions are given in [Rish et al. 2002] for the simplest member of the mini-bucket family, greedy forward assignment (no preprocessing) for finding MPE, on problems having high MPE probability, such as some problems with nearly deterministic CPTs (e.g., noisy-OR networks with low noise). Interestingly, the greedy approximation is guaranteed to give an exact solution when MPE values are high enough (a frequent situation in low-noise problems), but its accuracy drops dramatically after certain threshold MPE value. Investigating the behavior of more complex mini-bucket approximations under similar conditions remains the topic of further research.

ACKNOWLEDGMENT. We wish to thank Padhraic Smyth and Robert McEliece for insightful discussions and for providing information about the coding domain, Kalev Kask for his contribution to the joint work on coding problems [Kask et al. 1998], and to Nira Brand for useful editorial comments.

REFERENCES

ARNBORG, S. A. 1985. Efficient algorithms for combinatorial problems on graphs with bounded decomposability—A survey. *BIT 25*, 2–23.

BERROU, G., GLAVIEUX, A., AND THITIMAJSHIMA, P. 1993. Near Shannon limit error-correcting coding: Turbo codes. In *Proceedings of the 1993 International Conference on Communications* (Geneva, May).

BERTELE, U., AND BRIOSCHI, F. 1972. *Nonserial Dynamic Programming*. Academic Press, New York.

BODDY, M., AND DEAN, T. L. 1989. Solving time-dependent planning problems. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*. pp. 979–984.

BODLAENDER, H. L. 1997. Treewidth: Algorithmic techniques and results. In *Proceedings of 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS'97)*. Lecture Notes in Computer Science, vol. 1295. Springer-Verlag, New York, pp. 19–36.

BODLAENDER, H. L., KOSTER, A. M., EIJKHOF, F. V., AND VAN DER GAAG, L. C. 2001. Pre-processing for triangulation of probabilistic networks. In *Proceedings of 17th Annual Conference of Uncertainty in Artificial Intelligence (UAI01)*. pp. 32–39.

BOYEN, X., AND KOLLER, D. 1998. Tractable inference for complex stochastic processes. In *Proceedings of 14th annual conference on Uncertainty in AI (UAI)* (Madison, Wisc). pp. 33–42.

CANNINGS, C., THOMPSON, E. A., AND SKOLNICK, H. H. 1978. Probability functions on complex pedigrees. *Adv. Appl. Prob. 10*, 26–61.

CHENG, J.-F. 1997. Iterative decoding. Ph.D. dissertation. Caltech, Pasadena, Calif.

COOPER, G. F. 1990. The computational complexity of probabilistic inference using Bayesian belief networks. *Artif. Intel. 42*, 2–3, 393–405.

DAGUM, P., AND LUBY, M. 1993. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artif. Intel. 60*, 1, 141–155.

D'AMBROSIO, B. 1994. Symbolic probabilistic inference in large BN2O networks. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*. pp. 128–135.

DAVIS, M., AND PUTNAM, H. 1960. A computing procedure for quantification theory. *J. ACM 7*, 3.

DEAN, T. L., AND BODDY, M. 1988. An analysis of time-dependent planning. In *Proceedings of the 7th National Conference on Artificial Intelligence*. PP. 49–54.

DECHTER, R. 1992. Constraint networks. In *Encyclopedia of Artificial Intelligence*, 2nd ed. Wiley, New York, pp. 276–285.

DECHTER, R. 1996. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)*. pp. 211–219.

DECHTER, R. 1997a. Bucket elimination: A unifying framework for processing hard and soft constraints. *CONSTRAINTS: An Int. J. 2*, 51–55.

DECHTER, R. 1997b. Mini-buckets: A general scheme for generating approximations in automated reasoning. In *Proceedings of the 15th International Joint Conference of Artificial Intelligence (IJCAI-97)* (Japan). pp. 1297–1302.

DECHTER, R. 1998. Constraint satisfaction. In *MIT Encyclopedia of the Cognitive Sciences (MITECS)*. MIT Cambridge, Mass.

DECHTER, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artif. Intel. 113*, 41–85.

DECHTER, R., AND FROST, D. 1997. Backtracking algorithms for constraint satisfaction problems, a tutorial survey. Tech. rep., UCI.

DECHTER, R., KASK, K., AND MATEESCU, R. 2002. Iterative join-graph propagation. In *Proceedings of UAI-02*.

DECHTER, R., AND PEARL, J. 1987. Network-based heuristics for constraint satisfaction problems. *Artif. Intel. 34*, 1–38.

DECHTER, R., AND RISH, I. 1994. Directional resolution: The Davis–Putnam procedure, revisited. In *Principles of Knowledge Representation and Reasoning (KR-94)*. pp. 134–145.

DECHTER, R., AND RISH, I. 1997. A scheme for approximating probabilistic inference. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI97)*.

DECHTER, R., AND VAN BEEK, P. 1997. Local and global relational consistency. *Theoret. Comput. Sci.* pp. 283–308.

DRAPER, D. 1995. Localized partial evaluation of belief networks. Tech. rep. Ph.D. dissertation. University of Washington.

EL FATTAH, Y., AND DECHTER, R. 1995. Diagnosing tree-decomposable circuits. In *Proceedings of the International Joint Conference of Artificial Intelligence (IJCAI-95)* (Montreal, Ont., Canada, Aug.). pp. 1742–1748.

JENSEN, F., AND ANDERSEN, S. K.. 1990. Approximations in Bayesian belief universes for knowledge-based systems. In *Proceedings of the 6th Conference on Uncertainty in Artificial Intelligence*.

FREUDER, E. C. 1978. Synthesizing constraint expressions. *Commun. ACM 21*, 11, 958–965.
FREUDER, E. C. 1982. A sufficient condition for backtrack-free search. *J. ACM 21*, 11, 958–965.
FREY, B. J. 1998. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, Cambridge, Mass.
FREY, B. J. 1998. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, Cambridge, Mass.
FREY, B. J., AND MACKAY, D. J. C. 1998. A revolution: belief propagation in graphs with cycles. *Adv. Neural Inf. Proc. Syst. 10*.
GALLAGER, R. G. 1965. A simple derivation of the coding theorem and some applications. *IEEE Trans. Inf. Theory IT-11*, 3–18.
GUO, H., AND HSU, W. 2002. A survey on algorithms for real-time Bayesian network inference. In *Proceedings of the Joint AAAI-02/KDD-02/UAI-02 Workshop on Real-Time Decision Support and Diagnosis Systems* (Edmonton, Alberta, Canada).
HECKERMAN, D. AND BREESE, J. 1995. Causal independence for probability assessment and inference using Bayesian networks. Tech. Rep. MSR-TR-94-08, Microsoft Research.
HORVITZ, E. 1987. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 3rd Conference on Uncertainty in Artificial Intelligence (UAI-87)* (Seattle, Wash.). pp. 301–324.
HORVITZ, E. 1988. Reasoning under varying and uncertain resource constraints. In *Proceedings of the 4th Conference on Uncertainty in Artificial Intelligence (UAI-88)*. pp. 111–116.
HORVITZ, E. 1990. Computation and action under bounded resources. Ph.D. dissertation. Stanford Univ., Stanford, Calif.
HORVITZ, E., SUERMONDT, H. J., AND COOPER, G. F. 1989. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of the 5th Conference on Uncertainty in Artificial Intelligence (UAI-89)*. pp. 182–193.
KASK, K., RISH, I., AND DECHTER, R. 1998. Approximation algorithms for probabilistic decoding. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*.
JAAKKOLA, T. S., AND JORDAN, M. I. 1996. Recursive algorithms for approximating probabilities in graphical models. *Adv. Neural Inf. Proc. Syst. 9*.
JORDAN, M. I., GHAHRAMANI, Z., JAAKKOLA, T. S., AND SAUL, L. K. 1998. *An Introduction to Variational Methods for Graphical Models*. Kluwer Academic Publishers.
KASK, K. 2000. New search heuristics for max-csp. In *Proceedings of the Principles and Practice of Constraint Programming (CP2000)*. pp. 255–269.
KASK, K., AND DECHTER, R. 1999a. Mini-bucket heuristics for improved search. In *Proceedings of 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*.
KASK, K., AND DECHTER, R. 1999b. Stochastic local search for bayesian networks. In *Proceedings of Workshop on AI and Statistics (AISTAT'99)*. pp. 113–122.
KASK, K., AND DECHTER, R. 2001. A general scheme for automatic search heuristics from specification dependencies. *Artif. Intel.*
KASK, K., DECHTER, R., LARROSA, J., AND FABIO, G. 2001. Bucket-tree elimination for automated reasoning. *Artif. Intel. 125*, 91–131.
KJAERULFF, U. 1990. Triangulation of graph-based algorithms giving small total state space. Tech. Rep. 90-09, Department of Mathematics and Computer Science, Univ. Aalborg, Aalborg, Denmark.
KJAERULFF, U. 1992. Optimal decomposition of probabilistic networks by simulated annealing. *Stat. Comput. 2*, 7–17.
KJAERULFF, U. 1994. Reduction of computational complexity in Bayesian networks through removal of week dependencies. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI-94)*.
LARROSA, J. 2000. On the time complexity of bucket elimination algorithms. ICS Tech. rep.
LASSEZ, J.-L., AND MAHLER, M. 1992. On Fourier's algorithm for linear constraints. *J. Automat. Reas. 9*.
LAURITZEN, S. L., AND SPIEGELHALTER, D. J. 1988. Local computation with probabilities on graphical structures and their application to expert systems. *J. Roy. Stat. Soc., Seri. B 50*, 2, 157–224.
MACKAY, D. J. C. 1998. Introduction to Monte Carlo methods. In *Learning in Graphical Models*, M. I. Jordan, ed. Kluwer Academic Press.
MACKAY, D. J. C., AND NEAL, R. M. 1996. Near Shannon limit performance of low density parity check codes. *Electron. Lett. 33*, 457–458.

MACKWORTH, A. K. 1977. Consistency in networks of relations. *Artif. Intel. 8*, 1, 99–118.

MATEESCU, R., DECHTER, R., AND KASK, K. 2002. Tree approximation for belief updating. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-2002)* (Edmonton, Alberta, Canada). pp. 553–559.

MCELIECE, R. J., MACKAY, D. J. C., AND CHENG, J. F. 1997. Turbo decoding as an instance of Pearl's belief propagation algorithm. *IEEE J. Select. Areas Commun.*

MILLER, R. A., MASARIE, F. E., AND MYERS, J. 1986. Quick medical reference (QMR) for diagnostic assistance. *Medi. Comput. 3*, 5, 34–38.

MILLER, R. A., POPLE, H. E., AND MYERS, J. D. 1982. Internist-1: An experimental computer-based diagnostic consultant for general internal medicine. *New Eng. J. Med. 307*, 468–476.

MONTANARI, U. 1972. On the optimal approximation of discrete functions with low-dimensional tables. *Inf. Proc. 71*, 1363–1368.

PARK, J. 2002. MAP complexity results and approximation methods. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI)* (San Francisco, Calif.) Morgan-Kaufmann, San Mateo, Calif., pp. 388–396.

PARKER, R., AND MILLER, R. 1987. Using causal knowledge to create simulated patient cases: the CPCS project as an extension of INTERNIST-1. In *Proceedings of the 11th Symposium Computer Applications in Medical Care*, pp. 473–480.

PEARL, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan-Kaufmann, San Mateo, Calif.

PENG, Y., AND REGGIA, J. A. 1989. A connectionist model for diagnostic problem solving. *IEEE Trans. Syst., Man and Cybernet.*

POOLE, D. 1996. Probabilistic conflicts in a search algorithm for estimating posterior probabilities in Bayesian networks. *Artif. Intel. 88*, 69–100.

PRADHAN, M., PROVAN, G., MIDDLETON, B., AND HENRION, M. 1994. Knowledge engineering for large belief networks. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*.

RISH, I., BRODIE, M., AND MA, S. 2002. Accuracy vs. efficiency trade-offs in probabilistic diagnosis. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI2002)* (Edmonton, Alb., Canada).

RISH, I., AND DECHTER, R. 1998. On the impact of causal independence. Tech. rep., Information and Computer Science, Univ. California, Irvine, Irvine, Calif.

RISH, I., AND DECHTER, R. 2000. Resolution vs. search; Two strategies for SAT. *J. Automat. Reas. 24*, 1/2, 225–275.

ROBERTSON, N., AND SEYMOUR, P. 1995. Graph minor. XIII. The disjoint paths problem. *Combinat. Theory, Ser. B 63*, 65–110.

ROTH, D. 1996. On the hardness of approximate reasoning. *AI J. 82*, 1-2 (Apr.), 273–302.

SANTOS, E. 1991. On the generation of alternative explanations with implications for belief revision. In *Proceedings of the 7th Annual Conference on Uncertainty in Artificial Intelligence (UAI-91)*. pp. 339–347.

SANTOS, E. J., AND SHIMONY, S. E. 1998. Deterministic approximation of marginal probabilities in Bayes nets. *IEEE Trans. Syst. Man, Cybernet. 28*, 4, 377–393.

SHACHTER, R. D. 1986. Evaluating influence diagrams. *Oper. Res. 34*.

SHANNON, C. E. 1948. A mathematical theory of communication. *Bell Syst. Tech. J. 27*, 379–423, 623–656.

SHIMONY, S. E., AND CHARNIACK, E. 1991. A new algorithm for finding MAP assignments to belief networks. In *Uncertainty in Artificial Intelligence*, P. Bonissone, M. Henrion, L. Kanal, and J. Lemmer, Eds. pp. 185–193.

SHWE, M., MIDDLETON, B. F., HECKERMAN, D. E., HENRION, M., HORVITZ, E. J., LEHMANN, H., AND COOPER, G. F. 1991. Probabilistic diagnosis using a reformulation of the Internist-1/QMR knowledge base: I. The probabilistic model and inference algorithms. *Meth. Inf. Med. 30*, 241–255.

TATMAN, J. A., AND SHACHTER, R. D. 1990. Dynamic programming and influence diagrams. *IEEE Trans. Syst. Man, Cyberneti.*

VAN ENGELEN, R. A. 1997. Approximating Bayesian belief networks by arc removal. *IEEE Trans. Patt. Anal. Mach. Intel. 19*, 8, 916–920.

WEISS, Y. 1997. Belief propagation and revision in networks with loops. In *Proceedings of the NIPS-97 Workshop on Graphical Models*.

WELLMAN, M. P., AND LIU, C. 1994. State-space abstraction for anytime evaluation of probabilistic networks. In *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*. pp. 567–574.

YEDIDIA, J., FREEMAN, W. T., AND WEISS, Y. 2001. Generalized belief propagation. In *NIPS 13*. MIT Press, Cambridge, Mass, pp. 689–695.

ZHANG, N. L., AND POOLE, D. 1996. Exploiting causal independence in Bayesian network inference. *J. Artif. Intel. Res. 5*, 301–328.

RECEIVED NOVEMBER 2000; REVISED OCTOBER 2002; ACCEPTED OCTOBER 2002