Iterative Algorithms for Graphical Models [1]

Robert Mateescu
School of Information and Computer Science
University of California, Irvine
mateescu@ics.uci.edu
http://www.ics.uci.edu/∼mateescu

June 30, 2003

[1]This report was presented as part of the Ph.D. candidacy exam of Robert Mateescu

Abstract

Probabilistic inference in Bayesian networks, and even reasoning within error bounds are known to be NP-hard problems. Our research focuses on investigating approximate message-passing algorithms inspired by Pearl's belief propagation algorithm and by variable elimination. We study the advantages of bounded inference provided by anytime schemes such as Mini-Clustering (MC), and combine them with the virtues of iterative algorithms such as Iterative Belief Propagation (IBP). Our resulting hybrid algorithm Iterative Join-Graph Propagation (IJGP) is shown empirically to surpass the performance of both MC and IBP on several classes of networks. IJGP can also be viewed as a Generalized Belief Propagation algorithm, a framework which recently allowed connections with approximate algorithms from statistical physics, showing that convergence points are in fact stationary points of the Bethe (or the more general Kikuchi) free energy. Although there is still little understanding why or when IBP works well, it exhibits tremendous performance on different classes of problems, most notably coding and satisfiability problems. We investigate the iterative algorithms for Bayesian networks by making connections with well known constraint processing algorithms, which help explain when IBP infers correctly extreme beliefs. This study gives an account of why iterating helps, and identifies classes of easy and hard problems for IBP (and IJGP). Finally, we plan to investigate iterative message-passing algorithms in other graph-based frameworks such as influence diagrams and planning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Probabilistic inference is the principal task in Bayesian networks, and it is known to be an NP-hard problem [Cooper1990]. Most of the commonly used exact algorithms such as join-tree clustering [Lauritzen & Spiegelhalter1988, Jensen, Lauritzen, & Olesen1990] or variable-elimination [Dechter1996, N. L. Zhang & Poole1994], exploit the network structure. Yet, they are time and space exponential in a graph parameter called *induced width* (or *tree-width*), rendering them essentially intractable even for moderate size problems. Approximate algorithms are therefore necessary for most of the practical problems, although approximation within given error bounds is also NP-hard [Dagum & Luby1993, Roth1996].

Our research is focused primarily on graph algorithms for the task of belief updating. They are inspired by Pearl's belief propagation algorithm [Pearl1988], which is known to be exact for poly-trees, and by Mini-Buckets algorithm [Dechter & Rish1997], which an anytime version of variable elimination. As a distributed algorithm, belief propagation is also well defined for networks that contain cycles, and it can be applied iteratively in the form of Iterative Belief Propagation (IBP), also known as loopy belief propagation. When the networks contain cycles, IBP is no longer guaranteed to be exact, but in many cases it provides very good approximations upon convergence, most notably in the case of coding networks [R.J. McEliece & Cheng1997] and satisfiability zecchina02.

In our work, we investigate: 1. the quality of bounded inference in anytime schemes such as Mini-Clustering, which is a generalization of Mini-Buckets to arbitrary tree-decompositions; 2. the virtues of iterating message-passing algorithms, and the result of combining the two in algorithms such as Iterative Join-Graph Propagation (IJGP). Our recent work makes connections with well known understood consistency enforcing algorithms for constraint satisfaction, giving strong support for iterating messages, and helping identify cases of strong and weak inference power for IBP and IJGP.

The remaining of this chapter contains the necessary definitions and preliminaries. Chapter 2 discusses our past work. Section 2.1 describes the Mini-Clustering algorithm, Section 2.2 presents the Iterative Join-Graph Propagation algorithm, which is both anytime and iterative. Chapter 3 summarizes our current and futures directions of research. Section

3.1 gives a an account of why iterating helps by making connections to Arc-consistency algorithms from constraint satisfaction. It investigates the power of IBP to infer extreme beliefs. It also provides a solid ground in explaining why IBP works well for coding networks, a question which we plan to investigate further. Section 3.2 looks at the problem of partitioning in isolation, although this issue is relevant to many of the algorithms we develop and would impact other areas of our research. Finally, Section 3.3 briefly touches upon areas for our future work. We plan to study the applicability of our iterative algorithms, or to adjust them to other graphical-based frameworks such as influence diagrams and planning.

The published version of some of work presented in this report is given in Appendices A [Mateescu, Dechter, & Kask2002], B [Dechter, Mateescu, & Kask2002] and C [Dechter & Mateescu2003].

1.1 Preliminaries

The problems we address are in general expressed within two formalisms: *Belief Networks* and *Constraint Networks*. We provide below the basic definitions and concepts.

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. If $(X_i, X_j) \in E$, we say that X_i points to X_j. For each variable X_i, the set of parent nodes of X_i, denoted pa_{X_i}, or pa_i, comprises the variables pointing to X_i in G.

The family of X_i, F_i, includes X_i and its parent variables. A directed graph is *acyclic* if it has no directed cycles. In an *undirected graph*, the directions of the arcs are ignored: (X_i, X_j) and (X_j, X_i) are identical.

DEFINITION **1.1.1 (Induced-graphs and induced width)** *[Dechter & Pearl1987] An* ordered graph *is a pair (G, d) where G is an undirected graph, and $d = X_1, ..., X_n$ is an ordering of the nodes. The* width of a node *in an ordered graph is the number of the node's neighbors that precede it in the ordering. The* width of an ordering d*, denoted $w(d)$, is the maximum width over all nodes. The* induced width of an ordered graph, $w^*(d)$*, is the width of the induced ordered graph obtained as follows: nodes are processed from last to first; when node X is processed, all its preceding neighbors are connected. The* induced width of a graph, $w*$*, is the minimal induced width over all its orderings. The* tree-width *of a graph is the minimal induced width [Arnborg1985].*

1.2 Belief Networks

Belief networks provide a formalism for reasoning about partial beliefs under conditions of uncertainty. A belief network is defined by a directed acyclic graph over nodes representing random variables of interest.

DEFINITION **1.2.1 (belief network, moral graph)** *A* belief network *is a quadruple* $BN =< X, D, G, P >$[1] *where* $X = \{X_1, \ldots, X_n\}$ *is a set of random variables,* $D = \{D_1, ..., D_n\}$ *is the set of the corresponding domains,* G *is a directed acyclic graph over* X *and* $P = \{p_1, ..., p_n\}$*, where* $p_i = P(X_i|pa_i)$ *(*pa_i *are the parents of* X_i *in* G*) denote conditional probability tables (CPTs).*
The moral graph *of a belief network is obtained by connecting all parents nodes having a common child in* G *and removing directions.*

The belief network represents a probability distribution over X having the product form $P(x) = P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|x_{pa_i})$ where an assignment $(X_1 = x_1, \ldots, X_n = x_n)$ is abbreviated to $x = (x_1, \ldots, x_n)$ and where x_s denotes the restriction of a tuple x to the subset of variables S.

An evidence set e is an instantiated subset of variables. We use upper case letters for variables and nodes in a graph and lower case letters for values in variables' domains. Given a function f, we denote by $scope(f)$ the set of arguments of function f. The *moral graph* of a directed graph is the undirected graph obtained by connecting the parent nodes of each variable and eliminating direction.

Belief updating. The *belief updating* problem defined over a belief network (also referred to as *probabilistic inference*) is the task of computing the posterior probability $P(Y|e)$ of *query* nodes $Y \subseteq X$ given evidence e. We will focus on two cases:

1. when Y consists of a single variable X_i; namely on computing $Bel(X_i) = P(X_i = x|e),\ \forall X_i \in X,\ \forall x \in D_i$;
2. when Y consists of the scope of an original CPT; in this case, we compute $Bel(F_i) = P(F_i = t|e),\ \forall F_i\ family\ in\ \mathcal{B},\ \forall t \in \times_{X_i \in F_i} D_i$.

Extensions of our algorithms to computing the updated belief in the general case when Y is an arbitrary subset of X can be done by conditioning, and will be discussed later.

1.3 Constraint networks

DEFINITION **1.3.1 (constraint network)** *A* constraint network $\mathcal{R} = (X, D, C)$ *is defined over a set of variables* $X = \{X_1, ..., X_n\}$*, their respective domains of values* $D = \{D_1, ..., D_n\}$ *and a set of constraints* $C = \{C_1, ..., C_t\}$*. Each constraint is a pair* $C_i = (S_i, R_i)$*, where* $S_i \subseteq X$ *is the scope of the relation* R_i *defined over* S_i*, denoting the allowed combinations of values.*

In a *binary constraint network* each constraint is defined over pairs of variables X_i and X_j, denoted R_{ij}. The primary query over constraint networks is to determine if there exists a solution, namely, an assignment $x = (x_1, \ldots, x_n)$ to all the variables that satisfies

[1] Also abbreviated $< G, P >$ when X and D are clear.

all the constraints (i.e. $\forall i, x_{S_i} \in R_i$), and if so, to find one. A constraint networks can be associated with a constraint graph where each node represents a variable, and any two variables appearing in the same constraint's scope are connected. We say that a constraint network $\mathcal{R}$ represents its set of all solutions $sol(\mathcal{R})$.

Chapter 2

Past Work

This chapter contains most of our past work, which started by extending an anytime scheme to general tree-decompositions and then moved towards making it iterative. Section 2.1 contains the Mini-Clustering (MC) algorithm, which is inspired by Mini-Buckets algorithm [Dechter & Rish1997]. It is a message-passing algorithm guided by a user adjustable parameter called *i-bound*, offering a flexible tradeoff between accuracy and efficiency in anytime style (in general the higher the i-bound, the better the accuracy). MC algorithm operates on a tree-decomposition, and similar to Pearl's belief propagation algorithm [Pearl1988] it converges in two passes, up and down the tree. We were motivated by the success of Iterative Belief Propagation (IBP) in trying to make MC benefit from the apparent virtues of iterating. The resulting algorithm, Iterative Join-Graph Propagation (IJGP) is described in Section 2.2. IJGP is again a messages-passing algorithm, but it operates on a general join-graph decomposition which may contain cycles. It still provides a user adjustable *i-bound* that defines the maximum cluster size of the graph (and hence the complexity), so it is both anytime and iterative. Since both MC and IJGP are approximate schemes, empirical results on various classes of problems are included, shedding light on their average case performance.

2.1 Mini-Clustering

In this section we present a parameterized anytime approximation scheme for probabilistic inference called *Mini-Clustering (MC)*, which extends the partition-based approximation offered by mini-bucket elimination [Dechter & Rish1997], to general tree decompositions. The benefit of this algorithm is that all single-variable beliefs are computed (approximately) at once, using a two-phase message-passing process along the cluster tree. The resulting approximation scheme allows adjustable levels of accuracy and efficiency, in anytime style. Empirical evaluation against competing algorithms such as Iterative Belief Propagation and Gibbs Sampling demonstrates the potential of the Mini-Clustering approximation scheme: on several classes of problems (e.g. random noisy-or, grid networks and CPCS networks)

Mini-Clustering exhibited superior performance. A similar scheme was presented in a general way in [Kask2001], and for optimization tasks in [Dechter, Kask, & Larrosa2001]. Here we adapt the scheme for the specific task of belief updating, and present the first empirical evaluation for this specific task, showing its effectiveness.

2.1.1 Tree-decomposition schemes

We will describe our algorithms relative to a unifying tree-decomposition framework based on [Gottlob, Leone, & Scarcello2000]. It generalizes tree-decompositions to include join-trees, bucket-trees and other variants applicable to both constraint processing and probabilistic inference.

DEFINITION **2.1.1 (tree-decomposition, cluster tree)** *Let $BN =< X, D, G, P >$ be a belief network. A* tree-decomposition *for BN is a triple $< T, \chi, \psi >$, where $T = (V, E)$ is a tree, and χ and ψ are labeling functions which associate with each vertex $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq P$ satisfying:*

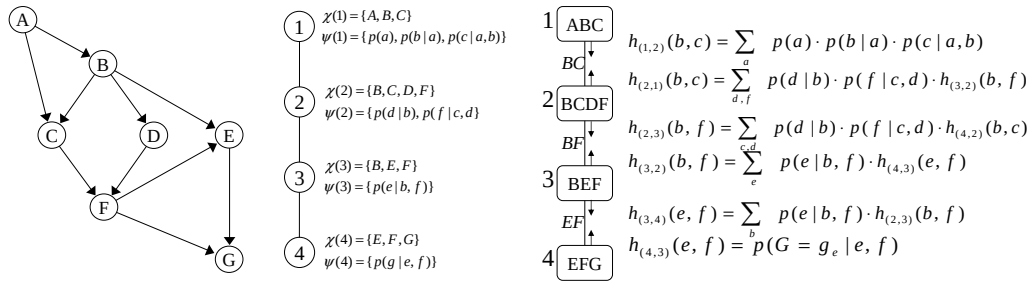
1. *For each function $p_i \in P$, there is* exactly *one vertex $v \in V$ such that $p_i \in \psi(v)$, and $scope(p_i) \subseteq \chi(v)$.*
2. *For each variable $X_i \in X$, the set $\{v \in V | X_i \in \chi(v)\}$ induces a connected subtree of T. This is also called the running intersection property.*

We will often refer to a node and its functions as a cluster *and use the term* tree-decomposition *and* cluster tree *interchangeably.*

DEFINITION **2.1.2 (tree-width, hyper-width, separator, eliminator)** *The* tree-width *[Arnborg1985] of a tree-decomposition $< T, \chi, \psi >$ is $max_{v \in V} |\chi(v)|$, and its* hyper-width *is $max_{v \in V} |\psi(v)|$. Given two adjacent vertices u and v of a tree-decomposition, the* separator *of u and v is defined as $sep(u, v) = \chi(u) \cap \chi(v)$, and the* eliminator *of u with respect to v is $elim(u, v) = \chi(u) - \chi(v)$.*

2.1.1.1 Join-trees and cluster-tree-elimination

In both Bayesian network and constraint satisfaction communities, the most used tree decomposition method is called join-tree decomposition [Lauritzen & Spiegelhalter1988, Dechter & Pearl1989]. Such decompositions can be generated by embedding the network's moral graph, G, in a chordal graph, often using a triangulation algorithm and using its maximal cliques as nodes in the join-tree. The triangulation algorithm assembles a join-tree by connecting the maximal cliques in the chordal graph in a tree. Subsequently, every CPT p_i is placed in one clique containing its scope. Using the previous terminology, a join-tree decomposition of a belief network (G, P) is a tree $T = (V, E)$, where V is the set of cliques of a chordal graph G' that contains G, and E is a set of edges that form a tree between cliques, satisfying the running intersection property [Maier1983]. Such a join-tree



(a) (b) (c)

Figure 2.1: a) A belief network; b) A join-tree decomposition; c)Execution of CTE-BU; no individual functions appear in this case

satisfies the properties of tree-decomposition, therefore our derivation using cluster trees is immediately applicable to join-trees.

There are a few variants for processing join-trees for belief updating [Jensen, Lauritzen, & Olesen1990, Shafer & Shenoy1990]. The variant which we use here, (similar to [Dechter, Kask, & Larrosa2001]), called cluster-tree-elimination (CTE) is applicable to tree-decompositions in general and is geared toward space savings. It is a message passing algorithm (either two-phase message passing, or in asynchronous mode). , where messages are computed by summation over the eliminator between the two clusters of the product of functions in the originating cluster. Algorithm CTE for belief updating denoted CTE-BU is given in Figure 2.12. The algorithm pays a special attention to the processing of observed variables since the presence of evidence is a central component in belief updating. When a cluster sends a message to a neighbor, the algorithm operates on all the functions in the cluster except the message from that particular neighbor. The message contains a single *combined* function and *individual* functions that do not share variables with the relevant eliminator. All the non-individual functions are *combined* in a product and summed over the eliminator.

Example 2.1.1 *Figure 2.1 describes a belief network (a) and a join-tree decomposition for it (b). Figure 2.1c shows the trace of running CTE-BU. In this case no individual functions appear between any of the clusters. To keep the figure simple, we only show the combined functions $h_{(u,v)}$ (each of them being in fact the only element of the set $H_{(u,v)}$ that represents the corresponding message between clusters u and v).*

THEOREM **2.1.2 (Complexity of CTE-BU)** *[Dechter, Kask, & Larrosa2001, Kask2001] The time complexity of CTE-BU is $O(deg \cdot (n + N) \cdot d^{w^*+1})$ and the space complexity is $O(N \cdot d^{sep})$, where deg is the maximum degree of a node in the tree, n is the number of variables, N is the number of nodes in the tree decomposition, d is the maximum domain size of a variable, w^* is the tree-width and sep is the maximum separator size.*

Algorithm **CTE for Belief-Updating (CTE-BU)**
Input: A tree decomposition $< T, \chi, \psi >$, $T = (V, E)$ for $BN =< X, D, G, P >$. Evidence variables $var(e)$.
Output: An augmented tree whose nodes are clusters containing the original CPTs and the messages received from neighbors. $P(X_i, e)$, $\forall X_i \in X$.

Denote by $H_{(u,v)}$ the message from vertex u to v, $ne_v(u)$ the neighbors of u in T excluding v.
$cluster(u) = \psi(u) \cup \{H_{(v,u)} | (v, u) \in E\}$.
$cluster_v(u) = cluster(u)$ excluding message from v to u.

• **Compute messages:**
For every node u in T, once u has received messages from all $ne_v(u)$, compute message to node v:

1. **Process observed variables:**
   Assign relevant evidence to all $p_i \in \psi(u)$
2. **Compute the combined function:**

$$h_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in A} f.$$

   where A is the set of functions in $cluster_v(u)$ whose scope intersects $elim(u, v)$.
   Add $h_{(u,v)}$ to $H_{(u,v)}$ and add all the individual functions in $cluster_v(u) - A$
   Send $H_{(u,v)}$ to node v.

• **Compute** $P(X_i, e)$**:**
For every $X_i \in X$ let u be a vertex in T such that $X_i \in \chi(u)$. Compute $P(X_i, e) = \sum_{\chi(u)-\{X_i\}} (\prod_{f \in cluster(u)} f)$

Figure 2.2: Algorithm Cluster-Tree-Elimination for Belief Updating (CTE-BU);

2.1.2 Mini-Clustering for belief updating

The time, and especially the space complexity of CTE-BU renders the algorithm infeasible for problems with large tree-width. In this section we introduce the Mini-Clustering, a partition-based anytime algorithm which computes approximate values or bounds on $P(X_i, e)$ for every variable X_i in the network. It is a natural extension of the mini-bucket idea to tree-decompositions. Rather than computing the mini-bucket approximation n times, one for each variable as would be required by the mini-bucket approach, the algorithm performs an equivalent computation with just two message passings along each arc of the cluster tree. The idea is to partition each cluster into mini-clusters having at most i variables, where i is an accuracy parameter. Node u partitions its cluster into p mini-

Procedure **MC for Belief Updating (MC-BU(i))**

2. **Compute the combined mini-functions:**

   Make an (i)-size mini-clusters partitioning of $cluster_v(u)$, $\{mc(1), \ldots, mc(p)\}$;

   $h^1_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in mc(1)} f$

   $h^i_{(u,v)} = \max_{elim(u,v)} \prod_{f \in mc(i)} f \quad i = 2, \ldots, p$

   add $\{h^i_{(u,v)} | i = 1, \ldots, p\}$ to $H_{(u,v)}$. Send $H_{(u,v)}$ to v.

Compute upper bounds on $P(X_i, e)$:
For every $X_i \in X$ let $u \in V$ be a cluster such that $X_i \in \chi(u)$. Make (i) mini-clusters from $cluster(u)$, $\{mc(1), \ldots, mc(p)\}$; Compute
$(\sum_{\chi(u)-X_i} \prod_{f \in mc(1)} f) \cdot (\prod_{k=2}^{p} \max_{\chi(u)-X_i} \prod_{f \in mc(k)} f)$.

Figure 2.3: Procedure Mini-Clustering for Belief Updating (MC-BU)

clusters $mc(1), \ldots, mc(p)$. Instead of computing $h_{(u,v)} = \sum_{elim(u,v)} \prod_{k=1}^{p} \prod_{f \in mc(k)} f$ as in CTE-BU, we can compute an upper bound by migrating the summation operator into each mini-cluster. However, this would give $\prod_{k=1}^{p} \sum_{elim(u,v)} \prod_{f \in mc(k)} f$ which is an unnecessarily large upper bound on $h_{(u,v)}$ in which each $\prod_{f \in mc(k)} f$ is bounded by its sum over $elim(u, v)$. Instead, we rewrite $h_{(u,v)} = \sum_{elim(u,v)} (\prod_{f \in mc(1)} f) \cdot (\prod_{i=2}^{p} \prod_{f \in mc(i)} f)$. Subsequently, instead of bounding $\prod_{f \in mc(i)} f, (i \geq 2)$ by summation over the eliminator, we bound it by its maximum over the eliminator, which yields $(\sum_{elim(u,v)} \prod_{f \in mc(1)} f) \cdot \prod_{k=2}^{p} (\max_{elim(u,v)} \prod_{f \in mc(k)} f)$. Therefore, if we are interested in an upper bound, we marginalize one mini-cluster by summation and the others by maximization. Note that the summation in the first mini-cluster must be over *all* variables in the eliminator, even if some of them might not appear in the scope of functions in $mc(1)$.

Consequently, the combined functions are approximated via mini-clusters, as follows. Suppose $u \in V$ has received messages from all its neighbors other than v (the message from v is ignored even if received). The functions in $cluster_v(u)$ that are to be combined are partitioned into mini-clusters $\{mc(1), \ldots, mc(p)\}$, each one containing at most i variables. One of the mini-clusters is processed by summation and the others by maximization over the eliminator, and the resulting combined functions as well as all the individual functions are sent to v.

2.1.2.1 Lower-bounds and mean approximations

We can also derive a lower-bound on beliefs by replacing the max operator with min operator (see above derivation for rationale). This allows, in principle, computing both an

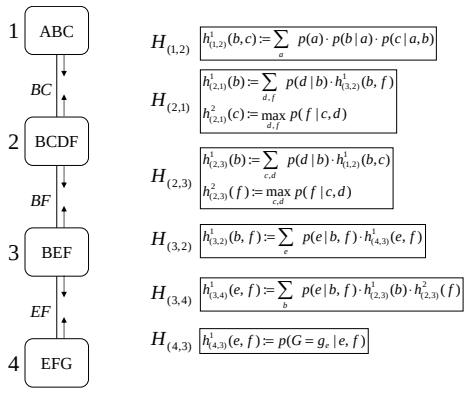


Figure 2.4: Execution of MC-BU for $i = 3$

upper bound and a lower bound on the joint beliefs. Alternatively, if we yield the idea of deriving a bound (and indeed the empirical evaluation encourages that) we can replace max by a $mean$ operator (taking the sum and dividing by the number of elements in the sum), deriving an approximation of the joint belief.

Algorithm MC-BU for upper bounds can be obtained from CTE-BU by replacing step 2 of the main loop and the final part of computing the upper bounds on the joint belief by the procedure given in Figure 2.3.

2.1.2.2 Partitioning strategies

In the implementation we used for the experiments reported here, the partitioning was done in greedy brute-force manner guided only by the sizes of the functions, and the choice of the first mini-cluster for upper bound computation was random. This had the advantage of adding a very small overhead to the Mini-Clustering algorithm. Clearly, more informed schemes that take into account the actual information in the tables of the functions may improve the overall accuracy. We discuss the greedy approach in more detail, as well as new heuristic approaches, that we are currently investigating in Section 3.2.

Example 2.1.3 *Figure 2.4 shows the trace of running MC-BU(3) on the problem in Figure 2.1. First, evidence $G = g_e$ is assigned in all CPTs. There are no individual functions to be sent from cluster 1 to cluster 2. Cluster 1 contains only 3 variables, $\chi(1) = \{A, B, C\}$, therefore it is not partitioned. The combined function $h^1_{(1,2)}(b, c) = \sum_a p(a)\cdot p(b|a)\cdot p(c|a, b)$ is computed and the message $H_{(1,2)} = \{h^1_{(1,2)}(b, c)\}$ is sent to node 2. Now, node 2 can send its message to node 3. Again, there are no individual functions. Cluster 2 contains 4 variables, $\chi(2) = \{B, C, D, F\}$, and a partitioning is necessary: MC-BU(3) can choose $mc(1) = \{p(d|b), h_{(1,2)}(b, c)\}$ and $mc(2) = \{p(f|c, d)\}$. The combined functions $h^1_{(2,3)}(b) = \sum_{c,d} p(d|b) \cdot h_{(1,2)}(b, c)$ and $h^2_{(2,3)}(f) = \max_{c,d} p(f|c, d)$ are computed and the message $H_{(4,3)} = \{h^1_{(2,3)}(b), h^2_{(2,3)}(f)\}$ is sent to node 3. The algorithm continues until every node has received messages from all its neighbors. An upper bound on $p(a, G = g_e)$*

can now be computed by choosing cluster 1*, which contains variable* A*. It doesn't need partitioning, so the algorithm just computes* $\sum_{b,c} p(a) \cdot p(b|a) \cdot p(c|a,b) \cdot h^1_{(2,1)}(b) \cdot h^2_{(2,1)}(c)$*. Notice that unlike CTE-BU which processes 4 variables in cluster 2, MC-BU(3) never processes more than 3 variables at a time.*

2.1.3 Properties of Mini-Clustering

THEOREM **2.1.4** *Algorithm MC-BU(i) with* max *(respectively* min*) computes an upper (respectively lower) bound on the joint probability* $P(X, e)$ *of each variable and each of its values.*

A similar mini-clustering scheme for combinatorial optimization was developed in [Dechter, Kask, & Larrosa2001] having similar performance properties as MC-BU.

THEOREM **2.1.5 (Complexity of MC-BU(**i**))** *[Dechter, Kask, & Larrosa2001] The time and space complexity of MC-BU(i) is* $O(n \cdot hw^* \cdot d^i)$ *where n is the number of variables, d is the maximum domain size of a variable and* $hw^* = max_u |\{f | scope(f) \cap \chi(u) \neq \phi\}|$*, which bounds the number of functions that may travel to a neighboring cluster via message-passing.*

2.1.3.1 Accuracy

For a given i, the accuracy of MC-BU(i) can be shown to be not worse than that of executing the mini-bucket algorithm MB(i) n times, once for each variable (an algorithm that we call nMB(i)). Given a specific execution of MC-BU(i), we can show that for every variable X_i, there exists an ordering of the variables and a corresponding partitioning such that MB(i) computes the same approximation value for $P(X_i, e)$ as does $MC - BU(i)$. In empirical analysis [Kask2001] it is shown that MC-BU has an up to linear speed-up over nMB(i).

2.1.3.2 Normalization

The MC-BU algorithm using max operator computes an upper bound $\overline{P}(X_i, e)$ on the joint probability $P(X_i, e)$. However, deriving a bound on the conditional probability $P(X_i|e)$ is not easy when the exact value of $P(e)$ is not available. If we just try to divide (multiply) $\overline{P}(X_i, e)$ by a constant, the result is not necessarily an upper bound on $P(X_i|e)$. In principle, if we can derive a lower bound $\underline{P}(e)$ on $P(e)$, then we can compute $\overline{P}(X_i, e)/\underline{P}(e)$ as an upper bound on $P(X_i|e)$. However, due to compound error, it is likely to be ineffective. In our empirical evaluation we experimented with normalizing the upper bound as $\overline{P}(X_i, e)/\sum_{X_i} \overline{P}(X_i, e)$ over the values of X_i. The result is not necessarily an upper bound on P($X_i|e$). Similarly, we can also normalize the values when using $mean$ or min operators. It is easy to show that normalization with the $mean$ operator is identical to normalization of MC-BU output when applying the summation operator in all the mini-clusters.

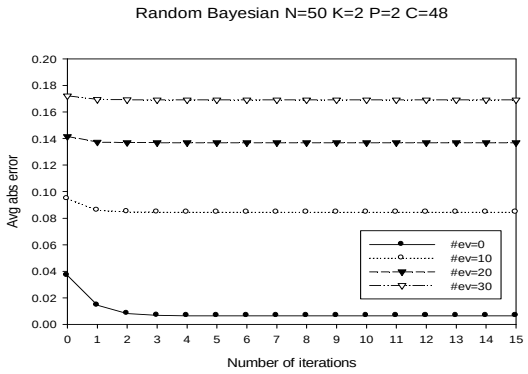


Figure 2.5: Convergence of IBP

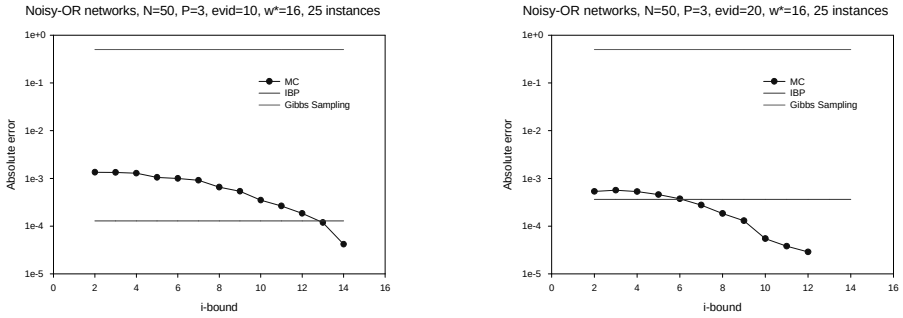


Figure 2.6: Absolute error for noisy-OR networks

2.1.4 Empirical results

We tested the performance of our scheme on random noisy-or networks, random coding networks, general random networks, grid networks, and three benchmark CPCS files with 54, 360 and 422 variables respectively (these are belief networks for medicine, derived from the Computer based Patient Case Simulation system, known to be hard for belief updating). On each type of network we ran Iterative Belief Propagation (IBP) - set to run at most 30 iterations, Gibbs Sampling (GS) and MC-BU(i), with i from 2 to the tree-width w^* to capture the anytime behavior of MC-BU.

We immediately observed that the quality of MC-BU in providing upper or lower bounds on the joint $P(X_i, e)$ was ineffective. Although the upper bound decreases as the accuracy parameter i increases, it still is in most cases greater than 1. Therefore, following the ideas explained in the previous subsection 2.1.3.2 we report the results with normalizing the upper bounds (called max) and normalizing the mean (called $mean$). We notice that MC-BU using the $mean$ operator is doing consistently better.

For noisy-or networks, general random networks, grid networks and for the CPCS networks we computed the exact solution and used three different measures of accuracy: 1. Normalized Hamming Distance (NHD) - We picked the most likely value for each variable

Noisy-OR networks, w*=10

N=50, P=2, 50 instances								
0 |e| 10 20	NHD		Abs. Error		Rel. Error		Time	
	max	mean	max	mean	max	mean	max	mean
		0		9.0E-09		1.1E-05		0.102
IBP		0		3.4E-04		4.2E-01		0.081
		0		9.6E-04		1.2E+00		0.062
	0	0	1.6E-03	1.1E-03	1.9E+00	1.3E+00	0.056	0.057
MC-BU(2)	0	0	1.1E-03	8.4E-04	1.4E+00	1.0E+00	0.048	0.049
	0	0	5.7E-04	4.8E-04	7.1E-01	5.9E-01	0.039	0.039
	0	0	1.1E-03	9.4E-04	1.4E+00	1.2E+00	0.070	0.072
MC-BU(5)	0	0	7.7E-04	6.9E-04	9.3E-01	8.4E-01	0.063	0.066
	0	0	2.8E-04	2.7E-04	3.5E-01	3.3E-01	0.058	0.057
	0	0	3.6E-04	3.2E-04	4.4E-01	3.9E-01	0.214	0.221
MC-BU(8)	0	0	1.7E-04	1.5E-04	2.0E-01	1.9E-01	0.184	0.190
	0	0	3.5E-05	3.5E-05	4.3E-02	4.3E-02	0.123	0.127

Noisy-OR networks, w*=16

N=50, P=3, 25 instances								
10 |e| 20 30	NHD		Abs. Error		Rel. Error		Time	
	max	mean	max	mean	max	mean	max	mean
		0		1.3E-04		7.9E-01		0.242
IBP		0		3.6E-04		2.2E+00		0.184
		0		6.8E-04		4.2E+00		0.121
	0	0	1.3E-03	9.6E-04	8.2E+00	5.8E+00	0.107	0.108
MC-BU(2)	0	0	5.3E-04	4.0E-04	3.1E+00	2.4E+00	0.077	0.077
	0	0	2.3E-04	1.9E-04	1.4E+00	1.2E+00	0.064	0.064
	0	0	1.0E-03	8.3E-04	6.4E+00	5.1E+00	0.133	0.133
MC-BU(5)	0	0	4.6E-04	4.1E-04	2.7E+00	2.4E+00	0.104	0.105
	0	0	2.0E-04	1.9E-04	1.2E+00	1.2E+00	0.098	0.095
	0	0	6.6E-04	5.7E-04	4.0E+00	3.5E+00	0.498	0.509
MC-BU(8)	0	0	1.8E-04	1.8E-04	1.1E+00	1.0E+00	0.394	0.406
	0	0	3.4E-05	3.4E-05	2.1E-01	2.1E-01	0.300	0.308
	0	0	2.6E-04	2.4E-04	1.6E+00	1.5E+00	2.339	2.378
MC-BU(11)	0	0	3.8E-05	3.8E-05	2.3E-01	2.3E-01	1.421	1.439
	0	0	6.4E-07	6.4E-07	4.0E-03	4.0E-03	0.613	0.624
	0	0	4.2E-05	4.1E-05	2.5E-01	2.4E-01	7.805	7.875
MC-BU(14)	0	0	0	0	0	0	2.075	2.093
	0	0	0	0	0	0	0.630	0.638

Table 2.1: Performance on Noisy-OR networks;

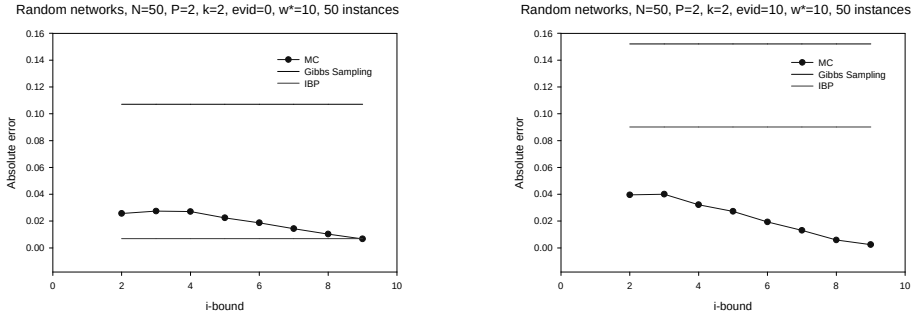


Figure 2.7: Absolute error for random networks

for the approximate and for the exact, took the ratio between the number of disagreements and the total number of variables, and averaged over the number of problems that we ran for each class. 2. Absolute Error (Abs. Error) - is the absolute value of the difference between the approximate and the exact, averaged over all values (for each variable), all variables and all problems. 3. Relative Error (Rel. Error) - is the absolute value of the difference between the approximate and the exact, divided by the exact, averaged over all values (for each variable), all variables and all problems. For coding networks, we report only one measure, Bit Error Rate (BER). In terms of the measures defined above, BER is the normalized Hamming distance between the approximate (computed by an algorithm) and the actual input (which in the case of coding networks may be different from the solution given by exact algorithms), so we denote them differently to make this semantic distinction. We also show the time taken by each algorithm.

In Figure 2.5 we show that IBP converges after about 5 iterations. So, while in our experiments we report its time for 30 iterations, its time is even better when sophisticated termination is used. These results are typical of all runs.

The random noisy-or networks and the random networks were generated using param-

Random networks, w*=10

N=50, P=2, 50 instances								
0 |e| 10 20	NHD		Abs. Error		Rel. Error		Time	
	max	mean	max	mean	max	mean	max	mean
IBP		0.01840		0.00696		0.01505		0.100
		0.19550		0.09022		0.34608		0.080
		0.27467		0.13588		3.13327		0.062
GS		0.50400		0.10715		0.26621		13.023
		0.51400		0.15216		0.57262		12.978
		0.51267		0.18066		4.71805		13.321
MC-BU(2)	0.11400	0.08080	0.03598	0.02564	0.07950	0.05628	0.055	0.055
	0.10600	0.08800	0.04897	0.03957	0.12919	0.10579	0.047	0.048
	0.08667	0.07333	0.04443	0.03639	0.13096	0.10694	0.041	0.042
MC-BU(5)	0.10120	0.06480	0.03392	0.02242	0.07493	0.04937	0.071	0.072
	0.06950	0.05850	0.03254	0.02723	0.08613	0.07313	0.063	0.065
	0.03933	0.03400	0.02022	0.01831	0.05533	0.04984	0.059	0.060
MC-BU(8)	0.05080	0.02680	0.01872	0.01030	0.04103	0.02262	0.216	0.221
	0.01550	0.01450	0.00743	0.00587	0.01945	0.01547	0.178	0.180
	0.00600	0.00400	0.00228	0.00200	0.00597	0.00542	0.129	0.134

Random networks, w*=16

N=50, P=3, 25 instances								
0 |e| 10 20	NHD		Abs. Error		Rel. Error		Time	
	max	mean	max	mean	max	mean	max	mean
IBP		0.03652		0.00907		0.01894		0.298
		0.25200		0.08319		0.22335		0.240
		0.34000		0.13995		0.91671		0.183
MC-BU(2)	0.17304		0.04377		0.09395		0.140	
	0.17600	0.11600	0.05930	0.04558	0.14706	0.11034	0.100	0.103
	0.15067	0.14000	0.07658	0.06683	0.23155	0.19538	0.075	0.078
MC-BU(5)	0.15652		0.04380		0.09398		0.158	
	0.15600	0.11800	0.05665	0.04320	0.13484	0.10221	0.124	0.129
	0.09467	0.09467	0.05545	0.05049	0.15000	0.13706	0.105	0.107
MC-BU(8)	0.16783		0.04166		0.08904		0.602	
	0.09800	0.08100	0.04051	0.03254	0.09923	0.07942	0.481	0.491
	0.05467	0.04533	0.02939	0.02691	0.07865	0.07237	0.385	0.393
MC-BU(11)	0.12087		0.03076		0.06550		2.986	
	0.05500	0.04700	0.02425	0.01946	0.05644	0.04533	2.307	2.345
	0.00800	0.00533	0.00483	0.00431	0.01307	0.01156	1.564	1.585
MC-BU(14)	0.06348		0.01910		0.04071		14.910	
	0.01400	0.01200	0.00542	0.00434	0.01350	0.01108	8.548	8.578
	0.00000	0.00000	0.00089	0.00089	0.00212	0.00211	3.656	3.676

Table 2.2: Performance on random networks;

	$\sigma = .22$		$\sigma = .26$		$\sigma = .32$		$\sigma = .40$		$\sigma = .51$		
BER	max	mean	max	mean	max	mean	max	mean	max	mean	Time
N=100, P=3, 50 instances, w*=7											
IBP	0.000	0.000	0.000	0.000	0.002	0.002	0.022	0.022	0.088	0.088	0.00
GS	0.483	0.483	0.483	0.483	0.483	0.483	0.483	0.483	0.483	0.483	31.36
MC-BU(2)	0.002	0.002	0.004	0.004	0.024	0.024	0.068	0.068	0.132	0.131	0.08
MC-BU(4)	0.001	0.001	0.002	0.002	0.018	0.018	0.046	0.045	0.110	0.110	0.08
MC-BU(6)	0.000	0.000	0.000	0.000	0.004	0.004	0.038	0.038	0.106	0.106	0.12
MC-BU(8)	0.000	0.000	0.000	0.000	0.002	0.002	0.023	0.023	0.091	0.091	0.19
N=100, P=4, 50 instances, w*=11											
IBP	0.000	0.000	0.000	0.000	0.002	0.002	0.013	0.013	0.075	0.075	0.00
GS	0.506	0.506	0.506	0.506	0.506	0.506	0.506	0.506	0.506	0.506	39.85
MC-BU(2)	0.006	0.006	0.015	0.015	0.043	0.043	0.093	0.094	0.157	0.157	0.19
MC-BU(4)	0.006	0.006	0.017	0.017	0.049	0.049	0.104	0.102	0.158	0.158	0.19
MC-BU(6)	0.005	0.005	0.011	0.011	0.035	0.034	0.071	0.074	0.151	0.150	0.29
MC-BU(8)	0.002	0.002	0.004	0.004	0.022	0.022	0.059	0.059	0.121	0.122	0.71
MC-BU(10)	0.001	0.001	0.001	0.001	0.008	0.008	0.033	0.032	0.101	0.102	1.87

Table 2.3: BER for coding networks

eters (N,K,C,P), where N is the number of variables (a square integer for grid networks), K is their domain size (we used only K=2), C is the number of conditional probability matrices and P is the number of parents in each conditional probability matrix. The grid networks have the structure of a square, with edges directed to form a diagonal flow (all parallel edges have the same direction). They were generated by specifying N (a square integer) and K (we used K=2). We also varied the number of evidence nodes, denoted by $|e|$ in the tables. The parameter values are reported in each table.

Comment: We should note that since our evaluation measures are based on comparing against exact figures, we had to restrict the instances to be relatively small or sparse enough to be managed by exact algorithms.

For all the problems, Gibbs sampling performed consistently poorly so we only include part of the results in the following tables and figures.

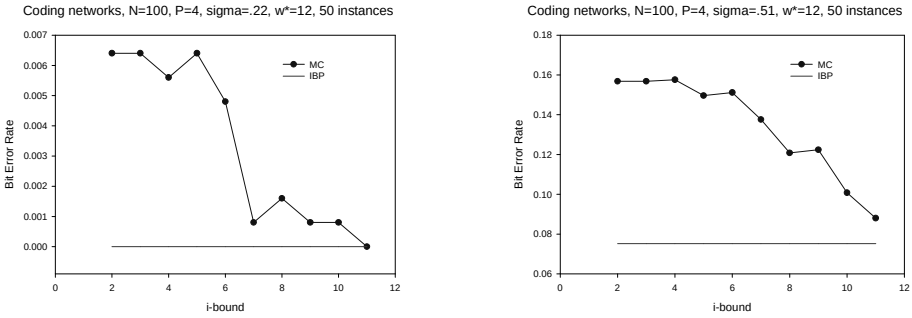


Figure 2.8: BER for coding networks

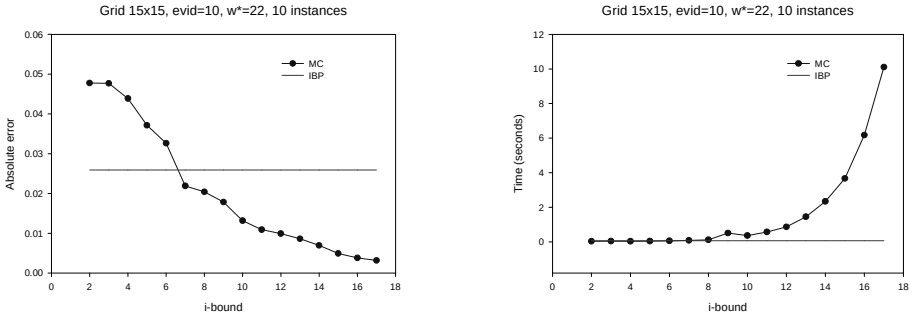


Figure 2.9: Absolute error and time for grid networks

Random noisy-or networks results are summarized in Table 2.1 and Figure 2.6. For NHD, both IBP and MC-BU gave perfect results. For the other measures, we noticed that IBP is more accurate for no evidence by about an order of magnitude. However, as evidence is added, IBP's accuracy decreases, while MC-BU's increases and they give similar results. We also notice that MC-BU gets better as the accuracy parameter i increases, which shows its anytime behavior. We also observed a similar pattern of behavior when experimenting with smaller noisy-or networks, generated with P=2 (w*=10).

General random networks results are summarized in Table 2.2 and Figure 2.7. They are in general similar to those for random noisy-or networks. NHD is non-zero in this case. Again, IBP has the best result only for few evidence variables. It is remarkable how quickly MC-BU surpasses the performance of IBP as evidence is added. We also experimented with larger networks generated with P=3 (w*=16) and observed a similar behavior.

Random coding networks results are given in Table 2.3 and Figure 2.8. The instances fall within the class of linear block codes, (σ is the channel noise level). It is known that IBP is very accurate for this class. Indeed, these are the only problems that we experimented with where IBP outperformed MC-BU throughout. The anytime behavior of MC-BU can again be seen in the variation of numbers in each column.

Grid 13x13, w*=21

N=169, 25 instances, $mean$ operator				
$\|e\|$ = 0, 10, 20, 30	NHD	Abs. Error	Rel. Error	Time
IBP	0.0102	0.0038	0.0083	0.053
	0.0745	0.0323	0.0865	0.047
	0.1350	0.0551	0.3652	0.044
	0.1672	0.0759	0.2910	0.044
GS	0.5172	0.1111	0.2892	6.634
	0.4901	0.1229	0.3393	6.667
	0.5205	0.1316	0.7320	6.787
	0.4921	0.1431	0.5455	6.806
MC-BU(2)	0.1330	0.0464	0.1034	0.044
	0.1263	0.0482	0.1103	0.028
	0.1388	0.0479	0.1117	0.026
	0.1168	0.0513	0.1256	0.024
MC-BU(6)	0.1001	0.0337	0.0731	0.044
	0.0863	0.0313	0.0697	0.040
	0.0805	0.0268	0.0605	0.041
	0.0581	0.0263	0.0610	0.036
MC-BU(10)	0.0402	0.0144	0.0310	0.235
	0.0330	0.0115	0.0252	0.220
	0.0223	0.0092	0.0211	0.206
	0.0224	0.0086	0.0195	0.191
MC-BU(14)	0.0151	0.0056	0.0123	1.246
	0.0151	0.0051	0.0113	1.340
	0.0137	0.0044	0.0101	1.306
	0.0124	0.0032	0.0073	1.256
MC-BU(17)	0.0088	0.0027	0.0059	6.916
	0.0045	0.0018	0.0040	5.889
	0.0030	0.0010	0.0022	5.219
	0.0023	0.0008	0.0018	4.354

Grid 15x15, w*=22

N=225, 10 instances, $mean$ operator				
$\|e\|$ = 0, 10, 20, 30	NHD	Abs. Error	Rel. Error	Time
	0.0094	0.0037	0.0080	0.071
IBP	0.0665	0.0665	0.0761	0.070
	0.1205	0.0463	0.1894	0.068
	0.1462	0.0632	0.1976	0.062
	0.5178	0.1096	0.2688	9.339
GS	0.5047	0.5047	0.3200	9.392
	0.4849	0.1232	0.4009	9.524
	0.4692	0.1335	0.4156	9.220
	0.1256	0.0474	0.1071	0.049
MC-BU(2)	0.1312	0.1312	0.1070	0.041
	0.1371	0.0523	0.1205	0.042
	0.1287	0.0512	0.1201	0.053
	0.1050	0.0356	0.0775	0.217
MC-BU(6)	0.0944	0.0944	0.0720	0.064
	0.0844	0.0313	0.0701	0.059
	0.0759	0.0286	0.0652	0.120
	0.0406	0.0146	0.0313	0.500
MC-BU(10)	0.0358	0.0358	0.0288	0.368
	0.0337	0.0122	0.0272	0.484
	0.0256	0.0116	0.0265	0.468
	0.0233	0.0081	0.0173	2.315
MC-BU(14)	0.0209	0.0209	0.0152	2.342
	0.0146	0.0055	0.0126	2.225
	0.0118	0.0046	0.0105	2.350
	0.0089	0.0031	0.0065	10.990
MC-BU(17)	0.0116	0.0116	0.0069	10.105
	0.0063	0.0022	0.0048	9.381
	0.0036	0.0017	0.0038	9.573

Table 2.4: Performance on grid networks;

N=54, 50 instances								
0	NHD		Abs. Error		Rel. Error		Time	
$\|e\|$ 10								
20	max	mean	max	mean	max	mean	max	mean
		0.01852		0.00032		0.00064		2.450
IBP		0.15727		0.03307		0.07349		2.191
		0.20765		0.05934		0.14202		1.561
		0.49444		0.07797		0.18034		17.247
GS		0.51409		0.09002		0.21298		17.208
		0.48706		0.10608		0.26853		17.335
	0.16667	0.07407	0.02722	0.01221	0.05648	0.02520	0.154	0.153
MC-BU(2)	0.11636	0.07636	0.02623	0.01843	0.05581	0.03943	0.096	0.095
	0.10529	0.07941	0.02876	0.02196	0.06357	0.04878	0.067	0.067
	0.18519	0.09259	0.02488	0.01183	0.05128	0.02454	0.157	0.155
MC-BU(5)	0.10727	0.07682	0.02464	0.01703	0.05239	0.03628	0.112	0.112
	0.08059	0.05941	0.02174	0.01705	0.04790	0.03778	0.090	0.087
	0.12963	0.07407	0.01487	0.00619	0.03047	0.01273	0.438	0.446
MC-BU(8)	0.06591	0.05000	0.01590	0.01040	0.03394	0.02227	0.369	0.370
	0.03235	0.02588	0.00977	0.00770	0.02165	0.01707	0.292	0.294
	0.11111	0.07407	0.01133	0.00688	0.02369	0.01434	2.038	2.032
MC-BU(11)	0.02818	0.01500	0.00600	0.00398	0.01295	0.00869	1.567	1.571
	0.00353	0.00353	0.00124	0.00101	0.00285	0.00236	0.867	0.869

Table 2.5: Performance on CPCS54 network, w*=15

Grid networks results are given in Table 2.4 and Figure 2.9. We only report results with $mean$ operator for a 15x15 grid for which the induced width is w*=22. We notice that IBP is more accurate for no evidence and MC is better as more evidence is added. The same behavior was consistently manifested for smaller grid networks that we experimented with (from 7x7 up to 14x14).

CPCS networks results We also tested on three CPCS benchmark files. The results are given in Tables 2.5 and 2.6 and in Figure 2.10. It is interesting to notice that the MC

CPCS360, w*=20

N=360, 5 instances, $mean$ operator				
$\|e\|$ = 0, 20, 40	NHD	Abs. Error	Rel. Error	Time
IBP	0.0000	0.0027	0.0054	82
	0.0112	0.0256	3.4427	76
	0.0363	0.0629	736.1080	60
MC-BU(8)	0.0056	0.0125	0.0861	16
	0.0041	0.0079	0.0785	14
	0.0113	0.0109	0.2997	9
MC-BU(11)	0.0000	0.0080	0.0636	38
	0.0000	0.0048	0.0604	39
	0.0088	0.0102	0.1733	33
MC-BU(14)	0.0000	0.0030	0.0192	224
	0.0012	0.0045	0.0502	232
	0.0056	0.0070	0.0693	200
MC-BU(17)	0.0000	0.0016	0.0073	1433
	0.0006	0.0026	0.0266	1455
	0.0013	0.0006	0.0045	904

CPCS422, w*=23

N=422, 1 instance, $mean$ operator				
$\|e\|$ = 0, 20, 40	NHD	Abs. Error	Rel. Error	Time
IBP	0.0024	0.0062	0.0150	2838
	0.0721	0.0562	7.5626	2367
	0.0654	0.0744	37.5096	2150
MC-BU(3)	0.0687	0.0455	1.4341	161
	0.0373	0.0379	0.9792	85
	0.0366	0.0233	2.8384	48
MC-BU(7)	0.0545	0.0354	0.1531	146
	0.0249	0.0253	0.3112	77
	0.0262	0.0164	0.5781	45
MC-BU(11)	0.0166	0.0175	0.0738	152
	0.0448	0.0352	0.6113	95
	0.0340	0.0237	0.6978	63
MC-BU(15)	0.0024	0.0039	0.0145	526
	0.0398	0.0278	0.5338	564
	0.0183	0.0113	0.5248	547

Table 2.6: Performance on CPCS360 and CPCS422 networks

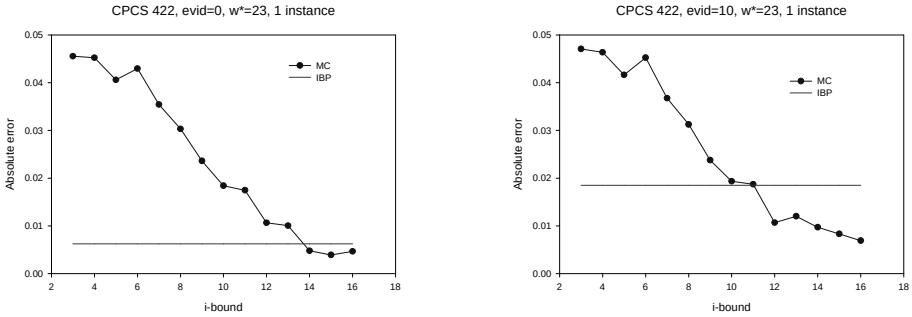


Figure 2.10: Absolute error for CPCS422

scheme scales up even to fairly large networks, like the real life example of CPCS422 (induced width 23). IBP is again slightly better for no evidence, but is quickly surpassed by MC when evidence is added.

2.1.5 Discussion

We presented in this chapter an approximation scheme for probabilistic inference, one of the most important task over belief networks. The scheme, called Mini-Clustering, is governed by a controlling parameter that allows adjustable levels of accuracy and efficiency in an anytime style.

We presented empirical evaluation of mini-cluster approximation on several classes of networks, comparing its anytime performance with competing algorithms such as Gibbs Sampling and Iterative Belief Propagation, over benchmarks of noisy-or random networks, general random networks, grid networks, coding networks and CPCS type networks. Our results show that, as expected, IBP is superior to all other approximations for coding networks. However, for random noisy-or, general random networks, grid networks and the CPCS networks, in the presence of evidence, the mini-clustering scheme is often superior even in its weakest form. Gibbs sampling was particularly bad and we believe that enhanced variants of Monte Carlo approach, such as likelihood weighting and importance

sampling, should be compared with [Cheng & Druzdzel2000]. The empirical results are particularly encouraging as we use an unoptimized scheme that exploits a universal principle applicable to many reasoning tasks. Our contribution beyond recent works in this area [Dechter & Rish1997, Dechter, Kask, & Larrosa2001] is in: 1. Extending the partition-based approximation for belief updating from mini-buckets to general tree-decompositions, thus allowing the computation of the updated beliefs for all the variables at once. This extension is similar to the one proposed in [Dechter, Kask, & Larrosa2001] but replaces optimization with probabilistic inference. 2. Providing for the first time empirical evaluation demonstrating the effectiveness of the partition-based idea for belief updating.

There are many potential ways for improving the MC scheme. Among the most important, the partitioning step can be further elaborated, and we discuss such approaches in 3.2. In the work presented here, we used only a brute-force approach for partitioning.

One extension of this work [Dechter, Mateescu, & Kask2002] is an iterative version of MC called Iterative Join-Graph Propagation (IJGP), which is both anytime and iterative and belongs to the class of generalized belief propagation methods [Yedidia, Freeman, & Weiss2001]. Rather than assuming an underlying join-tree, IJGP works on a join-graph that may contain loops. IJGP is related to MC in a similar way as IBP is related to BP (Pearl's belief propagation). Experimental work shows that in most cases iterating improves the quality of the MC approximation even further, especially for low i-bounds. We will discuss this algorithm in detail in Section 2.2.

2.2 Iterative Join-Graph Propagation

This section contains our work on Iterative Join-Graph Propagation. The original motivation for designing this algorithm was in trying to combine the anytime feature of Mini-Clustering (MC) and the iterative virtues of Iterative Belief Propagation (IBP). MC is an anytime algorithm but it works on tree-decompositions and it converges in two passes, so iterating doesn't change the messages. IBP is an iterative algorithm that converges in most cases, and when it converges it does so very fast. Allowing it more time doesn't improve the accuracy. IJGP was designed to benefit from both these directions. It works on a general join-graph which may contain cycles. The cluster size of the graph is user adjustable by the *i-bound* (providing the anytime nature), and the cycles in the graph allow iterating. The precise mechanics of the algorithm are given in the following sections. Empirical results are also provided, showing that in many cases IJGP is superior to both MC and IBP on several classes of problems.

2.2.1 Join-Graphs

We will describe our algorithms relative to a join-graph decomposition framework using recent notation proposed by [Gottlob, Leone, & Scarcello2000]. The notion of join-tree decompositions was introduced in relational databases [Maier1983].

DEFINITION **2.2.1 (join-graph decompositions)** *A* join-graph decomposition *for* $BN =< X, D, G, P >$ *is a triple* $D =< JG, \chi, \psi >$*, where* $JG = (V, E)$ *is a graph, and* χ *and* ψ *are labeling functions which associate with each vertex* $v \in V$ *two sets,* $\chi(v) \subseteq X$ *and* $\psi(v) \subseteq P$ *such that:*

1. *For each* $p_i \in P$*, there is* exactly *one vertex* $v \in V$ *such that* $p_i \in \psi(v)$*, and* $scope(p_i) \subseteq \chi(v)$*.*
2. *(connectedness) For each variable* $X_i \in X$*, the set* $\{v \in V | X_i \in \chi(v)\}$ *induces a connected subgraph of* G*. The connectedness requirement is also called the running intersection property.*

We will often refer to a node and its CPT functions as a *cluster*[1] and use the term *join-graph-decomposition* and *cluster graph* interchangeably. A *join-tree-decomposition* or a *cluster tree* is the special case when the join-graph *JG* is a tree.

2.2.1.1 Join-tree propagation

The well known join-tree clustering algorithm first converts the belief network into a cluster tree and then sends messages between clusters. We call the second message passing phase

[1]Note that a node may be associated with an empty set of CPTs

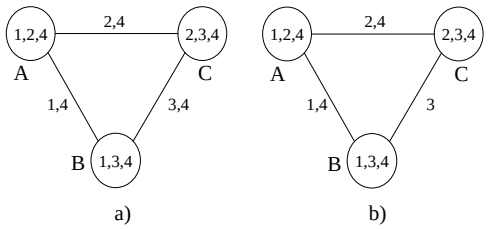


Figure 2.11: An arc-labeled decomposition

join-tree propagation. The complexity of join-tree clustering is exponential in the number of variables in a cluster (tree-width), and the number of variables in the intersections between adjacent clusters (separator-width), as defined below.

DEFINITION **2.2.2 (tree-width, separator-width)** *Let $D =< JT, \chi, \psi >$ be a tree decomposition of a belief network $< G, P >$. The* tree-width *of D [Arnborg1985] is $max_{v \in V} |\chi(v)|$. The tree-width of $< G, P >$ is the minimum tree-width over all its join-tree decompositions. Given two adjacent vertices u and v of JT, the* separator *of u and v is defined as $sep(u, v) = \chi(u) \cap \chi(v)$, and the* separator-width *is $max_{(u,v)} |sep(u, v)|$.*

The minimum tree-width of a graph G can be shown to be identical to a related parameter called *induced-width*. A join-graph decomposition D is *arc-minimal* if none of its arcs can be removed while still satisfying the connectedness property of Definition 2.2.1. If a graph-decomposition is not arc-minimal it is easy to remove some of its arcs until it becomes arc-minimal. In our preliminary experiments we observed immediately that when applying join-tree propagation on a join-graph iteratively, it is crucial to avoid cycling messages relative to every single variable. The property of arc-minimality is *not* sufficient to ensure such acyclicity though. What is required is that, for every node X, the arc-subgraph that contains X be a tree.

Example 2.2.1 *The example in Figure 2.11a shows an arc minimal join-graph which contains a cycle relative to variable 4, with arcs labeled with separators. Notice however that if we remove variable 4 from the label of one arc we will have no cycles (relative to single variables) while the connectedness property will still be maintained.*

To allow more flexible notions of connectedness we refine the definition of join-graph decompositions, when arcs can be labeled with a subset of their separator.

DEFINITION **2.2.3 ((minimal) arc-labeled join-graph decompositions)** *An* arc-labeled decomposition *for $BN =< X, D, G, P >$ is a four-tuple $D =< JG, \chi, \psi, \theta >$, where $JG = (V, E)$ is a graph, χ and ψ associate with each vertex $v \in V$ the sets $\chi(v) \subseteq X$ and $\psi(v) \subseteq P$ and θ associates with each edge $(v, u) \subset E$ the set $\theta((v, u)) \subseteq X$ such that:*

1. *For each function $p_i \in P$, there is* exactly *one vertex $v \in V$ such that $p_i \in \psi(v)$, and $scope(p_i) \subseteq \chi(v)$.*

2. *(arc-connectedness) For each arc* (u, v), $\theta(u, v) \subseteq sep(u, v)$, *such that* $\forall X_i \in X$, *any two clusters containing* X_i *can be connected by a path whose every arc's label includes* X_i.

Finally, an arc-labeled join-graph is minimal *if no variable can be deleted from any label while still satisfying the arc-connectedness property.*

DEFINITION **2.2.4 (separator, eliminator)** *Given two adjacent vertices* u *and* v *of* JG, *the* separator *of* u *and* v *is defined as* $sep(u, v) = \theta((u, v))$, *and the* eliminator *of* u *with respect to* v *is* $elim(u, v) = \chi(u) - \theta((u, v))$.

Arc-labeled join-graphs can be made minimal by deleting variables from the labels. It is easy to see that a *minimal arc-labeled join-graph* does not contain any cycle relative to any single variable. That is, any two clusters containing the same variable are connected by exactly one path labeled with that variable.

2.2.2 Algorithm IJGP

Applying join-tree propagation iteratively to join-graphs yields algorithm *Iterative Join-Graph Propagation (IJGP)* described in Figure 2.12. One iteration of the algorithm applies message-passing in a topological order over the join-graph, forward and back.

When node i sends a message (or messages) to a neighbor node j it operates on all the CPTs in its cluster and on all the messages sent from its neighbors excluding the ones received from j. First, all individual functions that share no variables with the eliminator are collected and sent to j. All the rest of the functions are *combined* in a product and summed over the eliminator between i and j.

It is known that:

THEOREM **2.2.2** 1. *[Lauritzen & Spiegelhalter1988] If* $IJGP$ *is applied to a join-tree decomposition it reduces to join-tree clustering and it therefore is guaranteed to compute the exact beliefs in one iteration.*

2. *[Larrosa, Kask, & Dechter2001] The time complexity of one iteration of IJGP is* $O(deg \cdot (n + N) \cdot d^{w^*+1})$ *and its space complexity is* $O(N \cdot d^{\theta})$, *where deg is the maximum degree of a node in the join-graph, n is the number of variables, N is the number of nodes in the graph decomposition, d is the maximum domain size,* w^* *is the maximum cluster size and* θ *is the maximum label size.*

However, when applied to a join-graph the algorithm is neither guaranteed to converge nor to find the exact posterior.

Proof. The number of cliques in the chordal graph G' corresponding to G is at most n, so the number of nodes in the join-tree is at most n. The complexity of processing a node u in the join-tree is $deg_u \cdot (|\psi(u)| + deg_u - 1) \cdot d^{|\chi(u)|}$, where deg_u is the degree of u. By bounding

Algorithm **Iterative Join Graph Propagation (IJGP)**

Input An arc-labeled join-graph decomposition $< JG, \chi, \psi, \theta >, JG = (V, E)$ for $BN =< X, D, G, P >$. Evidence variables $var(e)$.

Output An augmented graph whose nodes are clusters containing the original CPTs and the messages received from neighbors. Approximations of $P(X_i|e),\ \forall X_i \in X$.

Denote by $h_{(u,v)}$ the message from vertex u to v, $ne_v(u)$ the neighbors of u in JG excluding v.
$cluster(u) = \psi(u) \cup \{h_{(v,u)}|(v,u) \in E\}$.
$cluster_v(u) = cluster(u)$ excluding message from v to u.

• **One iteration of IJGP:**
For every node u in JG in some topological order d and back,do

1. **Process observed variables:**
   Assign relevant evidence to all $p_i \in \psi(u)$ $\chi(u) := \chi(u) - var(e),\ \ \forall u \in V$
2. **Compute individual functions:**
   Include in $H_{(u,v)}$ each function in $cluster_v(u)$ whose scope does not contain variables in $elim(u, v)$. Denote by A the remaining functions.
3. **Compute and send to v the combined function:** $h_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in A} f$.
   Send $h_{(u,v)}$ and the individual functions $H_{(u,v)}$ to node v.

Endfor
• **Compute $P(X_i, e)$:**

For every $X_i \in X$ let u be a vertex in T such that $X_i \in \chi(u)$.
Compute $P(X_i|e) = \alpha \sum_{\chi(u)-\{X_i\}} (\prod_{f \in cluster(u)} f)$

Figure 2.12: Algorithm Iterative Join-Graph Propagation (IJGP)

deg_u by deg, $|\psi(u)|$ by n and $\chi(u)$ by $w^* + 1$ and knowing that $deg < N$, by summing over all nodes, we can bound the entire time complexity by $O(deg \cdot (n + N) \cdot d^{w^*+1})$.

For each edge JTC records functions. Since the number of edges in bounded by n and the size of each message is bounded by d^{sep} we get space complexity of $O(n \cdot d^{sep})$. □

2.2.3 I-mappness of arc-labeled join-graphs

The success of IJGP, no doubt, will depend on the choice of cluster graphs it operates on. The following paragraphs provide some rationale to our choice of minimal arc-labeled join-graphs. First, we are committed to the use of an underlying graph structure that captures as many of the distribution independence relations as possible, without introducing new ones. That is, we restrict attention to cluster graphs that are I-maps of P [Pearl1988]. Second, we wish to avoid cycles as much as possible in order to minimize computational over-counting.

Indeed, it can be shown that any join-graph of a belief network is an I-map of the underlying probability distribution relative to node-separation. It turns out that arc-labeled

join-graphs display a richer set of independencies relative to arc-separation.

DEFINITION **2.2.5 (arc-separation in (arc-labeled) join-graphs)** *Let $D =< JG, \chi, \psi, \theta >$, $JG = (V, E)$ be an arc-labeled decomposition. Let $N_W, N_Y \subseteq V$ be two sets of nodes, and $E_Z \subseteq E$ be a set of edges in JG. Let W, Y, Z be their corresponding sets of variables ($W = \cup_{v \in N_W} \chi(v)$, $Z = \cup_{e \in E_Z} \theta(e)$). E_Z* arc-separates *N_W and N_Y in D if there is no path between N_W and N_Y in the graph JG with the edges in E_Z removed. In this case we also say that W is* separated *from Y given Z in D, and write $< W|Z|Y >_D$. Arc-separation in a regular join-graph is defined relative to its separators.*

THEOREM **2.2.3** *Any arc-labeled join-graph decomposition $D =< JG, \chi, \psi, \theta >$ of a belief network $BN =< X, D, G, P >$ is an I-map of P relative to arc-separation.*

Proof. Let MG be the moral graph of BN. Since MG is an I-map of P, it is enough to prove that JG is and I-map of MG.

Let N_W, N_Z, N_Y be three disjoint set of nodes in JG, and W, Z, Y be their corresponding sets of variables in MG. We will prove:

$$< N_W|N_Z|N_Y >_{JG} \Longrightarrow < W|Z|Y >_{MG}$$

by contradiction.

Since the sets W, Z, Y may not be disjoint, we will actually prove that $< W - Z|Z|Y - Z >_G$ holds, this being equivalent to $< W|Z|Y >_G$.

Supposing $< W - Z|Z|Y - Z >_{MG}$ is false, then there exists a path $\alpha = \gamma_1, \gamma_2, \ldots, \gamma_{n-1}, \beta = \gamma_n$ in MG that goes from some variable $\alpha = \gamma_1 \in W - Z$ to some variable $\beta = \gamma_n \in Y - Z$ without intersecting variables in Z.

Let N_v be the set of all nodes in JG that contain variable $v \in X$, and let's consider the set of nodes:

$$S = \cup_{i=1}^{n} N_{\gamma_i} - N_Z$$

We argue that S forms a connected sub-graph in JG.

First, the running intersection property ensures that every $N_{\gamma_i}, i = 1, \ldots, n$, remains connected in JG after pulling out the nodes in N_Z (otherwise, it must be that there was a path between the two disconnected parts in the original JG, which implies that a γ_i is part of Z, which is a contradiction).

Second, the fact that $(\gamma_i, \gamma_{i+1}), i = 1, \ldots, n-1$, is an edge in the moral graph MG implies that there is a conditional probability table (CPT) on both γ_i and $\gamma_{i+1}, i = 1, \ldots, n-1$ (and perhaps other variables). From property 1 of the definition of the join-graph, it follows that for all $i = 1, \ldots, n-1$ there exists a node in JG that contains both γ_i and γ_{i+1}. This proves the existence of a path in the mutilated join-graph (JG with N_Z pulled out) from a node in N_W containing $\alpha = \gamma_1$ to the node containing both γ_1 and γ_2 (N_{γ_1} is connected),

then from that node to the one containing both γ_2 and γ_3 (N_{γ_2} is connected), and so on until we reach a node in N_Y containing $\beta = \gamma_n$.

This shows that $< N_W|N_Z|N_Y >_{JG}$ is false, concluding the proof by contradiction. □

Interestingly however, removing arcs or labels from arc-labeled join-graphs whose clusters are fixed will not increase the independencies captured by arc-labeled join-graphs. That is:

Proposition 1 *Any two (arc-labeled) join-graphs defined on the same set of clusters, sharing (V, χ ψ), express exactly the same set of independencies relative to arc-separation.*

Consequently, all such decomposition are *correct* and are isomorphic I-maps.

THEOREM **2.2.4** *Any arc-labeled join-graph decomposition of a belief network $BN =< X, D, G, P >$ is a minimal I-map of P relative to arc-separation.*

Hence, the issue of minimizing computational over-counting due to cycles appears to be orthogonal to maximizing independencies via minimal I-mappness. Nevertheless, to avoid over-counting as much as possible, we still prefer join-graphs that minimize cycles relative to each variable. That is, we prefer to apply IJGP to *minimal* arc-labeled join-graphs.

2.2.4 Bounded join-graphs

Since we want to control the complexity of IJGP we will define it on decompositions having bounded cluster size. If the number of variables in a cluster is bounded by i, the time and space complexity of one full iteration of IJGP(i) is exponential in i. How can good graph-decompositions of bounded cluster size be generated?

Since we want the join-graph to be as close as possible to a tree, and since a tree has a tree-width 1, we may try to find a join-graph JG of bounded cluster size whose tree-width (as a graph) is minimized. While we will not attempt to optimally solve this task, we will propose one method for generating i-bounded graph-decompositions.

DEFINITION **2.2.6 (external and internal widths)** *Given an arc-labeled join-graph decomposition $D =< JG, \chi, \psi, \theta >$ of a network $< G, P >$, the internal width of D is $max_{v \in V}|\chi(v)|$, while the external width of D is the tree-width of JG as a graph.*

Clearly, if D is a tree-decomposition its external width is 1 and its internal width equals its tree-width. For example, an edge minimal dual decomposition has an internal width equal to the maximum scope of each function, m, and external width w^* which is the tree-width of the moral graph of G. On the other hand, a tree-decomposition has internal width of w^* and external width of 1.

Algorithm **Join-Graph Structuring(*i*)**

1. Apply procedure schematic mini-bucket(*i*).
2. Associate each resulting mini-bucket with a node in the join-graph, the variables of the nodes are those appearing in the mini-bucket, the original functions are those in the mini-bucket.
3. Keep the arcs created by the procedure (called out-edges) and label them by the regular separator.
4. Connect the mini-bucket clusters belonging to the same bucket in a chain by in-edges labeled by the single variable of the bucket.

Figure 2.13: Algorithm Join-Graph Structuring(*i*)

Procedure **Schematic Mini-Bucket(*i*)**

1. Order the variables from X_1 to X_n minimizing (heuristically) induced-width, and associate a bucket for each variable.
2. Place each CPT in the bucket of the highest index variable in its scope.
3. For $j = n$ to 1 do:
   Partition the functions in $bucket(X_j)$ into mini-buckets having at most i variables.
   For each mini-bucket mb create a new scope-function (message) f where $scope(f) = \{X | X \in mb\} - \{X_i\}$ and place scope(f) in the bucket of its highest variable. Maintain an arc between mb and the mini-bucket (created later) of f.

Figure 2.14: Procedure Schematic Mini-Bucket(*i*)

Using this terminology we can now state our target decomposition more clearly. Given a graph G, and a bounding parameter i we wish to find a join-graph decomposition of G whose internal width is bounded by i and whose external width is minimized. The bound i controls the complexity of one iteration of $IJGP$ while the external width provides some measure of its accuracy.

One class of such decompositions is partition-based. It starts from a given tree-decomposition and then partitions the clusters until the decomposition has clusters bounded by i. The opposite approach is grouping-based. It starts from an arc-minimal dual-graph decomposition (where each cluster contains a single CPT) and groups clusters into larger clusters as long as the resulting clusters do not exceed the given bound. In both methods we should attempt to reduce the tree-width of the generated graph-decomposition. Our partition-based approach inspired by the mini-bucket idea [Dechter & Rish1997] is as follows.

Given a bound i, algorithm *join-graph structuring(i)* applies procedure *schematic mini-*

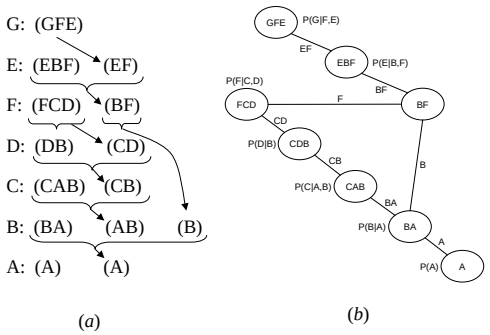


Figure 2.15: Join-graph decompositions

bucket(i), described in Figure 2.14. The procedure only traces the scopes of the functions that would be generated by the full mini-bucket procedure, avoiding actual computation. The algorithm then connects the mini-buckets' scopes minimally to obtain the running intersection property, as described in Figure 2.13.

Example 2.2.5 *Figure 2.15a shows the trace of procedure schematic mini-bucket(3) applied to the problem described in Figure 2.1a. The decomposition in Figure 2.15b is created by the algorithm graph structuring. The only cluster partitioned is that of F into two scopes (FCD) and (BF), connected by an in-edge labeled with F.*

Procedure schematic mini-bucket ends with a collection of trees rooted in mini-buckets of the first variable. Each of these trees is minimally arc-labeled. Then, *in-edges* are labeled with only one variable, and they are added only to obtain the running intersection property between branches of these trees. It can be shown that:

Proposition 2 *Algorithm join-graph structuring(i), generates a minimal arc-labeled join-graph decomposition having bound i.*

Example 2.2.6 *Figure 2.16 shows a range of arc-labeled join-graphs. On the left extreme we have a graph with smaller clusters, but more cycles. This is the type of graph IBP works on. On the right extreme we have a tree decomposition, which has no cycles but has bigger clusters. In between, there could be a number of join-graphs where maximum cluster size can be traded for number of cycles. Intuitively, the graphs on the left present less complexity for IJGP because the cluster size is small, but they are also likely to be less accurate. The graphs on the right side are computationally more complex, because of larger cluster size, but are likely to be more accurate.*

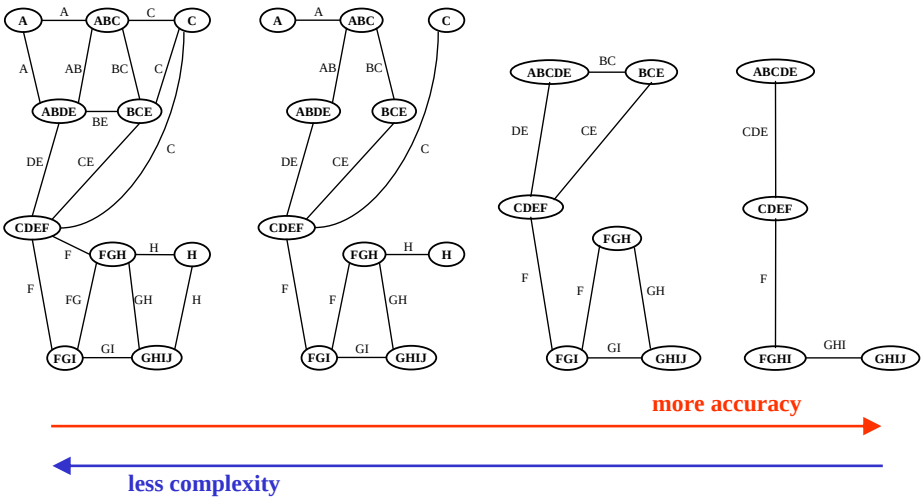


Figure 2.16: Join-graphs

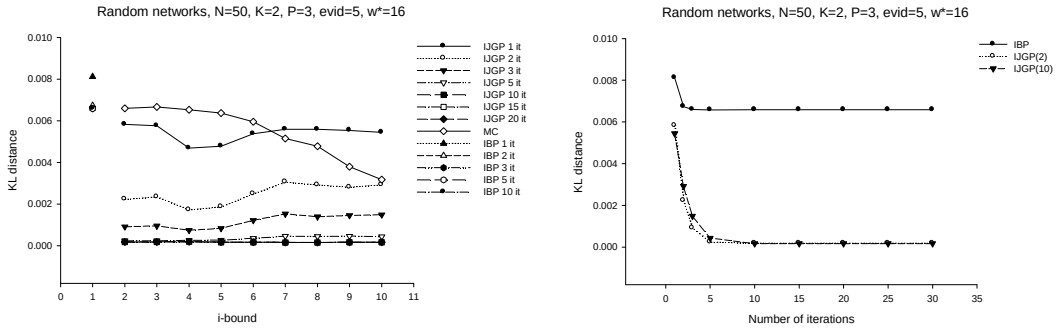
Table 2.7: Random networks: N=50, K=2, C=45, P=3, 100 instances, w*=16

		Absolute error				Relative error				KL distance				Time			
		IBP	IJGP			IBP	IJGP			IBP	IJGP			IBP	IJGP		
#it	#evid		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8
1	0	0.02988	0.03055	0.02623	0.02940	0.06388	0.15694	0.05677	0.07153	0.00213	0.00391	0.00208	0.00277	0.0017	0.0036	0.0058	0.0295
	5	0.06178	0.04434	0.04201	0.04554	0.15005	0.12340	0.12056	0.11154	0.00812	0.00582	0.00478	0.00558	0.0013	0.0040	0.0052	0.0200
	10	0.08762	0.05777	0.05409	0.05910	0.23777	0.18071	0.14278	0.15686	0.01547	0.00915	0.00768	0.00899	0.0013	0.0040	0.0036	0.0121
5	0	0.00829	0.00636	0.00592	0.00669	0.01726	0.01326	0.01239	0.01398	0.00021	0.00014	0.00015	0.00018	0.0066	0.0145	0.0226	0.1219
	5	0.05182	0.00886	0.00886	0.01123	0.12589	0.01967	0.01965	0.02494	0.00658	0.00024	0.00026	0.00044	0.0060	0.0120	0.0185	0.0840
	10	0.08039	0.01155	0.01073	0.01399	0.21781	0.03014	0.02553	0.03279	0.01382	0.00055	0.00042	0.00073	0.0048	0.0100	0.0138	0.0536
10	0	0.00828	0.00584	0.00514	0.00495	0.01725	0.01216	0.01069	0.01030	0.00021	0.00012	0.00010	0.00010	0.0130	0.0254	0.0436	0.2383
	5	0.05182	0.00774	0.00732	0.00708	0.12590	0.01727	0.01628	0.01575	0.00658	0.00018	0.00017	0.00016	0.0121	0.0223	0.0355	0.1639
	10	0.08040	0.00892	0.00808	0.00855	0.21782	0.02101	0.01907	0.02005	0.01382	0.00028	0.00024	0.00029	0.0109	0.0191	0.0271	0.1062
MC	0		0.04044	0.04287	0.03748		0.08811	0.09342	0.08117		0.00403	0.00435	0.00369		0.0159	0.0173	0.0552
	5		0.05303	0.05171	0.04250		0.12375	0.11775	0.09596		0.00659	0.00636	0.00477		0.0146	0.0158	0.0532
	10		0.06033	0.05489	0.04266		0.14702	0.13219	0.10074		0.00841	0.00729	0.00503		0.0119	0.0143	0.0470

MC(i) vs. IJGP(i). As can be hinted by our structuring of a bounded join-graph, there is a close relationship between MC(i) and IJGP(i). In particular, one iteration of IJGP(i) is similar to MC(i) (MC(i) is an algorithm that approximates join-tree clustering and was shown to be competitive with IBP and Gibbs Sampling [Mateescu, Dechter, & Kask2002]). Indeed, while we view IJGP(i) as an iterative version of MC(i), the two algorithms differ in several technical points, some may be superficial, due to implementation, others may be more principled. We will leave the discussion at that and will observe the comparison of the two approaches in the empirical section.

2.2.5 Empirical results

We tested the performance of IJGP(i) on random networks, on M-by-M grids, on two benchmark CPCS files with 54 and 360 variables, respectively (these are belief networks



a) Performance vs. i-bound b) Convergence with iterations

Figure 2.17: Random networks: KL distance

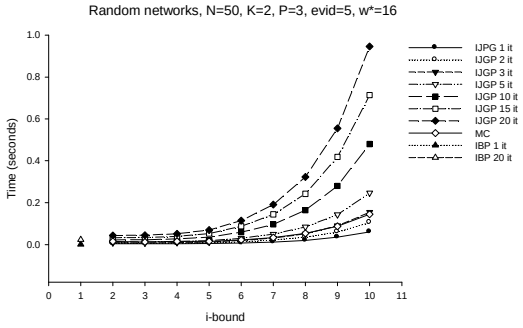


Figure 2.18: Random networks: Time

for medicine, derived from the Computer based Patient Case Simulation system, known to be hard for belief updating) and on coding networks. On each type of networks, we ran Iterative Belief Propagation (IBP), MC(i) and IJGP(i), while giving IBP and IJGP(i) the same number of iterations.

We use the partitioning method described in Section 2.2.4 to construct a join-graph. To determine the order of message computation, we recursively pick an edge (u,v), such that node u has the fewest incoming messages missing.

For each network except coding, we compute the exact solution and compare the accuracy of algorithms using: 1. Absolute error - the absolute value of the difference between the approximate and the exact, averaged over all values, all variables and all problems. 2. Relative error - the absolute value of the difference between the approximate and the exact, divided by the exact, averaged over all values, all variables and all problems. 3. KL distance - $P_{exact}(X = a) \cdot log(P_{exact}(X = a)/P_{approximation}(X = a))$ averaged over all values, all variables and all problems. We also report the time taken by each algorithm. For coding networks we report Bit Error Rate (BER) computed as follows: for each approximate algorithm we pick the most likely value for each variable, take the number of disagreements with the exact input, divide by the total number of variables, and average over all the instances of the problem. We also report time.

Table 2.8: 9x9 grid, K=2, 100 instances, w*=12

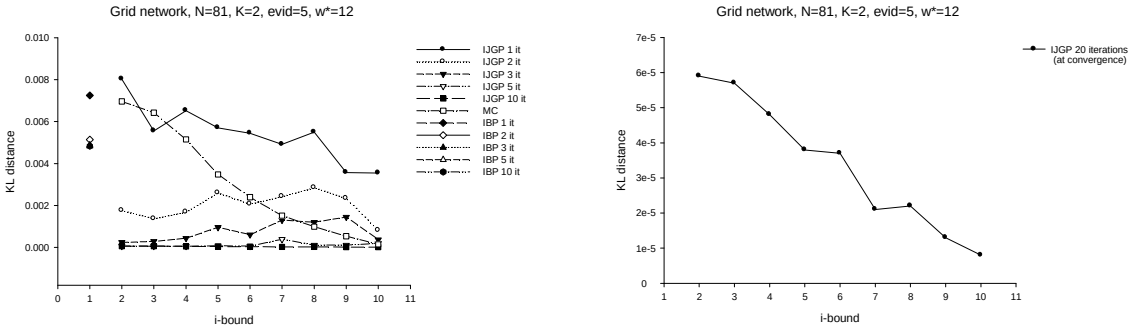
		Absolute error				Relative error				KL distance				Time			
		IBP	IJGP			IBP	IJGP			IBP	IJGP			IBP	IJGP		
#it	#evid		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8
1	0	0.03524	0.05550	0.04292	0.03318	0.08075	0.13533	0.10252	0.07904	0.00289	0.00859	0.00602	0.00454	0.0010	0.0053	0.0106	0.0426
	5	0.05375	0.05284	0.04012	0.03661	0.16380	0.13225	0.09889	0.09116	0.00725	0.00802	0.00570	0.00549	0.0016	0.0041	0.0092	0.0315
	10	0.07094	0.05453	0.04304	0.03966	0.23624	0.14588	0.12492	0.12202	0.01232	0.00905	0.00681	0.00653	0.0013	0.0038	0.0072	0.0256
5	0	0.00358	0.00393	0.00325	0.00284	0.00775	0.00849	0.00702	0.00634	0.00005	0.00006	0.00007	0.00010	0.0049	0.0152	0.0347	0.1462
	5	0.03224	0.00379	0.00319	0.00296	0.11299	0.00844	0.00710	0.00669	0.00483	0.00006	0.00007	0.00010	0.0053	0.0131	0.0309	0.1127
	10	0.05503	0.00364	0.00316	0.00314	0.19403	0.00841	0.00756	0.01313	0.00994	0.00006	0.00009	0.00019	0.0036	0.0127	0.0271	0.0913
10	0	0.00352	0.00352	0.00232	0.00136	0.00760	0.00760	0.00502	0.00293	0.00005	0.00005	0.00003	0.00001	0.0090	0.0277	0.0671	0.2776
	5	0.03222	0.00357	0.00248	0.00149	0.11295	0.00796	0.00549	0.00330	0.00483	0.00005	0.00003	0.00002	0.0096	0.0246	0.0558	0.2149
	10	0.05503	0.00347	0.00239	0.00141	0.19401	0.00804	0.00556	0.00328	0.00994	0.00005	0.00003	0.00001	0.0090	0.0223	0.0495	0.1716
MC	0		0.05827	0.04036	0.01579		0.13204	0.08833	0.03440		0.00650	0.00387	0.00105		0.0106	0.0142	0.0382
	5		0.05973	0.03692	0.01355		0.13831	0.08213	0.03001		0.00696	0.00348	0.00099		0.0102	0.0130	0.0342
	10		0.05866	0.03416	0.01075		0.14120	0.07791	0.02488		0.00694	0.00326	0.00075		0.0099	0.0116	0.0321

The random networks were generated using parameters (N,K,C,P), where N is the number of variables, K is their domain size, C is the number of conditional probability tables (CPTs) and P is the number of parents in each CPT. Parents in each CPT are picked randomly and each CPT is filled randomly. In grid networks, N is a square number and each CPT is filled randomly. In each problem class, we also tested different numbers of evidence variables. The coding networks are from the class of linear block codes, where σ is the channel noise level. Note that we are limited to relatively small and sparse problem instances since our evaluation measured are based on comparing against exact figures.

Random network results with networks of N=50, K=2, C=45 and P=3 are given in Table 2.7 and Figures 2.17 and 2.18. For IJGP(i) and MC(i) we report 3 different values of i-bound: 2, 5, 8; for IBP and IJGP(i) we report 3 different values of number of iterations: 1, 5, 10; for all algorithms we report 3 different values of number of evidence: 0, 5, 10. We notice that IJGP(i) is always better than IBP (except when i=2 and number of iterations is 1), sometimes as much as an order of magnitude, in terms of absolute and relative error and KL distance. IBP rarely changes after 5 iterations, whereas IJGP(i) solution can be improved up to 15-20 iterations. As we predicted, IJGP(i) is about equal to MC(i) in terms of accuracy for one iteration. But IJGP(i) improves as the number of iterations increases, and is eventually better than MC(i) by as much as an order of magnitude, although it clearly takes more time when the i-bound is large.

Figure 2.17a shows a comparison of all algorithms with different numbers of iterations, using the KL distance. Because the network structure changes with different i-bounds, we do not see monotonic improvement of IJGP with i-bound for a given number of iterations (as is the case with MC). Figure 2.17b shows how IJGP converges with iteration to smaller KL distance than IBP. As expected, the time taken by IJGP (and MC) varies exponentially with the i-bound (see Figure 2.18).

Grid network results with networks of N=81, K=2, 100 instances are very similar to those of random networks. They are reported in Table 2.8 and in Figure 2.19, where we



a) Performance vs. i-bound b) Fine granularity for KL

Figure 2.19: Grid 9x9: KL distance

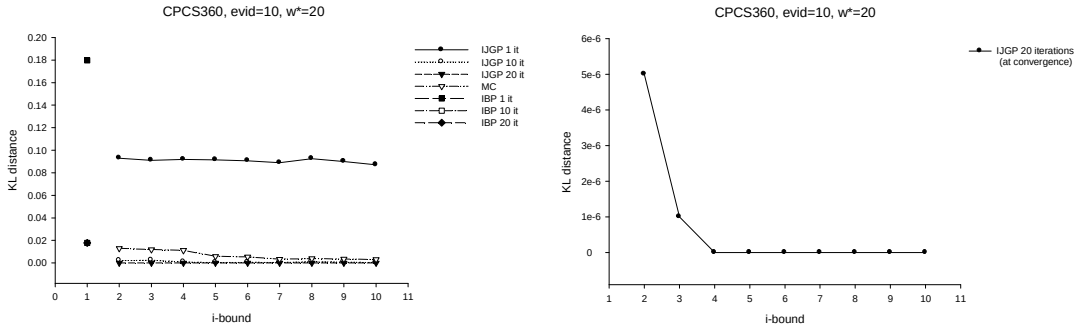
Table 2.9: CPCS networks: CPCS54 50 instances, w*=15; CPCS360 10 instances, w*=20

		Absolute error				Relative error				KL distance				Time			
		IBP	IJGP			IBP	IJGP			IBP	IJGP			IBP	IJGP		
#it	#evid		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8		i=2	i=5	i=8
CPCS54																	
1	0	0.01324	0.03747	0.03183	0.02233	0.02716	0.08966	0.07761	0.05616	0.00041	0.00583	0.00512	0.00378	0.0097	0.0137	0.0146	0.0275
	5	0.02684	0.03739	0.03124	0.02337	0.05736	0.09007	0.07676	0.05856	0.00199	0.00573	0.00493	0.00366	0.0072	0.0094	0.0087	0.0169
	10	0.03915	0.03843	0.03426	0.02747	0.08475	0.09156	0.08246	0.06687	0.00357	0.00567	0.00506	0.00390	0.005	0.0047	0.0052	0.0115
5	0	0.00031	0.00016	0.00123	0.00110	0.00064	0.00033	0.00255	0.00225	7.75e-7	0.00000	0.00002	0.00001	0.0371	0.0334	0.0384	0.0912
	5	0.01874	0.00058	0.00092	0.00098	0.04067	0.00124	0.00194	0.00203	0.00161	0.00000	0.00001	0.00001	0.0337	0.0215	0.0260	0.0631
	10	0.03348	0.00101	0.00139	0.00144	0.07302	0.00215	0.00298	0.00302	0.00321	0.00001	0.00003	0.00002	0.0290	0.0144	0.0178	0.0378
10	0	0.00031	0.00009	0.00014	0.00015	0.00064	0.00018	0.00029	0.00031	7.75e-7	0.0000	0.00000	0.00000	0.0736	0.0587	0.0667	0.1720
	5	0.01874	0.00037	0.00034	0.00038	0.04067	0.00078	0.00071	0.00080	0.00161	0.00000	0.00000	0.00000	0.0633	0.0389	0.0471	0.1178
	10	0.03348	0.00058	0.00051	0.00057	0.07302	0.00123	0.00109	0.00122	0.00321	4.0e-6	3.0e-6	4.0e-6	0.0575	0.0251	0.0297	0.0723
MC	0		0.02721	0.02487	0.01486		0.05648	0.05128	0.03047		0.00218	0.00171	0.00076		0.0144	0.0125	0.0333
	5		0.02702	0.02522	0.01760		0.05687	0.05314	0.03713		0.00201	0.00186	0.00098		0.0103	0.0126	0.0346
	10		0.02825	0.02504	0.01600		0.06002	0.05318	0.03409		0.00216	0.00177	0.00091		0.0094	0.0090	0.0295
CPCS360																	
1	10	0.26421	0.14222	0.13907	0.14334	7.78167	2119.20	2132.78	2133.84	0.17974	0.09297	0.09151	0.09255	0.7172	0.5486	0.5282	0.4593
	20	0.26326	0.12867	0.12937	0.13665	370.444	28720.38	30704.93	31689.59	0.17845	0.08212	0.08269	0.08568	0.6794	0.5547	0.5250	0.4578
10	10	0.01772	0.00694	0.00121	0.00258	1.06933	6.07399	0.01005	0.04330	0.017718	0.00203	0.00019	0.00116	7.2205	4.7781	4.5191	3.7906
	20	0.02413	0.00466	0.00115	0.00138	62.99310	26.04308	0.00886	0.01353	0.02027	0.00118	0.00015	0.00036	7.0830	4.8705	4.6468	3.8392
20	10	0.01772	0.00003	3.0e-6	3.0e-6	1.06933	0.00044	8.0e-6	7.0e-6	0.01771	5.0e-6	0.0	0.0	14.4379	9.5783	9.0770	7.6017
	20	0.02413	0.00001	9.0e-6	9.0e-6	62.9931	0.00014	0.00013	0.00004	0.02027	0.0	0.0	0.0	13.6064	9.4582	9.0423	7.4453
MC	10		0.03389	0.01984	0.01402		0.65600	0.20023	0.11990		0.01299	0.00590	0.00390		2.8077	2.7112	2.5188
	20		0.02715	0.01543	0.00957		0.81401	0.17345	0.09113		0.01007	0.00444	0.00234		2.8532	2.7032	2.5297

can see the impact of having evidence (0 and 5 evidence variables) on the algorithms. IJGP at convergence gives the best performance in both cases, while IBP's performance deteriorates with more evidence and is surpassed by MC with i-bound 5 or larger.

CPCS network results with CPCS54 and CPCS360 are given in Table 2.9 and Figure 2.20, and are even more pronounced than those of random and grid networks. When evidence is added, IJGP(i) is more accurate than MC(i), which is more accurate than IBP, as can be seen in Figure 2.20a.

Coding network results are given in Table 2.10. We tested on large networks of 400 variables, with tree-width w*=43, with IJGP and IBP set to run 30 iterations (this is more than enough to ensure convergence). IBP is known to be very accurate for this class of



a) Performance vs. i-bound b) Fine granularity for KL

Figure 2.20: CPCS360: KL distance

Table 2.10: Coding networks: N=400, P=4, 500 instances, 30 iterations, w*=43

Bit Error Rate

σ		i-bound 2	4	6	8	10	IBP
0.22	IJGP	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005
	MC	0.00501	0.00800	0.00586	0.00462	0.00392	
0.28	IJGP	0.00062	0.00062	0.00062	0.00062	0.00062	0.00064
	MC	0.02170	0.02968	0.02492	0.02048	0.01840	
0.32	IJGP	0.00238	0.00238	0.00238	0.00238	0.00238	0.00242
	MC	0.04018	0.05004	0.04480	0.03878	0.03558	
0.40	IJGP	0.01202	0.01188	0.01194	0.01210	0.01192	0.01220
	MC	0.08726	0.09762	0.09272	0.08766	0.08334	
0.51	IJGP	0.07664	0.07498	0.07524	0.07578	0.07554	0.07816
	MC	0.15396	0.16048	0.15710	0.15452	0.15180	
0.65	IJGP	0.19070	0.19056	0.19016	0.19030	0.19056	0.19142
	MC	0.21890	0.22056	0.21928	0.21904	0.21830	

Time

	IJGP	0.36262	0.41695	0.86213	2.62307	9.23610	0.019752
	MC	0.25281	0.21816	0.31094	0.74851	2.33257	

problems and it is indeed better than MC. It is remarkable however that IJGP converges to smaller BER than IBP even for small values of the i-bound. Both the coding network and CPCS360 show the scalability of IJGP for large size problems. Notice that here the anytime behavior of IJGP is not clear.

2.2.6 Discussion

In this section we presented an iterative anytime approximation algorithm called Iterative Join-Graph Propagation (IJGP(i)), that applies the message passing algorithm of join-tree clustering to join-graphs rather than join-trees, iteratively. The algorithm borrows the iterative feature from Iterative Belief Propagation (IBP) on one hand and is inspired by the anytime virtues of mini-clustering MC(i) on the other. We show that the success of IJGP is facilitated by extending the notion of join-graphs to minimal arc-labeled join-graphs, and provide a structuring algorithm that generates minimal arc-labeled join-graphs of bounded size.

The empirical results are extremely encouraging. We experimented with randomly gen-

erated networks, grid-like networks, medical diagnosis CPCS networks and coding networks. We showed that IJGP is almost always superior to both IBP and MC(i) and is sometimes more accurate by an order of several magnitudes. One should note that IBP cannot be improved with more time, while MC(i) requires a large i-bound for many hard and large networks to achieve reasonable accuracy. There is no question that the iterative application of IJGP is instrumental to its success. In fact, IJGP(2) in isolation appears to be the most cost effective variant.

One question which we did not answer in this section is why propagating the messages iteratively helps. Why is IJGP upon convergence, superior to IJGP with one iteration and is superior to MC(i)? One clue can be provided when considering deterministic constraint networks which can be viewed as "extreme probabilistic networks". It is known that constraint propagation algorithms, which are analogous to the messages sent by belief propagation, are guaranteed to converge and are guaranteed to improve with convergence. The propagation scheme presented here works like constraint propagation relative to the flat network abstraction of P, (where all non-zero entries are normalized to a positive constant), and is guaranteed to be more accurate for that abstraction at least. It is precisely these issues that we address in Section 3.1 of next chapter.

Chapter 3

Current and Future Work

3.1 The Inference Power of Iterative Belief Propagation

A good fraction of our current research is devoted to studying the properties of *Iterative Belief Propagation (IBP)*, and of the generalized belief propagation version *Iterative Join-Graph Propagation (IJGP)*. We are particularly interested in making connections to well known algorithms from constraint networks, like Arc-consistency, which may help explain when and why IBP has strong or weak inference power.

The belief propagation algorithm is a distributed algorithm that computes posterior beliefs for tree-structured Bayesian networks (poly-trees) [Pearl1988]. However, in recent years it was shown to work surprisingly well in many applications involving networks with loops, including turbo codes, when applied iteratively [R.J. McEliece & Cheng1997]. Another recent result [Mézard, Parisi, & Zecchina2002] shows impressive performance for an iterative message passing scheme used for very large satisfiability problems. While there is still very little understanding as to why and when IBP works well, some recent investigation shows that when IBP converges, it converges to a stationary point of the Bethe energy, thus making connections to approximation algorithms developed in statistical physics and to variational approaches to approximate inference [Welling & Teh2001, Yedidia, Freeman, & Weiss2001]. However, these approaches do not explain why IBP is successful where it is, and do not allow any performance guarantees on accuracy.

The work we present here is based on some some simple observations that may shed light on IBP's behavior, and on the more general class of IJGP algorithms. Zero-beliefs are variable-value pairs that have zero conditional probability given the evidence. We show that: if a value of a variable is assessed as having zero-belief in any iteration of IBP, it remains a zero-belief in all subsequent iterations; that IBP finitely converges relative to its set of zero-beliefs; and, most importantly that the set of zero-beliefs decided by any of the iterative belief propagation methods is sound. Namely any zero-belief determined by IBP corresponds to a true zero conditional probability relative to the given probability distribution expressed by the Bayesian network.

While each of these claims can be proved directly, our approach is to associate a belief network with a constraint network and show a correspondence between IBP applied to the belief network and an arc-consistency algorithm applied to the corresponding constraint network. Since arc-consistency algorithms are well understood this correspondence not only proves right away the targeted claims, but may provide additional insight into the behavior of IBP and IJGP. In particular, not only it immediately justifies the iterative application of belief propagation algorithms on one hand, but it also illuminates its "distance" from being complete, on the other.

3.1.1 Arc-consistency algorithms

Constraint propagation algorithms is a class of polynomial time algorithms that are at the center of constraint processing techniques. They were investigated extensively in the past three decades and the most well known versions are *arc-*, *path-*, and *i-consistency* [Dechter1992].

DEFINITION **3.1.1 (arc-consistency)** *[Mackworth1977] Given a binary constraint network (X, D, C), the network is arc-consistent iff for every binary constraint $R_{ij} \in C$, every value $v \in D_i$ has a value $u \in D_j$ s.t. $(v, u) \in R_{ij}$.*

When a binary constraint network is not arc-consistent, arc-consistency algorithms can enforce arc-consistency. The algorithms remove values from the domains of the variables that violate arc-consistency until an arc-consistent network is generated. A variety of improved performance arc-consistency algorithms were developed over the years, however we will consider here a non-optimal distributed version, which we call *distributed arc-consistency*.

DEFINITION **3.1.2 (distributed arc-consistency; DAC)** *The algorithm is a message passing algorithm. Each node maintains a current set of viable values D_i. Let $ne(i)$ be the set of neighbors of X_i in the constraint graph. Every node X_i sends a message to any node $X_j \in ne(i)$, which consists of the values in X_j's domain that are consistent with the current D_i, relative to the constraint that they share. Namely, the message that X_i sends to X_j, denoted by D_i^j, is:*

$$D_i^j \leftarrow \pi_j(R_{ji} \bowtie D_i) \tag{3.1}$$

(where, join ($\bowtie$) and project (π) are the usual relational operators) and in addition node i computes:

$$D_i \leftarrow D_i \cap (\bowtie_{k \in ne(i)} D_k^i) \tag{3.2}$$

Clearly the algorithm can be synchronized into iterations, where in each iteration every node computes its current domain based on all the messages received so far from its

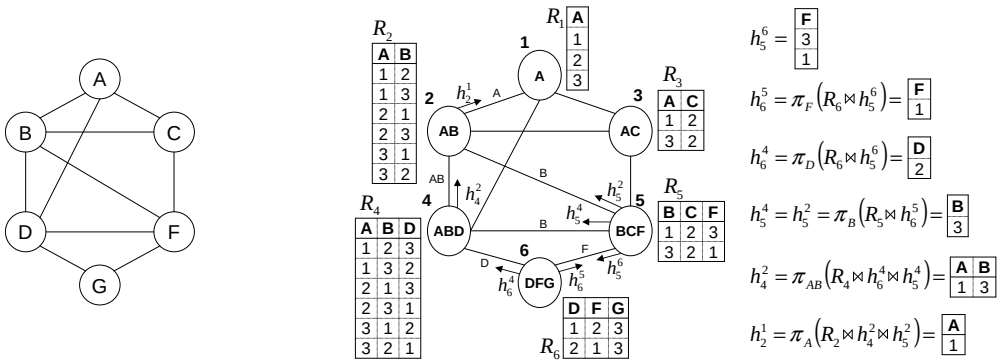


Figure 3.1: Part of the execution of DR-AC algorithm;

neighbors (eq. 3.2), and sends a new message to each neighbor (eq. 3.1). Alternatively, equations 3.1 and 3.2 can be combined. The message X_i sends to X_j is:

$$D_i^j \leftarrow \pi_j(R_{ji} \bowtie D_i \bowtie_{k \in ne(i)} D_k^i) \tag{3.3}$$

Let us mention again the definition of the dual graphs, which we will be using in this section:

DEFINITION **3.1.3 (dual graphs)** *Given a set of functions $F = \{f_1, ..., f_l\}$ over scopes $S_1, ..., S_l$, the dual graph of F is a graph $DG = (V, E, L)$ that associates a node with each function, namely $V = F$ and an arc connects any two nodes whose scope share a variable, $E = \{(f_i, f_j) | S_i \cap S_j \neq \phi\}$. L is a set of labels for the arcs, each arc being labeled by the shared variables of its nodes, $L = \{l_{ij} = S_i \cap S_j | (i, j) \in E\}$.*

The above distributed arc-consistency algorithm can be applied to the dual problem of any non-binary constraint network as well. This is accomplished by the following rule applied by each node in the dual graph. We call the algorithm distributed relational arc-consistency.

DEFINITION **3.1.4 (distributed relational arc-consistency; DR-AC)** *Let R_i and R_j be two constraints sharing scopes, whose arc in the dual graph is labeled by l_{ij}. The message R_i sends to R_j denoted h_i^j is defined by:*

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \tag{3.4}$$

and each node updates its current relation according to:

$$R_i \leftarrow R_i \cap (\bowtie_{k \in ne(i)} h_k^i) \tag{3.5}$$

Example 3.1.1 *Figure 3.1 describes part of the execution of DR-AC for a graph coloring problem, having the constraint graph shown on the left. All variables have the same domain, {1,2,3}, except for C which is 2, and G which is 3. The arcs correspond to* not equal *constraints, and the relations are R_A, R_{AB}, R_{AC}, R_{ABD}, R_{BCF}, R_{DFG}. The dual graph of this problem is given on the right side of the figure, and each table shows the initial constraints (there are unary, binary and ternary constraints). To initialize the algorithm, the first messages sent out by each node are universal relations over the labels. For this example, DR-AC actually solves the problem and finds the unique solution A=1, B=3, C=2, D=2, F=1, G=3.*

Proposition 3 *Distributed relational arc-consistency converges after $O(t \cdot r)$ iterations to the largest arc-consistent network that is equivalent to the original network, where t bounds the number of tuples in each constraint and r is the number of constraints.*

Proposition 4 (complexity) *The complexity of distributed arc-consistency is $O(r^2 t^2 \log t)$.*

Proof. One iteration can be accomplished in $O(r \cdot t \cdot \log t)$, and there can be at most $r \cdot t$ iterations. □

3.1.2 Iterative belief propagation over dual join-graphs

Iterative belief propagation (IBP) is an iterative application of Pearl's algorithm that was defined for poly-trees [Pearl1988]. Since it is a distributed algorithm, it is well defined for any network. In this section we will present IBP as an instance of join-graph propagation over variants of the *dual graph*.

Consider a Bayesian network $\mathcal{B} =< X, D, G, P >$. As defined earlier, the *dual graph* $\mathcal{D}_\mathcal{G}$ of the Belief network $\mathcal{B}$, is an arc-labeled graph defined over the CPTs as its functions. Namely, it has a node for each CPT and a labeled arc connecting any two nodes that share a variable in the CPT's scope. The arcs are labeled by the shared variables. A *dual join-graph* is a labeled arc subgraph of $\mathcal{D}_\mathcal{G}$ whose arc labels are subsets of the labels of $\mathcal{D}_\mathcal{G}$ such that the *running intersection property*, also called *connectedness property*, is satisfied. The running intersection property requires that any two nodes that share a variable in the dual join-graph be connected by a path of arcs whose labels contain the shared variable. Clearly the dual graph itself is a dual join-graph. An *arc-minimal* dual join-graph is a dual join-graph for which none of the labels can be further reduced while maintaining the connectedness property.

Interestingly, there are many dual join-graphs of the same dual graph and many of them are arc-minimal. We define Iterative Belief Propagation on a dual join-graph. Each node sends a message over an arc whose scope is identical to the label on that arc. Since Pearl's algorithm sends messages whose scopes are singleton variables only, we highlight arc-minimal singleton dual join-graph. One such graph can be constructed directly from the

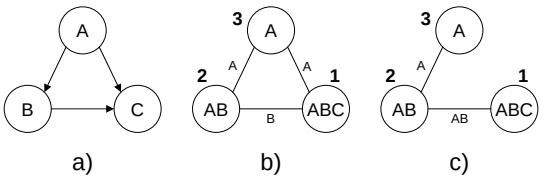


Figure 3.2: a) A belief network; b) A dual join-graph with singleton labels; c) A dual join-graph which is a join-tree;

graph of the Bayesian network, labeling each arc with the parent variable. It can be shown that:

Proposition 5 *The dual graph of any Bayesian network has an arc-minimal dual join-graph where each arc is labeled by a single variable.*

Example 3.1.2 *Consider the belief network on 3 variables A, B, C with CPTs 1.$P(C|A,B)$, 2.$P(B|A)$ and 3.$P(A)$, given in Figure 3.2a. Figure 3.2b shows a dual graph with singleton labels on the arcs. Figure 3.2c shows a dual graph which is a join tree, on which belief propagation can solve the problem exactly in one iteration (two passes up and down the tree).*

For complete reference, we will next present IBP algorithm that is applicable to any dual join-graph (Figure 3.3). The algorithm is a special case of IJGP introduced in [Dechter, Mateescu, & Kask2002]. It is easy to see that one iteration of IBP is time and space linear in the size of the belief network, and when IBP is applied to the singleton labeled dual graph it coincides with Pearl's belief propagation applied directly to the acyclic graph representation. For space reasons, we do not include the proof here. Also, when the dual join-graph is a tree IBP converges after one iteration (two passes, up and down the tree) to the exact beliefs.

3.1.3 Flattening the Bayesian network

Given a belief network $\mathcal{B}$ we will now define a flattening of the Bayesian network into a constraint network called $flat(\mathcal{B})$ where all the zero entries in the CPTs are removed from the corresponding relation. $flat(B)$ is a constraint network defined over the same set of variables and has the same set of domain values as $\mathcal{B}$. Formally, for every X_i and its CPT $P(X_i|pa_i) \in \mathcal{B}$ we define a constraint R_{F_i} over the family of X_i, $F_i = \{X_i\} \cup pa_i$ as follows: for every assignment $x = (x_i, x_{pa_i})$ to F_i,

$$(x_i, x_{pa_i}) \in R_{F_i} \ \ iff \ \ P(x_i|x_{pa_i}) > 0.$$

The evidence set $e = \{e_1, ..., e_r\}$ is mapped into unary constraints that assign the corresponding values to the evidence variables.

Algorithm **IBP**
Input: An arc-labeled dual join-graph $DJ = (V, E, L)$ for a Bayesian network $BN =< X, D, G, P >$. Evidence e.
Output: An augmented graph whose nodes include the original CPTs and the messages received from neighbors. Approximations of $P(X_i|e),\ \forall X_i \in X$. Approximations of $P(F_i|e),\ \forall F_i \in \mathcal{B}$.
Denote by: h_u^v the message from u to v; $ne(u)$ the neighbors of u in V; $ne_v(u) = ne(u) - \{v\}$; l_{uv} the label of $(u, v) \in E$; $elim(u, v) = scope(u) - scope(v)$.

• **One iteration of IBP**

For every node u in DJ in a topological order and back, do:

1. **Process observed variables**
   Assign evidence variables to the each p_i and remove them from the labeled arcs.
2. **Compute and send to v the function:**

$$h_u^v = \sum_{elim(u,v)} (p_u \cdot \prod_{\{h_i^u, i \in ne_v(u)\}} h_i^u)$$

Endfor

• **Compute approximations of $P(F_i|e)$, $P(X_i|e)$:**
For every $X_i \in X$ let u be the vertex of family F_i in DJ,
$P(F_i|e) = \alpha(\prod_{h_i^u, u \in ne(i)} h_i^u) \cdot p_u$;
$P(X_i|e) = \alpha \sum_{scope(u)-\{X_i\}} P(F_i|e)$.

Figure 3.3: Algorithm Iterative Belief Propagation;

THEOREM **3.1.3** *Given a belief network $\mathcal{B}$ and evidence e, for any tuple t: $P_{\mathcal{B}}(t|e) > 0 \Leftrightarrow t \in sol(flat(B, e))$.*

Proof. $P_{\mathcal{B}}(t|e) > 0 \Leftrightarrow \Pi_i P(x_i|x_{pa_i})|_t > 0 \Leftrightarrow \forall i,\ P(x_i|x_{pa_i})|_t > 0 \Leftrightarrow \forall i,\ (x_i, x_{pa_i})|_t \in R_{F_i}$ $\Leftrightarrow t \in sol(flat(B, e))$, where $|_t$ is the restriction to t. □

We next define an algorithm dependent notion of zero tuples.

DEFINITION **3.1.5 (IBP-zero)** *Given a CPT $P(X_i|pa_i)$, an assignment $x = (x_i, x_{pa_i})$ to its family F_i is* IBP-zero *if some iteration of IBP determines that $P(x_i|x_{pa_i}, e) = 0$.*

It is easy to see that when IBP is applied to a constraint network where sum and product are replaced by join and project, respectively, it becomes identical to distributed relational arc-consistency defined earlier. Therefore, a partial tuple is removed from a flat constraint by arc-consistency iff it is IBP-zero relative to the Bayesian network.

THEOREM **3.1.4** *When IBP is applied in a particular variable ordering to a dual join-graph of a Bayesian network $\mathcal{B}$, its trace is identical, relative to zero-tuples generation, to that of DR-AC applied to the corresponding flat dual join-graph. Namely, taking a snapshot at identical steps, any IBP-zero tuple in the Bayesian network is a removed tuple in the corresponding step of DR-AC over the flat dual join-graph.*

Proof. It suffices to prove that the first iteration of IBP and DR-AC generates the same zero tuples and removed tuples, respectively. We prove the claim by induction over the topological ordering that defines the order in which messages are sent in the corresponding dual graphs.
Base case: By the definition of the flat network, when algorithms IBP and DR-AC start, every zero probability tuple in one of the CPTs P_{X_i} in the dual graph of the Bayesian network, becomes a removed tuple in the corresponding constraint R_{F_i} in the dual graph of the flat network.
Inductive step: Suppose the claim is true after n correspondent messages are sent in IBP and DR-AC. Suppose the $(n+1)th$ message is scheduled to be the one from node u to node v. Indexing messages by the name of the algorithm, in the dual graph of IBP, node u contains p_u and ${h_{IBP}}_i^u, i \in ne_v(u)$, and in the dual graph of DR-AC, node u contains R_u and ${h_{DR-AC}}_i^u, i \in ne_v(u)$. By the inductive hypothesis, the zero tuples in p_u and ${h_{IBP}}_i^u, i \in ne_v(u)$ are the removed tuples in R_u and ${h_{DR-AC}}_i^u, i \in ne_v(u)$, respectively. Therefore, the zero tuples in the product $(p_u \cdot (\prod_{i \in ne_v(u)}) h_i^u)$ correspond to the removed tuples in the join $(R_u \bowtie (\bowtie_{i \in ne_v(u)}) h_i^u)$. This proves that the zero tuples in the message of IBP
${h_{IBP}}_u^v = \sum_{elim(u,v)} (p_u \cdot (\prod_{i \in ne_v(u)}) h_i^u)$, correspond to the removed tuples in the message of DR-AC
${h_{DR-AC}}_u^v = \pi_{l_{uv}} (R_u \bowtie (\bowtie_{i \in ne_v(u)}) h_i^u)$.
The same argument can now be extended for every iteration of the algorithms. □

Corollary 1 *Algorithm IBP zero-converges. Namely, its set of zero tuples does not change after $t \cdot r$ iterations.*

Proof. From Theorem 3.1.4 any IBP-zero is a no-good removed by arc-consistency over the flat network. Since arc-consistency converges, the claim follows. □

THEOREM **3.1.5** *When IBP is applied to a dual join-graph of a Bayesian network, any tuple t that is IBP-zero satisfies $P_{\mathcal{B}}(t|e) = 0$.*

Proof. From Theorem 3.1.4 if a tuple t is IBP zero, it is also removed from the corresponding relation by arc-consistency over $flat(\mathcal{B}, e)$. Therefore this tuple is a no-good of the network $flat(\mathcal{B}, e)$ and, from Theorem 3.1.3 it follows that $P_{\mathcal{B}}(t|e) = 0$. □

3.1.3.1 Zeros are sound for any IJGP

The results for IBP can be extended to the more general class of algorithms called *iterative join-graph propagation*, IJGP [Dechter, Mateescu, & Kask2002]. IJGP can be viewed as a

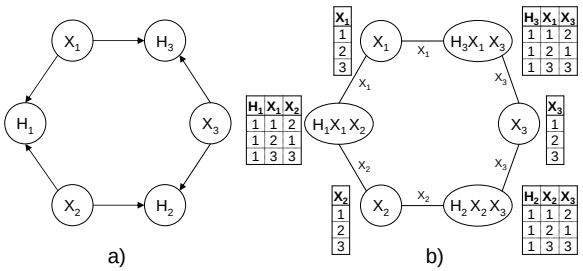


Figure 3.4: a) A belief network; b) An arc-minimal dual join-graph;

generalized belief propagation algorithm and was shown to benefit both from the virtues of iterative algorithms and from the anytime characteristics of bounded inference provided by mini-buckets schemes.

The message-passing of IJGP is identical to that of IBP. The difference is in the underlying graph that it uses. IJGP typically has an accuracy parameter i called i-bound, which restricts the maximum number of variables that can appear in a node (cluster). Each cluster contains a set of functions. IJGP performs message-passing on a graph called *minimal arc-labeled join-graph*.

It is easy to define a corresponding DR-AC algorithm that operates on a similar minimal arc-label join-graph. Initially, each cluster of DR-AC can contain a number of relations, which are just the flat correspondents of the CPTs in the clusters of IJGP. The identical mechanics of the message passing ensure that all the previous results for IBP can be extended to IJGP.

3.1.3.2 The inference power of IBP

We will next show that the inference power of IBP is sometimes very limited and other times strong, exactly wherever arc-consistency is weak or strong.

3.1.3.2.1 Cases of weak inference power

Example 3.1.6 *Consider a belief network over 6 variables $X_1, X_2, X_3, H_1, H_2, H_3$ where the domain of the X variables is $\{1, 2, 3\}$ and the domain of the H variables is $\{0, 1\}$ (see Figure3.4a). There are three CPTs over the scopes: $\{H_1, X_1, X_2\}$, $\{H_2, X_2, X_3\}$, and $\{H_3, X_1, X_3\}$. The values of the CPTs for every triplet of variables $\{H_k, X_i, X_j\}$ are:*

$$\begin{aligned} P(h_k = 1|x_i, x_j) &= \begin{cases} 1, & if \ \ (3 \neq x_i \neq x_j \neq 3); \\ 1, & if \ \ (x_i = x_j = 3); \\ 0, & otherwise; \end{cases} \\ P(h_k = 0|x_i, x_j) &= 1 - P(h_k = 1|x_i, x_j). \end{aligned}$$

Consider the evidence set $e = \{H_1 = H_2 = H_3 = 1\}$. *One can see that this Bayesian network expresses the probability distribution that is concentrated in a single tuple:*

$$P(x_1, x_2, x_3|e) = \begin{cases} 1, & if \ \ x_1 = x_2 = x_3 = 3; \\ 0, & otherwise. \end{cases}$$

In other words, any tuple containing an assignment of "1" or "2" for any X *variable has a zero probability. The flat constraint network of the above belief network is defined over the scopes* $S_1 = \{H_1, X_1, X_2\}$, $S_2 = \{H_2, X_2, X_3\}$, $S_3 = \{H_3, X_1, X_3\}$. *The constraints are defined by:* $R_{H_k,X_i,X_j} = \{(1,1,2), (1,2,1), (1,3,3), (0,1,1), (0,1,3), (0,2,2), (0,2,3), (0,3,1), (0,3,2)\}$. *Also, the prior probabilities for* X_i*'s become unary constraints equal to the full domain {1,2,3} (assuming the priors are non-zero). An arc-minimal dual join-graph which is identical to the constraint network is given in Figure 3.4b.*

In the flat constraint network, the constraints in each node are restricted after assigning the evidence values (see Figure 3.4b). In this case, DR-AC sends as messages the full domains of the variables and therefore no tuple is removed from any constraint. Since IBP infers the same zeros as arc-consistency, IBP will also not *infer any zeros for any family or any single variable. However, since the true probability of most tuples is zero we can conclude that the inference power of IBP on this example is weak or non-existent.*

The weakness of arc-consistency as demonstrated in this example is not surprising. Arc-consistency is known to be a weak algorithm in general. It implies the same weakness for belief propagation and demonstrates that IBP is very far from completeness, at least as long as zero tuples are concerned.

The above example was constructed by taking a specific constraint network with known properties and expressing it as a belief network using a known transformation. We associate each constraint R_S with a bi-valued new hidden variable X_h, direct arcs from the constraint variables to this new hidden variable X_h, and create the CPT such that:

$P(x_h = 1|x_{pa_h}) = 1 \ \ , iff \ \ x_{pa_h} \in R_S.$

while zero otherwise [Pearl1988]. The generated belief network conditioned on all the X_h variables being assigned "1" expresses the same set of solutions as the constraint network.

3.1.3.2.2 Cases of strong inference power The relationship between IBP and arc-consistency ensures that IBP is zero-complete whenever arc-consistency is. In general, if for a flat constraint network of a Bayesian network $\mathcal{B}$, arc-consistency removes all the inconsistent domain values (it creates minimal domains), then IBP will also discover all the true zeros of $\mathcal{B}$. We next consider several classes of constraints that are known to be tractable.

Acyclic belief networks. When the belief network is acyclic, namely when it has a dual join-graph that is a tree, the flat network is an acyclic constraint network that can be shown to be solvable by distributed relational arc-consistency [Dechter1992]. Note that acyclic Bayesian networks is a strict superset of polytrees. The solution requires only one iteration (two passes) of IBP. Therefore:

Proposition 6 *IBP is complete for acyclic networks, when applied to the tree dual join-graph (and therefore it is also zero-complete).*

Example 3.1.7 *We refer back to the example of Figure 3.2. The network is acyclic because there is a dual join-graph that is a tree, given in Figure 3.2c, and IBP will be zero-complete on it. Moreover, IBP is known to be complete in this case.*

Belief networks with no evidence. Another interesting case is when the belief network has no evidence. In this case, the flat network always corresponds to the *causal constraint network* defined in [Dechter & Pearl1991]. The inconsistent tuples or domain values are already explicitly described in each relation, and new zeros do not exist. Indeed, it is easy to see (either directly or through the flat network) that:

Proposition 7 *IBP is zero-complete for any Bayesian network with no evidence.*

In fact, it can be shown [Bidyuk & Dechter2001] that IBP is also complete for non-zero posterior beliefs of many variables when there is no evidence.

Max-closed constraints. Consider next the class of Max-closed relations defined as follows. Given a domain D that is linearly ordered let *Max* be a binary operator that returns the largest element among 2. The operator can be applied to 2 tuples by taking the pair-wise operation [Jeavons & Cooper1996].

DEFINITION **3.1.6 (Max-closed relations)** *A relation is Max-closed if whenever $t_1, t_2 \in R$ so is $Max(t_1, t_2)$. A constraint network is Max-closed if all its constraints are Max-closed.*

It turns out that if a constraint network is Max-closed, it can be solved by distributed arc-consistency. Namely, if no domain becomes empty by the arc-consistency algorithm, the network is consistent. While arc-consistency is not guaranteed to generate minimal domains, thus removing all inconsistent values, it can generate a solution by selecting the maximal value from the domain of each variable. Accordingly, while IBP will not necessarily discover all the zeros, all the largest non-zero values in the domains of each variable are true non-zeros.

Therefore, for a belief network whose flat network is Max-closed IBP is likely to be powerful for generating zero tuples.

Example 3.1.8 *Consider the following belief network: There are 5 variables $\{V, W, X, Y, Z\}$ over domains $\{1, 2, 3, 4, 5\}$. and the following CPTs:*

$$
\begin{array}{ll}
P(x|z,y,w) \neq 0, & iff \;\; 3x + y + z \geq 5w + 1 \\
P(w|y,z) \neq 0, & iff \;\; wz \geq 2y \\
P(y|z) \neq 0, & iff \;\; y \geq z + 2 \\
P(v|z) \neq 0, & iff \;\; 3v \leq z + 1 \\
P(Z = i) = 1/4, & i \in \{1, 2, 3, 4\}
\end{array}
$$

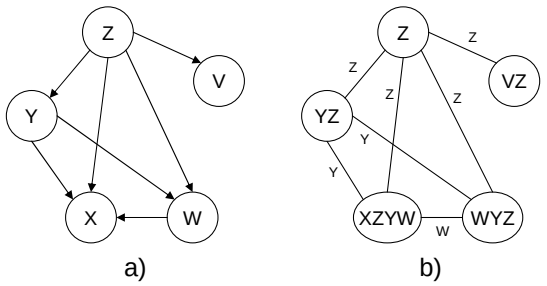


Figure 3.5: a) A belief network that corresponds to a Max-closed relation; b) An arc-minimal dual join-graph;

All the other probabilities are zero. Also, the domain of W does not include 3 and the domain z does not include 5. The problem's acyclic graph is given in Figure 3.5a. It is easy to see that the flat network is the set of constraints over the above specified domains: $w \neq 3$, $z \neq 5$, $3v \leq z+1$, $y \geq z+2$, $3x+y+z \geq 5w+1$, $wz \geq 2y$. *An arc-minimal dual join-graph with singleton labels is given in Figure 3.5b. It has 5 nodes, one for each family in the Bayesian network. If we apply distributed relational consistency we will get that the domains are:* $D_V = \{1\}$, $D_W = \{4\}$, $D_X = \{3,4,5\}$, $D_Y = \{4,5\}$ *and* $D_Z = \{2,3\}$. *Since all the constraints are Max-closed and since there is no empty domain the problem has a solution given by the maximal values in each domain:* $V = 1$, $W = 4$, $X = 5$, $Y = 5$, $Z = 3$. *The domains are not minimal however: there is no solution having* $X = 3$ *or* $X = 4$.

Based on the correspondence with arc-consistency, we know that applying IBP to the dual join-graph will indeed infer all the zero domains except those of X, *which validates that IBP is quite powerful for this example.*

The above example is suggested by a general scheme for creating belief networks that correspond to Max-closed constraints (or any other language of constraints): First, create an acyclic graph, then, associate with each node and its parents a max-closed probability constraint.

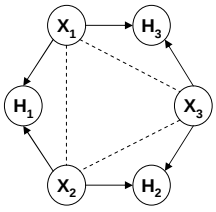
An interesting case for propositional variables is the class of Horn clauses. A Horn clause can be shown to be Min-closed (by simply checking its models). If we have an acyclic graph, and we associate every family with a Horn clause expressed as a CPT in the obvious way, then applying Belief propagation on a dual join-graph can be shown to be nothing but the application of unit propagation until there is no change. It is well known that unit propagation decides the consistency of a set of Horn clauses (even if they are cyclic). However, unit propagation will not necessarily generate the minimal domains, and thus not infer all the zeros, but it is likely to behave well.

Implicational constraints. Finally, a class that is known to be solvable by path-consistency is implicational constraints, defined as follows:

DEFINITION **3.1.7** *A binary network is implicational, iff for every binary relation every value of one variable is consistent either with only one or with all the values of the other variable [Kirousis1993]. A Bayesian network is implicational if its flat constraint networks is.*

Clearly, a binary function is an implicational constraint. Since IBP is equivalent to arc-consistency only, we cannot conclude that IBP is zero-complete for implicational constraints. This raises the question of what corresponds to path-consistency in belief networks, a question which we do not attempt to answer at this point.

3.1.3.3 A Finit



Prior for X_i

X_i	$P(X_i)$
1	.45
2	.45
3	.1

CPT for H_k

H_k	X_i	X_j	$P(H_k \mid X_i, X_j)$
1	1	2	1
1	2	1	1
1	3	3	1
1	...	...	0

#iter	Bel(X_i = 1)	Bel(X_i = 2)	Bel(X_i = 3)
1	.45	.45	.1
2	.49721	.49721	.00545
3	.49986	.49986	.00027
100	...	...	1e-129
200	...	...	1e-260
300	.5	.5	0
True belief	0	0	1

Figure 3.6: Example of a finite precision problem;

Algorithms should always be implemented with care on finite precision machines. We mention here a case where IBP's messages converge in the limit (i.e. in an infinite number of iterations), but they do not stabilize in any finite number of iterations. Consider again the example in Figure 3.4 with the priors on X_i's given in Figure 3.6. If all nodes H_k are set to value 1, the belief for any of the X_i variables as a function of iteration is given in the table in Figure 3.6. After about 300 iterations, the finite precision of our computer is not able to represent the value for $Bel(X_i = 3)$, and this appears to be zero, yielding the final updated belief $(.5, .5, 0)$, when in fact the true updated belief should be $(0, 0, 1)$. This does not contradict our theory, because mathematically, $Bel(X_i = 3)$ never becomes a true zero, and IBP never reaches a quiescent state.

3.1.4 Empirical results

We tested the performance of IBP and IJGP both on cases of strong and weak inference power. In particular, we looked at networks where probabilities are extreme and checked if the properties of IBP with respect to zeros also extend to ϵ small beliefs.

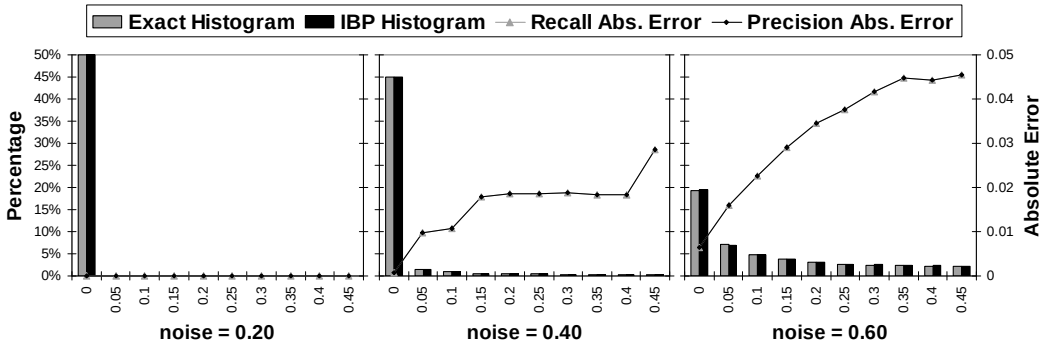


Figure 3.7: Coding, N=200, 1000 instances, w*=15;

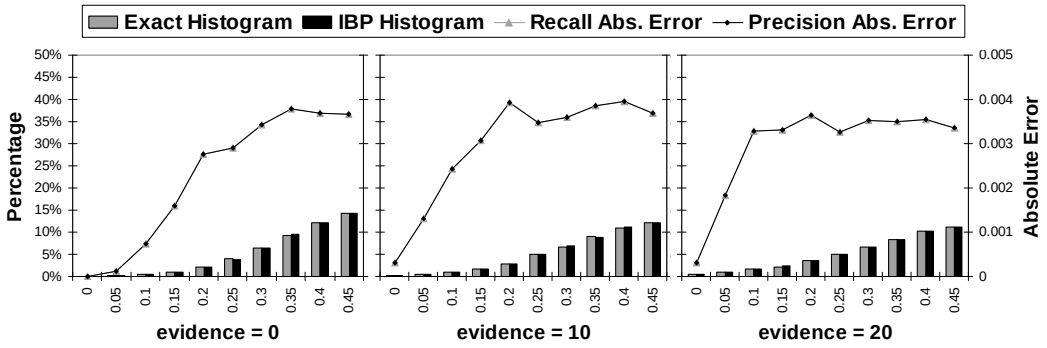


Figure 3.8: 10x10 grids, 100 instances, w*=15;

3.1.4.1 Accuracy of IBP across belief distribution

We investigated empirically the accuracy of IBP's prediction across the range of belief values from 0 to 1. Theoretically, zero values inferred by IBP are proved correct, and we hypothesize that this property extends to ϵ small beliefs. That is, if the flat network is easy for arc-consistency and IBP infers a posterior belief close to zero, then it is likely to be correct.

To capture the accuracy of IBP we computed its absolute error per intervals of $[0, 1]$. Using names inspired by the well known measures in information retrieval, we use *Recall Absolute Error* and *Precision Absolute Error*. *Recall* is the absolute error averaged over all the exact posterior beliefs that fall into the interval. For *Precision*, the average is taken over all the approximate posterior belief values computed by IBP that fall into the interval. Our experiments show that the two measures are strongly correlated. We also show the histograms of distribution of belief for each interval, for the exact and for IBP, which are also strongly correlated. The results are given in Figures 3.7-3.10. The left Y axis corresponds to the histograms (the bars), the right Y axis corresponds to the absolute error (the lines). All problems have binary variables, so the graphs are symmetric about 0.5 and we only show the interval [0, 0.5]. The number of variables, number of iterations and induced

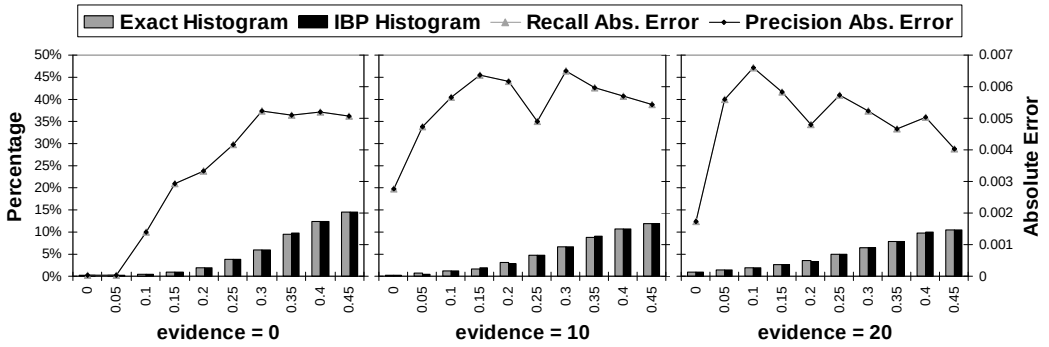


Figure 3.9: Random, N=80, 100 instances, w*=15;

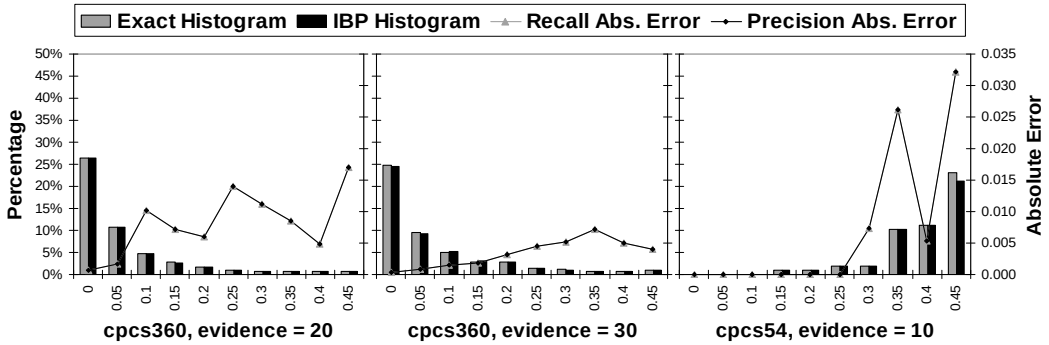


Figure 3.10: CPCS54, 100 instances, w*=15; CPCS360, 5 instances, w*=20;

width w* are reported for each graph.

Coding networks are the notorious case where IBP has impressive performance. The problems are from the class of linear block codes, with 50 nodes per layer and 3 parent nodes. Figure 3.7 shows the results for three different values of channel noise: 0.2, 0.4 and 0.6. For noise 0.2, all the beliefs computed by IBP are extreme. The Recall and Precision are very small, of the order of 10^{-11}. So, in this case, all the beliefs are very small (ϵ small) and IBP is able to infer them correctly, resulting in almost perfect accuracy (IBP is indeed perfect in this case for the bit error rate). When the noise is increased, the Recall and Precision tend to get closer to a bell shape, indicating higher error for values close to 0.5 and smaller error for extreme values. The histograms also show that less belief values are extreme as the noise is increased, so all these factors account for an overall decrease in accuracy as the channel noise increases.

Grid networks results are given in Figure 3.8. Contrary to the case of coding networks, the histograms show higher concentration around 0.5. The absolute error peaks closer to 0 and maintains a plateau, as evidence is increased, indicating less accuracy for IBP.

Random networks results are given in Figure 3.9. The histograms are similar to those of the grids, but the absolute error has a tendency to decrease towards 0.5 as evidence increases. This may be due to the fact that the total number of nodes is smaller (80) than

Table 3.1: Graph coloring type problems: 20 root variables

Absolute error				
	ϵ	H=40, w*=5	H=60, w*=7	H=80, w*=9
IBP	0.0	0.4373	0.4501	0.4115
	0.1	0.3683	0.4497	0.3869
	0.2	0.2288	0.4258	0.3832
IJGP(2)	0.0	0.1800	0.1800	0.1533
	0.1	0.3043	0.3694	0.3189
	0.2	0.1591	0.3407	0.3022
IJGP(4)	0.0	0.0000	0.0000	0.0000
	0.1	0.1211	0.0266	0.0133
	0.2	0.0528	0.1370	0.0916
IJGP(6)	0.0	0.0000	0.0000	0.0000
	0.1	0.0043	0.0000	0.0132
	0.2	0.0123	0.0616	0.0256

for grids (100), and the evidence can in many cases make the problem easier for IBP by breaking many of the loops (in the case of grids evidence has less impact in breaking the loops).

CPCS networks are belief networks for medicine, derived from the Computer based Patient Case Simulation system. We tested on two networks, with 54 and 360 variables. The histograms show opposing trends in the distribution of beliefs. Although irregular, the absolute error tends to increase towards 0.5 for cpcs54. For cpcs360 it is smaller around 0 and 0.5.

We note that for all these types of networks, IBP has very small absolute error for values close to zero, so it is able to infer them correctly.

3.1.4.2 Graph-coloring type problems

We also tested the behavior of IBP and IJGP on a special class of problems which were designed to be hard for belief propagation algorithms in general, based on the fact that arc-consistency is poor on the flat network.

We consider a graph coloring problem which is a generalization of example 3.1.6, with $N = 20$ X nodes, rather than 3, and a variable number of H nodes defining the density of the constraint graph. X variables are 3-valued root nodes, H variables are bi-valued and each has two parents which are X variables, with the CPTs defined like in example 3.1.6. Each H CPT actually models a binary constraint between two X nodes. All H nodes are assigned value 1. The flat network of this kind of problems has only one solution, where every X has value 3. In our experiments we also added noise to the H CPTs, making probabilities ϵ and $1 - \epsilon$ rather than 0 and 1.

The results are given in Table 3.1. We varied parameters along two directions. One was increasing the number of H nodes, corresponding to higher densities of the constraint network (the average induced width $w*$ is reported for each column). The other was increasing the noise parameter ϵ. We averaged over 50 instances for each combination of

these parameters. In each instance, the priors for nodes X were random uniform, and the parents for each node H were chosen randomly. We report the absolute error, averaged over all values, all variables and all instances. We should note that these are fairly small size networks (w*=5-9), yet they prove to be very hard for IBP and IJGP, because the flat network is hard for arc-consistency. It is interesting to note that even when ϵ is not extreme anymore (0.2) the performance is still poor, because the structure of the network is hard for arc-consistency. IJGP with higher i-bounds is good for $\epsilon = 0$ because it is able to infer some zeros in the bigger clusters, and these propagate in the network and in turn infer more zeros.

3.1.5 Discussion

The work presented in this section investigates the behavior of belief propagation algorithms by making analogies to well known and understood algorithms from constraint networks. By a simple transformation, called flattening of the Bayesian network, IBP (as well as any generalized belief propagation algorithm) can be shown to work similar to distributed relational arc-consistency relative to zero tuples generation. In particular we show that IBP's inference of zero beliefs converges and is sound.

Theorem 3.1.5 provides a justification for applying the belief propagation algorithm iteratively. We know that arc-consistency algorithms improve with iteration, generating the largest arc-consistent network that is equivalent to the original network. Therefore by applying IBP iteratively the set of zero tuples concluded grows monotonically until convergence.

While the theoretical results presented here are straightforward, they help identify new classes of problems that are easy or hard for IBP. Non-ergodic belief networks with no evidence, max-closed or implicational belief networks are expected to be cases of strong inference power for IBP. Based on empirical work, we observe that good performance of IBP and many small beliefs indicate that the flat network is likely to be easy for arc-consistency. On the other hand, when we generated hard networks for arc-consistency, IBP was very poor in spite of the presence of many zero beliefs. We believe that the success of IBP for coding networks can be explained by the presence of many extreme beliefs on one hand, and by an easy-for-arc-consistency flat network on the other. We plan to conduct more experiments on coding networks and study the influence of the good accuracy of IBP for extreme beliefs combined with the ϵ-cutset effect described in [Bidyuk & Dechter2001].

3.2 Partitioning Heuristics for Mini-Buckets

This section describes one of the issues we are currently investigating. The partitioning problem is relevant to many of the algorithms that we develop, and improving its accuracy would impact other areas of our research. We present here some preliminary results which may help suggest how much computational effort is worth investing in the partitioning step.

Both Mini-Buckets and Mini-Clustering algorithms rely on a procedure which is applied every time the number of variables to be processed in a bucket (cluster) exceeds the *i-bound*. This procedure takes the functions that are to be processed and partitions them into mini-buckets (or mini-clusters) such that the number of variables in each one of them does not exceed the *i-bound*. Although we presented the Mini-Clustering algorithm in this report, we will refer the relevant entities by *bucket* and *mini-bucket* from now on.

Obviously, in most cases the partitioning can be performed in many different ways, each one having a different impact on the overall accuracy of the algorithm. The problem we would like to address here is precisely the partitioning step. We will investigate the problem by describing it in isolation, although the benefit can only be fully understood when we plug it in the main algorithm and see the overall result.

In short, we want to find the best partitioning strategy for buckets into mini-buckets containing no more than i variables, such that the distance between the exact function to be computed and the approximate one which is computed is minimized. Formally, a bucket is a collection of functions $F = \{f_1, \ldots, f_k\}$. The exact Bucket Elimination algorithm computes the message $h_{exact} = \sum_{elim} \prod_{f \in F} f$, where *elim* are the variables to be eliminated. The Mini-Bucket algorithm partitions F into mini-buckets $mb(1), \ldots, mb(p)$, each containing at most i variables and computes a collection of messages $h^j_{approx} = \sum_{elim} \prod_{f \in mb(j)} f,\ j = 1, \ldots, p$. This collection of messages amounts to an approximate message $h_{approx} = \prod_{j=1}^{p} h^j_{approx}$. If $\mathcal{D}$ is a distance measure, the optimization problem we want to solve is to minimize $\mathcal{D}(h_{approx}, h_{exact})$ over all possible partitionings. The distance $\mathcal{D}$ can have different forms. Some of the most relevant ones for belief updating are the Kullback-Leibler (KL) distance and the absolute error.

We should note that finding the optimal partitioning with respect to $\mathcal{D}$ is a hard problem. However, expressing what we need to optimize suggests a number of heuristic approaches which can help guide the partitioning procedure. The time cost of such heuristic schemes is crucial, but we are right now interested to determine how much they gain in accuracy compared to the greedy procedure presented below.

The initial version of Mini-Bucket uses a greedy procedure for partitioning, described in Figure 3.11. *Procedure Greedy Partitioning* simply tries to create as few mini-buckets as possible. It sorts the functions by the size of their scopes, creates a mini-bucket for the largest function, and then places the other functions in already existing mini-buckets if the scope does not exceed the *i-bound*. If a function can be placed in more than one mini-bucket, the one with most functions is preferred. This is based on empirical observations which tend to show that unbalanced partitions yield better accuracy. If no existing mini-

bucket can accommodate the new function, a new mini-bucket is created.

Procedure **Greedy Partitioning**
Input: $\{h_1, \ldots, h_k\}$, *i-bound*;
Output: A partitioning $mb(1), \ldots, mb(p)$ such that every $mb(i)$ contains at most *i-bound* variables;
1. Sort functions by the size of their scopes. Let $\{h_1, \ldots, h_k\}$ be the sorted array of functions, with h_1 having the largest scope.
2. **for** $i = 1$ to k
  if h_i can be placed in existing mini-buckets without making the scope greater than the *i-bound*, place it in the one with the most functions.
  else create a new mini-bucket an place h_i in it.
endfor

Figure 3.11: Greedy Partitioning

Proposition 8 *The time complexity of the Greedy Partitioning is $O(k^2)$, where k is the number of functions in the bucket.*

Proof. Step 1 takes $O(k \log(k))$, and step 2 takes $O(k^2)$. □

This greedy procedure has the advantage of being very simple and adding very little overhead. It is important nevertheless to investigate whether more elaborate partitioning schemes could improve the accuracy at the expense of time. The greedy partitioning only takes into consideration the size of the tables (functions) that need to be processed, but does not use the contents of these tables. Therefore our goal is to improve the partitioning by using the contents of the tables. We chose the distance measure $\mathcal{D}$ to be the Kullback-Leibler (KL) distance.

The procedure *Heuristic KL Partitioning*, given in Figure 3.12, starts by placing each function in a mini-bucket of its own, and merging them by subsumption of their scopes. Then it combines two mini-buckets at a time, until no more mini-buckets can be merged. The procedure runs a *while* loop and decides at each step which is the best pair of mini-buckets that can be merged. The decision is taken by looking at this subpoblem in isolation, and studying the error in terms of KL distance between sending separate messages from each mini-bucket, and grouping them and sending a combined message. The function f_{approx} is the result of separate processing of the mini-buckets, and f_{exact} is the result of combining them and sending the exact message. Intuitively, if the error (KL distance) between f_{approx} and f_{exact} is small, then the algorithm should give lower priority to merging the two minibuckets. If the error is big however, the algorithm should try to combine them.

The error measure that we use is average KL distance. That is $f_{approx} \cdot \log(f_{approx}/f_{exact})$ averaged over all the the instantiations of the variables in the scope.

Procedure **Heuristic KL Partitioning**

Input: $\{h_1, \ldots, h_k\}$, *i-bound*;
Output: A partitioning $mb(1), \ldots, mb(p)$ such that every $mb(i)$ contains at most *i-bound* variables;
1. Initially, place each function in one mini-bucket.
2. Merge mini-buckets by subsumption of their scopes.
3. **while** (there exist mini-buckets that can be combined)
   for (every $mb(i)$, $mb(j)$ that can be combined, $i \neq j$)

$$Compute: \begin{cases} f_{approx} & = & (\sum_{elim} \prod_{h' \in mb(i)} h') \times (\sum_{elim} \prod_{h'' \in mb(j)} h'') \\ f_{exact} & = & (\sum_{elim} \prod_{h \in mb(i) \cup mb(j)} h) \end{cases}$$

$score[i,j] = AverageKL(f_{approx}, f_{exact})$.
   endfor
   Combine $mb(i)$, $mb(j)$ for which $score[i,j]$ is largest.
endwhile

Figure 3.12: Heuristic KL Partitioning

This allows us to make a reasonable comparison between KL distances taken over pairs of tables of different sizes.

Proposition 9 *The time complexity of the Heuristic KL Partitioning is $O(k^3 \cdot d^i)$ where k is the number of functions in the bucket, d is the maximum domain size of the variables in the bucket and i is the* i-bound.

Proof. Computing f_{approx} and f_{exact} and the KL distance amounts to $O(d^i)$, the *for* loop can be executed at most k^2 times, and the *while* loop at most k times. □

3.2.1 Empirical results

We investigated empirically the performance of Mini-Clustering when using the two different partitioning schemes. The measure of accuracy was KL distance with respect to the exact, for each of the schemes. More precisely, if P_{exact} is the exact posterior probability distribution and P_{approx} is the approximate one (greedy or heuristic), the KL distance is $P_{approx}(X = a) \cdot \log(P_{approx}(X = a)/P_{exact}(X = a))$ averaged over all values a for each variable, all variables X and all instances of the problems. We also used absolute error instead of KL distance in the heuristic procedure, and the results are very strongly correlated, yielding almost the same accuracy, and we do not report them here.

We tried different types of networks: CPCS, random uniform, random noisy-or, grids. The results are very similar for most of them, and we only include here random networks and CPCS.

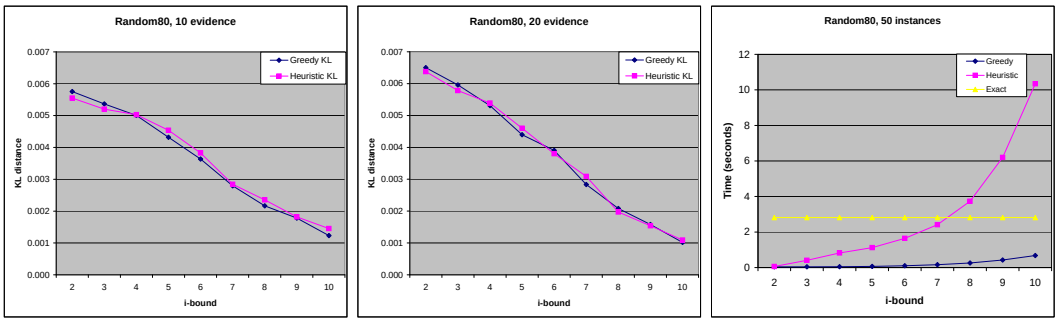


Figure 3.13: Random80, Heuristic KL vs. Greedy Partitioning

3.2.1.0.1 Random networks We generated networks with 80 variables, having induced width $w^* = 15$. We ran 50 instances for different values of evidence (0, 10, 20). Figure 3.13 shows the results for 10 and 20 evidence nodes. The heuristic partitioning is only better for small i-bounds, and this is a behavior we observed for all types of networks in general. The third graph in the figure shows time: the heuristic partitioning is slower, and takes more time even than the exact for i-bounds greater than 8.

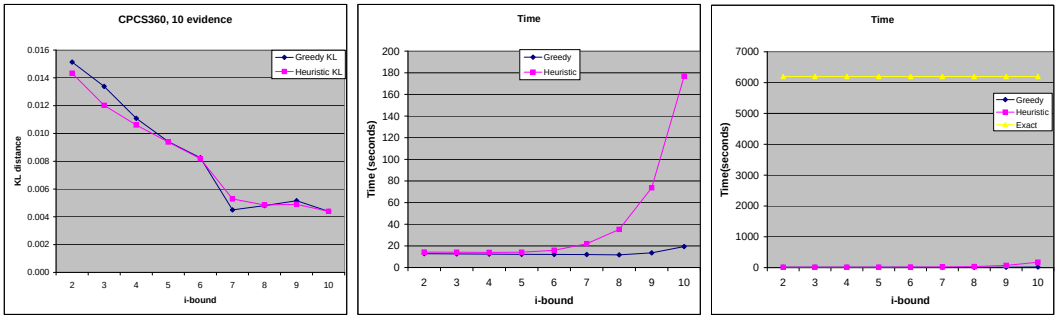


Figure 3.14: CPCS360, Heuristic KL vs. Greedy Partitioning

3.2.1.0.2 CPCS networks We show results for CPCS360 network, for 10 instances and 10 evidence in Figure 3.14. The induced width of the graph is $w^* = 20$ Again, the heuristic partitioning is slightly better for smaller values of the i-bound. The first time diagram shows, as expected, that the heuristic partitioning is slower than the greedy (by a cubic factor in the number of functions in a cluster). However, since this is a large network, the time taken by the heuristic is still small compared to the exact algorithm, even for $i = 10$, as can be seen in the second time graph.

3.2.2 Summary and future work

So far our preliminary results do not indicate that a more computationally intensive heuristic for partitioning improves the accuracy in a significant way. However we must experiment with networks that are less regular in order to determine the value of this method or any related one in terms of accuracy. If we will find that a more computationally intensive partitioning stategy yields substantial improvement in approximation we will then investigate ways of reducing the time overhead. The current approach does not look cost-effective at all based on our preliminary results. Based on our results so far, we can see that a conclusion of this work may either be a better partitioning approach, or a conclusive empirical demonstration that more computation in making the partitioning more effective is not the right approach. In fact, increasing the i-bound is likely to be a better way of investing any additional time or space.

3.3 Influence Diagrams and Planning

Influence diagrams [Howard & Matheson1984] provide a formalism for solving multiple decision problems in Bayesian theory. They extend belief networks by adding *decision variables* and *reward functional components*. An influence diagram is an acyclic graph with: random nodes, decision nodes and reward nodes. Formally

DEFINITION **3.3.1 (influence diagram)** *An* influence diagram *is a quadruple denoted by* $ID = (X, D, P, R)$*, where* $X = \{X_1, \ldots, X_n\}$ *is a set of chance variables on multivalued domains,* $D = \{D_1, \ldots, D_m\}$ *is a set of decision (action) nodes,* $P = \{P_1, \ldots, P_n\}$ *is a set of conditional probability tables (each* P_i *corresponding to* X_i*) and* $R = \{r_1, \ldots, r_j\}$ *is a set of rewards, each* r_i *being defined on a scope of chance and decision nodes.*

The main task in influence diagrams is to find the decision rules that maximize the total reward. In general, influence diagrams are required to satisfy a number of constraints. There must be a directed path that contains all the decision variables. Also, most variants assume a property of no-forgetting, in the sense that every decision node should depend on all the previous decision nodes. It is also possible not to enforce these requirements, and we give an example in which we do not insist upon no-forgetting.

Example 3.3.1 (oil wildcatter) *The graph in Figure 3.15 illustrates the famous problem of the oil wildcatter, adapted from [N. L. Zhang & Poole1994]. The chance variables are round, the decision variables are squared and the reward ones are diamonds. The graph shows that the decision to* Test *is made based on no information. The* Test results *depend on this decision and on the chance variable* Seismic structure*, which in turn depends on the unobservable variable* Oil underground*. The decision to* Drill *depends on the previous decision to* Test *and on the* Test result*. The decision on* Oil sale policy *is to be made considering the* Market information *and the amount of* Oil produced*. Other dependencies and independencies can be read from the graph in the same manner. There are four diamond nodes, the cost or reward nodes and the problem is to find the decisions that maximize the total reward.*

Influence diagrams are also a suitable framework for expressing planning problems under uncertainty, if they are formulated as *Markov Decision Processes (MDP)*. Influence diagrams subsume finite horizon MDPs and partially observable MDPs (POMDPs).

There are many variants of variable elimination algorithms for influence diagrams [Shachter1986, Shachter1988, Shachter1990, Tatman & Shachter1990, Shachter & Peot1992, Shenoy1992, Zhang1998, F. Jensen & Dittmer1994]. A bucket elimination framework which allows a complexity characterization based on graph parameters and an improvement over previous algorithms is presented in [Dechter2000b]. An initial investigation of anytime schemes for influence diagrams inspired by the mini-bucket idea is given in [Dechter2000a].

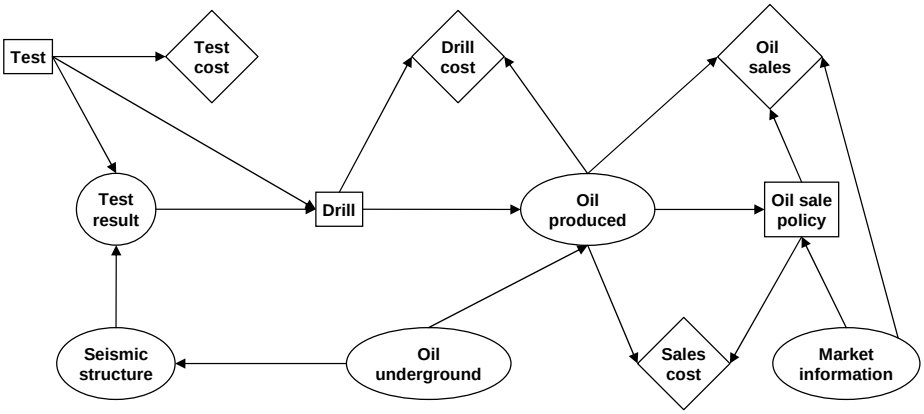


Figure 3.15: An influence diagram: the oil wildcatter problem

We plan to adapt and extend our algorithms for Bayesian networks to influence diagrams, and to investigate other types of possible hybrid algorithms which would be more suitable. An influence diagram without decision and reward nodes reduces to a Bayesian network. Therefore, influence diagrams provide a richer formalism and the extension of our algorithms is not necessarily an easy task. For example, for an iterative algorithm the semantics of the messages between different types of nodes (chance, decision, reward) needs to be addressed, as they are different than in Bayesian networks. Nevertheless, iterative algorithms similar to IBP have the advantage of being very fast and may provide good approximations for large networks.

Acknowledgments

I would like to thank my advisor, Professor Rina Dechter, for her guidance and support in doing this work. The ideas presented in section 3.2 emerged from discussions with Professor Sandy Irani. The experiments were done using the REES Toolkit developed by Radu Marinescu, with the base engine written by Kalev Kask. This work was supported in part by the NSF grant IIS-0086529 and MURI ONR award N00014-00-1-0617.

Appendix A

Tree Approximation for Belief Updating

Robert Mateescu, Rina Dechter and Kalev Kask

In Proceedings of The Eighteenth National Conference on Artificial Intelligence (AAAI-02), Edmonton, Canada.

Appendix B

Iterative Join-Graph Propagation

Rina Dechter, Kalev Kask and Robert Mateescu

In Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-02), Edmonton, Canada.

Appendix C

A Simple Insight into Iterative Belief Propagation's Success

Rina Dechter and Robert Mateescu

In Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-03), Acapulco, Mexico, forthcoming.

Bibliography

[Arnborg1985] Arnborg, S. A. 1985. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT* 25:2–23.

[Bidyuk & Dechter2001] Bidyuk, B., and Dechter, R. 2001. The epsilon-cuset effect in bayesian networks, http://www.ics.uci.edu/˜csp/r97a.pdf. Technical report, UCI.

[Cheng & Druzdzel2000] Cheng, J., and Druzdzel, M. 2000. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large bayesian networks. *Journal of Artificial Intelligence Research* 13:155–188.

[Cooper1990] Cooper, G. 1990. The computational complexity of probabistic inferences. *Artificial Intelligence* 393–405.

[Dagum & Luby1993] Dagum, P., and Luby, M. 1993. Approximating probabilistic inference in bayesian belief networks is np-hard. In *National Conference on Artificial Intelligence (AAAI-93)*.

[Dechter & Mateescu2003] Dechter, R., and Mateescu, R. 2003. A simple insight into iterative belief propagation's success. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI'03)*. To appear.

[Dechter & Pearl1987] Dechter, R., and Pearl, J. 1987. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence* 34:1–38.

[Dechter & Pearl1989] Dechter, R., and Pearl, J. 1989. Tree clustering for constraint networks. *Artificial Intelligence* 353–366.

[Dechter & Pearl1991] Dechter, R., and Pearl, J. 1991. Directed constraint networks: A relational framework for causal reasoning. In *IJCAI-91*, 1164–1170.

[Dechter & Rish1997] Dechter, R., and Rish, I. 1997. A scheme for approximating probabilistic inference. In *Proceedings of Uncertainty in Artificial Intelligence (UAI'97)*, 132–141.

[Dechter, Kask, & Larrosa2001] Dechter, R.; Kask, K.; and Larrosa, J. 2001. A general scheme for multiple lower bound computation in constraint optimization. *Principles and Practice of Constraint Programming (CP2000)*.

[Dechter, Mateescu, & Kask2002] Dechter, R.; Mateescu, R.; and Kask, K. 2002. Iterative join-graph propagation. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI'02)*, 128–136.

[Dechter1992] Dechter, R. 1992. Constraint networks. *Encyclopedia of Artificial Intelligence* 276–285.

[Dechter1996] Dechter, R. 1996. Bucket elimination: A unifying framework for probabilistic inference algorithms. In *Uncertainty in Artificial Intelligence (UAI'96)*, 211–219.

[Dechter2000a] Dechter, R. 2000a. An anytime approximation for optimizing policies under uncertainty. In *Workshop of Decision Theoretic Planning in (AIPS-2000)*.

[Dechter2000b] Dechter, R. 2000b. A new perspective on algorithms for optimizing policies under uncertainty. In *International Conference on Artificial Intelligence Planning Systems (AIPS-2000)*, 72–81.

[F. Jensen & Dittmer1994] F. Jensen, F. J., and Dittmer, S. 1994. From influence diagrams to junction trees. In *Tenth Conference on Uncertainty in Artificial Intelligence*, 367–363.

[Gottlob, Leone, & Scarcello2000] Gottlob, G.; Leone, N.; and Scarcello, F. 2000. A comparison of structural CSP decomposition methods. *Artificial Intelligence* 243–282.

[Howard & Matheson1984] Howard, R. A., and Matheson, J. E. 1984. *Influence diagrams*.

[Jeavons & Cooper1996] Jeavons, P. G., and Cooper, M. C. 1996. Tractable constraints on ordered domains. *Artificial Intelligence* 79:327–339.

[Jensen, Lauritzen, & Olesen1990] Jensen, F.; Lauritzen, S.; and Olesen, K. 1990. Bayesian updating in causal probabilistic networks by local computation. *Computational Statistics Quarterly* 4:269–282.

[Kask2001] Kask, K. 2001. Approximation algorithms for graphical models. Technical report, Ph.D. thesis, Information and Computer Science, University of California, Irvine, California.

[Kirousis1993] Kirousis, L. M. 1993. Fast parallel constraint satisfaction. *Artificial Intelligence* 64:147–160.

[Larrosa, Kask, & Dechter2001] Larrosa, J.; Kask, K.; and Dechter, R. 2001. Up and down mini-bucket: a scheme for approximating combinatorial optimization tasks. *Submitted*.

[Lauritzen & Spiegelhalter1988] Lauritzen, S., and Spiegelhalter, D. 1988. Local computation with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B* 50(2):157–224.

[Mackworth1977] Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence* 8(1):99–118.

[Mézard, Parisi, & Zecchina2002] Mézard, M.; Parisi, G.; and Zecchina, R. 2002. Analytic and algorithmic solution of random satisfiability problems. *Science* 297:812–815.

[Maier1983] Maier, D. 1983. The theory of relational databases. In *Computer Science Press, Rockville, MD*.

[Mateescu, Dechter, & Kask2002] Mateescu, R.; Dechter, R.; and Kask, K. 2002. Tree approximation for belief updating. In *Proceedings of The Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, 553–559.

[N. L. Zhang & Poole1994] N. L. Zhang, R. Q., and Poole, D. 1994. A computational theory of decision networks. *International Journal of Approximate Reasoning* 83–158.

[Pearl1988] Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

[R.J. McEliece & Cheng1997] R.J. McEliece, D. M., and Cheng, J.-F. 1997. Turbo decoding as an instance of pearl's belief propagation algorithm. *IEEE J. Selected Areas in Communication*.

[Roth1996] Roth, D. 1996. On the hardness of approximate reasoning. 82(1-2):273–302.

[Shachter & Peot1992] Shachter, R., and Peot, M. 1992. Decision making using probabilistic inference methods. In *Proceedings of Uncertainty in Artificial Intelligence (UAI92)*, 276–283.

[Shachter1986] Shachter, R. 1986. Evaluating influence diagrams. *Operations Research* 34.

[Shachter1988] Shachter, R. 1988. Probabilistic inference and influence diagrams. *Operations Research* 36.

[Shachter1990] Shachter, R. D. 1990. An ordered examination of influence diagrams. *Networks* 20:535–563.

[Shafer & Shenoy1990] Shafer, G. R., and Shenoy, P. 1990. Probability propagation. *Annals of Mathematics and Artificial Intelligence* 2:327–352.

[Shenoy1992] Shenoy, P. 1992. Valuation-based systems for bayesian decision analysis. *Operations Research* 40:463–484.

[Tatman & Shachter1990] Tatman, J., and Shachter, R. 1990. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics* 365–379.

[Welling & Teh2001] Welling, M., and Teh, Y. W. 2001. Belief optimization for binary networks: a stable alternative to loopy belief propagation. In *UAI2001*.

[Yedidia, Freeman, & Weiss2001] Yedidia, J. S.; Freeman, W.; and Weiss, Y. 2001. Generalized belief propagation. In *ANIPS-13*.

[Zhang1998] Zhang, N. L. 1998. Probabilistic inference in influence diagrams. *Computational Intelligence* 475–497.