Up and Down Mini-Buckets: A Scheme for Approximating Combinatorial Optimization Tasks

Kalev Kask, Javier Larrosa and Rina Dechter
Information and Computer Science
University of California at Irvine, USA

Abstract

The paper addresses the problem of computing lower bounds on the optimal costs associated with each unary assignment of a value to a variable in combinatorial optimization problems. This task is instrumental in probabilistic reasoning and is also important for the development of admissible heuristic functions that can guide search algorithms for optimal solutions. The paper presents UD-MB, a new algorithm that applies the mini-bucket elimination idea [Dechter and Rish, 1997] to accomplish this task. We show empirically that UD-MB may achieve a substantial speed up over a brute-force approximation method via mini-buckets.

1 Introduction

A *Combinatorial optimization problem* (COP) is defined by a finite set of *variables*, a set of finite *domains* and a set of *cost functions*. A *solution* is the *optimal cost* over the set of complete assignments of values to variables. It is well known that solving COP is NP-*hard*. COP arise in a large variety of domains including *artificial intelligence*, *chemistry*, *biology* and in numerous industrial applications. Solving techniques have been developed in fields such as *operations research* [Nemhauser and Wolsey, 1988], *constraint satisfaction* [Freuder and Wallace, 1992; Larrosa *et al.*, 1998] and *probabilistic reasoning* [Pearl, 1988].

In this paper we address the problem of approximating the *singleton-optimality problem*, defined as the optimal cost associated to each unary assignment of a value to a variable. This task is instrumental in probabilistic reasoning [Dechter and Rish, 1997] and is also important for the development of admissible heuristic functions that can guide search algorithms for optimal solutions. In particular, such lower bounds can be used in look-ahead methods for dynamic variable and value selections strategies that detect and prune infeasible values [Larrosa *et al.*, 1998].

The paper presents a new algorithm, UD-MB, that applies the mini-bucket elimination idea [Dechter and Rish, 1997]. It is built upon BTE, an exact algorithm for the singleton-optimality problem which is presented in a companion paper [Anon., 2001]. We show that by applying mini-bucket elimination twice, up and down, along a *bucket-tree* ordering, we may get a linear speed up over a brute-force approximation method. Such performance improvements are crucial if we are to apply the scheme at every node in search algorithms.

The structure of the paper is as follows: Section 2 introduces basic notation; Section 3 overviews BTE; Section 4 introduces algorithm UD-MB, the main contribution of this paper; Section 5 reports empirical results demonstrating the effectiveness of our scheme in Max-CSP and probabilistic decoding problems; and Section 6 concludes.

2 Preliminaries

A *combinatorial optimization problem* (COP) is defined by a tuple $\mathcal{P} =< X, D, F, \otimes, \Downarrow >$ where: $X = \{1, 2, \ldots, n\}$ is a set of variables. $D = \{D_1, \ldots, D_n\}$ is a collection of finite domains (D_i is the set of possible values for variable i). $F = \{f_1, \ldots, f_r\}$ is a set of *cost functions*[1] with *scopes* (i.e., arguments) $S = \{S_1, \ldots, S_r\}$ ($S_i \subseteq X$, $1 \leq i \leq r$). The domain of f_i is the Cartesian product of the domains of variables in S_i. Operator $\otimes$ combines functions (for example, *sum* or *product* of functions). Operator $\Downarrow$ projects a function over a subset of its scope (typically, eliminating variables by *minimization* or *maximization*)

Given a set of variables $Y \subseteq X$, the optimization task is

$$Opt_Y = \Downarrow_Y (\otimes_{j=1}^{r} f_j)$$

Typically, $Opt_\emptyset$ is the *problem solution*. For the sake of clarity and for its wide applicability, this paper will focus on *additive minimization problems* where variables are eliminated by minimization (i.e., $\Downarrow_Y \; f = \min_{S-Y}\{f\}$, S is the scope of f) and functions are combined by addition (i.e., $\otimes = +$). However, our results can be directly extended to general COP. We focus on the *singleton-optimality* task, defined as,

$$Opt_i = \min_{X-\{i\}} (\sum_{j=1}^{r} f_j), \quad \forall i \in X$$

Opt_i is a unary function over i which returns for each value $a \in D_i$ the optimum cost restricted to the assignment $i \leftarrow a$.

[1] In the *constraint satisfaction* community cost functions are called *soft constraints*.

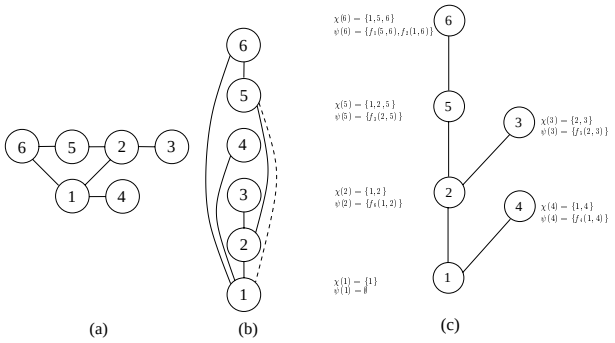

Figure 1: a) Primal graph of a COP instance; b) Its induced graph with respect the lexicographical ordering; c) Its bucket-tree.

Example 2.1 *Maximal Constraint Satisfaction (Max-CSP) is the problem of finding the minimum number of violated constraints of a total assignment in an overconstrained CSP [Freuder and Wallace, 1992]. Clearly, Max-CSP can be expressed as an additive minimization problem with constraints represented as 0/1 cost functions, where 0 denotes satisfaction and 1 denotes violation. In this context, Opt_i evaluated over $a \in D_i$ is the minimum number of constraint violations attainable if $i \leftarrow a$ is extended to a complete assignment.*

Definition 2.1 *[Dechter, 1999](primal and induced graph, induced width) A COP instance is associated with an undirected graph G called the* primal graph *such that problem variables are the graph nodes and two nodes are adjacent iff they are included in the same scope of some cost function. Given a graph G and an arbitrary ordering of its nodes o, its* induced graph $G^*(o)$ *is obtained by processing the nodes in reverse order. For each node all its earlier neighbors are connected, taking into account old and new edges created during the process. The induced width $w^*(o)$ is the maximum number of earlier neighbors over the nodes of $G^*(o)$.*

Example 2.2 *Figure 1.a depicts the primal graph of a COP instance defined over six variables $\{1, 2, \ldots, 6\}$ and six cost functions $\{f_1, \ldots, f_6\}$ with scopes: $S_1 = \{5, 6\}$, $S_2 = \{1, 6\}$, $S_3 = \{2, 5\}$, $S_4 = \{1, 4\}$, $S_5 = \{2, 3\}$ and $S_6 = \{1, 2\}$, respectively. Figure 1.b depicts the induced graph with respect to the lexicographical order. Solid lines depict original edges in the problem graph and the broken-line edge was added during the computation of the induced graph. The induced width along the ordering is 2.*

3 Bucket Tree Elimination (BTE)

Bucket Elimination (BE) [Dechter, 1999] is a unifying algorithmic framework that generalizes dynamic programming to accommodate probabilistic and deterministic reasoning tasks, as well as COP. It can be viewed as a one phase message-passing process along the so-called *bucket-tree* defined in terms of the induced graph as follows (details are given in [Anon., 2001]).

Definition 3.1 *[Anon., 2001] (bucket-tree, eliminator)*
Let $G^*(o)$ be the induced graph of a COP along ordering o. The bucket-tree *is a triplet $< \mathcal{T}, \chi, \psi >$:*

- *$\mathcal{T} = (V, E)$ is a tree whose nodes are the problem variables. The parent of i is the latest earlier neighbor of i in $G^*(o)$ (namely, the closest earlier neighbor of i in $G^*(o)$).*
- *χ is a labelling function for each node. $\chi(i)$ is the set of variables containing i and every earlier neighbor of i in $G^*(o)$.*
- *ψ is a labelling function for each node. $\psi(i)$ is the set of cost functions which contains every function having i as the highest variable in its scope.*

Let i and j be two adjacent nodes in $\mathcal{T}$. The eliminator *of i with respect j is defined as $elm(i, j) = \chi(i) - \chi(j)$,*

Example 3.1 *Figure 1.c depicts the bucket-tree of the induced graph in Figure 1.b. The parent of 5 is 2, $\psi(5)$ is $\{f_3(5, 2)\}$, $\chi(5)$ is $\{1, 2, 5\}$, the eliminator of 5 respect to 2 is $elm(5, 2) = \{5\}$ and the eliminator of 2 respect to 5 is $elm(1, 2) = \emptyset$.*

Note that visiting the nodes of $\mathcal{T}$ from last to first along ordering o produces a traversal from the leaves to the root of $\mathcal{T}$. In BE *messages* are computed in the nodes of the bucket-tree using functions contained in their *buckets*.

Definition 3.2 *(bucket, message) A message $m_{(i,j)}$ is a function computed in node i and sent to an adjacent node j. For each node i we define its* bucket *as the set of all functions in $\psi(i)$ and all messages received by i, namely,*

$$Bck(i) = \psi(i) \cup_{k \in N(i)} m_{(k,i)}$$

where $N(i)$ denotes the neighbors of i in $\mathcal{T}$. Message $m_{(i,j)}$ is computed when i has received the message $m_{(k,i)}$ from each of its neighbors other than j. The computation of $m_{(i,j)}$ consist of summing up all the functions in $Bck(i)$ excluding $m_{(j,i)}$ and subsequently eliminating the variables in the eliminator of i and j,

$$m_{(i,j)} = \min_{elm(i,j)} \{ \sum_{\{f \in Bck(i), f \neq m_{(j,i)}\}} f \}$$

Algorithm BE receives a bucket-tree $< \mathcal{T}, \chi, \psi >$ as input and it processes nodes from last to first (the bucket-tree is traversed in a top-down manner). Processing node i consists of computing message $m_{(i,j)}$, where j is the parent of i. Processing the root node yields an empty-scope function which is the problem solution. In general, the complexity of BE is exponential in the problem induced width. It was shown that,

Theorem 3.1 *[Dechter, 1999] Once BE terminates, the sum of all the functions in the first bucket yields function Opt_1. Namely,*

$$Opt_1 = \sum_{\{f \in Bck(1)\}} f$$

From Theorem 3.1 it is clear that the singleton-optimal functions can be derived by running BE n times, each one

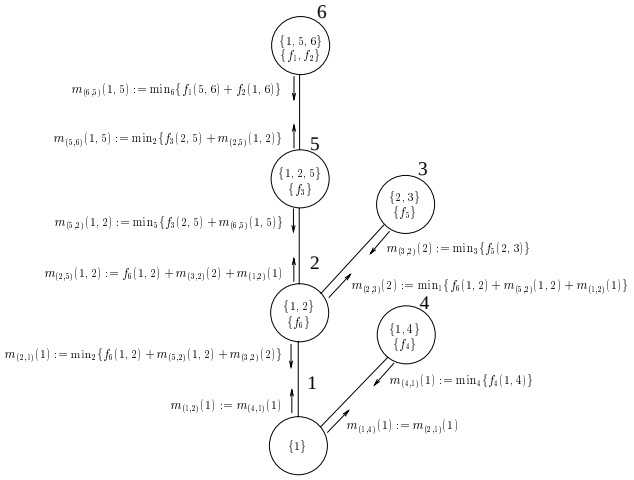

Figure 2: Execution of BTE with the problem of Figure 1.

with a different variable initiating the ordering. Each execution computes the singleton-optimality function for a different variable. However, in a companion paper [Anon., 2001] we introduce algorithm *Bucket Tree Elimination* (BTE) which accomplishes the task more efficiently. It extends BE by adding a second message-passing phase along the bucket-tree. At termination, each bucket can compute the singleton-optimality function for its variable.

In the second phase, nodes are processed from first to last (the tree is traversed in a bottom-up manner). Processing node i consist of computing message $m_{(i,j)}$ from i to every child of i, j.

Algorithm BTE also generalizes probabilistic reasoning algorithms over polytrees [Pearl, 1988]. In that domain, top-down and bottom-up messages are called λ and π, respectively. This distinction is convenient because the directionality of a message carries different semantics and it is the notation used in [Anon., 2001]. Here, we use common notation $m_{(i,j)}$ for both up and down messages, where $m_{(i,j)}$ is the message from i to j. This notation emphasizes that there is no algorithmic difference between messages going up and messages going down along the tree.

Algorithm 1 describes BTE. In the top-down phase (line 1.1) each node i computes the message $m_{(i,j)}$ to its parent j. In the bottom-up phase (line 1.2) the parent j of each node i computes the message $m_{(j,i)}$. After the second phase, every node has received the messages from all its neighbors (children and parent). Then BTE computes Opt_i, for every node i (line 1.3), by summing all the functions in $Bck(i)$ and subsequently eliminating every variable in $\chi(i)$ other than i. Formally,

$$Opt_i = \min_{\chi(i)-\{i\}} \Big(\sum_{\{f \in Bck(i)\}} f \Big)$$

Example 3.2 *Figure 2 shows the trace of running BTE for the problem in Figure 1. It depicts the bucket-tree and the flow of messages. Next to each edge (i, j) where j is the parent of i, there are the two messages: $m_{(i,j)}$ computed in the top-down phase and $m_{(j,i)}$ computed in the bottom-up phase. For instance, $m_{(6,5)} = \min_6\{f_1 + f_2\}$ and $m_{(5,6)} = \min_2\{f_3 + m_{(2,5)}\}$ are computed in the top-down and bottom-up phases, respectively. Function $Opt_5 = \min_{\{2,5\}}\{f_3 + m_{(6,5)} + m_{(2,5)}\}$ is the desired singleton optimality function for variable 5.*

Algorithm 1: BTE. $< \mathcal{T}, \chi, \psi >$ is the bucket-tree for the ordering $o = 1, 2, ..., n$. The first loop performs the top-down pass, the second loop performs the bottom-up pass and the third loop generates the singleton-optimality functions.

Procedure BTE$(< \mathcal{T}, \chi, \psi >)$
1.1 **for** $i = n$ *to* 1 **do**
  $j :=$ parent of i in $\mathcal{T}$;
  $m_{(i,j)} := \min_{elm(i,j)}\{\sum_{\{f \in Bck(i) - m_{(j,i)}\}} f\}$;
end
1.2 **for** $i = 1$ *to* n **do**
  $j :=$ parent of i in $\mathcal{T}$;
  $m_{(j,i)} := \min_{elm(j,i)}\{\sum_{\{f \in Bck(j) - m_{(i,j)}\}} f\}$;
end
1.3 **for** $i = 1$ *to* n **do**
  $Opt_i := \min_{\chi(i)-\{i\}}\{\sum_{\{f \in Bck(i)\}} f\}$;
end
return $\{Opt_i | \ 1 \leq i \leq n\}$

In [Anon., 2001], we showed that BTE can provide up to linear speed-up over the alternative of running BE n times.

4 Up and Down Mini-Bucket Elimination

4.1 UD-MB

The time and especially the space complexity of BTE renders the algorithm infeasible for problems with high induced width. In this section we introduce a new algorithm called *Up and Down Mini-Bucket Elimination* (UD-MB) that approximates the singleton-optimality functions by providing lower bounds. UD-MB computes a set of unary functions LB_i such that $LB_i \leq Opt_i$ for all $i \in X$ (we say that $f \leq g$ iff f and g have the same scope and for every tuple t, $f(t) \leq g(t)$). These approximation functions are computed using *mini-buckets* [Dechter and Rish, 1997]. The algorithm is parameterized by z which allows to trade time and space for accuracy. Increasing z, the algorithm provides more accurate approximations, at the cost of higher time and space complexity.

The idea is to approximate the computations by partitioning buckets into mini-buckets and by eliminating the appropriate variables in each mini-bucket, independently.

Consider an edge (i, j) of $\mathcal{T}$. The exact computation of $m_{(i,j)}$ was defined in the previous section as,

$$m_{(i,j)} = \min_{elm(i,j)} \Big(\sum_{\{f \in Bck(i), f \neq m_{(j,i)}\}} f \Big)$$

Its approximation, $M_{(i,j)}$, is based on a partition of $Bck(i) - \{m_{(j,i)}\}$ into mini-buckets $\{mb(1), \ldots, mb(p)\}$ such that the number of variables mentioned in each is bounded by z. In

each mini-bucket we compute,

$$m^k_{(i,j)} = \min_{elm(i,j)} \{ \sum_{f \in mb(k)} f \}$$

The sum of functions in the set $M_{(i,j)} = \{m^1_{(i,j)}, \ldots, m^p_{(i,j)}\}$ is the basis of a lower bound for $m_{(i,j)}$. Namely,

$$(\sum_{k=1}^{p} m^k_{(i,j)}) \leq m_{(i,j)}$$

In summary, a message in UD-MB is the set of functions $M_{(i,j)}$ whose sum is a lower bound function for $m_{(i,j)}$. The lower bound of Opt_i, denoted LB_i, is obtained by applying the same idea,

$$LB_i = \sum_{k=1}^{q} (\min_{\chi(i)-\{i\}} \{ \sum_{\{f \in mb(k)\}} f \})$$

Algorithm 2 describes UD-MB. Its structure is identical to BTE. The only difference is that in UD-MB computations are approximated via mini-buckets. The algorithm uses procedure ApMB(G, V, z), which approximates the sum of a set of functions G and the subsequent elimination of a set of variables V, subject to accuracy parameter z. First, functions in G mentioning variables in V are collected in a set A (line 2.4) and partitioned into mini-buckets (line 2.5). Next, variables in V are eliminated from each mini-bucket (line 2.6). The procedure returns the result of processing each mini-bucket and all the unprocessed functions.

Algorithm 2: UD-MB. $< \mathcal{T}, \chi, \psi >$ is the bucket-tree for the ordering $o = 1, 2, ..., n$ and z is the accuracy parameter.

Procedure UD-MB($< \mathcal{T}, \chi, \psi >, z$)
2.1 **for** $i = n$ *to* 1 **do**
  $j :=$ parent of i in $\mathcal{T}$;
  $M_{(i,j)} :=$ApMB($Bck(i) - M_{(j,i)}, elm(i,j), z$);
end
2.2 **for** $i = 1$ *to* n **do**
  $j :=$ parent of i in $\mathcal{T}$;
  $M_{(j,i)} :=$ApMB($Bck(j) - M_{(i,j)}, elm(j,i), z$);
end
2.3 **for** $i = 1$ *to* n **do** $LB_i :=$ApMB($Bck(j), \chi(i) - \{i\}, z$);
return $\{LB_i | \ 1 \leq i \leq n\}$

Procedure ApMB(G, V, z)
2.4 $A :=$functions in G that mention variables in V;
2.5 (z)mini-buckets from A, $\{mb(1), \ldots, mb(p)\}$;
2.6 **for** $k := 1$ *to* p **do** $g_k := \min_V \{\sum_{f \in mb(k)} f\}$;
return $(G - A \cup \{g_1, \ldots, g_p\})$

The top-down phase of UD-MB coincides with the original MB algorithm [Dechter and Rish, 1997], although its description looks somewhat different. The only difference, however, is that in our description mini-bucket functions do not *jump* to their destination bucket according to the *highest variable in the scope* rule. Instead, they *travel down* the bucket-tree.

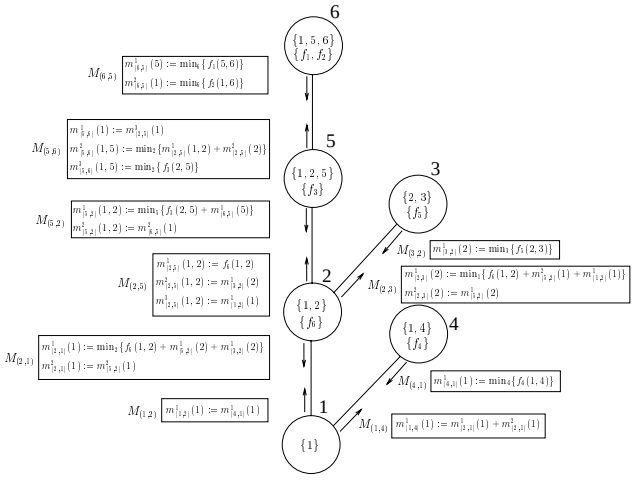

Figure 3: Execution of UD-MB with $z = 2$ with the problem in Figure 1.

4.2 Example

Figure 3 shows the trace of running UD-MB with $z = 2$ on the problem in Figure 1. The top-down phase starts computing $M_{(6,5)}$ by calling procedure ApMB(G, V, z) with $G = \{f_1(5,6), f_2(1,6)\}$ and $V = \{6\}$. Each function from G is put into one mini-bucket (their combined arity surpasses the value of z) and variable 6 is eliminated from each mini-bucket, generating two functions: $m^1_{(6,5)}(5) = \min_6\{f_1(5,6)\}$ and $m^2_{(6,5)}(1) = \min_6\{f_2(1,6)\}$. Message $M_{(6,5)} = \{m^1_{(6,5)}(5), m^2_{(6,5)}(1)\}$ is then sent to node 5. Next, $M_{(5,2)}$ is computed by procedure ApMB(G, V, z) with $G = \{f_3(2,5), m^1_{(6,5)}(5), m^2_{(6,5)}(1)\}$ and $V = \{5\}$. Only f_3 and $m^1_{(6,5)}$ mention variable 5. They can be placed in the same mini-bucket and variable 5 eliminated, producing a new function $m^1_{(5,2)}(2) = \min_5\{f_3(2,5) + m^1_{(6,5)}(5)\}$. Function $m^2_{(6,5)}(1)$, not processed in node 5, is passed down as $m^2_{(5,2)}(1) = m^2_{(6,5)}(1)$. The process continues until all top-down messages are computed.

The bottom-up phase starts computing $M_{(1,2)}$, which is obtained by procedure ApMB(G, V, z) with $G = \{m^1_{(4,1)}(1)\}$ and $V = \emptyset$. Since there is no variable to eliminate, the function is not processed ($m^1_{(1,2)}(1) = m^1_{(4,1)}(1)$). Message $M_{(1,2)} = \{m^1_{(1,2)}(1)\}$ is sent to node 2. Next, $M_{(2,3)}$ is computed by procedure ApMB(G, V, z) with $G = \{f_6(1,2), m^1_{(5,2)}(2), m^2_{(5,2)}(1), m^1_{(1,2)}(1)$ and $V = \{1\}$. All functions mentioning variable 1 can be placed in one mini-bucket where the variable is eliminated, yielding $m^1_{(2,3)}(2) = \min_1\{f_6(1,2) + m^2_{(5,2)}(1) + m^1_{(1,2)}(1)\}\}$. Function $m^1_{(5,2)}(2)$ is not processed ($m^2_{(2,3)}(2) = m^1_{(5,2)}(2)$), because it does not mention variable 1. Message $M_{(2,3)} = \{m^1_{(2,3)}(2), m^1_{(2,3)}(2)\}$ is sent to node 3.

After the bottom-up phase, lower bounds can be computed. For instance, LB_2 is com-

puted by procedure ApMB(G, V, z) with $G = \{f_6(1,2), m^1_{(5,2)}(2), m^2_{(5,2)}(1), m^1_{(3,2)}(2), m^1_{(1,2)}(1)\}$ and $V = \{1\}$. All functions mentioning variable 1 can be placed in one mini-bucket where the variable is eliminated, $g(2) = \min_1\{f_6(1,2) + m^2_{(5,2)}(1) + m^1_{(1,2)}(1)\}$. The lower bound function is obtained summing $g(2)$ with the unprocessed functions, $LB_2 = g(2) + m^1_{(5,2)}(2) + m^1_{(3,2)}(2)$.

4.3 Correcness and Complexity

The following two Lemmas are useful to prove the correctness of UD-MB. The first one proves that ApMB is a correct way to obtain lower bounds. The second one proves that messages in UD-MB are lower bounds of messages in BTE.

Lemma 4.1 *[Dechter and Rish, 1997] Let H be the output of ApMB(G, V, z). Then,*

$$\sum_{h \in H} h \leq \min_V \{\sum_{g \in G} g\}$$

Lemma 4.2 *Let $< \mathcal{T}, \chi, \psi >$ be a bucket-tree. For every edge (i,j) of $\mathcal{T}$,*

$$(\sum_{k=1}^{p} m^k_{(i,j)}) \leq m_{(i,j)}$$

where $m_{(i,j)}$ is the message computed by BTE and $M_{(i,j)} = \{m^1_{(i,j)}, \ldots, m^p_{(i,j)}\}$ is the message computed by UD-MB.

From Lemma4.2 it follows that,

Theorem 4.1 *UD-MB is correct. Namely,*

$$LB_i \leq Opt_i, \ \forall x_i \in X$$

The following Theorem establishes the complexity of UD-MB.

Theorem 4.2 *The complexity of UD-MB is $O(n \times r \times exp(z))$, where n is the number of variables, r is the number of cost functions and z is the accuracy parameter.*

It was shown in [Dechter and Rish, 1997; Larrosa, 2000] that the complexity of MB is $O(r \times exp(z))$ and it is easy to see that the brute force solving approach for the singleton optimality problem of running MB n times (to which we will refer as nMB) has complexity $O(n \times r \times exp(z))$. Consequently, the speed up of UD-MB over nMB is not captured by this worst-case bounds. However, the following result shows that UD-MB is never worse than nMB and the speed-up for particular instances ranges from n to 1.

Theorem 4.3 *Let t_{MB}, t_{UD-MB} and t_{nMB} be the time cost of executing MB, UD-MB and nMB with the same z value. Then, for all problem instance,*

$$t_{MB} \leq t_{UD-MB} \leq t_{nMB}$$

and the previous bound is tight. Namely, there are problem instances for which UD-MB is n times faster than nMB as well as instances where there is no speed-up.

5 Empirical Results

We have run a number of experiments in order to investigate the average speedup of UD-MB over the alternative of nMB. We will be comparing two algorithms: UD-MB(z) and nMB(z). For every problem instance, we run both UD-MB(z) and nMB(z), and record their running times $t_{UD-MB(z)}$ and $t_{nMB(z)}$. The speedup is defined as $t_{nMB(z)}/t_{UD-MB(z)}$.

Given a problem instance, we first create a *min-degree* ordering of the variables[2] that is used to create the bucket-tree for the UD-MB algorithm. nMB requires n different ordering. The ordering initiated by variable i is obtained by swapping the first variable with i in the min-degree ordering. We have tested the performance of UD-MB and nMB on two different domains: random binary Max-CSP problems and random probabilistic decoding problems.

5.1 Max-CSP

Max-CSP is an optimization version of Constraint Satisfaction and was described in Example 2.1. We used the well known four parameters random model, $< N, K, C, T >$, where N is the number of variables, K is the domain size, C is the number of constraints, and T is the tightness of each constraint (see [Larrosa *et al.*, 1998] for details).

In Tables 1 and 2 we have the results of experiments with two sets of Max-CSP problems: $N = 100$, $K = 3$, $200 \leq C \leq 400$ and $N = 50$, $K = 5$, $75 \leq C \leq 135$. In these experiments the value of T is irrelevant since the complexity of the algorithms do not depend on it. Each row in the table corresponds to problems with a fixed number of constraints (column 1). In column 2 we have the average induced width along the min-degree ordering. In columns 3 through 8 we report the average speed-up for different values of z. In our experiments, the average CPU time per problem for nMB(z) ranges from a fraction of a second ($z = 2$) to as much a 5 minutes ($z = 7$).

We observe that the speedup is sometimes as large as an order of magnitude. We also see that the speedup is correlated with the induced width w^* - the larger the induced width the smaller the speedup. This observation coincides with the theoretical result obtain for classes of problems (see proof of Theorem4.3) where the speed up was inversely proportional to the induced witdth of the problem. Another interesting observation from Table 1 is that, when the constraint graph is sparse ($C = 200$), the speedup decreases as z increases, while for dense graphs ($C = 400$) the speedup increases with z.

5.2 Probabilistic Decoding

Channel coding is a systematic way to add redundancy to a source of binary information to be transmitted through a noisy channel. Redundancy is used to correctly retrieve the information at destination. *Probabilistic decoding* is the task of finding the most probable source of information, given the received information and the redundancy pattern.

[2] The variable with the smallest degree is placed at the end of the ordering, all its neighbors are connected and it is removed from the graph. The process is repeated until every variable has been selected.

C	w^*	z=2	z=3	z=4	z=5	z=6	z=7
$N = 100$, $K = 3.\ 50$ instances.							
200	21.2	10.8	10.1	9.20	8.36	7.77	7.82
250	27.9	6.87	6.86	6.60	6.29	6.10	6.16
300	33.7	4.49	4.97	5.04	5.06	5.14	5.28
350	38.9	3.42	4.02	4.22	4.35	4.50	4.73
400	43.0	2.65	3.36	3.68	3.88	4.07	4.34

Table 1: Speedup of UD-MB(z) over nMB(z). Max-CSP.

C	w^*	z=2	z=3	z=4	z=5	z=6	z=7
$N = 50$, $K = 5.\ 50$ instances.							
75	7.10	7.63	6.63	6.36	6.49	7.11	8.93
90	9.48	5.98	4.64	4.59	4.76	5.11	5.44
105	11.1	4.49	3.68	3.64	3.79	3.97	4.34
120	13.9	3.72	3.17	3.12	3.32	3.44	3.70
135	16.3	3.29	2.73	2.67	2.81	3.02	3.21

Table 2: Speedup of UD-MB(z) over nMB(z). Max-CSP.

In our experiments we generate random vectors of N information bits, we add N additional redundant bits using random linear block codes (each redundant bit is the XOR of P randomly chosen source bits). Finally, we simulate the transmission by adding white Gaussian noise (this experimental setting has already been used in [Kask and Dechter, 1999]). The general task is to retrieve the source bits from the bits at destination. The problem can be formulated as a COP where the N source bits are the problem variables (with binary domains). There are $N \times P$ cost functions of arity P, each one defined by a probability distribution. Functions are combined by multiplication (i.e., $\otimes = \times$) and variables are eliminated by maximization (i.e., $\Downarrow_Y f = \max_{S-Y} f$)

We consider the task of computing the *most probable explanation value* (MPE) for each variable-value pair, which is equivalent to the singleton optimallity problem.

In Table 3 we have the results of experiments with a set of random coding problems with $N = 100$ variables and $3 \leq P \leq 7$. The results are similar to the case of Max-CSP. Again we observe that the speed-up is sometimes as large as an order of magnitude. We also see that the speedup is correlated with the induced width w^* - the larger the induced width the smaller the speedup.

6 Conclusions and Future Work

Efficient techniques for bounding the optimum of combinatorial optimization problems have been widely studied in Operations Research, Constraint satisfaction, heuristic search and probabilistic reasoning [**?**; Pearl, 1988; de Givry *et al.*, 1997; Schiex, 2000].

P	w^*	z=2	z=4	z=6	z=8	z=10
N=100. 50 instances.						
3	7.31	13.1	15.8	13.6	12.6	18.6
4	12.0	6.64	8.12	7.00	6.90	6.65
5	15.1	5.29	6.39	5.48	6.21	5.83
6	17.7	6.39	5.36	4.61	4.66	4.81
7	19.6	6.70	8.64	9.20	9.75	8.78

Table 3: Speedup of UD-MB(z) over nMB(z). Decoding.

The bucket-elimination scheme [Dechter, 1999] provides a unifying framework that facilitates the development of general methods for combinatorial optimization problems. It also helps to the cross-fertilization of ideas across different fields. Mini-bucket elimination [Dechter and Rish, 1997] is an approximation scheme based on bucket elimination. In this paper we have introduced UD-MB, a new algorithm that uses mini-buckets to compute lower bounds for the singleton-optimality problem. Approximating methods for this problem are useful in probabilistic reasoning. They are also useful as generic look-ahead procedures to be used within algorithms that search the optimum, where the lower bounds are used to detect and prune infeasible values as well as to guide the next step of search [Freuder and Wallace, 1992; Larrosa *et al.*, 1998].

We showed that UD-MB provides a linear speed-up over a brute-force application of mini-buckets for classes of problems. Our experiments on two different domains demonstrated the practical effectiveness of our approach. The integration of UD-MB with search methods remains as future work.

References

[Dechter and Rish, 1997] R. Dechter and I. Rish. A Scheme For Approximating Probabilistic Inference. In *Proceedings of UAI'97*.

[Dechter, 1999] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[Freuder and Wallace, 1992] E.C. Freuder and R.J. Wallace. Partial Constraint Satisfaction *Artificial Intelligence*, 58:21–70, 1992.

[de Givry *et al.*, 1997] S. de Givry, G. Verfaillie and T. Schiex. Bounding the optimum of Constraint Optimization Problems. In *Proceeding of CP'97*.

[Kask and Dechter, 1999] K. Kask and R. Dechter Branch and Bound with mini-bucket heuristics. *Proc. of IJCAI'99*.

[Anon., 2001] Anonymous. Unifying Tree-Decomposition Schemes for Automated Reasoning. Sub. to *IJCAI'2001*.

[Larrosa *et al.*, 1998] J. Larrosa, P. Meseguer and T. Schiex. Maintaining Reversible DAC for Max-CSP. *Artificial Intelligence*, 107:149–163, 1998.

[Larrosa, 2000] J. Larrosa. On the Time Complexity of Bucket Elimination Algorithms. *UCI Tech. Rep.*, 2000.

[Nemhauser and Wolsey, 1988] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, 1988.

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

[Schiex, 2000] T. Schiex. Soft Arc Consistency. In *Proceeding of CP'2000*