A General Scheme for Multiple Lower Bound Computation in Constraint Optimization*

Rina Dechter[1] and Kalev Kask[1] and Javier Larrosa[2]

[1] University of California at Irvine (UCI)
{dechter, kkask}@ics.uci.edu
[2] Universitat Politecnica de Catalunya (UPC)
larrosa@lsi.upc.es

Abstract. Computing lower bounds to the best-cost extension of a tuple is an ubiquous task in constraint optimization. A particular case of special interest is the computation of lower bounds to all singleton tuples, since it permits domain pruning in Branch and Bound algorithms. In this paper we introduce MCTE(z), a general algorithm which allows the computation of lower bounds to arbitrary sets of tasks. Its time and accuracy grows as a function of z allowing a controlled tradeoff between lower bound accuracy and time and space to fit available resources. Subsequently, a specialization of MCTE(z) called MBTE(z) is tailored to computing lower bounds to singleton tuples. Preliminary experiments on Max-CSP show that using MBTE(z) to guide dynamic variable and value orderings in branch and bound yields a dramatic reduction in the search space and, for some classes of problems, this reduction is highly cost-effective producing significant time savings and is competative against specialized algorithms for Max-CSP.

1 Introduction

One of the main successes in constraint satisfaction is the development of *local consistency* properties and their corresponding *consistency enforcing algorithms* [18, 11]. They allow to *infer* and make explicit constraints that are implicit in the problem. Most useful in practice are consistency enforcing algorithms that *filter out* values that cannot participate in a solution. Filtering algorithms can be embedded into a search-based solver, propagating the effect of the current assignment towards future variables by pruning infeasible values under the current assignment [19, 3, 6].

Several attempts have been made in recent years to extend the notion of local consistency to *constraint optimization* problems [4, 5, 20]. The main difficulty being that inferred soft constraints cannot be carelessly added to the problem, due to the non-idempotency of the operator used to aggregate costs. A whole line of research mitigates this problem by extending only directional local consistency

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

to soft constraints and focuses on its most practical use: detecting *lower bounds* for the best extension of tuples [22, 9, 16, 20, 13]. When there is an upper bound on the maximum cost of a solution, tuples having a lower bound higher than this bound cannot participate in an optimal solution and can be viewed as infeasible (i.e., a *nogood*). As in the CSP context, lower bounds for values (singleton tuples) are of special interest, because they can be used to filter out infeasible values.

This paper, introduces MCTE(z), a general decomposition method for *multiple* lower bound computation, and MBTE(z), its specialization to singleton tuples. Our scheme is built on top of *cluster-tree elimination* (CTE), a tree-based decomposition schema which unifies several approaches for automated reasoning tasks. Algorithm MCTE(z) approximates CTE using a partitioning idea similar to *mini-buckets* [9]. The parameter z controls its complexity (which is exponential in z) as well as its accuracy, and can therefore be tuned to best fit the available resources.

After describing CTE and introducing MCTE (sections 3 and 4), we describe in Section 5, MBTE(z), an instantiation of MCTE(z) which is designed to compute lower bounds for singleton tuples. As we show in the empirical section, MBTE(z) facilitates a parameterized dynamic look-ahead method for variable and value ordering heuristics in branch and bound. The parameter controls its pruning power and overhead, and can therefore adjust branch and bound to different levels of problem hardness: while low accuracy suffices for easy problems, higher accuracy may be more cost-effective when problems grow harder and larger.

The singleton optimality problem can be solved by n runs of the mini-bucket elimination MBE(z) [9] which we will call nMBE(z). We contrast MBTE(z) against this alternative nMBE(z). We argue that for the same level of accuracy (same parameter z), MBTE(z) is considerably more efficient (up to linear speed-up). Time efficiency is of the essence when the ultimate goal is to use these algorithms at every node of a branch and bound search. Indeed, our preliminary experiments on Max-CSP (Section 7) support theory-based expectations regarding MBTE(z)'s accuracy as a function of z as well as its speed-up relative to nMBE(z). Most significantly, however, we demonstrate the potential of embedding MBTE(z) in Branch and Bound in algorithm called BBBT(z) , showing a dramatic pruning power in search space relative to competitive Branch and Bound algorithms, which, for some (but not all) problem classes is highly cost-effective.

2 Preliminaries

Definition 1 (sum of functions, variable elimination). *Let f and g be two functions defined over $var(f)$ and $var(g)$, respectively. Then,*

1. *The* sum *of f and g, denoted $(f+g)$, is a new function defined over $var(f) \cup var(g)$ which returns for each tuple the sum of values given by f and g, $(f+g)(t) = f(t) + g(t)$*

2. *The* elimination *of x_i from f by minimization, denoted* $\min_{x_i} f$*, is a new function defined over $var(f)-\{x_i\}$ which returns for each tuple the minimum cost extension to f,* $(\min_{x_i} f)(t) = \min_{a \in D_i}\{f(t,a)\}$ *where D_i denotes the domain of variable x_i and $f(t,a)$ denotes the evaluation of f on the tuple t extended with value a assigned to x_i. We use* $(\min_S f)(t)$*, to denote the elimination of a set of variables $S \subseteq var(f)$.*

Definition 2 (lower bound function). *Let f and g be two functions defined over the same scope (same set of arguments). We say that g is a lower bound of f, denoted $g \leq f$, iff $g(t) \leq f(t)$, for all t.*

Definition 3 (constraint optimization problem (COP), constraint graph). *A* constraint optimization problem *(COP) is a triplet $P = < X, D, F >$, where $X = \{x_1, \ldots, x_n\}$ is a set of variables, $D = \{D_1, \ldots, D_n\}$ is a set of finite domains and $F = \{f_1, \ldots, f_m\}$ is a set of constraints. Constraints can be either* soft *(i.e, cost functions) or* hard *(i.e, sets of allowed tuples). Without loss of generality we assume that hard constraints are represented as (bi-valued) cost functions. Allowed and forbidden tuples have cost 0 and ∞, respectively. $var(f_i)$ denotes the scope of function f_i. The* constraint graph *of a problem P has the variables as its nodes, and two nodes are connected if they appear in a scope of a function in F.*

Definition 4 (optimization tasks,global and singleton). *Given a COP instance P, a set of optimization tasks is defined by $Z = \{Z_i\}_{i=1}^k$, where $Z_i \subseteq X$ and for each Z_i the task is to compute a function g_i over Z_i, such that $g_i(t)$ is the best cost attainable by extending t to X. Formally, $g_i(t) = \min_{X-Z_i}(\sum_{j=1}^m f_j)$. A* global optimization *is the task of finding the best global cost, namely $Z = \{\emptyset\}$.* Singleton optimization *is the task of finding the best-cost extension to every singleton tuple (x_i, a), namely $Z = \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\}$.*

Bucket elimination (BE) [7] is a decomposition algorithm for global optimization. The algorithm starts by partitioning the set of constraints into n buckets, one per variable. Then variables are processed one by one. For each variable x_i, a new constraint h_i is computed from its bucket, summarizing the effect of x_i on the rest of the problem. Variable x_i is eliminated and replaced by h_i. After the last elimination, only an empty-scope constraint (i.e, a constant function) containing the cost of the best solution remains in the problem. The time and space complexity of bucket-elimination algorithms is time and space exponential in a graph parameter called induced-width (to be defined later).

Mini-bucket elimination (MBE) [9] is an approximation of BE that mitigates its high time and space complexity. When processing variable x_i, its bucket is partitioned into mini-buckets. Each mini-bucket is processed independently, producing bounded arity functions which are cheaper to compute and store. This paper extends the idea of mini-bucket elimination from variable-elimination based architecture to tree-decomposition-based.

3 Cluster-Tree Elimination (CTE)

In this Section we present *cluster-tree elimination* (CTE), a general decomposition method for a variety of reasoning tasks. The algorithm is not new, it is a unifying description of variants of such algorithms appearing in the past 2 decades both in the constraints community and the probabilistic reasoning community [17, 8, 21, 12]. Nevertheless, we find it necessary to describe the scheme in some detail since it will allow presenting our approximation in the most general setting. We also provide refined complexity analysis. CTE is based on the concept of *tree-decomposition*. We use notation borrowed from [12].

Definition 5 (tree-decomposition, separator, eliminator). *Given a COP instance P, a* tree-decomposition *is a triplet $< T, \chi, \psi >$, 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 $\psi(v) \subseteq F$ that satisfy the following conditions:*

1. *For each function $f_i \in F$, there is* exactly *one vertex $v \in V$ such that $f_i \in \psi(v)$. Vertex v satisfies that $var(f_i) \subseteq \chi(v)$.*
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 called the* running intersection *property.*

Let (u, v) be an edge of a tree-decomposition, the separator *of u and v is defined as $sep(u, v) = \chi(u) \cap \chi(v)$; the* eliminator *of u and v is defined as $elim(u, v) = \chi(u) - sep(u, v)$.*

Definition 6 (tree-width, hyper-width, maximum separator size). *The tree-width of a tree-decomposition is $tw = \max_{v \in V} |\chi(v)| - 1$, its hyper-width is $hw = \max_{v \in V} |\psi(v)|$, and its maximum separator size is $s = \max_{(u,v) \in E} |sep(u, v)|$*

Definition 7 (valid tree-decomposition). *We say that the tree-decomposition $< T, \chi, \psi >$ is* valid *for a set of optimization tasks $Z = \{Z_i\}_{i=1}^k$ if there is a non-empty set of vertices for each task Z_i defined as $\{v \in V |\ \chi(v) = Z_i\}$. Such vertices are called* solution-vertices[1].

Example 1. Consider a constraint optimization problem P with six variables $\{x_1, \ldots, x_6\}$ and six constraints $\{f_1, \ldots, f_6\}$ with scopes: $var(f_1) = \{x_5, x_6\}$, $var(f_2) = \{x_1, x_6\}$, $var(f_3) = \{x_2, x_5\}$, $var(f_4) = \{x_1, x_4\}$, $var(f_5) = \{x_2, x_3\}$ and $var(f_6) = \{x_1, x_2\}$, respectively. Figure 2 depicts two different tree-decompositions for P. The tree on the left is valid for $Z = \{\{x_1, x_5, x_6\}, \{x_1, x_2, x_5\}\}$ (v_1 and v_2 are solution-vertices for the first and second tasks, respectively), the tree on the right is valid for $Z = \{\{x_1, x_4\}, \{x_2, x_6\}\}$.

In the tree of Fig. 2.a, the separator of v_2 and v_3 is $sep(v_2, v_3) = \{x_1, x_2\}$, the eliminator of v_2 and v_3 is $elim(v_2, v_3) = \{x_5\}$, the eliminator of v_3 and v_2 is $elim(v_3, v_2) = \{x_3, x_4\}$.

[1] Normally, solution-vertices are only implicitly required. In our formulation we require them explicitly in order to simplify the algorithmic presentation.

Procedure CTE
Input: A COP instance P, a set of tasks $Z = \{Z_i\}_{i=1}^{k}$ and a valid tree-decomposition $< T, \chi, \psi >$.
Output: An augmented tree such that each solution-vertex for Z_i contains the solution to task Z_i.

Repeat

1. Select and edge (u, v) such that $m_{(u,v)}$ has not been computed and u has received messages from all adjacent vertices other than v
2. $m_{(u,v)} \leftarrow \min_{elim(u,v)}(\sum_{g \in cluster(u), g \neq m_{(v,u)}} g)$ (where $cluster(u) = \psi(u) \cup \{m_{(w,u)} | (w, u) \in T\}$)

Until all messages have been computed

Fig. 1. Algorithm cluster-tree elimination (CTE)

Algorithm CTE (Figure 1) computes the solution to a set of tasks by processing a valid tree-decomposition. It works by computing *messages* that are sent along edges in the tree. Message $m_{(u,v)}$ is a function computed at vertex u and sent to vertex v. For each edge, two messages are computed. One in each direction. Message $m_{(u,v)}$ can be computed as soon as all incoming messages to u other than $m_{(v,u)}$ have been received. Initially, only messages at leaves qualify. As leaves compute their messages, their adjacent vertices also qualify for computation. The process goes on until all messages have been computed. The order in which qualifying messages are computed is irrelevant.

The set of functions associated with a vertex u augmented with the set of incoming messages is called a *cluster*,

$$cluster(u) = \psi(u) \cup_{(w,u) \in T} m_{(w,u)}$$

A message $m_{(u,v)}$ is computed as the sum of all functions in $cluster(u)$ excluding $m_{(v,u)}$ and the subsequent elimination of variables in the eliminator of u and v. Formally,

$$m_{(u,v)} = \min_{elim(u,v)} (\sum_{g \in cluster(u), g \neq m_{(v,u)}} g)$$

The algorithm terminates when all messages are computed. A solution to task Z_i is contained in any of its solution-vertices, as the sum of all functions in the cluster, $\sum_{g \in cluster(u)} g$.

Example 2. Figure 2 shows the execution trace of CTE with the two tree-decompositions, as the messages sent along the tree edges. For instance, in Figure 2.*a* only messages from leaves, $m_{(v_1,v_2)}$ and $m_{(v_3,v_2)}$, can be initially computed. Lets suppose that CTE starts computing $m_{(v_1,v_2)}$. Since $\psi(v_1) = \{f_1, f_2\}$ and the eliminator of v_1 and v_2 is $elim(v_1, v_2) = \{x_6\}$, the message is computed as $m_{(v_1,v_2)} = \min_{x_6}(f_1 + f_2)$. At this point we can compute either $m_{(v_3,v_2)}$ or $m_{(v_2,v_3)}$. The later is computed by summing all functions in $\psi(v_2)$ with $m_{(v_1,v_2)}$,

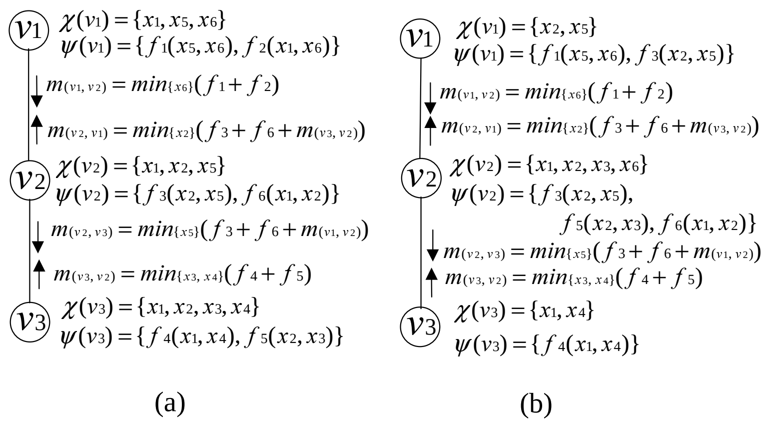

Fig. 2. Execution-trace of CTE on two different tree-decompositions.

and eliminating variables in $elim(v_2, v_3) = \{x_5\}$. The process goes on until all four messages have been computed.

Once messages are computed, solutions are contained in the solution-vertices. For instance, the solution to task $\{x_1, x_2, x_5\}$ is contained in $cluster(v_2)$ as $m_{(v_1,v_2)} + m_{(v_3,v_2)} + f_3 + f_6$. Similarly, the solution to task $\{x_1, x_5, x_6\}$ is contained in $cluster(v_1)$ as $f_1 + f_2 + m_{(v_2,v_1)}$.

Theorem 1 (correctness [17, 8, 21]). *Algorithm CTE is correct. Namely, for each solution-vertex v of Z_i, $\sum_{g \in cluster(v)} g = \min_{X-Z_i}(\sum_{j=1}^{n} f_j)$*

We can show that,

Theorem 2 (complexity). *The complexity of CTE is time $O(r \cdot (hw + dg) \cdot d^{tw+1})$ and space $O(r \cdot d^s)$, where r is the number of vertices in the tree-decomposition, hw is the hyper-tree width, dg is the maximum degree among the vertices, tw is the tree-width, d is the largest domain size in the problem and s is the maximum separator size.*

Proof. Regarding time, let $T_{CTE}(u, v)$ be the time required by CTE to compute $m_{(u,v)}$. It can be shown that $T_{CTE}(u, v)$ is bounded by $c_u \cdot d^{|\chi(u)|}$, where c_u is the size of cluster u, $c_u = |\psi(u)| + |\{(v, u) \in E\}|$. It is clear that $|\chi(u)| \leq tw$ and $c_u \leq hw + dg$. Therefore, $T_{CTE}(u, v)$ is $O((hw + dg) \times d^{tw})$. There are $r - 1$ edges, yielding $2(r - 1)$ different messages. Therefore, the total complexity is bounded by,

$$d^{tw} \cdot 2r \cdot (ht + dg)$$

Regarding space, message $m_{(u,v)}$ is defined over $sep(u, v)$, which has size bounded by the maximum separator size s. Therefore, the message has at most d^s entries. There are $2(r - 1)$ different messages to store, which yields $O(r \cdot d^s)$ entries. □

Procedure MiniBucketsApprox
Input: a set of functions G, a set of ordered variables V, parameter z
Output: a set of functions $\{h^j\}_{j=1}^k$ that provide a lower bound as $\sum_{j=1}^k h^j \leq \min_V(\sum_{g \in G} g)$
for each $x_i \in V$ from last to first **do**
    $G' \leftarrow \{g \in G \mid x_i \in var(g)\}$
    compute $\mathcal{P}(G') = \{\mathcal{P}_j\}_{j=1}^k$ a z-partition of G'
    $h^j \leftarrow \min_{x_i}(\sum_{g \in \mathcal{P}_j} g)$, for $j = 1..k$
    $G \leftarrow (G - G') \cup \{h^j\}$
Returns: G

Fig. 3. Procedure MiniBucketsApprox(V, G, z).

Since CTE is time and space exponential in tw and s, respectively, low width tree-decompositions are desirable(note that $tw + 1 \geq s$). Finding the minimum width decomposition-tree is known to be NP-complete [1], but various approximation algorithms are available [2].

4 Mini-Cluster-Tree Elimination (MCTE)

The time and especially the space complexity of CTE renders the method infeasible for high-width tree-decompositions. One way to decrease the algorithm's complexity is to bound the size of the messages' arity to a predefined size z. This idea, called *mini-buckets*, was first introduced in the bucket elimination context [9]. Here we extend it from approximating variable elimination to the more general setting of approximating tree-decomposition algorithms.

Let G be a set of functions having variable x_i in their scope. Suppose we want to compute a target function as the sum of functions in G and subsequently eliminate variable x_i (i.e, $\min_{x_i}(\sum_{g \in G} g)$). If exact computation is too costly, we can partition G into sets of functions $\mathcal{P}(G) = \{\mathcal{P}_j\}_{j=1}^k$ called mini-buckets, each one having a combined scope of size bounded by z. Such a partition is called a *z-partition*. If more than one partition is possible, any one is suitable. Subsequently, a bounded arity function h^j is computed at each mini-bucket $\mathcal{P}_j$ as the sum of all its included functions and the elimination of x_i (i.e, $h^j = \min_{x_i}(\sum_{g \in \mathcal{P}_j} g)$). The result is a set of functions $\{h^j\}_{j=1}^k$ which provides a lower bound to the target function. Namely, $\sum_j h^j \leq \min_{x_i} \sum_{g \in G} g$.

If more than one variable has to be eliminated, the process is repeated for each, according to a predefined ordering. Procedure MiniBucketsApprox(V, G, z) (Fig. 3) describes this process. Each iteration of the loop performs the elimination of one variable.[2]

[2] Another option is to eliminate all variables *at once* from each mini-bucket (i.e, $h^j = \min_V(\sum_{g \in \mathcal{P}_j} g)$). While correct, it will provide less accurate lower bounds.

Applying this idea to CTE yields a new algorithm called *mini-cluster-tree elimination* (MCTE(z)) The algorithm can be obtained by replacing line 2 in CTE by:

2. $M_{(u,v)} \leftarrow MiniBucketApprox(elim(u,v), cluster(u) - M_{(v,u)}, z)$
(where $cluster(u) = \psi(u) \cup \{M_{(w,u)} | (w,u) \in T\}$)

It works similar to CTE except that each time that a message has to be computed, the set of functions required for the computation are partitioned into *mini-buckets*, producing a set of *mini-messages* that are transmitted along the corresponding edge. Thus, in MCTE(z), a message is a *set* of bounded arity functions, $M_{(u,v)} = \{m^j_{(u,v)}\}$ (note that we use upper-case to distinguish MCTE(z) messages from CTE messages). A cluster is now the union of all messages and all functions in a node, $cluster(u) = \psi(u) \cup_{(w,u)\in T} M_{(w,u)}$ The message $M_{(u,v)}$ is computed by calling $\mathsf{MiniBucketsApprox}(V, G, k)$ with $V = elim(u,v)$ and $G = cluster(u) - M_{(v,u)}$. When all messages have been computed a lower bound to task Z_i is contained in its solution-vertex v as the sum of all the functions in its cluster, $\sum_{g \in cluster(v)} g$.

Example 3. Figure 4 shows the execution-trace of MCTE(2) with our running example and the tree-decomposition of Fig. 2.*a*. For instance, the computation of $M_{(v_3,v_2)}$ requires a 2-partition of $(cluster(v_3) - M_{(v_2,v_3)}) = \{f_4(x_1,x_4), f_5(x_2,x_3)\}$. The only 2-partition here is $\mathcal{P}_1 = \{f_4\}$ and $\mathcal{P}_2 = \{f_5\}$, which yields a two-functions message $M_{(v_3,v_2)} = \{\min_{x_4}(f_4), \min_{x_3}(f_5)\}$.

Lemma 1. *[9] Let H be the result of a call to procedure $\mathsf{MiniBucketsApprox}(V, G, z)$. Then,*

$$\sum_{h \in H} h \leq \min_V \left(\sum_{g \in G} g\right)$$

Theorem 3 (correctness). *MCTE(z) is correct. Namely, given a valid tree-decomposition, lower bounds for every task Z_i can be computed. Specifically, if u is a solution-vertex of task Z_i then, $\sum_{g \in cluster(u)} g \leq \min_{X - Z_i}(\sum_{j=1}^n f_j)$*

Proof (sketch).

Consider a tree-decomposition $< T, \chi, \psi >$ and the corresponding executions of CTE and MCTE(z). Let (u,v) be an arbitrary edge and $m_{(u,v)}$ and $M_{(u,v)} = \{m^j_{(u,v)}\}_{j=1}^k$ the messages computed by CTE and MCTE(z), respectively. From Theorem 1, we have that $m_{(u,v)} = \min_{X - Z_i}(\sum_{j=1}^n f_j)$. Therefore, it suffices to prove that $\sum_{j=1}^k m^j_{(u,v)} \leq m_{(u,v)}$.

Let $H = \cup_{(v,w)\in E, w \neq u} M_{(v,w)}$ and $I = \cup_{(v,w)\in E, w\neq u} m_{(v,w)}$. By definition,

$$\{m^j_{(u,v)}\}_{j=1}^k = \mathsf{MiniBucketsApprox}(elim(u,v), \psi(u) \cup H, z)$$

and

$$m_{(u,v)} = \min_{elim(u,v)} \left(\sum_{g \in \psi(u) \cup I} g\right)$$

Let D be the longest path from node u to a leaf, without passing through node v. The proof is by induction over $D(u,v)$.

– $D = 0$: node u is a leaf and $H = I = \emptyset$. From the definition of $M_{(u,v)}$ and the previous Lemma we have that

$$\sum_{j=1}^{k} m^j_{(u,v)} \leq \min_{elim(u,v)} (\sum_{g \in \psi(u)} g)$$

The right-hand part is $m_{(u,v)}$. Therefore, the claim is proved.
– $D > 0$: From the definition of $M_{(u,v)}$ and the Lemma we have that

$$\sum_{j=1}^{k} m^j_{(u,v)} \leq \min_{elim(u,v)} (\sum_{g \in \psi(u) \cup H} g)$$

From the induction hypothesis, $\sum m^j_{(v,w)} \leq m_{(v,w)}$, for all $(v,w) \in E, w \neq u$. Therefore,

$$\min_{elim(u,v)} (\sum_{g \in \psi(u) \cup H} g) \leq \min_{elim(u,v)} (\sum_{g \in \psi(u) \cup I} g)$$

The right-hand part is $m_{(u,v)}$. Therefore, the claim is proved.

□

In order to analyze the complexity of MCTE(z) we define a new labeling ψ^*, which depends on the tree-decomposition structure.

Definition 8 (ψ^*, induced hyper-width (hw^*)). *Let $P =< X, D, F >$ be a COP instance and $< T, \chi, \psi >$ be a tree-decomposition. We defined a labeling function ψ^* over nodes in the tree as, $\psi^*(v) = \{f \in F | \ var(f) \cap \chi(v) \neq \emptyset\}$, where $v \in T$. The* induced hyper-width *of a tree-decomposition is defined as, $hw^* = \max_{v \in V} |\psi^*(v)|$*

Observe that $\psi^*(u)$ is a superset of $\psi(u)$ which includes those cost functions not in $\psi(u)$ that *may travel* to cluster u via message-passing. It can be shown that the *induced hyper-width* bounds the maximum number of functions that can be in a cluster, and therefore the number of mini-buckets in a cluster. Namely $hw^* \geq \max_{v \in V} |cluster(v)|$. Note that $hw \leq hw^* \leq m$, where hw is the hyper-width and m is the number of input functions.

Lemma 2. *Algorithm MiniBucketsApprox(V, G, z) is time $O(|G| \cdot d^z)$*

Proof. The set of functions G defines a constraint optimization problem $P' =< X', D', G >$, where the *implicit* set of variables X' is the set of variables mentioned in G. Observe that running MiniBucketsApprox(X', G, z) is equivalent to running MBE(z) on P'. In [14] it was proved that MBE(z) is time $O(r \cdot d^z)$, where r is the number of input functions. Therefore, MiniBucketsApprox(X', G, z) is $O(|G| \cdot d^z)$. MiniBucketsApprox(V, G, z) cannot be more costly, since only a subset V of X' is eliminated. □

$$v_1 \quad \chi(v_1) = \{x_1, x_5, x_6\}$$
$$\psi(v_1) = \{f_1(x_5, x_6), f_2(x_1, x_6)\}$$
$$\downarrow M_{(v_1,v_2)} = \{m^1_{(v_1,v_2)} = min_{\{x_6\}}(f_1),\ m^2_{(v_1,v_2)} = min_{\{x_6\}}(f_2)\}$$
$$\uparrow M_{(v_2,v_1)} = \{m^1_{(v_2,v_1)} = min_{\{x_2\}}(m^2_{(v_3,v_2)} + f_6),\ m^2_{(v_2,v_1)} = f_6,\ m^3_{(v_2,v_3)} = m^2_{(v_1,v_2)}\}$$
$$v_2 \quad \chi(v_2) = \{x_1, x_2, x_5\}$$
$$\psi(v_2) = \{f_3(x_2, x_5), f_6(x_1, x_2)\}$$
$$\downarrow M_{(v_2,v_3)} = \{m^1_{(v_2,v_3)} = min_{\{x_5\}}(m^1_{(v_1,v_2)} + f_3),\ m^2_{(v_2,v_1)} = min_{\{x_2\}}(f_3),\ m^3_{(v_2,v_1)} = m^2_{(v_3,v_2)}\}$$
$$\uparrow M_{(v_3,v_2)} = \{m^1_{(v_3,v_2)} = min_{\{x_4\}}(f_4),\ m^2_{(v_3,v_2)} = min_{\{x_3\}}(f_5)\}$$
$$v_3 \quad \chi(v_3) = \{x_1, x_2, x_3, x_4\}$$
$$\psi(v_3) = \{f_4(x_1, x_4), f_5(x_2, x_3)\}$$

Fig. 4. An execution trace of MCTE(2).

Theorem 4 (complexity). *Given a problem P and a tree-decomposition T having induced hyper width hw^*, MCTE(z) is time and space $O(r \cdot hw^* \cdot d^z)$, where r is the number of nodes in T, and d bounds the domain size.*

Proof. Let $T_{MCTE}(u, v)$ denote the time required by MCTE(z) to compute $M_{(u,v)}$. Since $M_{(u,v)}$ is computed by calling MiniBucketsApprox with $G = cluster(u) - M_{(v,u)}$, and $|cluster(u) - M_{(v,u)}| \leq hw^*$, the previous Lemma establishes that $T_{MCTE}(u, v)$ is $O(hw^* \cdot d^z)$. MCTE computes $2(r - 1)$ messages. Therefore, the total complexity is

$$O(d^z \times r \times hw^*)$$

□

Clearly, increasing z is likely to provide better lower bounds at a higher cost. Therefore, MCTE(z) allows trading lower bound accuracy for time and space complexity. There is no guaranteed improvement however. The actual lower bounds provided by MCTE(z) depend on z and the z-partitions used during the execution.

5 MBTE(z): Computing Bounds to Singleton Tuples

There are a variety of ways in which valid tree-decompositions can be obtained. We analyze a special decomposition called bucket-trees, which is particularly suitable for the multiple singleton optimality task. The concept of bucket-tree is inspired from viewing bucket-elimination algorithms as message-passing along a tree [7]. A *bucket-tree* can be defined over the *induced graph* relative to a variable ordering.

Definition 9 (induced graph, induced width [7]). *An* ordered constraint graph *is a pair (G, o), where G is a constraint graph and $o = x_1, ..., x_n$ is an* *ordering of its nodes. Its* induced graph *is obtained by processing the nodes recursively, from last to first. When node x_i is processed, all its earlier neighbors are connected. The* induced width $w^*(o)$ *is the maximum number of lower indexed neighbors over all vertices of the induced graph.*

Definition 10 (bucket-tree). *Given an ordered induced graph $G^*(o)$ of a problem P along o, a* bucket-tree $< T, \chi, \psi >$ *is defined as follows. i. There is a vertex v_i associated with each variable x_i. The parent of v_i is v_j iff x_j is the latest earlier neighbor of x_i in $G^*(o)$ (namely, x_j is the closest earlier neighbor of x_i in $G^*(o)$). ii. $\chi(v_i)$ contains x_i and every earlier neighbor of x_i in $G^*(o)$. 3. $\psi(v_i)$ contains every constraint having x_i as the highest variable in its scope.*

Notice that in a bucket-tree, vertex v_1, the root, is a solution-vertex for the task $\{x_1\}$.

The bucket-tree can be augmented with solution-vertices for each singleton-optimality task. A vertex u_i with $\chi(u_i) = \{x_i\}$ and $\psi(u_i) = \emptyset$ is added for $i = 2..n$. Vertex v_i is the parent of u_i. Subsequently, we define algorithm *Bucket-tree elimination* (BTE) to be CTE applied to the augmented bucket-tree.

Example 4. Figure 5 shows the execution-trace of BTE on our running example. Observe that messages from u-nodes to v-nodes do not need to be sent because they are null functions ($\psi(u_i) = \emptyset$).

Observe that BTE computes the exact singleton optimality problem. Observe also that BTE can be viewed as a two-phases algorithm. : a first phase in which top-down messages are computed and a second phase in which bottom-up messages are computed. The first phase (top-down) is equivalent to *bucket elimination* (BE) [7]: Cluster v_i is the bucket of x_i. Incoming messages are new functions derived from higher buckets and added to the bucket of x_i. Computing message $m_{(v_i,p(v_i))}$, where $p(v_i)$ is the parent of v_i, performs the elimination of variable v_i and produces a new function (the message) that is sent to a lower bucket (the parent of v_i).

Next, *Mini bucket tree elimination* (MBTE(z)) is defined by approximating BTE via mini-buckets or, equivalently, by executing MCTE(z) over the augmented bucket-tree. BTE and MBTE(z) process the same tree-decomposition but in MBTE(z) clusters are z-partitioned, producing mini-bucket-based messages (BTE and MBTE are not explicitly described). From Theorem 4 we can conclude,

Theorem 5 (complexity). *Given a variable ordering, the complexity of MBTE(z) is $O(n \cdot hw^* \cdot d^z)$, when n is the number of variables and hw^* is the bucket-tree induced hyper-width.*

MBTE vs nMBE: Given a variable ordering, it is easy to see that the variable-elimination based, *mini-bucket elimination* MBE(z) is equivalent to the top-down message-passing of MBTE(z). In particular, running MBE(z) n times, an algorithm that we call nMBE(z), each time having a different variable initiating the ordering, is an alternative for the singleton optimality problem. MBTE and

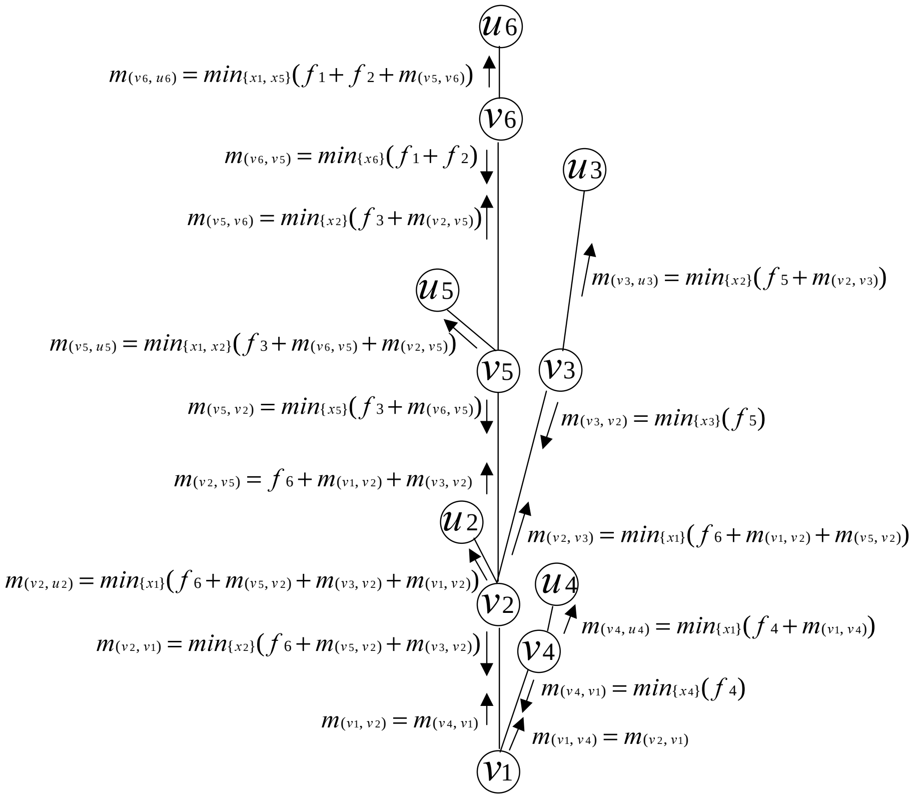

Fig. 5. An execution trace of BTE for the task of computing the best extension of all singleton tuples. If only top-down messages are considered, the algorithm is equivalent to BE

nMBE are closely related in terms of accuracy. Specifically, if MBE(z) is executed each time with the appropriate variable orderings, both approaches will produce exactly the same bounds, when using the same bucket-partitioning strategy. Clearly, however, MBTE(z) is always more efficient than multiple executions of MBE(z), since MBE(z) repeats message computation at different executions. The following Theorem summarizes these properties.

Theorem 6. *Let P be a constraint optimization problem and o a variable ordering. Lets consider the execution of MBTE(z) over the bucket-tree relative to o.*

- *(Accuracy) For each variable x_i, there is an ordering o_i initiated by x_i such that executing MBE(z) with o_i produces the same lower bound as MBTE(z) for task $\{x_i\}$, provided that both algorithms use the same criterion to select z-partitions.*
- *(time comparison) Let nMBE(z) be applied using the n previously defined o_i orderings. Then, every message computed by MBTE(z) is also computed*

by nMBE(z), and there are some messages that are computed multiple times (up to n) by nMBE(z).

Proof (sketch). We only define ordering o_i. Checking that MBE(z) with o_i produces the same lower bounds and computes the same messages comes quite straightforward. We need some operations over sequences. Let α and S be a sequence and a subset of variables, respectively (for instance $\alpha = (x_3, x_8, x_2, x_5)$ and $S = \{x_2, x_3\}$), then: α' is its reverse ($\alpha' = (x_5, x_2, x_8, x_3)$), $\alpha \downarrow S$ is the projection of the sequence onto the set ($\alpha \downarrow S = (x_3, x_2)$) and $\alpha \downarrow \overline{S}$ is the projection onto the complementary of S ($\alpha \downarrow \overline{S} = (x_8, x_5)$)

Let $o = \alpha, x_i, \beta$ (namely, α and β are the subsequences that come prior and posterior to x_i in the ordering o). Let $anc(i)$ be the set of ancestors of variable x_i in the bucket-tree relative to o. We define:

- $\gamma = \alpha \downarrow \overline{anc(i)}$
- $\delta = \alpha \downarrow \underline{(anc(i) \cap \psi(v_i)})$
- $\epsilon = \alpha \downarrow anc(i) - \psi(v_i)$

The ordering o_i is defined as $o_i = x_i, \delta, \epsilon', \gamma, \beta$ □

Thus, MBTE(z) is never worse than nMBE(z). Since the complexity of running MBE(z) n times is $O(n \cdot m \cdot d^z)$ and MBTE(z) is $O(n \cdot hw^* \cdot d^z)$, significant gains are expected when hw^* is smaller relative to m.

6 Comparison of MBTE with Soft Arc-consistency

Soft arc-consistency (SAC) [20] is the most general of a sequence of bounds for singleton optimization based in different forms of arc-consistency. In this Section we show through two examples that there is no dominance relation between them. Therefore, a potencial superiority of one over the other can only be empirically demonstrated. In our discussion, we consider the most general algorithm for SAC. Namely, the algorithm that after achieving soft arc-consistency, is allowed to iterate non-deterministically projecting and extending cost functions in order to increase, if possible, the available bounds.

Example 5. Consider a COP instance with three variables x_1, x_2, x_3, two values per domain a, b and a soft constraint $f_{i,j}(x_i, x_j)$ for each pair of variables such that its micro-structure is depicted in Figure 6.a. Non-zero costs between pairs of values are represented by links. All links indicate cost 1 (the default), except the link between values a of variables x_2 and x_3, which has cost 2.

Lets consider the singleton bound for the tuple (x_1, a). SAC can project the f_{23} onto x_2 producing the equivalent problem depicted in Figure 6.b (unary costs are depicted next to their values). Then, the algorithm can project what *remains* in f_{23} onto x_3 (Figure 6.c) . Subsequently, unary costs in variables x_2 and x_3 can be extended towards f_{21} and f_{31} (Figure 6.d). Finally, f_{21} and f_{31} can be projected onto x_1 (Figure 6.e). The resulting lower bound for the tuple (x_1, a) is 2. It can be easily observed that MBTE(2) computes a lower bound of 1, only.

The reason of the superiority of SAC in this example is that MBTE(2) need to transform the problem into an acyclic structure and each cost function has a single path to be propagated through. On the other hand, SAC works with the (possibly cyclic) constraint graph and may propagate different projections of the same function through different paths that may eventually converge. In the example, SAC *projects* information from f_{23} to x_2 and x_3. This information converges in x_1. MBTE(2), depending on the variable ordering will project f_{23} to either x_2 or x_3, but not to both of them.

Example 6. Consider a COP instance with three variables, two-value domains and three binary functions f_{ij}, one for each pair of variables. Function f_{ij} assigns cost 0 to tuples with the same value for the two variables, and cost 1 to tuples with different variables. SAC is completely useless in this problem (no projection or extension is possible) Lets consider MBTE(3) with the lexicographic ordering. It can be shown that it obtains a lower bound of one 1 for all singleton tuples. It could be argued that the superiority of MBTE is because larger scopes are allowed (as a matter of fact, in this example MBTE(3) provides exact values for singleton optimization). However, it is not the case, the reason is that SAC can only project cost functions one by one, while MBTE can sum several functions as long as the arity does not surpass z and then project from the result. In a simplistic way, SAC is only allowed to compute bounds by using $\sum_{f \in F} \min_V f$, while MBTE can perform $\min_V \sum_{f \in F} f$. Clearly, there are many situations where $\sum_{f \in F} \min_V f$ produces null functions, while $\min_V \sum_{f \in F} f$ produces non-null functions. This fact allows MBTE to outperform SAC in some problem instances.

7 Empirical Results

We performed a preliminary experimental evaluation of the performance of MBTE(z) on solving the singleton optimality task. *i*) we have investigated the performance of MBTE(z) against its obvious brute-force alternative – nMBE(z), and showed that MBTE(z) achieves a significant speedup over nMBE(z). *ii*) we demonstrated that as expected, MBTE(z) accuracy grows as function of z, thus allowing a trade-off between accuracy and complexity. *iii*) we evaluated the effectiveness of MBTE(z) in improving Branch and Bound search.

All our experiments are done using the Max-CSP task as a sample domain. Max-CSP is an optimization version of Constraint Satisfaction and its task is to find an assignment that satisfies the most constraints. We use its formulation as a minimization problem where each constraint is a cost function that assigns a cost 1 to each nogood and cost 0 to each allowed tuple. We used the well known four parameter 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 [15] for details).

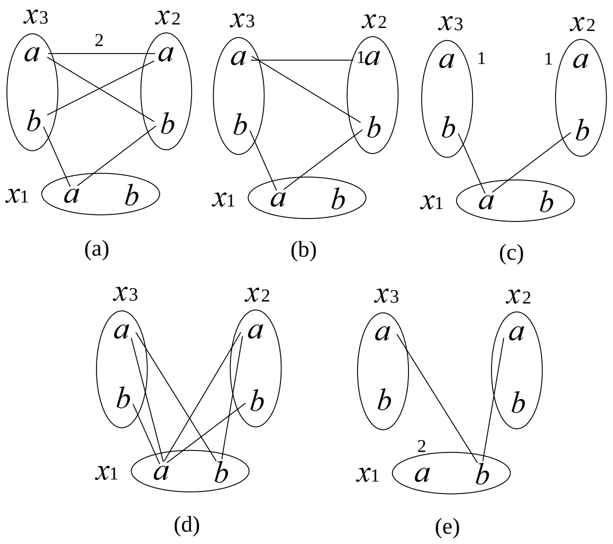

Fig. 6. Tree iterations of SAC.

7.1 Speedup of MBTE(z) vs. nMBE(z)

We have run a number of experiments in order to investigate the speedup of MBTE(z) over nMBE(z). We will be comparing two algorithms: MBTE(z) and nMBE(z). For every problem instance, we run both MBTE(z) and nMBE(z), and record their running times $t_{MBTE(z)}$ and $t_{nMBE(z)}$. The speedup is defined as $t_{nMBE(z)}/t_{MBTE(z)}$. Given a problem instance, we use a *min-degree* ordering of the variables[3] for creating the bucket-tree for the MBTE algorithm.

In Tables 1 and 2 we have the results of experiments with two sets of Max-CSP problems: $N = 100$, $K = 3$, $200 \le C \le 400$ and $N = 50$, $K = 5$, $75 \le C \le 135$. 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 nMBE(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. Another interesting observation from Table 1 is that, when the constraint graph is sparse ($C = 200$), the speedup

[3] 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.

$N = 100,\ K = 3$. 50 instances.							
C	w^*	z=2	z=3	z=4	z=5	z=6	z=7
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 MBTE(z) over nMBE(z).

$N = 50,\ K = 5$. 50 instances.							
C	w^*	z=2	z=3	z=4	z=5	z=6	z=7
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 MBTE(z) over nMBE(z).

decreases as z increases, while for dense graphs ($C = 400$) the speedup increases with z.

7.2 Accuracy of MBTE(z)

One of the main advantages of MBTE(z) is that it represents a whole family of algorithms, parameterized by z. When incorporated within a search algorithm as a heuristic function, MBTE(z) allows a trade-off between complexity of heuristic computation and its accuracy. Small values of z yield heuristics that are easy to compute, but have low accuracy. Higher values of z yield heuristics that are harder to compute, but have higher accuracy.

In Tables 3 and 4 we have the results of experiments with two sets of Max-CSP problems: N=50, K=3, C=90, $1 \leq T \leq 8$, and N=40, K=5, C=65, $10 \leq T \leq 20$. On each problem instance we ran MBTE(z) for different values of z, as well as the exact algorithm BTE. We compute the cost for each value of each variable, and compare approximated values computed by MBTE(z) against the exact value computed by BTE.

In column 1 we have the tightness and in the last column we report the exact average cost per singleton assignment. In the remaining columns we have the average running time of MBTE(z) for each z, and the average absolute error of the singleton variable cost computed by MBTE(z) for each z and T. By comparing the exact cost with the absolute error, we can get an idea of the relative error.

As expected, as z increases, the running time of MBTE(z) increases, and the error decreases, eventually reaching 0.

$N = 50$, $K = 3$, $C = 90$. $w^* = 9.5$. 100 instances.										
	z=2	z=3	z=4	z=5	z=6	z=7	z=8	z=9	z=10	
	time									
	0.02	0.03	0.06	0.11	0.26	0.55	1.24	2.70	5.25	
T	abs error									cost
1	0	0	0	0	0	0	0	0	0	0.003
2	0.007	0.006	0.005	0.004	0.003	0.002	0.002	0.0006	0.0003	0.016
3	0.66	0.64	0.61	0.57	0.50	0.42	0.33	0.23	0.14	0.89
4	4.46	4.26	3.71	3.31	2.68	2.18	1.49	0.88	0.28	5.27
5	9.16	8.34	6.61	5.39	4.30	3.34	2.41	1.45	0.57	12.7
6	12.7	10.4	7.81	6.49	5.11	3.88	2.76	1.65	0.74	22.9
7	13.2	10.8	7.97	6.44	5.12	3.78	2.67	1.58	0.61	34.9
8	10.7	10.1	7.63	6.17	4.92	3.60	2.57	1.59	0.63	51.1

Table 3. Accuracy of MBTE(z).

$N = 40$, $K = 5$, $C = 65$. $w^* = 6.8$. 100 instances.										
	z=2	z=3	z=4	z=5	z=6	z=7	z=8	z=9	z=10	
	time									
	0.03	0.06	0.19	0.63	2.02	6.19	14.0	18.7	18.7	
T	abs error									cost
10	0.013	0.011	0.007	0.004	0.003	0.001	0.0002	0	0	-
11	0.045	0.039	0.031	0.026	0.017	0.008	0.002	0	0	-
12	0.171	0.157	0.14	0.11	0.073	0.029	0.0035	0	0	-
13	0.699	0.664	0.61	0.51	0.37	0.18	0.024	0	0	-
14	1.79	1.72	1.53	1.22	0.88	0.40	0.093	0.002	0	-
15	3.16	2.94	2.46	1.83	1.16	0.47	0.06	0	0	-
16	4.97	4.49	3.51	2.54	1.69	0.70	0.10	0.006	0	-
17	6.51	5.52	4.06	2.85	1.75	0.77	0.14	0	0	-
18	7.84	6.20	4.46	3.11	1.92	0.83	0.18	0.006	0	-
19	9.27	7.03	4.97	3.42	2.22	0.88	0.16	0	0	-
20	10.2	7.67	5.51	3.75	2.37	1.12	0.22	0.02	0	-

Table 4. Accuracy of MBTE(z).

7.3 BBBT: Branch and Bound with MBTE(z)

$N = 50$, $K = 5$, $C = 150$. $w^* = 17.6$. 10 instances. time = 600sec.							
	BBMB					BBBT	PFC-MPRDAC
T	z=2	z=3	z=4	z=5	z=6	z=2	
	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks
5	6 45 1.11M	7 54 1.51M	6 6.2 177K	9 75 2.29M	10 6.2 123K	10 1.9 55	10 0.01 436
7	4 134 5.86M	5 150 4.62M	7 213 5.3M	8 208 5.14M	9 97 2.1M	10 2.5 94	10 1.7 15K
9	-	-	1 325 7.4M	3 227 4.97M	3 229 4.85M	10 14.3 2.1K	10 27.3 242K

Table 5. BBBT(z) vs. BBMB(z).

$N = 100$, $K = 5$, $C = 300$. $w^* = 33.9$. 10 instances. time = 600sec.								
	BBMB						BBBT	PFC-MPRDAC
T	z=2	z=3	z=4	z=5	z=6	z=7	z=2	
	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks
3	6 6 150K	6 6 150K	6 6 150K	6 5 115K	8 6.8 115K	8 15 8	10 7.73 60	10 0.03 750
5	2 36 980K	2 32 880K	2 24 650K	2 5.3 130K	3 38 870K	3 33 434K	10 14.3 114	10 0.06 1.5K
7	0	0	0	0	0	0	10 29 331	6 267 1.6M

Table 6. BBBT(z) vs. BBMB(z).

Since MBTE(z) computes lower bounds for each singleton-variable assignment, when incorporated within a Branch-and-Bound search, MBTE(z) can facilitate domain pruning and dynamic variable ordering. In this section we investigate the performance of such a new algorithm, called BBBT(z) (Branch-and-Bound with Bucket-Tree heuristics), and compare it against BBMB(z) [13].

BBMB(z) [13] is a Branch-and-Bound search algorithm that uses Mini-Bucket Elimination (MBE(z)) as a pre-processing step. MBE(z) generates intermediate functions that are used to compute a heuristic value for each node in the search space. Since these intermediate functions are pre-computed, before search starts, BBMB(z) uses the same fixed variable ordering as MBE(z). Unlike BBBT(z), BBMB(z) does not prune domains of variables. In the past [13] we showed that BBMB(z) was effective and competitive with alternative state-of-the-art algorithms for Max-CSP.

BBBT(z) is a Branch-and-Bound search algorithm that uses MBTE(z) at each node in the search space. Unlike BBMB(z), BBBT(z) has no pre-processing step. At each node in the search space, MBTE(z) is used to compute lower bounds for each variable-value assignment of future variables. These lower bounds are used for domain pruning – whenever a lower bound of a variable-value assignment is not less than the global upper bound, the value is deleted. BBBT(z) backtracks whenever an empty domain of a future variable is created. BBBT(z) also uses dynamic variable ordering – when picking the next variable to instantiate, it selects a variable with the smallest domain size. Ties are broken by picking a variable with the largest sum of lower bounds associated with each value. In addition, for value selection, BBBT(z) selects a value with the smallest lower bound.

In Tables 5-7 we have the results of experiments with three sets of Max-CSP problems: N=50, K=5, C=150, $5 \leq T \leq 9$, N=100, K=5, C=300, $3 \leq T \leq 7$, and N=50, K=5, C=100, T=15. On each problem instance we ran BBMB(z) for different values of z, as well as BBBT(2). We also ran BBBT(z) for larger values of z, but BBBT(2) was most cost effective on these problems. For comparison, we also report the results with PFC-MPRDAC [16] that is currently one of the best algorithms for Max-CSP.

$N = 50$, $K = 5$, $C = 100$. $w^* = 10.6$. 10 instances. time = 600sec.							
T	BBMB z=4	BBMB z=6	BBMB z=8	BBBT z=2	BBBT z=5	BBBT z=8	PFC-MPRDAC
	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks
15	8 184 13M	10 4.51 120K	10 12.4 24K	3 394 22K	9 190 565	8 91 30	9 108 1.0M

Table 7. BBBT(z) vs. BBMB(z).

In column 1 we have the tightness, in the last two columns we report BBBT(2) and PFC-MPRDAC, and in the middle columns we have BBMB(z). For each set of problems, we report the number of problems solved (within the time bound of 600 seconds), the average CPU time and number of deadends for solved problems. For example, we see from Table 5 (N=50, K=5, C=150), that when tightness T is 5, BBMB(6) solved all 10 problems, taking 6.2 seconds and 123 thousand backtracking steps, on the average, whereas BBBT(2) also solved all 10 problems, taking 1.9 seconds and 55 backtracking steps, on the average.

We see from Tables 5 and 6 that on these two sets of problems, BBBT(2) is vastly superior to BBMB(z), especially as the tightness increases. Average CPU time of BBBT(2) is as much as an order of magnitude less than BBMB(z). Sporadic experiments with 200 and 300 variable instances showed that BBBT(2) continues to scale up very nicely on these problems. BBBT(2) is also faster than PFC-MPRDAC on tight constraints.

The experiments also demonstrate the pruning power of MBTE(z). The number of backtracking steps used by BBBT(2) is up to three orders of magnitude less than BBMB(z). For example, we see from Table 5 that when tightness T is 7, BBMB(6) solved 9 problems out of 10, taking 2.1 million backtracking steps in 97 seconds, whereas BBBT(2) solved all 10 problems, taking 94 backtracking steps in 2.5 seconds.

We observed a different behavior on problems having sparser constraint graphs and tight constraints. While still very effective in pruning the search space, BBBT was not as cost-effective as BBMB(z) (which invests in heuristic computation only once). Table 3 exhibits a typical performance (N=50,C=100, K=5, T=15). We observe that here BBBT's performance exhibit a U-shape, improving with z up to an optimal z value. However, BBBT's slope of improvement is much more moderate as compared with BBMB.

8 Conclusions and future work

Since constraint optimization is NP-hard, approximation algorithms are of clear practical interest. In the paper we extend the mini-bucket scheme proposed for variable elimination to tree-decomposition. We have introduced a new algorithm for lower bound computation. MCTE(z) is a general method for computing compute lower bounds to arbitrary sets of tasks. The parameter z allows trading accuracy for complexity and can be adjusted to best fit the available resources. MBTE(z) is an instantiation of MCTE(z) for the computation of lower bounds to singleton optimization. This task is relevant in the context of branch and bound solvers. Both algorithms have been derived using CTE, a tree-decomposition schema for reasoning tasks which unifies a number of approaches appearing in the past 2 decades in the constraint satisfaction and probabilistic reasoning context.

We have shown that bounds obtained with MBTE(z) have the same accuracy as if computed with n runs of plain mini-buckets. The quality of such accuracy has already been demonstrated in a number of domains [9]. We have also shown that MBTE(z) can be up to n times faster than the alternative of running plain mini-buckets n times. This speed-up is essential if the algorithm is to be used at every node within a branch and bound solver. Our preliminary experiments suggest that MBTE(z) is a very promising approach. It generates good quality bounds at a reasonable cost. Within branch and bound, it reduces dramatically the search space and sometimes the reduction translates into great time savings. Taking into account that our implementation is general and has not yet been optimized, these results are quite promising.

Our approach leaves plenty of room for future improvement, which are likely to make it more cost effective in practice. For instance, it can be modified to treat separately hard and soft constraints, since hard constraints can be more efficiently processed and propagated [10]. As a matter of fact, even if the original problem has no hard constraints, our approach can be used to infer them (i.e, detect infeasible tuples). A second line of improvement is to exploit lower bound redundancy. Namely, to use tree-decompositions with several solution-vertices for each task. Since each vertex provides a different lower bound, we can select the best. Finally, currently our partitioning to mini-buckets was always random. Investigating heuristics for partitioning may increase the accuracy of the algorithms.

References

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

[2] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *UAI96*, pages 81–89, 1996.

[3] C. Bessiere and J.C. Regin. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *CP96*, pages 61–75, 1996.

[4] Stefano Bistarelli, Hélène Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, and Gerard Verfaillie. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints*, 4:199–240, 1999.

[5] S. Bistarelly, R. Gennari, and F. Rossi. Arc consistency for soft constraints. In *Proc. of the* 6^{th} *CP*, pages 83–97, Singapore, 2000.

[6] Romuald Debruyne and Christian Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *IJCAI99*, pages 412–417, 1999.

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

[8] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[9] Rina Dechter and Irina Rish. A scheme for approximating probabilistic inference. In *UAI97*, pages 132–141, 1997.

[10] Rina Dechter and Peter van Beek. Local and global relational consistency. *Theoretical Computer Science*, 173(1):283–308, 20 February 1997.

[11] Eugene Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29:24–32, March 1982.

[12] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. In *IJCAI99*, pages 394–399, 1999.

[13] K. Kask. new search heuristics for max-csp. In *CP2000*, pages 262–277, 2000.

[14] J. Larrosa. On the time complexity of bucket elimination algorithms. Technical report, University of California at Irvine, 2001.

[15] J. Larrosa and P. Meseguer. Partial lazy forward checking for max-csp. In *Proc. of the* 13^{th} *ECAI*, pages 229–233, Brighton, United Kingdom, 1998.

[16] Javier Larrosa, Pedro Meseguer, and Thomas Schiex. Maintaining reversible DAC for max-CSP. *Artificial Intelligence*, 107(1):149–163, 1999.

[17] S. L. Lauritzen and D. J. Spiegelhalter. Local computation with probabilities on graphical structures and their applications to expert systems. *Journal of the Royal Statistical Society, Series B*, 34:157–224, 1988.

[18] A. Mackworth. Consistency in networks of constraints. *Artificial Intelligence*, 8, 1977.

[19] B. Nudel. Tree search and arc consistency in constraint satisfaction algorithms. *Search in Artifical Intelligence*, 999:287–342, 1988.

[20] T. Schiex. Arc consistency for soft constraints. In *CP2000*, pages 411–424, Singapore, 2000.

[21] P.P. Shenoy. Binary join-trees for computing marginals in the shenoy-shafer architecture. *International Journal of Approximate Reasoning*, 2-3:239–263, 1997.

[22] G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search. In *AAAI96*, pages 181–187, 1996.