Combining tree decomposition and local consistency in Max-CSPs

Simon de Givry[1], Thomas Schiex[1], Gérard Verfaillie[2]

[1]INRA - UBIA, [2]ONERA - DCSD, Toulouse, France
{degivry,tschiex}@toulouse.inra.fr, verfaillie@cert.fr

Abstract. Most current state-of-the-art complete Max-CSP or weighted CSP solvers can be described as a basic depth-first branch and bound search (DFBB) that maintain some form of arc consistency during the search. In this paper, we study some structural solving methods such as BTD that have a better time complexity than DFBB. We introduce state-of-the-art forms of arc consistency in BTD in order to reduce the search effort and the memory space actually used. We show it can be the case on random and real-world instances.

1 Introduction

Max-CSP is a well-known problem of finding a complete assignment with a minimum number of unsatisfied constraints. In general, this problem is NP-hard. Max-CSP is usually solved by a depth-first branch and bound algorithm that has a linear space complexity but has an exponential time complexity in the number of problem variables. Other solving methods that exploit the structure of the constraint graph have their time complexity exponential in a given graph parameter whose value is often better than the number of variables. For instances, Bucket Elimination [5] has a time complexity exponential in the *induced width*, pseudo-tree search [6] and AND/OR tree search [12] have a time complexity exponential in the *tree-height*, and the BTD algorithm [15, 7] has a time complexity exponential in the *tree-width*, i.e. the size of the largest cluster of a tree decomposition of the constraint graph, and a space complexity exponential in the maximum separator size, i.e. the size of the largest intersection between two clusters. Another important aspect is the way of computing the lower bound during the search. Enforcing local consistency properties as defined in [13, 11, 2, 10, 4] produces a lower bound that can be used in a branch and bound algorithm to reduce the search effort. However the current BTD algorithm uses a limited form of local consistency only, i.e. forward checking. The goal of this paper is to introduce stronger forms of local consistency such as soft arc consistency inside BTD. Our aim is that local consistency will reduce the search effort of cluster explorations and the memory space actually used by its recording mechanism.

2 Preliminaries

For simplicity reasons, we consider in the sequel a basic soft constraint framework consisting of binary Max-CSPs, although all the methods presented in this paper are also valid for weighted CSPs. A Max-CSP is a triplet (X, D, W). $X = \{1, \ldots, n\}$ is a set of n variables. Each variable $i \in X$ has a finite domain $D_i \in D$ of values than can be assigned to it. The maximum domain size is d. W is a set of k binary soft constraints. A binary soft constraint $W_{ij} \in W$ is a function $W_{ij} : D_i \times D_j \mapsto \{0, 1\}$ such that the value 1 means the constraint is violated with a cost one and 0 it is satisfied.

Before introducing soft local consistencies in Section 5, we add to our problem definition a unary cost function for every variable such that $W_i : D_i \mapsto \{0, k\}$. k corresponds to the maximum violation cost of a Max-CSP. Unary cost functions initially return zero. A zero arity constraint $W_\varnothing$ is also introduced, which initially returns zero. W_i and $W_\varnothing$ will be used during local consistency enforcement. $W_\varnothing$ will store the current problem lower bound. The goal is to find a complete assignment with minimum cost: $\min_{(a_1,a_2,\ldots,a_n)\in D_1\times D_2\times\cdots\times D_n}\{W_\varnothing + \sum_{i=1}^{n} W_i(a_i) + \sum_{W_{ij}\in W} W_{ij}(a_i, a_j)\}$. In general, this combinatorial optimization problem is NP-hard.

A tree decomposition of a graph $G = (X, E)$ is a pair (C, T). $C = \{C_1, \ldots, C_m\}$ is a set of m subsets of X. T is a tree having m nodes which are the m clusters of C. C and T verifies the following properties: (i) $\bigcup_{C_e \in C} C_e = X$, (ii) for each edge $\{i, j\} \in E$, there exists $C_e \in C$ such that $i, j \in C_e$, and (iii) for all $C_e, C_f, C_g \in C$, if C_g is on a path from C_e to C_f in T, then $C_e \cap C_f \subseteq C_g$ (*running intersection property*). The tree-width of a tree decomposition (C, T) is equal to $\max_{C_e \in C}\{|C_e|\} - 1$, denoted by w in the sequel. The tree-width w^* of G is the minimal tree-width over all the tree decomposition of G. Finding a minimal tree-width is NP-hard in general.

3 The BTD method

Given a Max-CSP $P = (X, D, W)$, BTD [15, 7] exploits a tree decomposition (C, T) of the constraint graph $G = (X, \{\{i, j\} \text{ s.t. } W_{ij} \in W\})$. We consider the first cluster $C_1 \in C$ as the root of T in the sequel. Such a rooted cluster tree allows to define a partition of the variables and the constraints. Although a variable may belong to several clusters of C, we are interested in finding the closest cluster from the root that contains the variable. $N(i)$ denotes the index of this cluster for the variable i. We note $V_e = \{i \in X \text{ s.t. } N(i) = e\}$ the set of variables associated to the cluster $C_e \in C$. In the same manner, we associate each constraint W_{ij} to the closest cluster from the root that contains both variables. Let $Sons(C_e)$ be the set of son clusters of $C_e \in C$ in the rooted tree T. Similarly, $Father(e)$ denotes the father cluster of $C_e \in C$ in T.

For each cluster $C_e \in C$, we associate a subproblem P_e defined by the variables in V_e and all the variables V_f related to the descendants C_f of C_e in T, and by all the constraints the scope of which belong to the variables of P_e. A first property is that for any cluster $C_e \in C$, its subproblem P_e is constrained by the rest of the problem $P \backslash P_e$ only by the assignment of variables in $C_e \cap C_{Father(e)}$. It is possible to solve P_e for any assignment of $C_e \cap C_{Father(e)}$ and to record its optimum value which avoids to solve P_e again for the same assignment. Secondly, if $C_e, C_f, C_g \in C$ such that $C_f, C_g \in Sons(C_e)$, then, after the assignment of V_e, P_f and P_g are two independent subproblems that can be solved sequentially because they do not share any variable or constraint. BTD is based on these two properties.

```
Function BTD(A,C_e,V,clb,cub) : integer
    if (V = ∅) then
        S ⟵ Sons(C_e) ;
1       clb ⟵ clb + Σ_{C_f ∈ S} LB_{A[C_f]} ;
2       while (S ≠ ∅ and clb < cub) do
            Choose  C_f ∈ S ;
            S ⟵ S\C_f ;
            if (LB_{A[C_f]} < UB_{A[C_f]}) then
                /* No information for A[C_f]: LB_{A[C_f]} = 0 and UB_{A[C_f]} = +∞ */ ;
3               clb' ⟵ BTD(A, C_f, V_f, 0, +∞) ;
                clb ⟵ clb + clb' ;
                LB_{A[C_f]} ⟵ clb' ;
                UB_{A[C_f]} ⟵ clb' ;
        return clb ;
    else
        Choose  i ∈ V ;
        d ⟵ D_i ;
        while (d ≠ ∅ and clb < cub) do
            Choose  a ∈ d ;
            d ⟵ d\{a} ;
4           l ⟵ Σ_{W_ij ∈ W s.t. j assigned by A} W_ij(a, A[j]) ;
            if (clb + l < cub) then
                cub ⟵ min{cub,BTD(A ∪ {i ⟵ a}, C_e, V\{i}, clb + l, cub) } ;
        return cub ;
```

Algorithm 1: BTD algorithm [7]. First call is BTD($\emptyset$,C_1,V_1,0,$+\infty$).

BTD explores the cluster tree T in a depth-first search manner, starting from C_1. For each visited cluster $C_e \in C$, a depth-first search is performed on the variables V_e. Whenever a cluster $C_f \in Sons(C_e)$ is visited, its associated subproblem P_f will be completely solved before visiting another son of C_e.

The BTD algorithm is given in Fig. 1. BTD(A,C_e,V_e,0,$+\infty$) returns the optimum value of subproblem P_e knowing the current partial assignment A of past variables. *clb* (resp. *cub*) is a lower (resp. upper) bound of the current subproblem. *clb* and *cub* are used to prune the search tree following the branch and bound principle. For simplicity reasons, we present BTD using backward checking to compute the lower bound (see line 4). A specific data structure $LB_{A[C_e]}$ (resp. $UB_{A[C_e]}$) records a lower (resp. upper) bound of P_e for a given assignment of $C_e \cap C_{Father(e)}$ (denoted by $A[C_e]$). Thanks to the branch and bound principle and the lower bound computation, BTD may not visit all the assignments of $C_e \cap C_{Father(e)}$. LB and UB can be sparse data structures. In our implementation, we use hash tables. Initially, LB is set to zero and UB to $+\infty$. In the original BTD algorithm, each visited subproblem P_e is completely solved. Thus, we have $LB_{A[C_e]} = UB_{A[C_e]}$ equal to the optimum value of P_e for a given assignment $A[C_e]$. In line 1, BTD uses the knowledge of previously solved subproblems to possibly increase clb.

BTD has a time complexity in $O(ms^2 k \log(d) d^{w+1})$, where m is the number of clusters, s is the size of the largest intersection between two clusters, k is the number of constraints, d is the maximum domain size, and w is the tree-width of the tree decomposition. And it has a space complexity in $O(msd^s)$ [15].

4 Exploiting local cuts in BTD

Function $\mathsf{BTD}^+(A, C_e, V, clb, cub)$: integer
  if $(V = \emptyset)$ **then**
    $S \longleftarrow Sons(C_e)$;
    $clb \longleftarrow clb + \sum_{C_f \in S} LB_{A[C_f]}$;
5   **while** $(S \neq \emptyset$ and $clb < cub)$ **do**
      $Choose \quad C_f \in S$;
      $S \longleftarrow S \backslash C_f$;
      if $(LB_{A[C_f]} < UB_{A[C_f]})$ **then**
6         $\boldsymbol{cub' \longleftarrow cub - clb + LB_{A[C_f]}}$;
7         $clb' \longleftarrow \mathsf{BTD}^+(A, C_f, V_f, 0, \boldsymbol{cub'})$;
        $clb \longleftarrow clb + clb' \boldsymbol{- LB_{A[C_f]}}$;
8         $LB_{A[C_f]} \longleftarrow clb'$;
9   **if** $\boldsymbol{(clb' < cub')}$ **then** $UB_{A[C_f]} \longleftarrow clb'$;
    return clb ;
  else
    $Choose \quad i \in V$;
    $d \longleftarrow D_i$;
    while $(d \neq \emptyset$ and $clb < cub)$ **do**
      $Choose \quad a \in d$;
      $d \longleftarrow d \backslash \{a\}$;
      $l \longleftarrow \sum_{W_{ij} \in W \text{ s.t. } j \text{ assigned by } A} W_{ij}(a, A[j])$;
      if $(clb + l < cub)$ **then**
        $cub \longleftarrow \min\{cub, \mathsf{BTD}^+(A \cup \{i \longleftarrow a\}, C_e, V \backslash \{i\}, clb + l, cub)\}$;
    return cub ;

Algorithm 2: BTD$^+$ algorithm. First call is BTD$^+$($\emptyset$,C_1,V_1,0,$+\infty$).

When BTD solves a subproblem in line 3, it does not impose any initial upper bound. The resulting optimum value may be much greater than the current allowed branch and bound gap $cub - clb$ tested in line 2. In order to reduce the search effort when solving a subproblem P_f such that $C_e, C_f \in C, C_f \in Sons(C_e)$, we can impose an initial upper bound equal to the difference between the parental upper bound of P_e and a lower bound of the remaining part of the problem $P \backslash P_f$. The corresponding algorithm called BTD^+ is given in Fig. 2. The code is colored in gray and black in order to highlight the differences between BTD^+ and BTD. After solving a subproblem at line 7 with an initial upper bound computed in line 6, either there exists a solution with a cost strictly lower than this upper bound and this solution is optimal (then LB and UB are updated in lines 8,9), or no solution has been found and only a lower bound can be deduced and recorded for the subproblem (in line 8). In this case, the lower bound LB is equal to the initial upper bound and UB remains equal to $+\infty$.

Exploiting an initial upper bound when solving a subproblem has the potential drawback that the same subproblem can be solved several times in comparison with the original BTD algorithm that never solves the same subproblem more than once thanks to the memorization of its optimum value. However each new lower bound found by BTD^+ for a given subproblem is strictly greater than the previous recorded lower bound for this subproblem ($clb' = cub' = cub - clb + LB_{A[C_f]} > LB_{A[C_f]}$ because $cub - clb > 0$), otherwise the test in the while loop in line 5 would have failed. It means that the worst-case time complexity of BTD^+ is k (the maximum violation cost) times the time complexity of BTD. Both algorithms have the same space complexity.

This approach of trying different upper bounds for the same problem is common in branch and bound optimization and can be effective in practice. For instance, Iterative Deepening Search [8] uses increasing upper bounds starting from zero until the problem optimum is found.

A possible way of reducing subproblem repetition is by providing a good value ordering heuristic. By choosing a good value first, the current subproblem lower bound clb is kept as small as possible in the left-most branch of the depth-first search tree, resulting in a larger branch and bound gap $cub-clb$ and higher initial upper bounds when solving for the first time any subproblem $P_e, e \in \{1, \ldots, m\}$.

Another observation is that we do not need to record the exact subproblem upper bound (UB) but only the fact that we have found the optimum or just a lower bound. We use this result in our implementation of all the algorithms in order to save memory space by only recording LB associated to a boolean to know if it is the optimum or not.

5 Combining BTD with local consistency restricted to the current cluster subtree

In this paper, we are interested in combining BTD^+ with local consistency in order to reduce the search effort when exploring a cluster and also to save memory space by recording less subproblem lower bounds. We consider the following local consistencies developed in the soft constraint framework: soft arc consistency (AC) [13, 9], soft directional arc consistency (DAC) [3, 2], soft full directional arc consistency (FDAC) [3, 10, 2], and soft existential directional arc consistency (EDAC) [4].

Two WCSPs defined over the same variables are said to be *equivalent* if they define the same cost distribution on complete assignments. Enforcing local consistency on a given problem is done by changing the values returned by its cost functions. Projection and extension are the only two basic operations used to transform a given problem into an equivalent but possibly more explicit one (having a better lower bound $W_\varnothing$ and less values in the domains). See [13, 2] for a definition and examples of projection and extension operations. An important result of enforcing local consistency is to produce a problem lower bound in $W_\varnothing$ that can be used during the search as the current lower bound (stronger than backward or forward checking).

The difficulty for BTD is that these transformations preserve the semantic of the whole problem but can modify the semantic of some subproblems, by moving cost from one subproblem to another one. During the search, the optimum of a future subproblem can change due to local consistency enforcement, resulting in useless recorded lower and upper bounds.

We now analyze how to combine the four basic local consistency operations as described in [4] with BTD such that recorded lower and upper bounds are used in a safe manner.

5.1 Binary projection and extension

A binary projection $Project(i, a, j, \alpha)$ moves cost α from binary cost function W_{ij} to unary cost function W_i:

$$a \in D_i, \forall b \in D_j, W_{ij}(a, b) \longleftarrow W_{ij}(a, b) - \alpha, W_i(a) \longleftarrow W_i(a) + \alpha$$

If variables i, j are associated to the same cluster in the tree decomposition ($N(i) = N(j)$), there is nothing to do. If $N(i) \neq N(j)$, then clusters $C_{N(i)}$ and $C_{N(j)}$ are on a common path from the root due to the running intersection property of tree decomposition. Either cluster $C_{N(j)}$ is an ascendant of cluster $C_{N(i)}$, then the binary and unary cost functions belong to the same cluster $C_{N(i)}$ and thus the same subproblem. Or, $C_{N(i)}$ is an ascendant of $C_{N(j)}$, and the binary projection results in decreasing W_{ij} in the subproblem $P_{N(j)}$ and increasing W_i in the subproblem $P_{N(i)}$. Because $P_{N(i)}$ already contains $P_{N(j)}$, this does not change its optimum. On the contrary, the optimum of $P_{N(j)}$ is decreased by α. This is also the case for all the subproblems P_e on the path between $C_{N(j)}$ and $C_{N(i)}$ in the tree decomposition because their associated cluster C_e contains variables i, j. We store these cost modifications in a specific *backtrackable* data structure ΔW. Initially, ΔW is set to zero. The modified function $Project(i, a, j, \alpha)$ does the following:

$$\forall C_e \in C \text{ in the path from } C_{N(j)} \text{ included to } C_{N(i)} \text{ excluded,}$$
$$\Delta W_i^e(a) \longleftarrow \Delta W_i^e(a) + \alpha$$

During the search, we apply a correction to the recorded lower and upper bounds thanks to the current ΔW information. $\overline{\Delta W}_{A[C_e]}$ represents the total cost that has been moved out the subproblem P_e for a given assignment $A[C_e]$ of $C_e \cap C_{Father(e)}$ variables: $\overline{\Delta W}_{A[C_e]} = \sum_{\{i,a\} \in A \text{ s.t. } i \in C_e \cap C_{Father(e)}} \Delta W_i^e(a)$. This total cost is subtracted to the recorded lower bound in order to obtain the current true lower bound.

A binary extension $Extend(i, a, j, \alpha)$ moves cost α from unary cost function W_i to binary cost function W_{ij}. It is equivalent to a binary projection $Project(i, a, j, -\alpha)$. We apply the same modifications to $Extend$ as previously described for $Project$.

5.2 Unary projection

A unary projection $ProjectUnary(i)$ moves the minimum cost $\min_{a \in D_i}\{W_i(a)\}$ from unary cost function W_i to the problem lower bound $W_\varnothing$. Instead of having one zero-arity constraint for representing the problem lower bound, we split it into one zero-arity constraint per cluster in the tree decomposition. $W_\varnothing^e$ is a lower bound for the subproblem composed of V_e variables and the constraints whose scopes are inside V_e. Thus, local consistency enforcement provides a current lower bound $\overline{W}_\varnothing^e$ of subproblem P_e which is the sum of all the lower bounds of clusters included in P_e. We have $\overline{W}_\varnothing^1 = W_\varnothing$. For any subproblem $P_e, e \in \{1, \ldots, m\}$ and any assignment $A[C_e \cap C_{Father(e)}]$, we have two lower bounds, one provided by the BTD recording mechanism and the other one provided by local consistency. In the following codes, we always use the maximum of these two bounds.

5.3 Value removal

Value removal $PruneVar(i)$ removes values $a \in D_i$ such that $W_\varnothing + W_i(a) \geq gub$, with gub, a global upper bound of the problem. We replace this global condition by a local one $\overline{W}_\varnothing^e + W_i(a) \geq cub$ where $\overline{W}_\varnothing^e$ and cub are the current lower and upper bounds of the currently visited cluster C_e. We apply this pruning rule only to variables in the current subproblem P_e. By doing so, we ensure that any removed value in a given subproblem P_e is also forbidden in all the subproblem P_f included in P_e. Thus the set of removed values for one subproblem P_e, is still valid in all the subproblems P_f such that $C_f \in Sons(C_e)$, and all the propagations made during the exploration of C_e are still valid when exploring $C_f \in Sons(C_e)$. The proof is obtained by the fact that $cub - \overline{W}_\varnothing^e$ is monotonically decreasing when following a path in the cluster tree from the root to a leaf:

$$
\begin{aligned}
&\forall C_e \in C, \forall C_f \in Sons(C_e),\\
&\qquad cub_{C_f} - \overline{W}_\varnothing^f = cub - clb + \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}_\varnothing^f\} - \overline{W}_\varnothing^f\\
&\qquad\qquad = cub - W_\varnothing^e\\
&\qquad\qquad\quad - \sum_{C_g \in Sons(C_e) \setminus C_f} \max\{LB_{A[C_g]} - \overline{\Delta W}_{A[C_g]}, \overline{W}_\varnothing^g\}\\
&\qquad\qquad\quad - \overline{W}_\varnothing^f\\
&\qquad\qquad \leq cub - \overline{W}_\varnothing^e
\end{aligned}
$$

Function LC-BTD$^+$(A,C_e,V,clb,cub) : integer

if ($V = \emptyset$) **then**

$S \longleftarrow Sons(C_e)$;

10 $clb \longleftarrow W^e_\varnothing + \sum_{C_f \in S} \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\}$;

while ($S \neq \emptyset$ and $clb < cub$) **do**

$Choose \quad C_f \in S$;

$S \longleftarrow S \backslash C_f$;

if ($LB_{A[C_f]} < UB_{A[C_f]}$) **then**

$cub' \longleftarrow cub - clb + \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\}$;

11 $clb' \longleftarrow$LC-BTD$^+$(A, C_f, V_f, $\overline{W}^f_\varnothing$, cub') ;

$clb \longleftarrow clb + clb' - \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\}$;

12 $LB_{A[C_f]} \longleftarrow clb' + \overline{\Delta W}_{A[C_f]}$;

13 **if** ($clb' < cub'$) **then** $UB_{A[C_f]} \longleftarrow clb' + \overline{\Delta W}_{A[C_f]}$;

return clb ;

else

$Choose \quad i \in V$;

$d \longleftarrow D_i$;

while ($d \neq \emptyset$ and $clb < cub$) **do**

$Choose \quad a \in d$;

$d \longleftarrow d \backslash \{a\}$;

14 Enforce LC with $\{i \longleftarrow a\}$ on subproblem P_e;

if ($\overline{W}^e_\varnothing < cub$) **then**

$cub \longleftarrow \min\{cub,$LC-BTD$^+(A \cup \{i \longleftarrow a\}, C_e, V \backslash \{i\}, \overline{W}^e_\varnothing, cub)\}$;

return cub ;

Algorithm 3: LC-BTD$^+$ algorithm. Initial problem is made LC consistent beforehand. First call is LC-BTD$^+$($\emptyset$,C_1,V_1,$\overline{W}^1_\varnothing$,$+\infty$).

5.4 LC-BTD+ algorithm

The resulting algorithm called LC-BTD$^+$ combines BTD$^+$ with any local consistency $LC \in \{NC, AC, DAC, FDAC, EDAC\}$ and is described in Fig. 3 (again, differences with BTD$^+$ are highlighted). In line 10, the lower bound of the current subproblem is the sum of assignment cost for cluster C_e ($W^e_\varnothing$) plus the sum of the maximum of the two lower bounds (obtained by lower bound recording or by propagation) for each cluster son. In line 14, the local consistency LC is enforced on the current subproblem P_e resulting in a new lower bound $\overline{W}^e_\varnothing$ of P_e. In lines 12 and 13, we apply the right corrections in order to record valid lower and upper bounds. Notice that in line 11, we already have a lower bound

$\overline{W}^f_\varnothing$ of P_f that is given as input to LC-BTD$^+$. The rest of the code is identical to BTD$^+$.

The time and space complexities of LC-BTD$^+$ are identical to BTD$^+$. Time complexity proof is based on the fact that when a new lower bound $clb' + \overline{\Delta W}_{A[C_f]}$ is recorded, it is equal to $cub_{C_f} + \overline{\Delta W}_{A[C_f]} = cub - clb + \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\} + \overline{\Delta W}_{A[C_f]} > LB_{A[C_f]}$.

6 Combining BTD with unrestricted local consistency

In the previous algorithm, local consistency was restricted to the current subproblem in order to keep the set of forbidden values still valid when going down from one cluster C_e to another cluster $C_f \in Sons(C_e)$ during the search. Now, we show how it is possible to remove this restriction by using local and global bounds.

A value is removed if $\overline{W}^e_\varnothing + W_i(a) \geq cub$ or $\overline{W}^1_\varnothing + W_i(a) \geq gub$. Recall that $\overline{W}^e_\varnothing$ (resp. $\overline{W}^1_\varnothing$) is the current lower bound of subproblem P_e (resp. problem P). gub is a global upper bound of the whole problem. Although the second global condition takes into account all the information produced by local consistency enforcement, it does not necessarily imply the first local condition because cub includes the knowledge of recorded lower bounds. So both conditions are needed.

A major difficulty when using two conditions during the exploration of a cluster C_e is that we are not sure to find the optimum of P_e even if the initial upper bound is greater than the optimum. This is due to the fact that propagation in $P \backslash P_e$ can increase the current lower bound of P such that $\overline{W}^1_\varnothing \geq gub$ before a solution is found in P_e. Moreover, even if a solution is found in P_e, there is no guarantee that we will find the optimal solution of P_e. This surprising result is due to the fact that soft local consistencies such as AC, FDAC or EDAC are not confluent. Depending on the way $\overline{W}^e_\varnothing$ increases during the search, it can result in different value removal orders in $P \backslash P_e$ conducting to different projection/extension operation orders and finally different $\overline{W}^1_\varnothing$ values. Our solution is to collect during the exploration of a subproblem P_e its minimum lower bound and the cost of the best solution found at all the leaf nodes of the search tree developed for solving P_e. Notice that this minimum lower bound cannot be greater than the initial subproblem upper bound due to previous value removals. Moreover, value removals can occur during a cluster exploration, due to the global cut $\overline{W}^1_\varnothing + W_i(a) \geq gub$ which is equal to $\overline{W}^e_\varnothing + \sum_{C_g \in P \backslash P_e} W^g_\varnothing + W_i(a) \geq gub$. Therefore, the minimum lower bound of P_e when a failure occurs (at a leaf node) cannot be greater than $\overline{W}^e_\varnothing + W_i(a) \geq gub - \sum_{C_g \in P \backslash P_e} W^g_\varnothing = gub - \overline{W}^1_\varnothing + \overline{W}^e_\varnothing$.

The resulting algorithm called LC-BTD* is presented in Fig. 4 (again, differences with LC-BTD$^+$ are highlighted). In line 21 and 22, the exploration of the current cluster takes into account local and global bounds. This is also the case for computing the initial upper bound of a subproblem in line 16. Instead of returning a lower bound *or* the optimum of P_e as it is done in LC-BTD$^+$,

Function LC-BTD*(A,C_e,V,clb,cub,$\boldsymbol{glb}$,$\boldsymbol{gub}$:in/out) : (integer,integer)

if $(V = \emptyset)$ **then**

$S \longleftarrow Sons(C_e)$;

$clb \longleftarrow W^e_\varnothing + \sum_{C_f \in S} \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\}$;

15 $\boldsymbol{cub''} \longleftarrow \boldsymbol{W^e_\varnothing}$;

while ($S \neq \emptyset$ and $clb < cub$) **do**

$Choose \quad C_f \in S$;

$S \longleftarrow S \backslash C_f$;

if $(LB_{A[C_f]} < UB_{A[C_f]})$ **then**

16 $cub' \longleftarrow \min\{cub - clb + \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\},$
$\boldsymbol{gub} - \boldsymbol{glb} + \overline{\boldsymbol{W}}^f_\varnothing, UB_{A[C_f]}\}$;

$(\boldsymbol{clb'}, \boldsymbol{cub'}) \longleftarrow$LC-BTD*($A$, C_f, V_f, $\overline{W}^f_\varnothing$, cub', $\boldsymbol{glb}$, $\boldsymbol{gub}$) ;

$clb \longleftarrow clb + clb' - \max\{LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}, \overline{W}^f_\varnothing\}$;

17 $\boldsymbol{cub''} \longleftarrow \boldsymbol{cub''} + \boldsymbol{cub'}$;

$LB_{A[C_f]} \longleftarrow \max\{\boldsymbol{LB_{A[C_f]}}, clb' + \overline{\Delta W}_{A[C_f]}\}$;

$UB_{A[C_f]} \longleftarrow \min\{\boldsymbol{UB_{A[C_f]}}, \boldsymbol{cub'} + \overline{\Delta W}_{A[C_f]}\}$;

else

18 $\boldsymbol{cub''} \longleftarrow \boldsymbol{cub''} + LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]}$

19 **return** $(\boldsymbol{clb}, \boldsymbol{cub''})$;

else

20 $(\boldsymbol{clb'}, \boldsymbol{cub'}) \longleftarrow (\boldsymbol{cub}, +\infty)$;

$Choose \quad i \in V$;

$d \longleftarrow D_i$;

21 **while** ($d \neq \emptyset$ and $clb < cub$ and $\boldsymbol{glb} < \boldsymbol{gub}$) **do**

$Choose \quad a \in d$;

$d \longleftarrow d \backslash \{a\}$;

Enforce LC with $\{i \longleftarrow a\}$;

22 **if** ($\overline{W}^e_\varnothing < cub$ and $\overline{\boldsymbol{W}}^1_\varnothing < \boldsymbol{gub}$) **then**

$(\boldsymbol{l}, \boldsymbol{u}) \longleftarrow$LC-BTD*($A \cup \{i \longleftarrow a\}$, C_e, $V \backslash \{i\}$, $\overline{W}^e_\varnothing$, cub, $\overline{\boldsymbol{W}}^1_\varnothing$, $\boldsymbol{gub}$) ;

23 $\boldsymbol{clb'} \longleftarrow \min\{\boldsymbol{clb'}, \boldsymbol{l}\}$;

24 $\boldsymbol{cub'} \longleftarrow \min\{\boldsymbol{cub'}, \boldsymbol{u}\}$;

$\boldsymbol{cub} \longleftarrow \min\{\boldsymbol{cub}, \boldsymbol{u}\}$;

if $(C_e = C_1)$ **then** $\boldsymbol{gub} \longleftarrow \min\{\boldsymbol{gub}, \boldsymbol{u}\}$;

else

25 $\boldsymbol{clb'} \longleftarrow \min\{\boldsymbol{clb'}, \overline{\boldsymbol{W}}^e_\varnothing, \boldsymbol{gub} - \overline{\boldsymbol{W}}^1_\varnothing + \overline{\boldsymbol{W}}^e_\varnothing\}$;

26 **return** $(\boldsymbol{clb'}, \boldsymbol{cub'})$;

Algorithm 4: LC-BTD^* algorithm. Initial problem is made LC consistent beforehand. First call is LC-BTD*($\emptyset$,C_1,V_1,$\overline{W}^1_\varnothing$,$+\infty$,$\overline{W}^1_\varnothing$,$+\infty$).

$\mathsf{LC\text{-}BTD}^*$ returns in lines 19 and 26 a lower bound *and* the cost of the best solution found after solving P_e. cub'' represents the cost of the best solution found in P_e and is the sum of the current cluster cost $W^e_\varnothing$ and the total cost of the best solution found for each subproblem P_f such that $C_f \in Sons(C_e)$ (see lines 15,17,18). When exploring the current cluster, we maintain the minimum lower bound (clb') and the cost of the best solution found (cub') at all the leaf nodes of the search tree in lines 20,23,24, and 25.

The space complexity of $\mathsf{LC\text{-}BTD}^*$ is the same as for BTD. Its time complexity is no more bounded by the tree-width because we don't have the property that the recorded lower bounds will increase monotonically. However, $\mathsf{LC\text{-}BTD}^*$ has a time complexity in $O(d^h)$, where h is the tree-height of the tree decomposition, i.e. the maximum number of variables in a path from the root node to any leaf node of the cluster tree. This time complexity is also valid for all the BTD-like algorithms that follow a variable ordering compatible with the cluster tree and exploit the independence property of subproblems.

7 Taking into account recorded lower bounds as soon as possible

When exploring a cluster C_e, as soon as the variables in $C_f \cap C_e$ are assigned for a cluster $C_f \in Sons(C_e)$, we take into account the recorded lower bound $LB_{A[C_f]}$ in the current lower bounds $\overline{W}^e_\varnothing$ and $\overline{W}^1_\varnothing$. This can result in further propagations and better pruning. More precisely, we add to these lower bounds the cost $LB_{A[C_f]} - \overline{\Delta W}_{A[C_f]} - \overline{W}^f_\varnothing$ if it is positive, i.e. if the corrected recorded lower bound is better (*optimal or strictly greater*) than the lower bound found by propagation. If it is the case, then we must disconnect the subproblem P_f from the propagation in order to avoid counting the same constraint cost twice. Moreover, after the complete exploration of a subproblem P_f, s.t. $C_f \in Sons(C_e)$, we can take into account as previously the newly updated recorded lower bound when backtracking in C_e. All these improvements have been done to enhance $\mathsf{LC\text{-}BTD}^+$ (resp. $\mathsf{LC\text{-}BTD}^*$), resulting in the algorithm called $\mathsf{LC\text{-}BTDo}^+$ (resp. $\mathsf{LC\text{-}BTDo}^*$) in the following Section.

8 Experimental results

In this Section, we perform an empirical comparison of our various versions of the BTD method with classic depth-first branch and bound (MFDAC [10]) and Bucket Elimination (BE) [5] on random and real-world instances. For efficiency reasons, our BTD methods are using forward checking for BTD and BTD^+, and soft full directional arc consistency (FDAC) for the other versions ($\mathsf{FDAC\text{-}BTD}^+$, $\mathsf{FDAC\text{-}BTD}^*$, $\mathsf{FDAC\text{-}BTDo}^+$, and $\mathsf{FDAC\text{-}BTDo}^*$). For comparison purpose, we examine the case of using a weaker local consistency (soft node consistency, $\mathsf{NC\text{-}BTDo}^+$) and the case of not recording any lower bound (FDAC-PTS is derived from $\mathsf{FDAC\text{-}BTD}^*$), in the spirit of pseudo-tree search [6]. For variable selection, we used the *dom/deg* heuristic which selects the variable with the smallest ratio of domain size divided by future degree. For value selection we consider values in increasing order of unary cost W_i. The variable ordering for directional arc consistency is lexicographic. The tree decomposition method is based on the maximum cardinality search (MCS) ordering heuristic [14]. The root node is chosen such that the tree-height is minimized.

For random instances, we limit to 5 minutes the time spent for solving a given instance (for unsolved instances, we consider that the running time is 5 min). For real-world instances, the limit is 10 hours and 4 billion of visited nodes. Our implementation of all the algorithms is based on a free weighted CSP solver called ToolBar (C code)[1]. The experiments were all performed on a 2.4 GHz Xeon computer with 4 GB.

8.1 Randomly generated instances

Random instances are clique trees. We use the following basic parametric model (w, s, h', d, t) such that each clique has $w + 1$ variables with domain size equal to d, each separator has s variables, each binary constraint has the same tightness equal to t (the ratio between the number of forbidden tuples and d^2), and the resulting clique tree is a complete binary tree composed of $2^{h'} - 1$ cliques. Depending on the parameters (w, s, h'), the total number of variables is equal to $n = s + (w + 1 - s)(2^{h'} - 1)$. Our tree decomposition method produces a tree-width equal to w and a tree-height equal to $h = h'(w + 1 - s) + s$. We have generated and solved $(w = 9, h' = 3, d = 5)$ clique trees with varying separator sizes $s \in \{2, 5, 7\}$ and varying constraint tightness $t \in [20, 80]\%$. Samples have 50 instances and we report average values. Time results in seconds and memory space in number of recorded lower bounds are given in Fig. 1. We do not report the results for FDAC-BTD* (resp. FDAC-BTDo*) because they were almost identical to FDAC-BTD$^+$ (resp. FDAC-BTDo$^+$). Exploiting local cuts as in FC-BTD$^+$ saved time (except for $s = 2, t \in \{50, 60\}\%$) and space compared with the original FC-BTD algorithm. Introducing stronger soft local consistency such as FDAC in BTD had mainly the effect of reducing memory usage and the number of visited nodes (not reported here for the lack of space) significantly compared to FC-BTD. However, FDAC-BTD$^+$ was slower than FC-BTD for $s \in \{2, 5\}$. The reason for this is that the same subproblem can be solved several times (up to 99 times and mean repetition in $[0.0, 4.3]$ for all the instances and BTD versions), and also FDAC time complexity is in $O(knd^3)$ [10] compared to FC in $O(kd)$. Taking into account recorded lower bounds as soon as possible greatly improved the results. FDAC-BTDo$^+$ algorithm got the best results other all BTD-like algorithms (except for $s = 2$ where NC-BTDo$^+$ was faster), up to two-orders of magnitude for $s = \{2, 7\}$ compared with FC-BTD. Not using lower bound recording as in FDAC-PTS could result in very poor performance ($s \in \{2, 5\}$). MFDAC was unable to solve the instances with small separator size and large number

[1] ToolBar is available at the *Algorithms* link of the *SoftCSP* web site `http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP`.

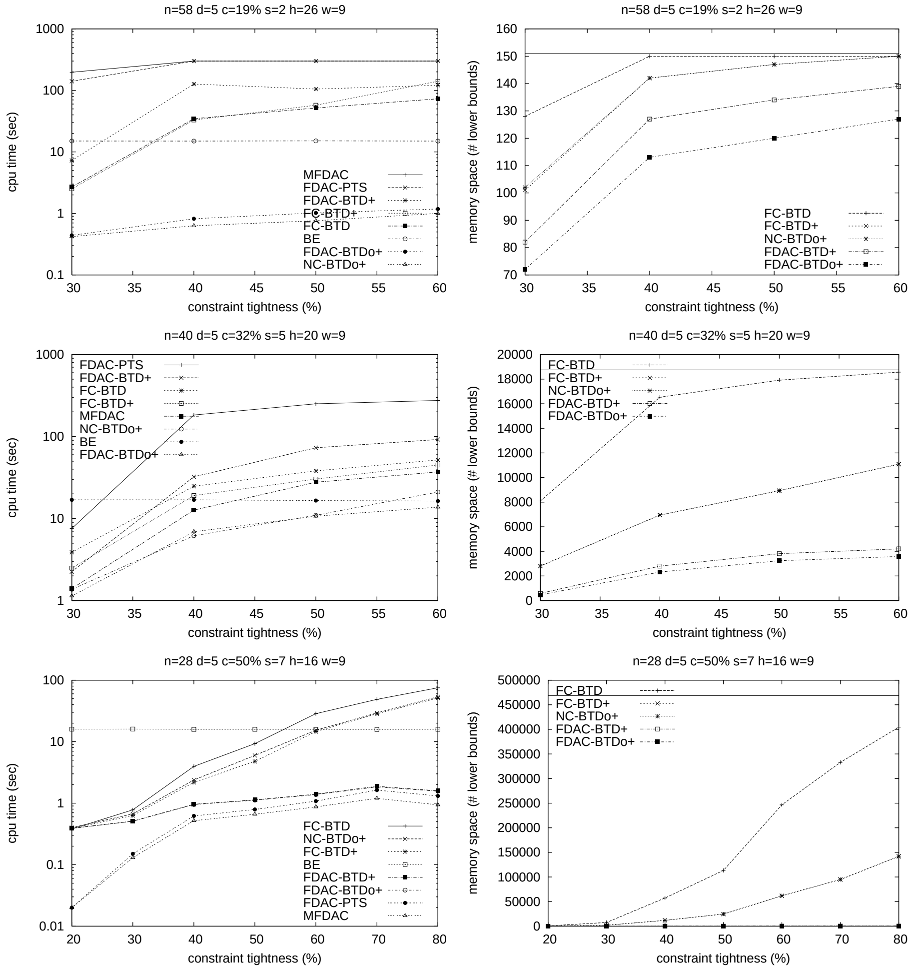

Fig. 1. Time in seconds (y-axis in log-scale) and memory space in number of recorded lower bounds (the theoretical maximum is given by an horizontal line) for solving (without an initial upper bound) random clique trees with clique size equal to 10 (tree width equals to $w = 9$) and separator size s equal to 2,5, and 7 (corresponding tree heights h equal to 26, 20, 16). Methods are sorted from the worst (top) to the best (bottom). n is the number of variables with domain sizes equal to d and c is the constraint graph connectivity (100% means a complete graph).

of variables ($s = 2, n = 58$) in less than 5 min. On the contrary, it performed the best with $s = 7, n = 28$. Finally, bucket elimination took constant time and space ($d^w = 5^9 = 1.910^6$).

8.2 Real-world instances

The Radio Link Frequency Assignment Problem (RLFAP) [1] is a resource allocation problem where the goal is to assign frequencies to a set of radio links in such a way that all the links may operate together without noticeable interference. Some RLFAP instances can be naturally cast as weighted CSPs with binary soft constraints. We focussed on subinstances of SCEN-06 and SCEN-07 which have constraint costs in $\{1, 10, 10^2, 10^3\}$ and $\{1, 10^2, 10^4, 10^6\}$ respectively. For efficiency reasons, we provided the optimum value as the initial global upper bound, we used the *min degree* heuristic for tree decomposition, and we employed *2-sided Jeroslow like* heuristic for variable selection. SUBCELAR$_i$ instances contain a subset of the SCEN-06 variables. Instance SCEN-06-24r has been obtained by removing 24 values per domain from the original instance and relaxing the constraints such that the optimum is a lower bound of SCEN-06[2]. SCEN-07-10^4-30r has been obtained by the same process and also by removing all the constraints with costs 1 and 10^2. In the following Table, we give the resulting time in seconds and number of recorded LB. BTD methods using FDAC saved a lot of space compared to methods using a weaker local consistency (e.g. 97 times for SCEN-06-24r between NC-BTDo$^+$ and FDAC-BTDo$^+$). Moreover these methods were able to solve all the instances within the time and node limits (up to 23 repetition times and mean repetition ≤ 3.9). On the largest instance, FDAC-BTDo$^+$ was faster than FDAC-BTDo* having more subproblem repetitions. Bucket elimination didn't solve any instance due to the 4 GB limit.

Method	SUBCELAR$_1$ $n = 14, d = 44$ $w = 9, h = 14$		SUBCELAR$_2$ $n = 16, d = 44$ $w = 10, h = 14$		SUBCELAR$_3$ $n = 18, d = 44$ $w = 12, h = 15$		SCEN-06-24r $n = 99, d = 20$ $w = 10, h = 37$		SCEN-07-10^4-30r $n = 196, d = 14$ $w = 12, h = 32$	
	time	#goods	time	#goods	time	#goods	time	#goods	time	#goods
FC-BTD	7489	4344	9234	2252	23560	69808	-	3002547	-	109311451
NC-BTDo$^+$	2647	4344	2442	1560	11342	9024	9231	6326854	38	83557
FDAC-BTDo$^+$	66	0	44	0	653	0	4789	65270	15	5874
FDAC-BTDo*	65	0	43	0	638	0	4573	64571	30	9247
FDAC-PTS	67	0	42	0	675	0	-	0	20277	0
MFDAC	14	0	20	0	193	0	-	0	-	0

9 Conclusion

In this paper, we have progressively introduce state-of-the-art soft local consistency in the BTD algorithm [15, 7] resulting in several new versions having different time complexities. All versions have the same space complexity as BTD.

[2] See http://www.inra.fr/bia/ftp/T/VCSP.

The combination of unrestricted soft local consistency with BTD makes the time complexity exponential in the tree-height, instead of the tree-width as in the original BTD. However, it has been shown on randomly generated clique trees and RLFAP subinstances (up to 196 variables) that these new versions using FDAC can be several orders of magnitude faster than BTD using forward checking, especially when they take into account recorded lower bounds as soon as possible. Moreover, FDAC-BTD algorithms offer important savings in terms of memory space actually used by the recording mechanism.

Further experiments on other problems should be done in order to better characterize the performance of FDAC-BTD compared to FC-BTD, especially in the case of unrestricted soft local consistency. In the future, we want to study and if possible limit subproblem repetition.

References

1. B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints Journal*, 4:79–89, 1999.
2. M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154:199–227, 2004.
3. M.C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3):311 – 342, 2003.
4. S. de Givry, M. Zytnicki, F. Heras, and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *Proc. of IJCAI-05*, Edinburgh, Scotland, 2005.
5. Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.
6. Eugene C. Freuder and Michael J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proc. of the 9^{th} IJCAI*, pages 1076–1078, Los Angeles, CA, 1985.
7. P. Jégou and C. Terrioux. Decomposition and good recording. In *Proc. of ECAI-2004*, pages 196–200, Valencia, Spain, 2004.
8. R. E. Korf. Depth first iterative deepening : An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.
9. J. Larrosa. On arc and node consistency in weighted CSP. In *Proc. AAAI'02*, pages 48–53, Edmondton, (CA), 2002.
10. J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. of the 18^{th} IJCAI*, pages 239–244, Acapulco, Mexico, 2003.
11. J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc-consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.
12. R. Marinescu and R. Dechter. And/or tree search for constraint optimization. In *CP-2004 workshop on Soft Constraints and Preferences*, Toronto, Canada, 2004.
13. T. Schiex. Arc consistency for soft constraints. In *Proc. of CP-2000*, volume 1894 of *LNCS*, pages 411–424, Singapore, 2000.
14. R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.
15. C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proc. of CP-2003*, pages 709–723, Kinsale, Ireland, 2003.