Russian Doll Search with Tree Decomposition

M. Sanchez, D. Allouche, S. de Givry, and T. Schiex

INRA, Applied Math. & CS Dept., France
{msanchez,allouche,degivry,tschiex}@toulouse.inra.fr

Abstract. Depth-First Branch and Bound is commonly used to solve constraint optimization problems such as weighted constraint satisfaction problems. The worst-case time complexity of this algorithm can be improved by exploiting the constraint graph structure, identifying independent subproblems and caching their optima. However, the exploitation of the structure is done *a posteriori*: each time a new subproblem occurs, it has to be solved before its optimum can be used. We propose to solve a relaxation of every subproblem before solving the whole problem, in the spirit of the Russian Doll Search algorithm. This relaxation allows to exploit subproblem lower bounds in a more proactive way. Experimental results on three large structured benchmarks show the benefit of this approach.

1 Introduction

Graphical model processing is a central problem in AI. The optimization of the combined cost of local cost functions, central in the valued CSP framework [23], captures problems such as weighted MaxSAT, Weighted CSP or Maximum Probability Explanation in probabilistic networks. It has applications in e.g., *resource allocation* [25, 1] and *bioinformatics* [19, 21].

In the last years, in order to solve satisfaction, optimization or counting problems, several algorithms have been proposed that simultaneously exploit a decomposition of the graph of the problem and the propagation of hard information using local consistency enforcing. This includes algorithms such as Recursive Conditioning [8], Backtrack bounded by Tree Decomposition (BTD) [24], AND-OR tree and graph search [18, 19], all related to Pseudo-Tree Search [11].

In this paper, we further explore the approach developed in [9] based on an extension of BTD to optimization [13]. The main drawback of BTD is a restriction on the dynamic variable ordering heuristic. In [12], the authors extend BTD by partially removing the variable ordering restriction, while bounding the worst-case time complexity by a user-controlled parameter. In [14], the restriction is completely removed but at the possible cost of loosing the time complexity bound. We propose to keep the variable ordering restriction but to take more advantage of the problem decomposition. Such a decomposition defines a set of subproblems whose size is less than the number of variables. By solving relaxations of these subproblems successively, it is possible to exploit a better lower bound which is crucial in optimization. We present this extension in the following sections. In the experiments, we show the benefit of this approach and give some practical ways to overcome the variable ordering restriction by shifting from a tree decomposition to a path decomposition or by merging clusters.

2 Weighted Constraint Satisfaction Problem

A Weighted CSP (WCSP) is a quadruplet (X, D, W, m). X and D are sets of n variables and finite domains, as in a standard CSP. The domain of variable i is denoted D_i. The maximum domain size is d. For a set of variables $S \subset X$, we note $\ell(S)$ the set of tuples over S. W is a set of cost functions. Each cost function (or soft constraint) w_S in W is defined on a set of variables S called its scope and assumed to be different for each cost function. A cost function w_S assigns costs to assignments of the variables in S i.e. $w_S : \ell(S) \rightarrow [0, m]$. The set of possible costs is $[0, m]$ and $m \in \{1, \ldots, +\infty\}$ represents an intolerable cost. Costs are combined by the bounded addition $\oplus$, such as $a \oplus b = \min\{m, a + b\}$ and compared using $\geq$. The operation $\ominus$ subtracts a cost b from a larger cost a where $a \ominus b = (a - b)$ if $a \neq m$ and m otherwise.

For unary/binary cost functions, we use simplified notations: a unary (resp. binary) cost function on variable(s) i (resp. i and j) is denoted w_i (resp. w_{ij}). If they do not exist, we add to W a unary cost function w_i for every variable i, and a nullary cost function, noted $w_\varnothing$ (a constant cost payed by any assignment). All these additional cost functions have initial cost 0, leaving the semantics of the problem unchanged.

The cost of a complete assignment $t \in \ell(X)$ in a problem $P = (X, D, W, m)$ is $Val_P(t) = \bigoplus_{w_S \in W} w_S(t[S])$ where $t[S]$ denotes the usual projection of a tuple on the set of variables S. The problem of minimizing $Val_P(t)$ is an optimization problem with an associated NP-complete decision problem.

Enforcing a given local consistency property on a problem P consists in transforming $P = (X, D, W, m)$ in a problem $P' = (X, D, W', m)$ which is equivalent to P ($Val_P = Val_{P'}$) and which satisfies the considered local consistency property. This enforcing may increase $w_\varnothing$ and provide an improved lower bound on the optimal cost. Enforcing is achieved using Equivalence Preserving Transformations (EPTs) moving costs between different scopes [22, 16, 7, 10, 4, 6, 5].

A classical complete solving method is Depth-First Branch and Bound (DFBB). We give its pseudo-code in Algorithm 1. It enforces at each search node a given local consistency property Lc (line 1). The pruning condition is applied if the resulting $w_\varnothing \geq m$ (line 2). m is updated to the cost of the last solution found (line 3). The initial call is DFBB(P, X, $\emptyset$). It assumes an already local consistent problem P and returns its optimum. P/A denotes the subproblem P under assignment A. The operator . is used to get an element of P. Function pop(S) returns an element of S and remove it from S.

DFBB worst-case time complexity is $O(d^n)$ and it uses linear space. In the next section, we briefly present how DFBB can be extended to exploit the problem structure.

3 Depth-First Branch and Bound with tree decomposition

Assuming connected problems, a tree decomposition of a WCSP is defined by a tree (C, T). The set of nodes of the tree is $C = \{C_1, \ldots, C_k\}$ where each C_e is a set of variables ($C_e \subset X$) called a cluster. T is a set of edges connecting clusters and forming a tree (a connected acyclic graph). The set of clusters C must cover all the variables ($\bigcup_{C_e \in C} C_e = X$) and all the cost functions ($\forall w_S \in W, \exists C_e \in C \text{ s.t. } S \subset C_e$). Furthermore, if a variable i appears in two clusters C_e and C_g, i must also appear in all the clusters C_f on the unique path from C_e to C_g in T.

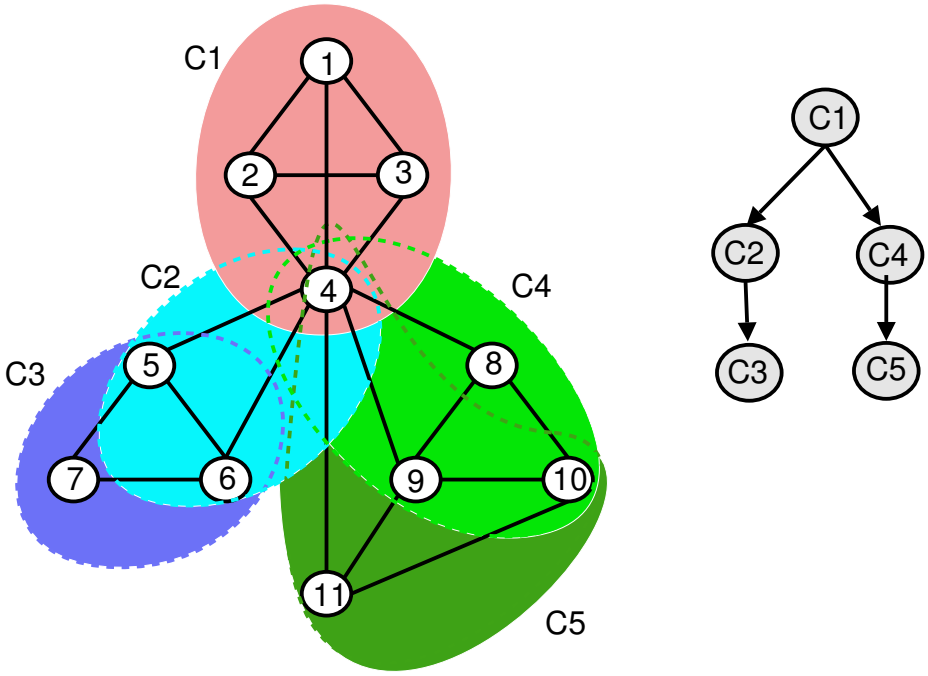

Fig. 1. The constraint graph of Example 1 and its associated tree decomposition.

For a given WCSP, we consider a rooted tree decomposition (C,T) with an arbitrary root C_1. We denote by $Father(C_e)$ (resp. $Sons(C_e)$) the parent (resp. set of sons) of C_e in T. The separator of C_e is the set $S_e = C_e \cap Father(C_e)$. The set of proper variables of C_e is $V_e = C_e \setminus S_e$.

The essential property of tree decompositions is that assigning S_e separates the initial problem in two subproblems which can then be solved independently. The first subproblem, denoted P_e, is defined by the variables of C_e and all its descendant clusters in T and by all the cost functions involving *at least* one proper variable of these clusters. The remaining cost functions, together with the variables they involve, define the remaining subproblem.

Example 1. Consider the MaxCSP problem depicted in Figure 1. It has eleven variables with two values (a,b) in their domains. Binary cost functions of difference ($w_{ij}(a,a) = w_{ij}(b,b) = 1, w_{ij}(a,b) = w_{ij}(b,a) = 0$) are represented by edges connecting the corresponding variables. In this problem, the optimal cost is 5 and it is attained with e.g. the assignment (a,b,b,a,b,b,a,b,b,a,b) in lexicographic order. A C_1-rooted tree decomposition with clusters $C_1 = \{1,2,3,4\}, C_2 = \{4,5,6\}, C_3 = \{5,6,7\}, C_4 = \{4,8,9,10\}$, and $C_5 = \{4,9,10,11\}$, is given on the right hand-side in Figure 1. For instance, C_1 has sons $\{C_2, C_4\}$, the separator of C_3 with its father C_2 is $S_3 = \{5,6\}$, and the set of proper variables of C_3 is $V_3 = \{7\}$. The subproblem P_3 has variables $\{5,6,7\}$ and cost functions $\{w_{5,7}, w_{6,7}, w_7\}$ (w_7 initially empty). P_1 corresponds to the whole problem.

Depth-First Branch and Bound with Tree Decomposition (BTD) exploits this property by restricting the variable ordering. Imagine all the variables of a cluster C_e are assigned before any of the remaining variables in its son clusters and consider a current assignment A. Then, for any cluster $C_f \in Sons(C_e)$, and for the current assignment A_f of the separator S_f, the subproblem P_f under assignment A_f (denoted P_f/A_f) can be solved independently from the rest of the problem. If memory allows, the *optimal cost* of P_f/A_f may be recorded which means it will never be solved again for the same assignment of S_f.

In [9], we show how to exploit a better initial upper bound for solving P_f. However this has the side-effect that the optimum of P_f may be not computed but only a lower bound. The lower bound and the fact it is optimal can be recorded in LB_{P_f/A_f} and Opt_{P_f/A_f} respectively, initially set to 0 and *false*.

As in DFBB, BTD enforces local consistency during search. However, local consistency may move costs between clusters, thereby invalidating previously recorded information. We store these cost moves in a specific *backtrackable* data structure ΔW as defined in [9]. During the search, we can obtain the total cost that has been moved out of the subproblem P_f/A_f by summing up all the $\Delta W_i^f(a)$ for all values (i,a) in the separator assignment A_f and correct any recorded information: $LB'_{P_f/A_f} = LB_{P_f/A_f} \ominus \bigoplus_{i \in S_f} \Delta W_i^f(A_f[i])$.

Moreover, we keep the nullary cost function local to each cluster: $w_\varnothing = \bigoplus_{C_e \in C} w_\varnothing^e$.

For pruning the search, BTD uses the maximum between local consistency and recorded lower bounds as soon as their separator is completely assigned by the current assignment A. We denote by $lb(P_e/A)$ this lower bound:

$$lb(P_e/A) = w_\varnothing^e \oplus \bigoplus_{C_f \in Sons(C_e)} \max(lb(P_f/A), LB'_{P_f/A_f}) \quad (1)$$

Example 2. In the problem of Example 1, variables $\{1,2,3,4\}$ of C_1 are assigned first, e.g. using a dynamic variable ordering *min domain / max degree* inside each cluster.

Let assume $A = \{(4,a),(1,a),(2,b),(3,b)\}$ be the current assignment[1]. Enforcing EDAC local consistency [10] on P_1/A produces $w_\varnothing^1 = 2, w_\varnothing^2 = w_\varnothing^4 = 1, w_\varnothing^3 = w_\varnothing^5 = 0$, resulting in $lb(P_1/A) = \bigoplus_{C_e \in C} w_\varnothing^e = 4$ (no lower bound recorded yet).

Then, subproblems $P_2/\{(4,a)\}$ and $P_4/\{(4,a)\}$ are solved independently, resulting in $LB_{P_2/\{(4,a)\}} = 1$, $LB_{P_4/\{(4,a)\}} = 2$, $Opt_{P_2/\{(4,a)\}} = Opt_{P_4/\{(4,a)\}} = true$ (no initial upper bound) which are recorded. A first complete assignment of cost $w_\varnothing^1 \oplus LB_{P_2/\{(4,a)\}} \oplus LB_{P_4/\{(4,a)\}} = 5$ (all ΔW costs are zero in this case) is found.

In Algorithm 1, we present the pseudo-code of the BTD algorithm combining tree decomposition and a given level of local consistency Lc. This algorithm uses our initial enhanced upper bound (line 4), value removal based on local cuts [9] and lower bound recording (lines 6 and 7). The initial call is BTD(P_1, V_1, $\emptyset$, 0), with $P_1 = P$, an already local consistent problem, returning its optimum.

The lower bound $lb(P_e/A)$ of Equation 1 does not take into account a possible recorded lower bound LB_{P_e/A_e}, which may exist if $Opt_{P_e/A_e} = false$ and the same subproblem is solved again. We therefore ensure a monotonically increasing lower bound

[1] Variable 4 has been selected first as it has the highest degree in C_1.

during the search by passing the best lower bound found recursively (line 5 and 9), resulting in a stronger pruning condition (line 8).

BTD worst-case time complexity is $O(md^{w+1})$ with $w = \max_{C_e \in C} |C_e| - 1$, the maximum cluster size minus one, called the tree-width of the tree decomposition (C, T). Its memory complexity is bounded by $O(d^s)$ with $s = \max_{C_e \in C} |S_e|$, the maximum separator size [9].

4 Russian Doll Search with tree decomposition

The original Russian Doll Search (RDS) algorithm [25] consists in solving n nested subproblems of an initial problem P with n variables. Given a fixed variable order, it starts by solving the subproblem with only the last variable. Next, it adds the preceding variable in the order and solves this subproblem with two variables, and repeats this process until the complete problem is solved. Each subproblem is solved by a DFBB algorithm with a static variable ordering heuristic following the nested subproblem decomposition order. The lower bound combines the optimum of the previously solved subproblems with the lower bound produced by enforcing soft local consistency (only a basic form of local consistency similar to Node Consistency [16] in the original RDS algorithm).

We propose to apply the RDS principle to a tree decomposition (RDS-BTD). The main difference with RDS is that the set of subproblems to solve is defined by a rooted tree decomposition (C, T).

We define P_e^{RDS} as the subproblem defined by the proper variables of C_e and all its descendant clusters in T and by all the cost functions involving *only* proper variables of these clusters. P_e^{RDS} has no cost function involving a variable in S_e, the separator with its father, and thus its optimum is a lower bound of P_e for any assignment of S_e.

RDS-BTD solves $|C|$ subproblems ordered by a depth-first traversal of T, starting from the leaves to the root $P_1^{RDS} = P_1$.

Each subproblem P_e^{RDS} is solved by BTD instead of DFBB. This allows to exploit decomposition and caching done by BTD. Because caching is only performed on completely assigned separators, and considering all possible assignments of S_e would be too costly in memory and time, we assign S_e before solving P_e^{RDS}. This is needed since otherwise, caching on P_f, a descendant of C_e, with $S_f \cap S_e \neq \emptyset$, would use a partially assigned A_f. To assign S_e, we use the fully supported value of each domain[2] (maintained by EDAC [10]) as temporary values used for caching purposes only.

Another better alternative would be to cache lower bounds for partial assignments. It should improve the pruning efficiency in subsequent searches, but this involves a more complex cache management that we leave for future work.

The advantage of using BTD is that recorded lower bounds can be reused during the next iterations of RDS-BTD. However, the optimum found by BTD for a given subproblem P_f when solving P_e^{RDS} is no more valid in $P_{Father(e)}^{RDS}$ due to possible cost

[2] Fully supported value $a \in D_i$ such that $w_i(a) = 0$ and $\forall w_S \in W$ *with* $i \in S, \exists t \in \ell(S)$ *with* $t[i] = a$ such that $w_S(t) = 0$.

Algorithm 1: DFBB, BTD, and RDS-BTD algorithms.

Function DFBB(P, V, A) : $[0,+\infty]$
  if $(V=\emptyset)$ **then**
    return $P.w_\varnothing$ /* *A new solution is found for P* */ ;
  else
    i :=pop(V) /* *Choose an unassigned variable of P* */ ;
    $d := P.D_i$;
    /* *Enumerate every value in the domain of i* */ ;
    while $(d \neq \emptyset$ and $P.w_\varnothing < P.m)$ **do**
      a :=pop(d) /* *Choose a value* */ ;
1   P' :=Lc($P/A \cup \{(i,a)\}$) /* *Enforce local consistency on* $P/A \cup \{(i,a)\}$ */ ;
2   **if** $(P'.w_\varnothing < P.m)$ **then**
3       $P.m$:=DFBB(P', V, $A \cup \{(i,a)\}$) ;
    return $P.m$;

Function BTD(P_e, V, A, blb) : $[0,+\infty]$
  if $(V=\emptyset)$ **then**
    $S := Sons(C_e)$;
    /* *Solve all cluster sons whose optima are unknown* */ ;
    while $(S \neq \emptyset$ and $lb(P_e/A) < P_e.m)$ **do**
      C_f :=pop(S) /* *Choose a cluster son* */ ;
      if $(not(Opt_{P_f/A_f}))$ **then**
4       $P_f.m := P_e.m \ominus lb(P_e/A) \oplus lb(P_f/A_f)$;
5   res :=BTD(P_f, V_f, A, $lb(P_f/A_f)$) ;
6       $LB_{P_f/A_f} := res \oplus \bigoplus_{i \in S_f} \Delta W_i^f(A[i])$;
7       $Opt_{P_f/A_f} := (res < P_f.m)$;
    return $lb(P_e/A)$ /* *A new solution is found for* P_e */ ;
  else
    i :=pop(V) /* *Choose an unassigned variable in* C_e */ ;
    $d := P_e.D_i$;
    /* *Enumerate every value in the domain of i* */ ;
    while $(d \neq \emptyset$ and $\max(blb, lb(P_e/A)) < P_e.m)$ **do**
      a :=pop(d) /* *Choose a value* */ ;
      P'_e :=Lc($P_e/A \cup \{(i,a)\}$) /* *Enforce local consistency on* $P_e/A \cup \{(i,a)\}$ */ ;
8   **if** $(\max(blb, lb(P'_e/A \cup \{(i,a)\})) < P_e.m)$ **then**
9       $P_e.m$:=BTD(P'_e, V, $A \cup \{(i,a)\}$, $\max(blb, lb(P'_e/A \cup \{(i,a)\}))$) ;
    return $P_e.m$;

Function RDS-BTD(P, P_e^{RDS}) : $[0,+\infty]$
  foreach $C_f \in Sons(C_e)$ **do**
    RDS-BTD(P, P_f^{RDS}) ;
10 $P_e^{RDS}.m := P.m \ominus lb(P/\emptyset) \oplus lb(P_e^{RDS}/\emptyset)$;
11 $LB_{P_e^{RDS}}$:=BTD(P_e^{RDS}, V_e, $\{(i,$EAC(i)$) | i \in S_e\}$, $lb(P_e^{RDS}/\emptyset)$) ;
12 Set to *false* all recorded $Opt_{P_f/A}$ such that C_f is a descendant of C_e, $S_f \cap S_e \neq \emptyset$, $A \in \ell(S_f)$;
  return $LB_{P_e^{RDS}}$;

functions between variables in $C_{Father(e)}$ and in P_f. At each iteration of RDS-BTD, after P_e^{RDS} is solved, we reset all Opt_{P_f/A_f} such that $S_f \cap S_e \neq \emptyset$ (line 12).

During search, RDS-BTD exploits the maximum between local consistency, recorded, and RDS lower bounds. Let $LB_{P_e^{RDS}}$ denote the optimum of P_e^{RDS} found by one iteration of RDS-BTD. Because costs can be moved between clusters, this information has to be corrected in order to be valid in the next iterations of RDS-BTD. For that, we use the maximum of ΔW on each current domain of the (possibly unassigned) separator variables. The lower bound corresponding to the current assignment A is then:

$$lb(P_e/A) = w_{\varnothing}^e \oplus \bigoplus_{C_f \in Sons(C_e)} \max(lb(P_f/A), LB'_{P_f/A_f}, LB_{P_f^{RDS}} \ominus \bigoplus_{i \in S_f} \max_{a \in D_i} \Delta W_i^f(a)) \quad (2)$$

Example 3. Applied on the problem of Example 1, RDS-BTD solves five subproblems $(P_3^{RDS}, P_2^{RDS}, P_5^{RDS}, P_4^{RDS}, P_1)$ successively. For instance, P_3^{RDS} has variable $\{7\}$ and cost function $\{w_7\}$. Before solving P_3^{RDS}, RDS-BTD assigns variables $\{5,6\}$ of the separator S_3 to their fully supported value ($\{(5,a),(6,a)\}$ in this example). In solving P_2^{RDS}, it can record e.g. the optimum of $P_3/\{(5,a),(6,a)\}$, equal to zero (recall that $w_{5,6}$ does not belong to P_3), that can be reused when solving P_1. In solving P_4^{RDS}, it can record e.g. the optimum of $P_5/\{(4,a),(9,a),(10,a)\}$, also equal to zero. However, due to the fact that variable 4 belongs to $S_5 \cap S_4$ and P_4^{RDS} does not contain $w_{4,11}$, this recorded information is only a lower bound for subsequent iterations of RDS-BTD. So, we set $Opt_{P_5/\{(4,a),(9,a),(10,a)\}}$= *false* before solving P_1. The resulting optima are: $LB_{P_3^{RDS}} = LB_{P_5^{RDS}} = 0, LB_{P_2^{RDS}} = LB_{P_4^{RDS}} = 1$ and $LB_{P_1^{RDS}} = 5$, the optimum of P_1.

In this simple example, for $A = \{(4,a),(1,a),(2,b),(3,b)\}$, $lb(P_1/A)$ using Equation 1 or 2 is the same because EDAC propagation provides lower bounds equal to RDS lower bounds. In the contrary, for $A = \emptyset$, $lb(P_1/\emptyset) = LB_{P_2^{RDS}} \oplus LB_{P_4^{RDS}} = 2$ using Equation 2 and $lb(P_1/\emptyset) = 0$ using Equation 1 (assuming EDAC local consistency in preprocessing and no initial upper bound).

We present the pseudo-code of the RDS-BTD algorithm in Algorithm 1. RDS-BTD call BTD to solve each subproblem P_e^{RDS} (line 11), using Equation 2 instead of Equation 1 to compute lower bounds. An initial upper bound for P_e^{RDS} is deduced from the global problem upper bound and the already computed RDS lower bounds (line 10). It initially assigns variables in S_e to their fully supported value (given by EAC function at line 11) as discussed above. The initial call is RDS-BTD(P, P_1^{RDS}). It assumes an already local consistent problem $P_1^{RDS} = P$ and returns its optimum.

Notice that as soon as a solution of P_e^{RDS} is found having the same optimal cost as $lb(P_e^{RDS}/\emptyset) = \bigoplus_{C_f \in Sons(C_e)} LB_{P_f^{RDS}}$, then the search ends thanks to the initial lower lound given at line 11.

The time and space complexity of RDS-BTD is the same as BTD. Notice that RDS-BTD without caching nor local consistency (except node consistency) and a pseudo-tree based tree decomposition (i.e. a cluster for each variable, implying a static variable ordering) is equivalent to Pseudo-Tree RDS [15]. If we further restrict the algorithm to use a specific tree decomposition (C,T) such that $|C| = n, \forall e \in [1,n], C_e = \{1,\ldots,e\}$, and $\forall e \in [2,n], Father(C_e) = C_{e-1}$, then it becomes equivalent to RDS.

5 Experimental results

5.1 Implementation details

We implemented DFBB, BTD, RDS-BTD, and RDS in an open-source C++ solver named `toulbar2`[3]. Dynamic variable ordering (*min domain / max degree*, breaking ties with maximum unary cost) is used inside clusters (BTD and RDS-BTD) and by DFBB. EDAC local consistency is enforced on binary [10] and ternary [21] cost functions during search, and for the CELAR and TagSNP benchmarks, VAC [5] is also enforced in preprocessing. RDS enforces Node Consistency [16] only. We use the Maximum Cardinality Search heuristic (except for the SPOT5 benchmark) to build a tree decomposition and choose the largest cluster as the root. A variable ordering compatible with this rooted tree decomposition is used for DAC local consistency [7] and RDS.

Recorded (and if available RDS) lower bounds are exploited by local consistency enforcing as soon as their separator variables are fully assigned. If the recorded lower bound is optimal ($Opt_{P_e/A_e} = true$) or strictly greater than the one produced by local consistency, i.e. $\max(LB'_{P_e/A_e}, LB_{P_e^{RDS}} \ominus \bigoplus_{i \in S_e} \Delta W_i^e(A[i]) > \bigoplus_{P_f \subseteq P_e} w_\varnothing^f$, then the corresponding subproblem (P_e/A_e) is deconnected from local consistency enforcing and the positive difference in lower bounds is added to its parent cluster lower bound ($w_\varnothing^{Father(C_e)}$), allowing possible new value removals by node consistency enforcing on the remaining problem.

All the solving methods exploit a binary branching scheme depending on the domain size d of the branching variable. If $d > 10$ then it splits the *ordered* domain into two parts (by taking the middle value), else the variable is assigned to its EDAC fully supported value or this value is removed from the domain. In both cases, it selects the branch which contains the fully supported value first, except for RDS and BTD-like methods where it selects the branch which contains the value corresponding to the last solution(s) found first if available.

The dynamic variable ordering heuristic is modified by a conflict back-jumping heuristic as suggested in [17]. It branches on the same variable again if the first branch in the binary branching scheme was directly pruned by propagation.

No initial upper bound is provided. The next tables show CPU time in seconds on a 3 GHz computer with 16GB (2.2GHZ and 4 GB for the TagSNP benchmark) for the four methods to find the optimum and prove its optimality on three large structured benchmarks, including a new proposed benchmark in the field of bioinformatics.

5.2 More freedom to the dynamic variable ordering

In order to relax the restriction imposed by BTD and RDS-BTD on the dynamic variable ordering heuristic, we propose to merge clusters with their parent if their separator is too large, as proposed in [12]. For that, we add a parameter r to limit the maximum size of the separators. Starting from the leaves of a given tree decomposition, we merge a cluster with its parent if the separator size is strictly greater than r. If $r = +\infty$, no separator restriction applies.

[3] Version 0.7 available at http://mulcyber.toulouse.inra.fr/gf/project/toulbar2

By definition, BTD and RDS-BTD solve independent subproblems (i.e. the sons of a father cluster P_e) sequentially (at line 5 of Algorithm 1) although the overall cost of these problems is bounded by a local upper bound (the upper bound $P_e.m$ associated to the father cluster). If the known (local consistency, recorded, and RDS) lower bound $lb(P_e/A)$ including these son clusters is weak, then BTD, and even RDS-BTD, may spend time by solving a son cluster to optimality whereas they could rapidly, e.g. with a dynamic variable ordering on all the remaining variables, produce a local lower bound greater than the father upper bound. This bad phenomenon has also been observed in [15] and recently in [14]. In order to avoid this problem occuring with multiple sons, we propose to exploit a path decomposition. A path decomposition of a WCSP is a tree decomposition (C,T) where T is a path. In the experiments, we use the same variable elimination ordering to build a path decomposition as to build a tree decomposition. The keyword *path* in the following tables corresponds to the results obtained with a path decomposition (and no cluster merging).

5.3 Radio Link Frequency Assignment

The problem consists in assigning frequencies to a set of radio links in such a way that all the links may operate together without noticeable interference [1]. We select two difficult instances provided by the French military institute CELAR. These instances contain binary cost functions only. Instance *scen*07*m* is a relaxation of the original instance *scen*07 where domain values have been merged into abstract values and costs lower than 10000 have been discarded. A "-" means the instance was not solved in less than ten hours. Among the different methods, only RDS-BTD can solve the two instances. Keeping small separators only increases CPU times by a factor of two, while using a path decomposition exceeds the time limit.

Instance	n	d	tree-width (w)			DFBB	BTD	RDS-BTD			RDS
			$r=+\infty$	$r=4$	*path*			$r=+\infty$	$r=4$	*path*	
scen06	100	44	11	19	54	4768	**665**	795	1639	-	-
scen07m	196	14	15	41	45	-	-	**3**	5.5	-	-

5.4 Earth Observation Satellite Scheduling

This benchmark deals with the daily management of SPOT5 earth observation satellite [25]. The goal is to select a set of feasible photographs maximizing a weighted sum, each photograph having a weight expressing its importance. Variables represent candidate photographs along the temporal axis. We keep this temporal order as our variable elimination ordering to build a tree or a path decomposition. Each domain corresponds to the possible instruments to take a photograph and a special value to reject this photograph. Maximum domain size is 4. Hard binary and ternary constraints express non overlapping and minimal transition time between two successive photographs on the same instrument. A unary cost function per variable express its importance.

For RDS-BTD, we tried two different local consistencies during search: EDAC, and a weaker form, soft AC [22, 16]. We observed better results using AC for methods exploiting RDS lower bounds. This is due to the ΔW correction applied on RDS

lower bounds. In the case of AC, this correction is always zero because there are only hard constraints. AC moves infinite (m) costs between clusters, which does not change subproblem optima, whereas EDAC can move finite (unary) costs.

RDS-BTD performed often better using a path decomposition rather than a tree decomposition on this benchmark. A "-" means the instance was not solved in less than thrity minutes. Merging clusters is not a good strategy here. The best results are still obtained by the original RDS algorithm which uses a completely static variable ordering.

Ins.	n	tree-width (w)			DFBB	BTD	RDS-BTD						RDS
		$r=+\infty$	$r=4$	$path$			$r=+\infty$		$r=4$		$path$		
							EDAC	AC	EDAC	AC	EDAC	AC	
29	82	17	31	18	1	0	0	0	0	0	0	0	**0**
42	190	36	66	56	-	-	769	692	1057	-	1457	669	**2**
54	67	14	21	20	0	0	0	0	0	0	0	**0**	0
404	100	22	39	30	168	0	0	0	0	0	1	**0**	1
408	200	55	127	64	-	-	-	-	-	-	-	62	**10**
412	300	55	228	79	-	-	-	-	-	-	-	94	**22**
414	364	110	289	150	-	-	-	-	-	-	-	478	**63**
503	143	19	43	30	-	29	3	18	1	6	1	**0**	1
505	240	39	175	59	-	-	1695	1473	-	-	323	23	**10**
507	311	76	236	103	-	-	-	-	-	-	1098	1032	**49**
509	348	97	273	134	-	-	-	-	-	-	-	619	**85**

These preliminary results show that BTD can take advantage of RDS and path decompositions, although some work is still needed to reach the efficiency of RDS.

5.5 Tag SNP selection

This problem occurs in genetics. SNP (Single Nucleotide Polymorphism) are typos mismatch in DNA sequences between members of a species. There are important factors responsible of polymorphism in species. They may explain, for instance, a portion of the heritable risk of common diseases and can affect respond to pathogens, chemicals, drugs, vaccines, and other agents. However, their greatest importance in biomedical research is for comparing regions of the genome between cohorts (such as with matched cohorts with and without a disease). The study of SNPs is also important in crop and livestock breeding programs.

The TagSNP problem consists in selecting a subset of SNP markers along a chromosome such as the selected markers, called tag SNPs, are mostly representative of the genetic information of the whole set of markers. The goal is to capture a maximally informative subset of SNPs for further screening of larger population.

For each pair of markers, linkage disequilibrium (LD) measures the pairwise correlation between the two SNPs. LD is a distance measure obtain from statistical analysis of occurrence of SNPs in an initial cohort.

A marker can represent another marker if and only if it is selected and the distance between the two markers is lower than a user-defined maximum threshold. A given

threshold - usually 0.8 [3] - allows to define a graph where each node is a marker and the edges are labelled by the LD distance linking pairs of nodes. Edges are filtered if their label is lower than the threshold. Thus, the initial graph after filtering and decomposition gives a set of connected components. See an example in Figure 2 of a connected component. In the present benchmark, each connected component is processed separately as an independent graph G corresponding to a WCSP instance.

Interestingly, the instance collection covers a large diversity of problem size and structure, the size varies from few tens to several hundred of variables, and the graph connectivity spans from 18% to 93%.

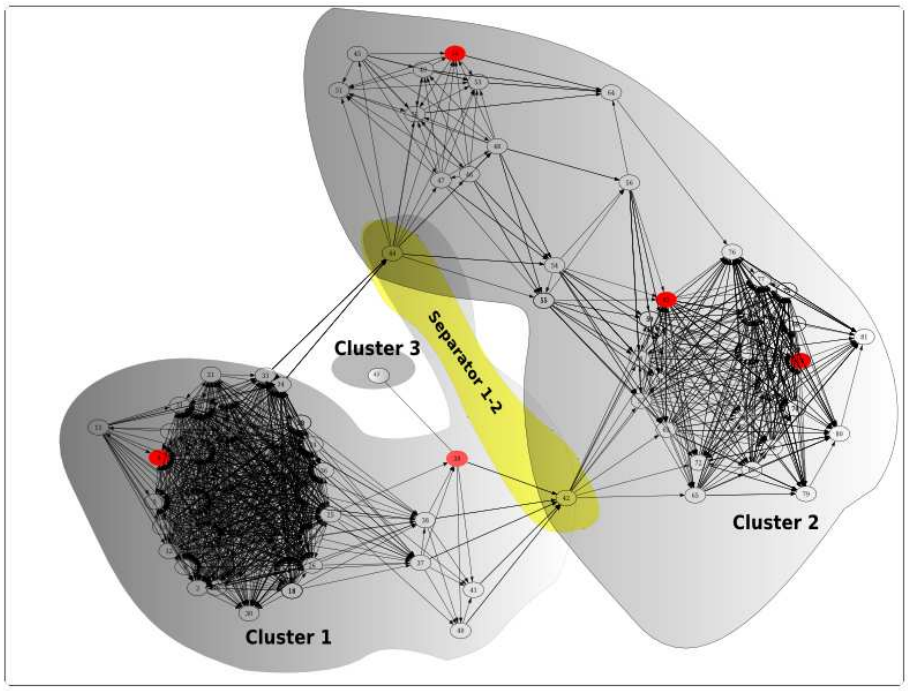

Fig. 2. The constraint graph of tag SNP selection instance 14481 covered by three clusters (maximum separator size $r = 4$). Root cluster is C_1. The separator $S_2 = C_1 \cap C_2$ is highlighted and contains two variables $S_2 = \{42, 44\}$. An optimal solution is given in red color.

For each connected component, we define a binary WCSP from G. Each variable represents a node/marker. Its domain represents the fact that the marker is selected or that it is represented by another marker in the neighborhood of the node in G. So, domain size is equal to the degree of the considered node in G plus one. There is a unary cost function for each node of G such that the problem is to minimize the number of selected (tag SNP) markers. There is a binary cost function for each edge of G. The

costs make sure that if a marker is represented by another marker then this marker must be selected, and also take into account additional secondary coverage criteria based on LD as defined in [20] (for the same minimum number of tag SNPs, maximize weighted coverage sum of unselected markers and maximize dispersion between tag SNPs). Note that this problem is similar to a set covering problem with additional binary costs.

The benchmark is composed of 80 instances coming from human chromosome 1 by courtesy of Steve Qin.

The next table shows the number of instances solved by each method in less than two hours. The second line gives the total amount of CPU time in seconds to solve the 47 instances solved by all the methods, except RDS. We observed that RDS-BTD with cluster merging ($r = 4$) is the best option.

	DFBB	BTD	RDS-BTD			RDS
			$r = +\infty$	$r = 4$	$path$	
Nb. solved	68	56	55	72	57	2
Total time	2007	4416	3460	1748	6882	*N/A*

Detail results (except RDS) of CPU time in seconds for all the instances sorted by increasing number of variables are given in Figure 3 and 4. For each instance, the best solving time is given in bold.

Future work remains to be done to understand when it is worthwhile to exploit the problem structure on this benchmark.

6 Conclusion

The decomposability of problems offered by treewidth is not always easy to successfully exploit in the context of optimization problems. In strongly decomposable problems, with small separators, exploiting decomposition is always a win. For more complex situations, the possible loss in lower bound quality for a given subproblem may adversely affect the efficiency. By precomputing subproblem based lower bounds as in the RDS algorithm, RDS-BTD can offer more overall robustness than other algorithms such as RDS or DFBB. The influence of other parameters such as the maximum separator size or the type of decomposition used (path or tree decomposition) requires additional analysis.

Future work remains to be done to implement caching on partial assignments. It will reduce redundant searches between successive RDS-BTD iterations. This is also important if we further combine RDS-BTD with Iterative Deepening as done in [2].

Acknowledgments This research has been partly funded by the French *Agence Nationale de la Recherche* (STALDECOPT project).

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.

Fig. 3. Tag SNP selection instances with less than 80 variables.

Instance	n	d	tree-width (w)			DFBB	BTD	RDS-BTD		
			$r=+\infty$	$r=4$	$path$			$r=+\infty$	$r=4$	$path$
17289	33	16	11	12	16	**0**	0	0	0	0
31481	33	22	12	18	26	0	**0**	0	0	0
16864	35	25	14	23	29	**1**	1	1	1	1
29401	35	21	16	20	21	0	**0**	0	0	0
20927	38	15	9	14	26	**0**	0	0	0	0
25023	40	21	16	24	20	**0**	0	0	0	0
9881	42	21	13	18	20	0	0	0	**0**	0
27116	43	23	19	21	24	1	0	0	**0**	0
31968	43	19	12	17	18	0	0	0	0	**0**
19684	45	23	16	21	28	**0**	0	0	0	0
31675	46	23	21	23	23	1	1	1	1	**1**
11053	47	31	23	29	43	4	2	2	**1**	16
30162	48	27	19	27	26	**1**	1	1	1	1
10734	49	20	17	21	19	0	1	**0**	0	3
19674	50	28	20	22	32	**0**	0	0	0	0
29043	50	42	35	47	41	5	**3**	7	5	4
6656	54	29	20	39	28	**1**	1	1	1	132
2480	55	49	31	52	48	9	11	**8**	8	10
28188	55	37	30	37	36	1	-	-	**1**	2
35624	55	25	21	24	25	0	0	**0**	0	3
42042	55	43	33	44	42	3	**2**	2	2	3
8892	55	33	24	33	33	1	322	**1**	1	4
3935	57	43	26	45	43	**1**	2	2	2	2
34916	59	40	26	29	45	28	154	172	**4**	106
13106	61	52	34	40	52	**35**	36	41	41	37
21758	63	58	32	61	57	5	4	**3**	4	10
22707	63	32	20	32	32	9	527	49	**1**	522
23384	63	54	37	38	53	**2**	2	2	2	2
1968	64	48	28	49	48	28	20	21	**5**	70
28202	64	30	19	63	37	86	5922	-	**38**	-
19655	65	37	27	63	57	**11**	19	15	23	-
27505	65	46	32	61	58	3	3	2	**2**	2
30799	67	62	52	62	61	367	**128**	135	282	624
3567	67	51	37	51	51	**1**	2	2	1	2
20037	75	68	55	72	70	**14**	14	15	15	17
38928	75	69	53	73	68	**7**	9	9	8	8
35418	77	52	35	72	66	**55**	-	-	6633	635
30644	78	68	38	77	73	13	32	34	**8**	-
18941	79	63	52	66	62	154	-	-	**37**	293

Fig. 4. Tag SNP selection instances with more than 80 variables.

Instance	n	d	tree-width (w)			DFBB	BTD	RDS-BTD		
			$r=+\infty$	$r=4$	$path$			$r=+\infty$	$r=4$	$path$
18827	80	74	62	76	75	**15**	-	-	15	27
18948	80	51	34	40	58	-	-	-	**7**	20
5013	80	66	43	76	70	3982	-	-	**23**	250
30395	81	75	70	77	74	274	324	335	282	**269**
14481	82	40	34	42	39	**39**	680	911	44	-
13809	84	65	52	74	68	**9**	11	11	35	220
3007	84	80	42	71	79	**104**	115	118	136	1877
12929	85	60	43	82	59	27	-	-	**23**	-
30393	85	70	49	84	69	322	-	1318	**318**	-
30557	87	37	27	36	48	33	138	**6**	38	72
29624	88	36	29	41	35	42	29	**29**	30	684
19648	91	74	43	43	73	**3**	4	4	4	4
30495	91	68	52	72	67	40	-	-	15	**14**
26578	93	38	26	43	38	89	25	**11**	47	-
14976	94	35	27	47	34	-	-	-	**49**	-
38837	94	89	81	93	92	**36**	44	45	42	40
8681	95	31	25	65	41	157	-	-	**41**	245
28454	96	56	38	40	61	288	-	-	**132**	-
22631	99	95	64	73	94	**183**	192	184	208	217
17904	100	79	54	94	79	-	-	-	-	-
18603	104	57	47	58	56	99	166	87	**83**	92
17271	107	76	70	73	75	438	**437**	438	440	441
23626	107	65	52	53	64	**13**	16	-	14	15
20125	113	106	64	111	106	36	1663	1687	**19**	1262
15282	114	35	27	39	41	202	22	**18**	21	81
23129	114	83	65	108	85	**852**	-	-	956	2126
37610	114	65	38	56	93	13	**6**	7	10	-
18016	116	113	109	112	112	**18**	24	21	21	39
15183	119	104	75	118	113	-	-	-	-	-
17929	119	113	89	114	112	-	493	541	**174**	-
43205	121	113	98	120	112	**860**	-	-	875	-
31466	123	91	57	118	90	24	23	**22**	64	-
32987	123	92	43	49	92	408	-	-	**9**	-
39997	124	100	85	113	102	-	-	-	-	-
30933	132	75	63	87	75	-	**77**	-	-	-
23659	150	114	105	112	117	**268**	-	-	887	1768
19984	174	97	69	145	118	-	-	-	-	-
37188	187	104	89	105	103	-	-	71	**65**	-
28203	215	120	75	213	148	-	-	-	-	-
9727	238	79	-	-	-	-	-	-	-	-
14194	251	175	-	249	-	-	-	-	-	-

2. B. Cabon, S. de Givry, and G. Verfaillie. Anytime Lower Bounds for Constraint Optimization Problems. In *Proc. of CP-98*, pages 117–131, Pisa, Italy, 1998.
3. C. S. Carlson, M. A. Eberle, M. J. Rieder, Q. Yi, L. Kruglyak, and D. A. Nickerson. Selecting a maximally informative set of single-nucleotide polymorphisms for association analyses using linkage disequilibrium. *Am. J. Hum. Genet.*, 74(1):106–120, 2004.
4. M. Cooper. High-order consistency in valued constraint satisfaction. *Constraints*, 10(3):283–305, 2005.
5. M. Cooper, S. de Givry, M. Sanchez, T. Schiex, and M. Zytnicki. Virtual arc consistency for weighted csp. In *Proc. of AAAI-08*, Chicago, IL, 2008.
6. M. Cooper, S. de Givry, and T. Schiex. Optimal soft arc consistency. In *Proc. of IJCAI-07*, pages 68–73, Hyderabad, India, 2007.
7. M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154:199–227, 2004.
8. A. Darwiche. Recursive Conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.
9. S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proc. of AAAI-06*, Boston, MA, 2006.
10. 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*, pages 84–89, Edinburgh, Scotland, 2005.
11. E. Freuder and M. 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.
12. P. Jégou, S. N. Ndiaye, and C. Terrioux. Dynamic management of heuristics for solving structured csps. In *Proc. of CP-07*, pages 364–378, Providence, USA, 2007.
13. P. Jégou and C. Terrioux. Decomposition and good recording. In *Proc. of ECAI-2004*, pages 196–200, Valencia, Spain, 2004.
14. M. Kitching and F. Bacchus. Exploiting decomposition in constraint optimization problems. In *Proc. of CP-08*, Sydney, Australia, 2008.
15. J. Larrosa, P. Meseguer, and M. Sanchez. Pseudo-tree search with soft constraints. In *Proc. of ECAI-02*, pages 131–135, Lyon, France, 2002.
16. J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc-consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.
17. C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Last conflict based reasoning. In *Proc. of ECAI-2006*, pages 133–137, Trento, Italy, 2006.
18. R. Marinescu and R. Dechter. And/or branch-and-bound for graphical models. In *Proc. of IJCAI-05*, pages 224–229, Edinburgh, Scotland, 2005.
19. R. Marinescu and R. Dechter. Memory intensive branch-and-bound search for graphical models. In *Proc. of AAAI-06*, Boston, MA, 2006.
20. Z. S. Qin, S. Gopalakrishnan, and G. R. Abecasis. An efficient comprehensive search algorithm for tagsnp selection using linkage disequilibrium criteria. *Bioinformatics*, 22(2):220–225, 2006.
21. M. Sanchez, S. de Givry, and T. Schiex. Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1):130–154, 2008.
22. T. Schiex. Arc consistency for soft constraints. In *Proc. of CP-2000*, pages 411–424, Singapore, 2000.
23. T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: hard and easy problems. In *Proc. of the 14^{th} IJCAI*, pages 631–637, Montréal, Canada, 1995.
24. C. Terrioux and P. Jegou. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146(1):43–75, 2003.
25. G. Verfaillie, M. Lemaître, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *Proc. of AAAI-96*, pages 181–187, Portland, OR, 1996.