RAIRO-Oper. Res. 47 (2013) 91–123
DOI: 10.1051/ro/2013030

RAIRO Operations Research
www.rairo-ro.org

EXPLOITING TREE DECOMPOSITION FOR GUIDING NEIGHBORHOODS EXPLORATION FOR VNS

Mathieu Fontaine[1], Samir Loudni[1]
and Patrice Boizumault[1]

Abstract. Tree decomposition introduced by Robertson and Seymour aims to decompose a problem into clusters constituting an acyclic graph. There are works exploiting tree decomposition for complete search methods. In this paper, we show how tree decomposition can be used to efficiently guide local search methods that use large neighborhoods like `VNS`. We propose `DGVNS` (Decomposition Guided `VNS`) which uses the graph of clusters in order to build neighborhood structures enabling better diversification and intensification. Second, we introduce tightness dependent tree decomposition which allows to take advantage of both the structure of the problem and the constraints tightness. Third, experiments performed on random instances (`GRAPH`) and real life instances (`CELAR`, `SPOT5` and `tagSNP`) show the appropriateness and the efficiency of our approach. Moreover, we study and discuss the influence of the width of the tree decomposition on our approach and the relevance of removing clusters with very few proper variables from the tree decomposition.

Keywords. Variable neighborhood search, tree decomposition, maximum cardinality search, cost functions netwok, constraint propagation.

Mathematics Subject Classification. 68T20.

Received January 28, 2013. Accepted February 11, 2013.

[1] Université de Caen Basse-Normandie, CNRS, UMR 6072 GREYC, 14032 Caen, France.
`samir.loudni@unicaen.fr`

1. Introduction

Many real–life problems, such as frequency assignment [7], or the daily management of an earth observation satellite [4], are very large and exhibit a highly structured constraints graph. Exploiting such structural properties may lead these problems to be tractable.

Tree decomposition introduced by Robertson and Seymour [37] aims to decompose a problem into subproblems (called clusters) constituting an acyclic graph. Each cluster corresponds to a subset of variables that are strongly connected. As each subproblem is significantly smaller in size than the original one, it can be solved more efficiently. The interest for exploiting structural properties of a problem has been attested in various domains: for checking satisfiability in SAT [36,36], for solving CSP (`CTE` [13]), in Bayesian or probabilistic networks (`AND/OR graph search` [29]), in relational databases [17,19], for constraint optimization (`BTD` [43], `Lc-BTD`$^+$ [11], `RDS-BTD` [38], `DB` [24]). All these proposals exploit tree decomposition for complete search methods.

For local search methods that use large neighborhoods, as *Large Neighborhood Search* (`LNS`) [41] or *Variable Neighborhood Search* (`VNS`) [30], the design of neighborhood structures is crucial, since they provide a way to intensify/diversify the search in order to explore promising regions of the search space. To the best of our knowledge, no proposal exploits tree decomposition for such methods.

Cost Functions Network (CFN) [26] is a generic framework used to model and solve constrained optimization problems which allows to deal with over–constrained problems as well as preferences between solutions. They can be solved by local, hybrid or tree search methods. They have been successfully applied to resource allocation [7], scheduling [4,5], combinatorial auctions [40], bio–informatics [39] and probabilistic reasoning [33].

In this paper, we first show how tree decomposition can be used to efficiently guide the exploration of `VNS`. We propose `DGVNS` (Decomposition Guided `VNS`) [15] which uses the graph of clusters in order to build neighborhood structures enabling better diversification and intensification. Second, we introduce *tightness dependent tree decomposition* which allows to take advantage of both the structure of the problem and the constraints tightness. Our idea is to exploit decompositions built on a subset of constraints of the original problem that may impact the quality of assignments. Experiments performed on random instances (`GRAPH`) and real life instances (`CELAR`, `SPOT5` and `tagSNP`) show the appropriateness and the efficiency of our approach. Third, we study and discuss the influence of the width of the tree decomposition on our approach and the relevance of removing clusters with very few proper variables from the tree decomposition. This can be achieved by bounding the maximum number of variables that can be shared between clusters (*i.e.* the maximum separator size $s_{\max}$) and by merging clusters with separators sizes higher than $s_{\max}$ (see Sects. 6.4 and 6.5).

Our aim is to characterize a tree decomposition in terms of topological properties (width, separators size) in order to build more relevant neighborhood structures

that will improve the diversification of DGVNS. From this study, we show that our approach is very effective on problems that decompose into *weakly connected clusters of reasonable size*. Morover, removing useless clusters (those with very few proper variables) enables to better refine the relevance of the tree decomposition and to improve the performances of DGVNS on some instances.

Experiments performed on random instances (GRAPH) and real life instances (CELAR, SPOT5 and tagSNP) show that exploiting such decompositions enables to clearly outperform VNS/LDS+CP [28] and ID-Walk [31]. To the best of our knowledge, our proposal constitutes the first attempt to use tree decomposition to efficiently guide the exploration of local search methods that use large neighborhoods like VNS.

Section 2 introduces the context. Section 3 presents how to exploit tree decomposition within VNS. Tightness Dependent Tree decomposition is introduced in Section 4. Section 5 presents the problem instances we used for our experiments. Section 6 is devoted to experimentations. Finally, we conclude and draw some perspectives.

2. Definitions and notations

2.1. Cost Functions Network

A Cost Functions Network (CFN) is a pair (X, W) where $X = \{x_1, \ldots, x_n\}$ is a set of n variables (with a maximum domain size d) and W is a set of e cost functions (see Fig. 3). Each variable $x_i \in X$ has a finite domain D_i of values that can be assigned to it. A value a in D_i is denoted (x_i, a). For a set of variables $S \subseteq X$, D^S denotes the cartesian product of the domains of the variables in S. A *complete* assignment $t{=}(a_1, ..., a_n)$ is an assignment of all variables; on the contrary, it will be called a *partial* assignment. For a given complete assignment t, $t[S]$ denotes the projection of t over S. A cost function $w_S \in W$, with scope $S \subseteq X$, is a function $w_S : D^S \mapsto [0, k_\top]$ where , $k_\top$ is a maximum integer cost (finite or not) used to represent forbidden assignments (expressing hard constraints). Costs are combined using the bounded addition defined by $\alpha \oplus \beta = \min(k_\top, \alpha + \beta)$. Figure 1 shows an example of a CFN. Tuples with non zero costs are in bold.

The central problem in CFN is to find a complete assignment t minimizing $\oplus_{w_S \in W} w_S(t[S])$. This optimization problem has an associated NP-complete decision problem and restrictions to Boolean variables and binary constraints are known to be APX–hard [32].

2.2. Tree decomposition

The constraints graph of a CFN is a graph $G{=}(X,E)$ with one vertex for each variable and one edge (u, v) for every cost function $w_S \in W$, such that $u, v \in S$.

Definition 2.1. A *tree decomposition* [37] of $G{=}(X,E)$ is a pair (C_T, T) where:

- $T = (I, A)$ is a tree with nodes set I and edges set A,

A	B	E	$w_{A,B,E}$
a	**a**	**b**	**10**
a	**b**	**b**	**10**
a	**a**	**c**	**10**
a	**b**	**c**	**100**
b	**a**	**b**	**500**
b	**b**	**b**	**200**
b	a	c	0
b	b	c	0

$t(w_{A,B,E}) = 0.75$

A	C	$w_{A,C}$
a	a	0
a	c	0
b	**a**	**1000**
b	c	0

$t(w_{A,C}) = 0.25$

C	D	$w_{C,D}$
a	**b**	**10**
a	c	0
c	**b**	**120**
c	**c**	**350**

$t(w_{C,D}) = 0.75$

B	D	$w_{B,D}$
a	**b**	**1000**
a	c	0
b	b	0
b	**c**	**10**

$t(w_{B,D}) = 0.5$

D	F	$w_{D,F}$
b	**a**	**1000**
b	**c**	**1000**
c	**a**	**1000**
c	c	0

$t(w_{D,F}) = 0.75$

FIGURE 1. Example of a CFN with six variables X = {A, B, C, D, E, F}, having as domains $D_A = D_B = \{a, b\}$, $D_C = D_F = \{a, c\}$ and $D_D = D_E = \{b, c\}$. There are one ternary cost function $w_{A,B,E}$ and four binary cost functions.

- $C_T = \{C_i \mid i \in I\}$ is a family of subsets of X (called *clusters*) such that:
  - $\cup_{i \in I} C_i = X$,
  - $\forall (u, v) \in E$, $\exists C_i \in C_T$ s.t. $u, v \in C_i$,
  - $\forall i, j, k \in I$, if j is on the path from i to k in T, then $C_i \cap C_k \subseteq C_j$.

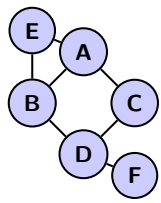
Definition 2.2. The intersection of two clusters C_i and C_j is called a *separator*, and noted $sep(C_i, C_j)$. Two clusters are *adjacent* if they share at least one variable. We denote by $s_{\max}$ the maximum size of the separators for a tree decomposition (C_T, T). Variables belonging to one, and only one, cluster are called *proper variables*.

Definition 2.3. Let C_i be a cluster of C_T. The cluster degree of C_i is the number of clusters connected (*i.e.* adjacent) to it.

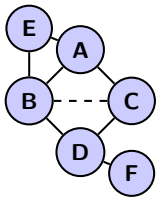
Definition 2.4. A *graph of clusters* for a tree decomposition (C_T, T) is an undirected graph $G_T = (C_T, E_T)$ that has a vertex for each cluster $C_i \in C_T$, and there is an edge $(C_i, C_j) \in E_T$ when $sep(C_i, C_j) \neq \emptyset$. The edges are labeled by the shared variables.

Definition 2.5. The *width of a tree decomposition* (C_T, T), is defined as $w^- = \max_{i \in I}(|C_i| - 1)$. The *treewidth* $tw(G)$ *of a graph* G is defined as the smallest width of all possible tree decompositions of G.

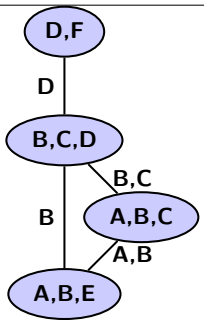
As finding an optimal tree decomposition is NP–hard [2], approximate tree decompositions using *triangulation* of a given graph are often exploited. Several effective heuristics that rely on the notion of graph *triangulation* have been proposed (see [25] for an introduction to triangulated graphs). Such heuristics provide upper bounds for the treewidth.



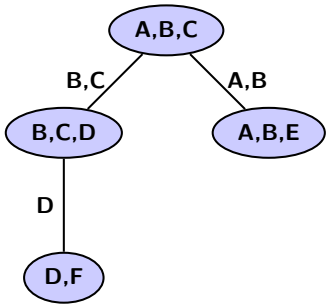
(a) Initial graph G.



(b) Example of a triangulation of G.



(c) Maximal cliques corresponding to the triangulated graph (graph of clusters).



(d) Tree decomposition of G with width 2.

FIGURE 2. Steps for computing a tree decomposition of a graph G.

2.3. COMPUTING A TREE DECOMPOSITION

In this section, we present a tree decomposition method that relies on the concept of graph triangulation.

Definition 2.6. A *graph is chordal* (or triangulated) if all cycles of length no less than four have a chord, *i.e.* an edge connecting two non-adjacent vertices of the cycle.

Theorem 2.7 [16]**.** *Let $G=(V,E)$ be an undirected graph, and let K be the set of maximal cliques of G, with K_v the set of all maximal cliques that contain vertex v of G. The following statements are equivalent:*

i) G is chordal;
ii) There exists a clique tree $T = (K,A)$ whose vertex set is the set of maximal cliques of G such that each of the induced subgraphs $T[K_v]$ is connected.

The link between triangulated graphs and tree decompositions is obvious. Indeed, if G is a chordal graph, then any clique tree of G is also a tree decomposition of G. However, the converse is not necessarily true. It follows from this that the set of maximal[2] cliques of G corresponds to the family of subsets associated with

[2]A clique is maximal iff it is not included in another clique.

a tree decomposition. Computing a tree decomposition for a graph is equivalent to finding a triangulation of this graph, *i.e.* finding a suitable set of edges to add to the graph to obtain a chordal graph [22].

Figure 2 illustrates the three steps for computing a tree decomposition for a graph G associated to the CFN of Figure 1 (see Part a). First, triangulation is performed on G by adding edge BC (see Part b). Then, maximal cliques in the chordal graph are determined in order to build the graph of clusters (see Part c). Finally, tree decomposition is achieved (see Part d).

In this paper, we have used the heuristic called *Maximum Cardinality Search* (MCS) [42]. This heuristic provides a good compromise between the width of the tree decomposition and the time required to compute it [22].

Algorithm 1: VNS/LDS+CP

```
   function VNS/LDS+CP(X, W, k_init, k_max, δ_max) ;
   begin
1      S ← genInitSol() ;
2      k ← k_init ;
3      while (k < k_max) ∧ (notTimeOut) do
4          X_un ← Hneighborhood(X, N_k, S) ;
5          A ← S\{(x_i, a) | x_i ∈ X_un} ;
6          S' ← Rebuild(A, X_un, δ_max, f(S), S) ;
7          NeighbourhoodChange(S, S', k);
8      return S ;
   end
   procedure NeighbourhoodChange (S, S', k);
   begin
9      if f(S') < f(S) then
10         S ← S' ;
11         k ← k_init ;
12     else k ← k + 1 ;
13     return ;
   end
```

2.4. VNS/LDS+CP

VNS/LDS+CP [28] is a local search method based on the variable neighbourhood decomposition search (VNDS[3]) method [20]. Neighborhoods are obtained by unfixing a part of the current solution according to a neighborhood heuristic. Then the exploration of the search space, related to the unfixed part of the current solution, is performed by a partial tree search LDS (*Limited Discrepancy Search*, [21]) with Constraint Propagation (CP).

[3]VNDS extends the basic VNS into a two-level VNS scheme based upon decomposition of the problem.

Algorithm 1 shows the pseudo–code of VNS/LDS+CP. Let X be the set of variables, and let N_k be the neighborhood structure of dimension k; N_k denotes the set of all subsets of k variables among X. VNS/LDS+CP starts from an initial solution S which is randomly generated (line 1). A subset of k variables $\mathcal{X}_{un}$ is selected in X by the neighborhood heuristic Hneighborhood (line 4). A partial assignment $\mathcal{A}$ is generated from the current solution S by unassigning the k selected variables; the $(n-k)$ non–selected variables keep their current value in S (line 5). Then, unassigned variables are rebuilt (line 6) by a partial tree search (LDS) combined with Constraint Propagation (CP). Pruning is achieved using lower bounds provided by maintaining consistencies such as AC* [27], FDAC* [26], EDAC* [10] and VAC [9]. In our case, we used EDAC.

If LDS+CP finds a solution of better quality S' in the neighborhood of S (line 9), then S' becomes the current solution and k is reset to k_{init} (lines 10–11). Otherwise, VNS/LDS+CP looks for improvements in N_{k+1} (neighborhood structure where $(k+1)$ variables of X will be unassigned (line 12)). This treatment is achieved by procedure NeighborhoodChange (S, S', k) (line 7). Indeed, the higher the dimension of the neighborhood, the larger the search space and more likely to contain better solutions than the current one. However, since the size of neighborhoods may quickly grow, finding the best neighbor may require too much effort. That is why, in order to efficiently explore parts of the search space, we use LDS+CP, a partial tree search combined with constraint propagation. The search stops when it reaches the maximal dimension size allowed or the $TimeOut$ (line 3).

2.5. Neighborhood heuristics

The neighborhood heuristic used to select variables to be unassigned is crucial, since its drives VNDS to explore regions of the search space in order to improve the current solution. However, defining efficient heuristics is difficult and requires deep specific knowledge of the problem. There exist very few generic neighborhood heuristics. We have selected ConflictVar because it is one of the simplest and popular ones: for a given dimension of neighborhood k, ConflictVar randomly selects k variables to unassign among conflicted ones[4]. Such a heuristic which is mainly based on random choices, allows to diversify the search and to quickly escape from local minima. In [34], the authors have proposed PGLNS which uses propagation to define generic neighborhood heuristics (see [34] for more details).

3. Intensification/diversification using tree decomposition

In this section, we present the first contribution of our paper: DGVNS (Decomposition Guided VNS) [15] which uses the graph of clusters in order to build neighborhood structures enabling a better diversification and a better intensification than

[4]A variable is said to be conflicted if it occurs in at least one unsatisfied constraint.

VNS/LDS+CP. For both DGVNS and VNS/LDS+CP, the rebuilding step is performed using LDS+CP, but neighborhoods are managed in a different way in order to take advantage of the graph of clusters.

Instead of the neighborhood structures N_k used by VNS/LDS+CP (see Sect. 2.4), DGVNS uses neighborhood structures $N_{k,i}$, where k is the neighborhood dimension and C_i is the cluster where the variables will be selected from. Algorithm 2 depicts the pseudo–code of DGVNS.

Algorithm 2: DGVNS

```
   function DGVNS(X, W, k_init, k_max, δ_max);
   begin
1      let G be the constraints graph of (X, W) ;
2      let (C_T, T) be a tree decomposition of G ;
       let C_T = {C_1, C_2, ..., C_p} ;
3      S ← genInitSol() ;
4      k ← k_init ;
5      i ← 1 ;
6      while (k < k_max) ∧ (notTimeOut) do
7          C_s ← CompleteCluster(C_i, k) ;
8          X_un ← Hneighborhood(C_s, N_{k,i}, S) ;
9          A ← S\{(x_i, a) | x_i ∈ X_un} ;
10         S' ← Rebuild(A, X_un, δ_max, f(S), S) ;
11         NeighbourhoodChange(S, S', k, i);
12     return S ;
   end
   procedure NeighbourhoodChange (S, S', k, i);
   begin
13     if f(S') < f(S) then
14         S ← S' ;
15         k ← k_init, i ← succ(i) ;
16     else k ← k + 1 ; i ← succ(i) ;
17     return ;
   end
```

DGVNS **favors moves on regions that are closely linked.** The concept of cluster embodies this criterion, because of its size (smaller than the original problem), and by the strong connection of the variables it contains. The set of candidate variables C_s to be unassigned are selected in a same cluster C_i. If $(k > |C_i|)$, then we complete C_s by adding the clusters C_j adjacent to C_i in order to take into account the topology of the graph of clusters. This treatment is achieved by function CompleteCluster(C_i, k) (line 7). So the neighborhood structure $N_{k,i}$ is constituted by the set of all subsets of k variables among C_s (line 8).

The aim of diversification is to sample a large number of different regions to ensure that the search space has been properly explored, and to locate the region containing the global optimum. To achieve a better diversification, we consider

successively all the C_i. This treatment is achieved by procedure `Neighbourhood-Change`(S, S', k, i) (line 11).

Let p the total number of clusters, *succ* a successor function[5], and $N_{k,i}$ the current neighborhood structure: if `LDS+CP` finds a solution of better quality S' in the neighborhood of S (line 13), then S' becomes the current solution (line 14), k is reset to k_{init} (line 15), and the next cluster is considered (line 15). Otherwise, `DGVNS` looks for improvements in $N_{(k+1),succ(i)}$ (neighborhood structure where $(k+1)$ variables of C_s will be unassigned (line 16)). The search stops when it reaches the maximal dimension size allowed or the $TimeOut$ (line 6).

First, diversification performed by moving from cluster C_i to cluster $C_{succ(i)}$ is necessary. Experiments we performed have shown that remaining in the same cluster leads to lower ameliorations: selecting a new cluster enables to improve the quality of the solution by visiting new parts of the search space. Second, when a local minimum is found in the current neighborhood, moving from k to $(k+1)$ will also provide some diversification by enlarging the neighborhood size.

The aim of intensification is to find the best solution contained within a relatively small region. This is performed by rebuilding the partial solutions using `LDS+CP` and by resetting the size of the neighborhood to k_{init} at each improvement. This will accelerate the search for complete assignments in small neighborhoods.

4. Tightness dependent tree decomposition

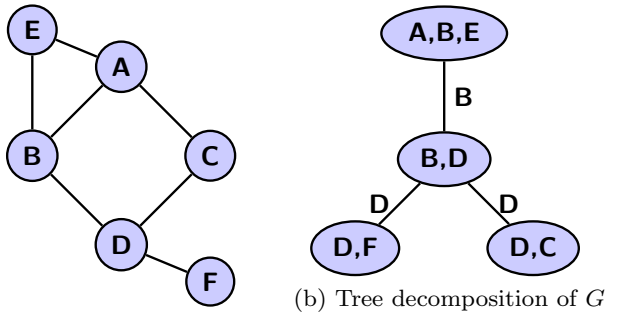
Optimization problems often involve constraints much harder to satisfy than others. Indeed, the higher the tightness of a constraint, the more difficult it is to satisfy it. So, removing constraints having a *low tightness* will ensure that the clusters of a decomposition are built taking account constraints that can have a great impact on the quality of assignments, while keeping the resulting constraints graph enough dense. Moreover, tight constraints enable a more efficient filtering on the set of variables to be selected in a cluster.

In this section, we show how to exploit constraints tightness to build more relevant tree decompositions.

Definition 4.1. Let w_S be a cost function and D^S the Cartesian product of the domains of the variables in S. The tightness $t(w_S)$ of w_S is

$$t(w_S) = \frac{|\{t \mid w_S(t) > 0, t \in D^S\}|}{|DS|}.$$

This definition of tightness does not take into account costs, since $t(w_S)$ depends only on the number of tuples with non zero cost in the table of w_S (see further works, Sect. 7).



(a) Initial graph G.

(b) Tree decomposition of G after having removed edge AC.

FIGURE 3. TD–tree decomposition of a graph G for $\lambda = 0.3$.

4.1. Computing TD–tree decomposition

The Tightness Dependent (TD) tree decomposition of a constraints graph G for a threshold value (noted λ) is performed in two steps:

1. All cost functions of tightness lower than λ are removed from G leading to subgraph G'.
2. Tree decomposition is applied to subgraph G'.

Figure 3 depicts the TD–tree decomposition of graph G for a threshold λ set to 0.3. First, the tightness $t(w_S)$ of each cost function w_S is determined (see Fig. 1). Second, $w_{A,C}$ is removed since its tightness is lower than λ. Finally, tree decomposition is applied to the remaining subgraph (see Part (b)).

Figure 4 shows the influence of the threshold λ on the Scen06 decomposition. Column 1 gives the different values of λ. Column 2 indicates the percentage of dropped constraints. Column 3 denotes the total number of clusters. Columns 4–12 respectively report, for each of the three parameters *cluster size*, *cluster degree* (*cf.* Def. 2.3) and *separator size*, their minimal, average and maximal value. Finally, the last column indicates the total number of separators. The tree decomposition performed on the initial constraints graph corresponds to $\lambda = 0$.

For $0.2 \leq \lambda \leq 0.5$, the TD–tree decomposition often yields a large number of clusters of relatively small size, while still retaining most of the important constraints of the initial problem. Moreover, clusters have (on average) a higher degree[6] (*i.e.* a higher connectivity) than those obtained with $\lambda = 0$. This highlights the interest and the importance of the TD–tree decomposition. However, for high values of λ ($\lambda \geq 0.6$), TD–tree decomposition is not pertinent since too many constraints would be removed, leading to much more isolated clusters of negligible

[5] if $i < p$ then $succ(i) = i + 1$ else $succ(p) = 1$.

[6] Removing constraints from the initial constraint graph G enables to split many clusters into new clusters of smaller sizes, thus increasing the degree of all the clusters sharing variables with these new clusters.

λ	% dropped	$\lvert C_T \rvert$	Cluster size			Cluster degree			Separator size			
			min.	avg.	max.	min.	avg.	max.	min.	avg.	max.	nb
0	0	55	2	4.9	12	1	9.6	25	1	1.77	8	266
0.1	1	59	2	5.1	11	1	10.27	26	1	1.9	10	303
0.2	3	61	2	4.9	11	1	10.06	26	1	1.89	10	307
0.3	5	61	2	4.91	11	1	10.06	26	1	1.8	10	307
0.4	8	60	1	4.75	11	0	7.8	21	1	1.9	10	234
0.5	14	56	1	4.3	11	0	5.57	14	1	1.75	9	156
0.6	28	65	1	3.93	10	0	5.29	14	1	1.93	9	172
0.7	53	74	1	2.98	10	0	4.10	10	1	1.61	6	152
0.8	54	72	1	2.97	10	0	3.63	9	1	1.64	6	131
0.9	76	80	1	1.85	9	0	1.6	7	1	1.18	5	64
1	100	100	1	1	1	0	0	0	0	0	0	0

FIGURE 4. TD–tree decompositions of Scen06.

size. Results obtained for other CELAR instances are not reported in this paper, since they are similar.

Moreover, for SPOT5 instances, TD–tree decomposition is pertinent for $0.1 \leq \lambda \leq 0.3$ (see Sect. 6.3 for more details). Results for GRAPH instances are given Section 6.4. Finally, TD–tree decomposition has not been applied on the tagSNP problem, since all the considered instances contain cost functions with the same tightness.

4.2. Setting the value of parameter λ

Determining the best value of parameter λ is problem–dependent. Our approach determines empirically the best setting of λ by evaluating several possible values ranging from 0.1 to 0.6. Values greater than 0.6 lead to much more disconnected clusters. However, despite its empirical character, our approach remains applicable and does not require extensive knowledge as well as a lot of experiments (for effective tuning), since parameter λ conveys a certain semantic for which its setting can be readily understandable.

5. Benchmark problems

Experiments have been performed on instances of four different problems.

RLFAP instances: The CELAR (Centre d'Electronique de l'Armement) has made available a set of instances for the Radio Link Frequency Assignment Problem (RLFAP) [7]. They consist in assigning a limited number of frequencies to a set of radio links defined between pairs of sites, in order to minimize interferences due to the re-use of frequencies. We report experiments on the most difficult instances: Scen06, Scen07 and Scen08.

GRAPH instances: The GRAPH generator (Generating Radio link frequency Assignment Problems Heuristically) has been developed by the CALMA project [44] in order to provide structured random instances close to RLFAP ones.

SPOT5 instances: The daily management of an earth observation satellite such as SPOT5 consists in selecting a subset of candidate photographs to fit physical limitations and maximize the importance of the selected photographs [4]. We report experiments on seven instances from those without hard capacity constraint.

tagSNP instances: A Single Nucleotide Polymorphism (SNP) is a DNA sequence variation occurring when a single nucleotide - A, T, C or G - in the genome differs between members of a biological species or paired chromosomes in an individual [8]. SNPs act as biological markers that may help predict risk of developing particular diseases. The tagSNP problem consists in selecting a small subset of SNPs, called tagSNPs, that captures most of the genetic information.

A correlation measure r^2 between any pair of SNPs has been introduced in [14]. A tagSNP p_i is said to be representative of another SNP p_j if p_i and p_j are considered as enough correlated (*i.e.* $r^2(p_i, p_j) \geq r_0$, where r_0 is a minimum threshold). The tagSNP problem consists in selecting a minimum number of SNPs such that all SNPs are covered. Other criteria [35,38] can also be considered: (i) maximizing the weighted coverage sum of unselected SNPs and (ii) maximizing the dispersion between selected SNPs (*i.e.* tagSNPs).

This problem is modeled as a binary CFN (see Sect. 2.1). Two variables i_s and i_r are associated to each SNP p_i: i_s is a boolean variable indicating whether p_i is a tagSNP; i_r is a variable representing the tagSNP covering p_i (the finite domain of r_i is the set of neighbors of p_i together with p_i itself). For each pair of SNPs (p_i, p_j) s.t. $r^2(p_i, p_j) \geq r_0$, the following (hard) constraints are enforced: $i_s \Rightarrow (i_r = p_i)$ and $(i_r = p_j) \Rightarrow j_s$. Such constraints are encoded as binary cost functions (with 0 or $k_\top$ costs). Preferences (i) and (ii) are respectively captured by unary and binary cost functions (see [38] for more details).

We have selected the tagSNP problem for two main reasons. First, the tagSNP problem is known to be very hard to solve, due to its close relation to the *set covering problem* (NP–hard). Second, the instances are reasonably large: up to $n = 1550$ variables with max domain size d ranging from 30 to 266, and up to $e = 250\,000$ cost functions. We report experiments on thirteen challenging instances derived from human chromosome–1–data[7] with $r_0 = 0.5$. Eight instances are medium-sized, while the five other instances are large ones.

6. Experiments

6.1. Experimental protocol

Each instance has been solved by each method, with a discrepancy of 3 for LDS, which is the best value found on RLFAP instances (see [28]). $k_{\min}$ and $k_{\max}$ have been respectively set to 4 and n (the total number of variables), and $TimeOut$ fixed to 3600 s, except for tagSNP instances, where $TimeOut = 2$ h (resp. 4 h) for medium-sized (resp. large) instances. A set of 50 runs per instance has been

[7] http://www.costfunction.org/benchmark

Instance	Method	Succ.	Time	Avg
Scen06	DGVNS	50/50	112	3,389
$n = 100$, $d = 44$	TDGVNS-0.2	**50/50**	**58**	**3,389**
$e = 1,222$	VNS/LDS+CP	15/50	83	3,399
$S^* = 3,389$	ID-Walk	NA	840	3,447 (3,389)
Scen07	DGVNS	40/50	317	345,614
$n = 200$, $d = 44$	TDGVNS-0.4	**49/50**	**221**	**343,600**
$e = 2,665$	VNS/LDS+CP	1/50	461	355,982
$S^* = 343,592$	ID-Walk	NA	360	373,334 (343,998)
Scen08	DGVNS	3/50	1,811	275
$n = 458$, $d = 44$,	TDGNVS-0.5	**9/50**	**442**	**272**
$e = 5,286$	VNS/LDS+CP	0/50	-	394 (357)
$S^* = 262$	ID-Walk	NA	3,000	291 (267)

FIGURE 5. Comparing the four methods on RLFAP instances.

performed on an AMD opteron with 2.1 GHz CPU and 256 GB of RAM. All search strategies have been implemented in C++ using the library toulbar2[8].

For each instance and each method, we report the number of successful runs to reach the optimum, "succ. runs/total runs", the average CPU time (in seconds) for the successful runs, the average cost over the 50 runs and the best cost (between brackets) for unsuccessful runs. We also give the *mean performance profiles* of the evolution of the solution quality over time (see Figs. 15, 16, 17, 18 and 19 in Appendix B)

First, we compare DGVNS with VNS/LDS+CP and ID-Walk [31], one of the most performing local search methods on RLFAP instances (see Sect. 6.2). Second, we compare DGVNS with its TD–tree decomposition version (namely TDGVNS-λ); comparisons are performed with different values of λ (see Sect. 6.3). Third, we study and discuss the influence of the width of the tree decomposition and the relevance of removing clusters with very few proper variables from the tree decomposition. This can be achieved by bounding the maximum number of variables that can be shared between clusters (*i.e.* the maximum separator size $s_{\max}$) and by merging clusters with separators sizes higher than $s_{\max}$ (see Sects. 6.4 and 6.5). Our aim is to characterize a tree decomposition in terms of topological properties (width, separators size) in order to build more relevant neighborhood structures that will improve the diversification of DGVNS.

Finally, note that comparing CPU times for our approach with those for complete methods that exploit tree decompositions would be rather difficult. In fact, all reported CPU times include both finding an optimal solution and proving its optimality. These two tasks take generally about a few days [38].

[8] http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP

Instance	Method	Succ.	Time	Avg
#408	DGVNS	49/50	117	6,228
$n = 200,\ d = 4$	TDGVNS-0.1	**50/50**	**72**	**6,228**
$e = 2,232$	VNS/LDS+CP	26/50	149	6,228
$S^* = 6,228$	ID-Walk	**50/50**	**3**	**6,228**
#412	DGVNS	36/50	84	32,381
$n = 300,\ d = 4$	TDGVNS-0.1	**38/50**	**71**	**32,381**
$e = 4,348$	VNS/LDS+CP	32/50	130	32,381
$S^* = 32,381$	ID-Walk	10/50	2102	3,238
#414	DGVNS	38/50	554	38,478
$n = 364,\ d = 4$	SDGNVS-0.1	**42/50**	**430**	**38,478**
$e = 10,108$	VNS/LDS+CP	12/50	434	38,481
$S^* = 38,478$	ID-Walk	0/50	-	38,481
#505	DGVNS	**50/50**	**63**	**21,253**
$n = 240,\ d = 4$	TDGVNS-0.1	48/50	92	21,253
$e = 2,242$	VNS/LDS+CP	41/50	143	21,253
$S^* = 21,253$	ID-Walk	**50/50**	**358**	**21,253**
#507	DGVNS	33/50	71	27,390
$n = 311,\ d = 4$	TDGVNS-0.1	**45/50**	**62**	**27,390**
$e = 5,732$	VNS/LDS+CP	11/50	232	27,391
$S^* = 27,390$	ID-Walk	7/50	1,862	27,391
#509	DGVNS	**40/50**	**265**	**36,446**
$n = 348,\ d = 4$	TDGVNS-0.2	**39/50**	**313**	**36,446**
$e = 8,624$	VNS/LDS+CP	12/50	598	36,448
$S^* = 36,446$	ID-Walk	0/50	-	36,450

FIGURE 6. Comparing the four methods on SPOT5 instances.

6.2. Contribution of the tree decomposition

6.2.1. RLFAP instances

First, DGVNS clearly outperforms VNS/LDS+CP on RLFAP instances (see Fig. 5). DGVNS reaches the optimum with success rates of 100%, 80% and 6% on Scen06, Scen07 and Scen08 respectively. VNS/LDS+CP gets successful runs only very few times on the first two instances, and is not able to find the optimum for Scen08: the best solution found has a cost 357.

Second, DGVNS clearly outperforms ID-Walk[9], particularly on the two challenging instances Scen07 and Scen08 (see Fig. 5). For Scen07 (resp. Scen08), DGVNS obtains solutions with a *mean deviation* (percentage deviation from the optimum) of 0.55% (resp. 5%) above the optimum, while ID-Walk only finds solutions whose average costs are respectively 8% and 11% above the optimum.

[9]Results for RLFAP instances are taken from [31]. The number of successful runs and the computing time are not available (NA in Fig. 5), Thus, we only report the time per trial, the average cost over 10 trials and the best cost found. For other instances (SPOT5, GRAPH and tagSNP) results were obtained using ID-Walk in the library INCOP [1], with the same experimental protocol as described above.

6.2.2. SPOT5 instances

This trend is confirmed by SPOT5 instances (see Fig. 6), where DGVNS outperforms VNS/LDS+CP, both in terms of success rates (with a gain of 40% on average) and CPU times. Indeed, DGVNS reaches the optimum for each run on two instances (#408 and #505). For the other instances (except for instance #507), the success rate is at least 70%. VNS/LDS+CP gets an average success rate of 67% on instances #408 and #505, while for the other instances (except on #412), the success rate is at most 24%.

Once again, DGVNS clearly dominates ID-Walk (except for instance #408) (see Fig. 6), particularly on the two large instances #414 and #509 where ID-Walk does not reach the optimum. For these two instances, the best solution found has a cost 38 479 and 36 447 respectively.

6.2.3. tagSNP instances

Figure 7 compares DGVNS with VNS/LDS+CP on tagSNP instances. ID-Walk exhibited poor performances that are not reported here. On *medium-sized instances*, DGVNS clearly outperforms VNS/LDS+CP. DGVNS reaches the optimum for each of the 50 runs (*i.e.* success rates of 100%). VNS/LDS+CP gets the same success rates on three instances (#6835, #15 757 and #16 706), but DGVNS is on average 3.6 times faster. On the two instances #3792 and #8956, VNS/LDS+CP gets successful runs only very few times (*i.e.* success rates of 30% and 24% respectively). For comparison, DGVNS improves very significantly this success rate about 70% (from 30% to 100%) for instance #3792 and 76% (from 24% to 100%) for instance #8956. For the other instances, VNS/LDS+CP remains less competitive both in terms of success rates and CPU times.

On *large instances*, DGVNS clearly dominates VNS/LDS+CP (see Fig. 7), particularly on the three instances #10 442, #14 226 and #17 034, where VNS/LDS+CP is not able to find the optimum. For these three instances, the best costs found (between brackets) are about 4% above the optimum. On the two remaining instances, even though no method reaches the optimum, DGVNS performs better than VNS/LDS+CP on instance #9150 and worse on #6858. For instance #9150, the deviation (resp. mean deviation) above the optimum of the best (resp. average) cost found decreases from 19% to 0.0003% (resp. from 22% to 3%). For instance #6858, VNS/LDS+CP obtains the lowest cost, and the deviation above the optimum of the best solution found decreases from 33% to 16%. Let us note that the two methods are quite similar on average.

6.2.4. Performance profiles

Figures 15, 16, Figures 17 and 18 (see Appendix B) compare performance profiles of DGVNS and VNS/LDS+CP.

For RLFAP and SPOT5 instances, DGVNS clearly outperforms VNS/LDS+CP (except for instance #505). From an anytime point of view, two important observations can be made. First, the curve for DGVNS shows a significant steep initial slope. Second,

Instance	Method	Succ.	Time	Avg
#3792, $n = 528$, $d = 59$, $e = 12,084$	DGVNS	**50/50**	**954**	**6,359,805**
$S^* = 6,359,805$	VNS/LDS+CP	15/50	2,806	6,359,856
#4449, $n = 464$, $d = 64$, $e = 12,540$	DGVNS	**50/50**	**665**	**5,094,256**
$S^* = 5,094,256$	VNS/LDS+CP	48/50	2,616	5,094256
#6835, $n = 496$, $d = 90$, $e = 18,003$	DGVNS	**50/50**	**2,409**	**4,571,108**
$S^* = 4,571,108$	VNS/LDS+CP	50/50	7,095	4,571,108
#8956, $n = 486$, $d = 106$, $e = 20,832$	DGVNS	**50/50**	**4,911**	**6,660,308**
$S^* = 6,660,308$	VNS/LDS+CP	12/50	8,665	6,660327
#9319, $n = 562$, $d = 58$, $e = 14,811$	DGVNS	**50/50**	**788**	**6,477,229**
$S^* = 6,477,229$	VNS/LDS+CP	47/50	2,434	6,477,229
#15757, $n = 342$, $d = 47$, $e = 5,091$	DGVNS	**50/50**	**60**	**2,278,611**
$S^* = 2,278,611$	VNS/LDS+CP	50/50	229	2,278,611
#16421, $n = 404$, $d = 75$, $e = 12,138$	DGVNS	50/50	**2,673**	**3,436,849**
$S^* = 3,436,849$	VNS/LDS+CP	37/50	3,146	3,436,924
#16706, $n = 438$, $d = 30$, $e = 6,321$	DGVNS	**50/50**	**153**	**2,632,310**
$S^* = 2,632,310$	VNS/LDS+CP	50/50	629	2,632,310
#6858, $n = 992$, $d = 260$, $e = 103,056$	DGVNS	0/50	-	26,882,588 (26,879,268)
$S^* = 20,162,249$	VNS/LDS+CP	0/50	-	**26,815,733 (23,524,452)**
#9150, $n = 13,52$, $d = 121$, $e = 44,217$	DGVNS	0/50	-	**44,754,916 (43,302,028)**
$S^* = 43,301,891$	VNS/LDS+CP	0/50	-	52,989,981 (51,677,673)
#10442, $n = 908$, $d = 76$, $e = 28,554$	DGVNS	**50/50**	**4,552**	**21,591,913**
$S^* = 21,591,913$	VNS/LDS+CP	0/50	-	22,778,811 (22,490,938)
#14226, $n = 1,058$, $d = 95$, $e = 36,801$	DGVNS	**46/50**	**7,606**	**25,688,751**
$S^* = 25,665,437$	VNS/LDS+CP	0/50	-	28,299,904 (26,830,579)
#17034, $n = 1142$, $d = 123$, $e = 47,967$	DGVNS	**41/50**	**8,900**	**38,563,232**
$S^* = 38,318,224$	VNS/LDS+CP	0/50	-	41,352,709 (39,850,974)

FIGURE 7. Comparing DGVNS and VNS/LDS+CP on tagSNP instances.

the decelerating of the curve for VNS/LDS+CP is very important as compared to DGVNS.

For tagSNP instances (Figs. 17 and 18) the same observations can be made, particularly on large instances, where the decelerating of the curve for VNS/LDS+CP is greatly amplified as compared to DGVNS. Moreover, the improvement speed of the quality of solutions provided by DGVNS, with respect to the initial cost, is very important. For comparison, VNS/LDS+CP needs much more CPU time and much more moves to reach the same improvement in quality. This confirms the importance of exploiting tree decomposition for guiding VNS.

6.2.5. Synthesis

These experiments clearly demonstrate the efficiency of our approach compared with both VNS/LDS+CP and ID-Walk on structured problems like RLFAP and SPOT5. For tagSNP instances, DGVNS clearly outperforms VNS/LDS+CP, particularly on large instances, where VNS/LDS+CP is not able to reach the optimum. Results for GRAPH are given in Figure 10, Section 6.4.

6.3. Contribution of tightness dependent tree decomposition

Figures 5 and 6 enable to compare TDGVNS (DGVNS applied to TD–tree decomposition) with DGVNS. Only results for the best setting for λ are reported. We have not applied TDGVNS on the tagSNP problem, since all the instances considered contain cost functions with the same tightness. Results for GRAPH are given in Figure 11, Section 6.4.

6.3.1. Comparing TDGVNS with DGVNS

On RLFAP instances (see Fig. 5), the impact of the TD–tree decomposition is very significant, particularly on Scen07 and Scen08, for which very large improvements are gained by TDGVNS. For Scen07, TDGVNS improves significantly the success rate about 18% (from 80% to 98%) as well as the mean deviation above the optimum (from 0.55% to 0.002%). For Scen08, the success rate is improved about 12% (from 6% to 18%), the mean deviation decreases from 5% to 3.80% and TDGVNS is 4 times faster than DGVNS. For Scen06, TDGVNS is 2 times faster than DGVNS.

The same trend is confirmed by SPOT5 instances (see Fig. 6), where TDGVNS is very effective. On three large instances (#412, #414 and #507), TDGVNS improves on average the success rate as well as the average CPU time about 12% and 25% respectively. For the other instances, both methods perform similarly, however TDGVNS is at least 1.5 faster than DGVNS. For performance profiles (Figs. 15 and 16), similar behaviors can be observed (except for Scen06 and instance #505), that is TDGVNS is always better than DGVNS.

6.3.2. Comparing various values of λ for RLFAP and SPOT5 instances

Figure 8 gives the impact of λ on the TD–tree decomposition. The best value for this parameter depends on the instance. Experiments show that, for RLFAP instances, values of λ ranging from 0.2 to 0.5 are the most appropriate. For SPOT5 instances, best results are obtained with λ ranging from 0.1 to 0.3. In fact, these instances seem highly constrained since only a small number of candidate photographs are selected in an optimal solution. $\lambda = 0.3$ (resp. 0.1) provides a more homogeneous behavior among the RLFAP (resp. SPOT5) instances. Finally, for RLFAP (resp. SPOT5) instances, values of λ greater than 0.6 (resp. 0.4) did not pay off since too many constraints would be removed. So, results are not reported here.

6.3.3. Synthesis

These experiments clearly demonstrate the relevance of TD–tree decomposition to design appropriate neighborhood structures. Moreover, for RLFAP instances, values of λ ranging from 0.2 to 0.5 are the most appropriate, while for SPOT5 instances, best results are obtained with λ ranging from 0.1 to 0.3. Results for GRAPH are given in Figure 11, Section 6.4.

Instance	λ	Succ.	Time	Avg
Scen06	0.2	**50/50**	**58**	**3,389**
	0.3	**50/50**	**61**	**3,389**
	0.4	50/50	110	3,389
	0.5	49/50	122	3,389
Scen07	0.2	45/50	271	344,603
	0.3	**47/50**	**229**	**344,198**
	0.4	**49/50**	**221**	**343,600**
	0.5	45/50	244	344,603
Scen08	0.2	**7/50**	**327**	273
	0.3	5/50	323	273
	0.4	6/50	344	274
	0.5	**9/50**	**442**	**272**
#408	0.1	**50/50**	**72**	**6,228**
	0.2	**44/50**	**90**	**6,228**
	0.3	10/50	105	6,229
	0.4	11/50	129	6,229

Instance	λ	Succ.	Time	Avg
#412	0.1	**38/50**	**71**	**32,381**
	0.2	32/50	135	32,384
	0.3	8/50	1337	32,384
	0.4	2/50	1290	32,383
#414	0.1	**42/50**	**430**	**38,478**
	0.2	**38/50**	**471**	**38,478**
	0.3	4/50	704	38,480
	0.4	1/50	747	38,480
#505	0.1	**48/50**	**92**	**21,253**
	0.2	**18/50**	**148**	**21,253**
	0.3	4/50	319	21,256
	0.4	1/50	436	21,256
#507	0.1	**45/50**	**62**	**27,390**
	0.2	**28/50**	**134**	**27,390**
	0.3	3/50	418	27,391
	0.4	3/50	203	27,391
#509	0.1	**36/50**	**286**	**36,446**
	0.2	**39/50**	**313**	**36,446**
	0.3	2/50	583	36,448
	0.4	4/50	848	36,449

FIGURE 8. Influence of λ on the TD–tree decomposition (`RLFAP` and `SPOT5` instances).

6.4. Impact of the width of tree decompositions

As stated in the introduction, real–life problems frequently exhibit particular structures like weakly interconnected clusters (*e.g.* see Appendix A, Fig. 14). Structures are related to some topological properties of the tree decomposition and can be characterized by three measures: width, separators size and clusters degree. The width of a tree decomposition gives a good indication on the size of sub–problems, while the last two measures provide information about the connectivity between clusters. Indeed, the smaller their values are, the less the clusters are interconnected. As pointed out in [12], finding an optimal tree decomposition with regard to these three measures is an open issue. We have chosen `MCS` [42] because this heuristic appears to be a good compromise between the quality of the tree decomposition and the CPU time required to compute it [22].

6.4.1. Defining two criteria for analyzing a tree decomposition

For a given instance, let w^- be the width of a tree decomposition obtained using `MCS`. To study the impact of the width of a tree decomposition, we have defined two criteria:

(i) the decomposability of a problem ($\frac{w^-}{n}$), estimated by the ratio between the width of a tree decomposition and the number of variables. The lower the ratio, the more likely the problem decomposes into clusters of small size.

Instance	n	w^-	s_{max}	mc	$\frac{w^-}{n}$	$\frac{w^-}{mc}$	$\mid C_T \mid$
Scen06	100	11	8	10	0.11	1.1	55
Scen07	200	22	22	10	0.11	2.2	110
Scen08	458	26	26	10	0.05	2.6	259
Graph05	100	32	32	8	0.32	4	58
Graph06	200	62	61	8	0.31	7.75	123
Graph11	340	117	116	7	0.34	19.5	191
Graph13	458	155	153	6	0.34	25.8	262
#3792	528	89	74	40	0.168	2.225	207
#4449	464	91	87	52	0.196	1.75	157
#6835	496	89	88	59	0.179	1,5	161
#8956	486	141	124	83	0.290	1,6	179
#9319	562	72	70	46	0.128	1,56	253
#15757	342	49	43	29	0.143	1,69	126
#16421	404	76	65	58	0.188	1,31	147
#16706	438	31	29	26	0.070	1,19	186
#10442	908	116	95	68	0.127	1,7	320
#14226	1058	115	103	81	0.108	1,42	374
#17034	1,142	158	151	84	0.138	1,88	388
#6858	992	302	302	131	0.304	2,3	275
#9150	1,352	132	127	86	0.097	1,53	500

FIGURE 9. Smallest width, maximum separator size, maximum clique size and ratios for RLFAP, GRAPH and tagSNP instances.

(ii) the precision of a decomposition ($\frac{w^-}{mc}$), defined as the ratio between the width of a tree decomposition and the maximum clique size (mc)[10] for the initial graph.

The maximum clique size (mc) minus one is a lower bound for the treewidth. It gives an effective idea about the quality of the obtained decomposition. Indeed, the closer the ratio ($\frac{w^-}{mc}$) to one, the better the decomposition identifies strong informative structures of the CFN, enabling to build more relevant neighborhoods.

Figure 9 reports, for each instance among RLFAP ones, GRAPH ones, and tagSNP ones, the size (n, number of variables), the width (w^-) provided by MCS, the maximum separator size ($s_{\max}$), the maximum clique size (mc) for the initial graph, the two ratios ($\frac{w^-}{n}$) and ($\frac{w^-}{mc}$), and the number of clusters. Since results for SPOT5 instances are similar to those for RLFAP ones, they are not reported here.

6.4.2. Analysing the instances

(i) For RLFAP instances (see Fig. 9), the width (w^-) increases with the problem size (n), while the ratio ($\frac{w^-}{n}$) decreases. This means that for large instances, clusters are more sparse. Moreover, these problems exhibit a (very) low value for $\frac{w^-}{n}$ (it varies from 5% to 11%). This is not the case for GRAPH instances (particularly for the two instances Graph11 and Graph13), where values of w^-, as well as their

[10]It can be computed within reasonable time [3].

ratio, are large compared to those for RLFAP instances. Figure 9 also shows (very) large gaps (>100) between lower and upper bounds for the largest GRAPH instances (*i.e.* very high values for $\frac{w^-}{mc}$), while for RLFAP instances this gap remains small. This remarkable difference between large RLFAP and GRAPH instances is inherent to the origin of the instances: the RLFAP instances are taken from real–life whereas the GRAPH instances are random ones.

(ii) For tagSNP instances (see Fig. 9), we observe a very low value for $\frac{w^-}{n}$ (≤ 0.2) on medium-sized instances, except for the instance #8956. Moreover, the gap between the maximum clique size (mc) and the width of the tree decomposition (w^-) is relatively small. This probably explains the good performances of DGVNS on these problems (see Fig. 7), which appear well suited for exploiting of the decomposition. For large instances, the same remarks can be drawn, except for the instance #6858, where VNS/LDS+CP obtains a lower cost than DGVNS. The weak performances of DGVNS on instance #6858 are certainly due to the high value for $\frac{w^-}{n}$ and the large gap between mc and w^-. It is difficult for MCS to identify a pertinent structure that decomposes into more sparse clusters of small size. For the second particular instance #9150 (see Fig. 7), even though the value of $\frac{w^-}{n}$ as well as the gap between mc and w^- are relatively small, DGVNS is not able to reach the optimum. For this instance, the constraints graph highlights a large sub–part of the graph which is dense. Such a structure, which tends to be a random one, does not benefit to DGVNS as a large number of clusters greatly overlap.

(iii) For GRAPH instances. As shown in Sections 6.2 and 6.3, DGVNS and TDGVNS clearly outperform both VNS/LDS+CP and ID-Walk on RLFAP and SPOT5 instances (*i.e.* instances having small values for $\frac{w^-}{n}$). This trend is also confirmed by Graph05 and Graph06, both in terms of success rates and solution quality (see Fig. 10) as well as in terms of performance profiles (see Appendix B, Fig. 19).

But, for Graph11 and Graph13, VNS/LDS+CP outperforms both DGVNS and TDGVNS (see Fig. 10). This is certainly due to the fact that clusters are very large and strongly connected. Thus, the impact of our diversification is weaker since most clusters have few proper variables and tend to be quite similar. These results show the limits of MCS to reveal pertinent structures (*i.e.* weakly connected clusters of reasonable size). However, if we compare performance profiles (see Fig. 19), we can observe that TDGVNS behaves well in terms of the evolution of solution quality (on average) over time compared to VNS/LDS+CP. So, further works have to be performed in order to draw more precise and definitive conclusions.

Finally, Figure 11 reports the impact of λ on the TD–tree decomposition for GRAPH instances. The best results are obtained with large values of λ (0.7). Indeed, as the clusters are relatively dense, a large number of constraints must be removed to obtain pertinent and appropriate decompositions.

6.4.3. Synthesis

The width of the tree decomposition has a great impact on the performances of DGVNS. Our experimental analysis demonstrates clearly the effectiveness of our

Instance	Method	Succ.	Time	Avg
Graph05	DGVNS	50/50	10	221
$n = 100$	TDGVNS-0.7	**50/50**	**8**	**221**
$e = 1,034$	VNS/LDS+CP	50/50	17	221
$s* = 221$	ID-Walk	0/50	-	10,584 (2,391)
Graph06	DGVNS	50/50	367	4,123
$n = 200$	TDGVNS-0.7	50/50	261	4,123
$e = 1,970$	VNS/LDS+CP	**50/50**	**218**	4,123
$s* = 4,123$	ID-Walk	0/50	-	18,949 (13,001)
Graph11	DGVNS	8/50	3,046	4,234
$n = 340$	TDGVNS-0.5	12/50	2,818	3,268
$e = 3,417$	VNS/LDS+CP	**44/50**	2,403	3,090
$s* = 3,080$	ID-Walk	0/50	-	41,604 (27,894)
Graph13	DGVNS	0/50	-	22,489 (18,639)
$n = 458$	TDGVNS-0.7	0/50	-	11,449 (10,268)
$e = 4,915$	VNS/LDS+CP	**3/50**	3,477	14,522
$s* = 10110$	ID-Walk	0/50	-	58,590 (47,201)

FIGURE 10. Comparing the four methods on GRAPH instances.

Instance	λ	Succ.	Time	Avg
Graph05	0.5	50 / 50	12	221
	0.6	50 / 50	13	221
	0.7	**50 / 50**	**8**	**221**
Graph06	0.5	50 / 50	339	4,123
	0.6	49 / 50	259	4,123
	0.7	**50 / 50**	**261**	**4,123**
Graph11	0.5	**12 / 50**	**2,818**	**3,643**
	0.6	**7 / 50**	**2,446**	**3,123**
	0.7	5 / 50	1417	3,223
Graph13	0.5	0 / 50	-	24,637 (18,833)
	0.6	0 / 50	-	19,340 (14,318)
	0.7	**0 / 50**	**-**	**11,449 (10,268)**

FIGURE 11. Influence of λ on the TD–tree decomposition (GRAPH instances).

approach on problems characterized by a low value for $\frac{w^-}{n}$ and by a small gap between mc and w^-, *i.e.* problems that decompose into clusters of reasonable size. For these problems, MCS provides relevant decompositions that reflect the original structure of the CFN, enabling a better diversification.

6.5. IMPACT OF REMOVING CLUSTERS WITH FEW PROPER VARIABLES

Experiments we performed have shown that most of the decompositions we obtained by MCS contain a large number of clusters with (very) few proper variables. To increase the proportion of proper variables in the clusters of a tree

decomposition, we propose to limit the maximum size of separator between clusters (*i.e.*, the maximum number of variables that can be shared). This is achieved by merging clusters with separator sizes higher than the maximum size allowed. Our main motivation is to refine tree decompositions obtained by MCS, by removing useless clusters (those with very few proper variables) and by reducing the number of clusters that overlap heavily.

To generate new alternative decompositions from the MCS derived tree decomposition, we followed the same scheme as proposed in [23], by bounding the maximum separator size $s_{\max}$. First, we computed a tree decomposition[11] using the same MCS order as in [38], by setting $s_{\max}$ to $+\infty$ (the size of separators is not restricted). Then, starting from the leaves of the MCS tree decomposition, we merge a cluster with its parent if the separator size is strictly greater than $s_{\max}$. For our experiments, we considered different values for $s_{\max}$: $s_{\max} \in \{4, 8, 12, 16, 24, 32\}$.

6.5.1. Comparing the performance of DGVNS *for different MCS derived tree decompositions with bounded separators sizes*

Figures 12 and 13 compare the performance of DGVNS for different tree decompositions obtained on tagSNP instances corresponding to different values of $s_{\max}$.

On six instances (see Fig. 13), merging clusters of the MCS derived tree decomposition (*w.r.t.* the value of $s_{\max}$) decreases significantly the performances of DGVNS in terms of success rates as well as in CPU times, compared to $s_{\max}=+\infty$. Moreover, the trend is greatly amplified for small values of $s_{\max}$ (≤ 12). For instance #6835, the average CPU time is increased by 37% for $s_{\max} = 32$, and by 56% for $s_{\max} = 24$. For medium values of $s_{\max}$ (12 and 16), average CPU time is multiplied by more than 3, while for small values of $s_{\max}$ the success rate is decreased by 8%. Similar behaviors can be observed on other instances, particularly on instances #14 226 and #17 034, where DGVNS performs bad for small values of $s_{\max}$.

The weak results of DGVNS on these instances are probably due to the fact that, by merging clusters according to the value of $s_{\max}$, the width of the decomposition tends to increase, degrading its quality. Indeed, the smaller the value of $s_{\max}$ is, the more likely the size of the largest cluster and the width of the tree decomposition (*i.e.* w^-) are high. Thus, the impact of diversification of DGVNS is very limited as the neighborhoods are too large and the number of explored regions (number of clusters) is too small. For instance #6835, with $s_{\max} = 4$, w^- increases from 108 to 468 and the number of clusters decreases from 56 to 3. For instance #6858, this trend is greatly amplified: w^- increases from 350 to 988 and the number of clusters decreases from 105 to 2.

On the contrary, for five instances (see Fig. 12), merging clusters of the MCS derived tree decomposition enables to improve the performances of DGVNS. For most of these instances (except #16706), $s_{\max} = 32$ gives the best results. On instance #16421, the average CPU time is divided by 2 compared to $s_{\max} = +\infty$, while

[11] This decomposition is different from the one used for experiments in Section 6.2.3, since it uses a different MCS order.

Inst.	s_{max}	Succ.	Time	Avg	$\|C_T\|$	Cluster size avg.	Cluster size max.
#3792	4	50 / 50	2,381	6,359,805	11	49.36	283
	8	50 / 50	1,498	6,359,805	20	29.8	255
	12	50 / 50	1,391	6,359,805	29	23.68	157
	16	50 / 50	959	6,359,805	37	21.64	139
	24	50 / 50	1,028	6,359,805	42	21.42	133
	32	**50 / 50**	**832**	**6,359,805**	**49**	**22.18**	**121**
	$+\infty$	50 / 50	954	6,359,805	70	30.37	95
#4449	4	50 / 50	2,399	5,094,256	7	67.85	305
	8	50 / 50	1,396	5,094,256	16	33	195
	12	50 / 50	1,216	5,094,256	17	31.70	195
	16	50 / 50	996	5,094,256	19	29.73	159
	24	50 / 50	663	5,094,256	26	27.07	121
	32	**50 / 50**	**587**	**5,094,256**	**49**	**22.18**	**121**
	$+\infty$	50 / 50	665	5,094,256	56	36.71	101
#9315	4	50 / 50	1,027	6,477,229	14	41.85	197
	8	50 / 50	853	6,477,229	22	28.72	173
	12	50 / 50	897	6,477,229	30	23.93	171
	16	50 / 50	698	6,477,229	35	22.48	115
	24	50 / 50	656	6,477,229	40	22	115
	32	**50 / 50**	**555**	**6,477,229**	**44**	**22.5**	**115**
	$+\infty$	50 / 50	788	6,477,229	62	34.41	89
#16421	4	50 / 50	2,170	3,436,849	5	82.2	207
	8	50 / 50	2,273	3,436,849	7	60.14	201
	12	50 / 50	2,209	3,436,849	11	42.27	201
	16	50 / 50	2,369	3,436,849	14	36.28	133
	24	50 / 50	1,253	3,436,849	17	33.47	133
	32	**50 / 50**	**1038**	**3,436,849**	**20**	**32.8**	**133**
	$+\infty$	50 / 50	2,673	3,436,849	35	42.54	113
#16706	4	50 / 50	131	2,632,310	11	41.36	129
	8	50 / 50	132	2,632,310	19	26.68	125
	12	50 / 50	105	2,632,310	33	19.66	55
	16	**50 / 50**	**99**	**2,632,310**	**37**	**19**	**51**
	24	50 / 50	124	2,632,310	46	18.91	49
	32	50 / 50	137	2,632,310	49	19.45	49
	$+\infty$	49 / 50	153	2,632,310	49	19.45	49

FIGURE 12. Comparing the performance of DGVNS for different tree decompositions of bounded separators sizes.

for #9315, the gain in average CPU time is about 41%. For the other instances, this gain is about 13%.

For these instances, merging clusters allows to obtain clusters of reasonable size and to remove clusters that greatly overlap, leading to pertinent neighborhoods (*i.e.* weakly connected clusters of reasonable sizes), since most of the merged clusters have few proper variables. Moreover, for these instances, the size of the largest cluster and the number of clusters for the different decompositions are varying lightly.

6.5.2. Synthesis

From this study, we show that removing useless clusters (those with very few proper variables), by bounding the maximum separator size $s_{\max}$ and by merging clusters with separator size strictly greater than $s_{\max}$, enables to better refine the relevance of the MCS derived tree decomposition and to improve the performances

Inst.	s_{max}	Succ.	Time	Avg	$\lvert C_T \rvert$	Cluster size avg.	Cluster size max.
#6835	4	46 / 50	10,781	4,571,140	3	167	469
	8	46 / 50	9,925	4,571,122	3	167	469
	12	50 / 50	8,075	4,571,108	7	76.71	443
	16	50 / 50	7,966	4,571,108	7	76.71	443
	24	50 / 50	3,775	4,571,108	16	44.62	235
	32	50 / 50	3,321	4,571,108	23	39.26	229
	$+\infty$	**50 / 50**	**2,409**	**4,571,108**	**56**	**53.5**	**109**
#6858	4	0 / 50	-	26,882,903 (26,882,903)	2	496	989
	8	0 / 50	-	26,882,903 (26,882,903)	2	496	989
	12	0 / 50	-	26,882,862 (26,882,785)	3	334.33	951
	16	0 / 50	-	26,882,851 (26,882,785)	3	334.33	951
	24	0 / 50	-	26,882,836 (26,882,785)	6	177.66	945
	32	0 / 50	-	26,882,790 (26,882,697)	7	156.71	901
	$+\infty$	0 / 50	-	**26,882,588 (26,879,268)**	**105**	**156.23**	**351**
#8956	4	42 / 50	9,037	6,660,310	9	56.11	215
	8	46 / 50	7,109	6,660,309	14	38.42	215
	12	44 / 50	7,592	6,660310	15	36.6	215
	16	42 / 50	7,580	6,660,310	16	35.12	215
	24	41 / 50	7,015	6,660,311	26	29.53	215
	32	46 / 50	7,192	6,660,309	30	29.33	215
	$+\infty$	**50 / 50**	**4,911**	**6,660,308**	**54**	**52.03**	**185**
#14226	4	26 / 50	11,973	26131481	20	54.7	237
	8	13 / 50	11,737	26,387,778	31	37.19	185
	12	28 / 50	11,657	25,758,768	38	32.21	185
	16	35 / 50	11,333	25,828,540	42	30.52	179
	24	46 / 50	10,930	25,712,052	58	27.48	179
	32	22 / 50	10,434	26,108,198	66	27.54	179
	$+\infty$	**46 / 50**	**7,606**	**25,688,751**	**94**	**38.19**	**159**
#15757	4	50 / 50	195	2,278,611	13	27.61	215
	8	50 / 50	82	2,278,611	19	21	101
	12	**50 / 50**	**54**	**2,278,611**	**25**	**18.44**	**101**
	16	50 / 50	64	2,27,8611	29	17.82	73
	24	50 / 50	72	2,278,611	41	18.31	73
	32	50 / 50	100	2,278,611	43	18.62	71
	$+\infty$	50 / 50	60	2,278,611	45	20.24	67
#17034	4	5 / 50	13,096	39,635,091	21	56.23	445
	8	2 / 50	11,979	39,665,758	30	41.26	445
	12	5 / 50	12,419	39,573,906	42	32.42	233
	16	14 / 50	12,149	39,359,507	55	27.98	231
	24	24 / 50	11,407	39,114,487	63	27.19	231
	32	32 / 50	11,058	38,838,844	79	27.12	229
	$+\infty$	**41 / 50**	**8,900**	**38,563,232**	**139**	**52.95**	**189**

FIGURE 13. Comparing the performance of DGVNS for different tree decompositions of bounded separators sizes.

of DGVNS. However, this method suffers from its rigidity, since the criterion used to merge clusters cannot take into account the structure of the instance. Indeed, restricting $s_{\max}$ to 32 on problems where the size of separators are very large would lead to merge all the clusters. Moreover, one may merge two clusters of size 100 with $s_{\max} = 32$, while keeping those of size 32 with a separator size equal to 31. This probably explains the weak results of DGVNS on instances of Figure 13. It is thus necessary to find more relevant criteria that take into account these information.

7. Conclusion

We have shown that *tree decomposition* can be used to efficiently guide the exploration of the search space. Moreover, we have introduced *tightness dependent tree decomposition* which allows to take advantage of both the structure of the problem and the constraints tightness. Our intensification/diversification scheme is generic and can be applied to other local search methods such as `LNS`. Finally, experiments show that, exploiting such decompositions, enables to clearly outperform `VNS/LDS+CP` and `ID-Walk`.

We are currently investigating two directions: using *separators* and *proper variables of clusters* to drive the search, and parallelizing the exploration of clusters.

Furthermore, we want to study two other promising directions:

(i) our current definition of constraint tightness does not take into account costs. Consider the two cost functions w_1 and w_2 defined on the same subset of variables: w_1 assigns a cost of 1 to every tuple, while w_2 assigns a cost of 1000 to every tuple. Both cost functions will be considered with the same tightness. However w_2 has much more influence than w_1. As tightness dependent tree decomposition is appropriate, it would be interesting to take into account costs, as in [24], to drive the decomposition.

(ii) using hyper–tree decomposition methods [18]. Contrary to tree decomposition which consists in grouping the vertices in clusters (*i.e.* variables in subproblems), hyper–tree decomposition consists in grouping the constraints (or hyper-edges) in nodes of the hyper–tree.

Acknowledgements. This work is partly supported by the ANR (French Research National Agency) funded project FiCOLOFO ANR-10-BLA-0214. We would like to thank David Allouche for his help on the `tagSNP` instances.

Appendix A

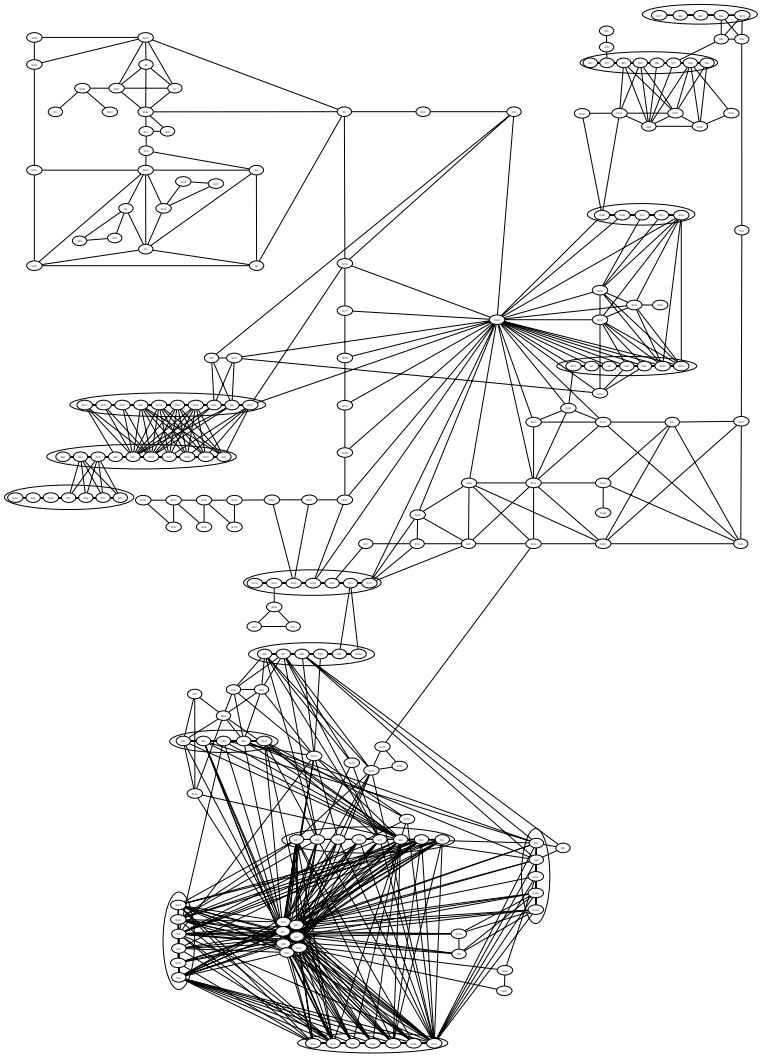


Figure 14. The constraints graph for instance Scen07. Variables contained in a same ellipse form a cluster.

Appendix B

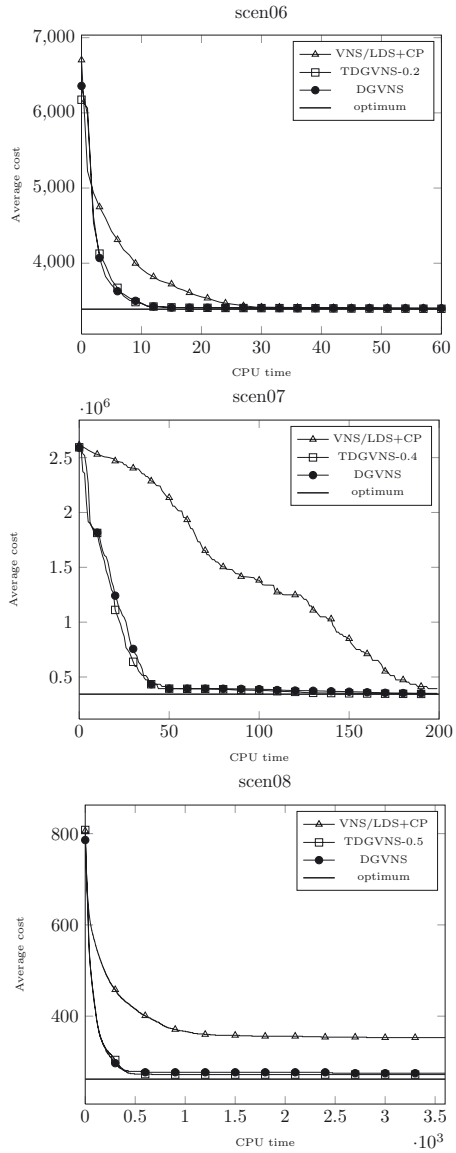


Figure 15. Comparison of performance profiles on `RLFAP` instances.

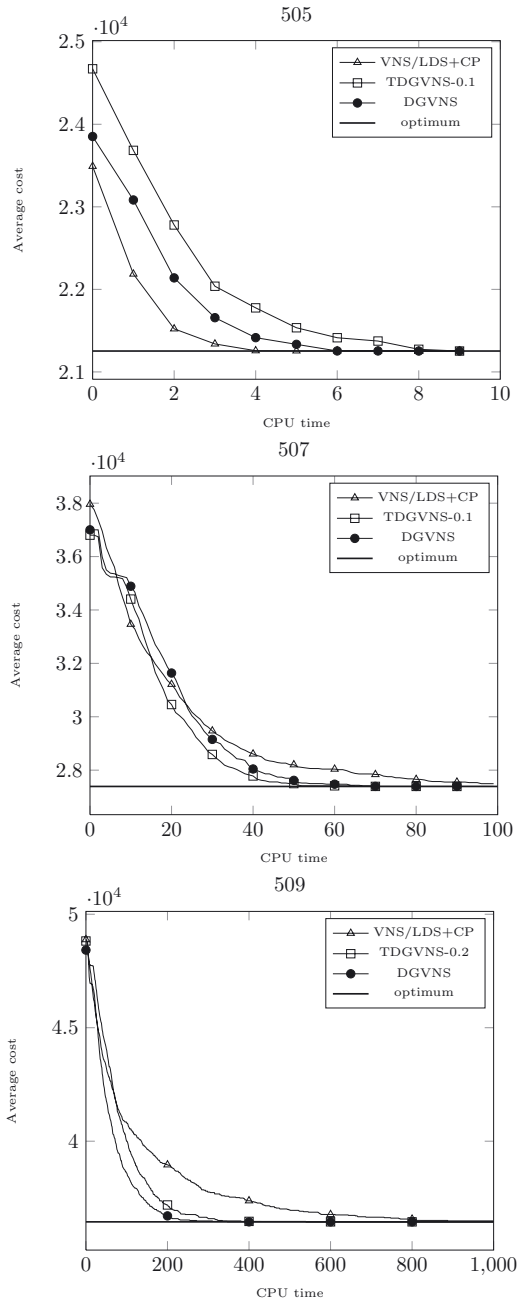


FIGURE 16. Comparison of performance profiles on SPOT5 instances.

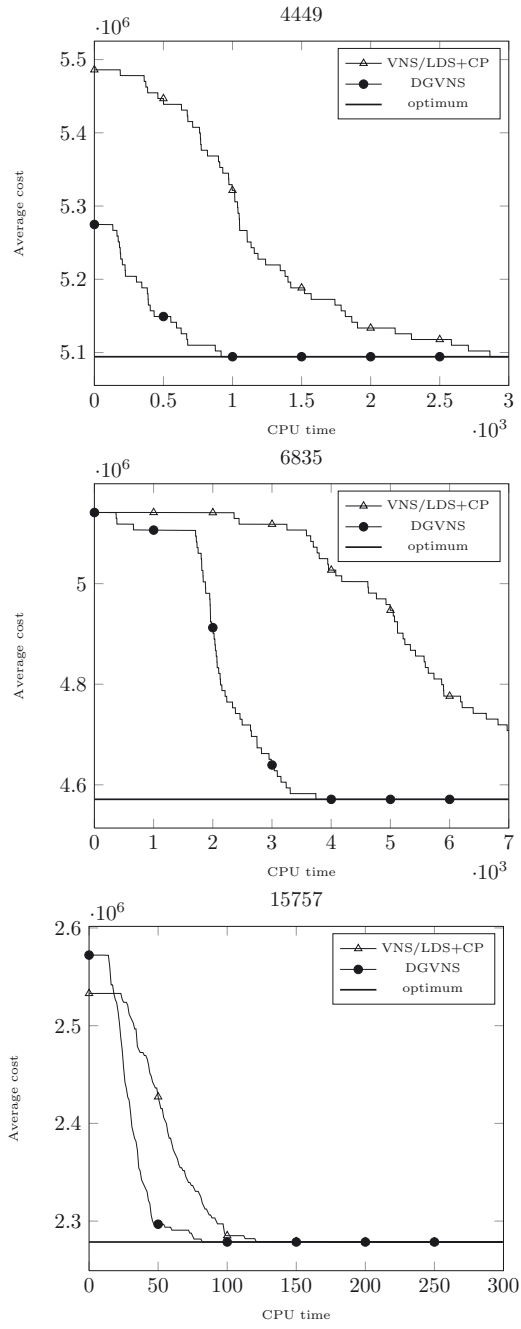


FIGURE 17. Comparison of performance profiles on `tagSNP` instances of medium-sized.

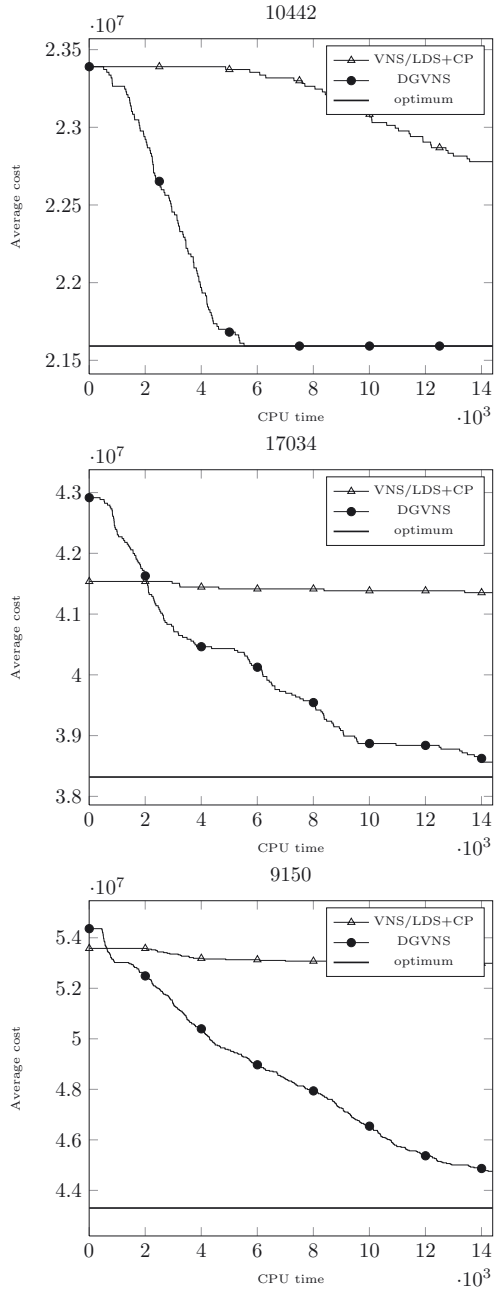


FIGURE 18. Comparison of performance profiles on `tagSNP` large instances.

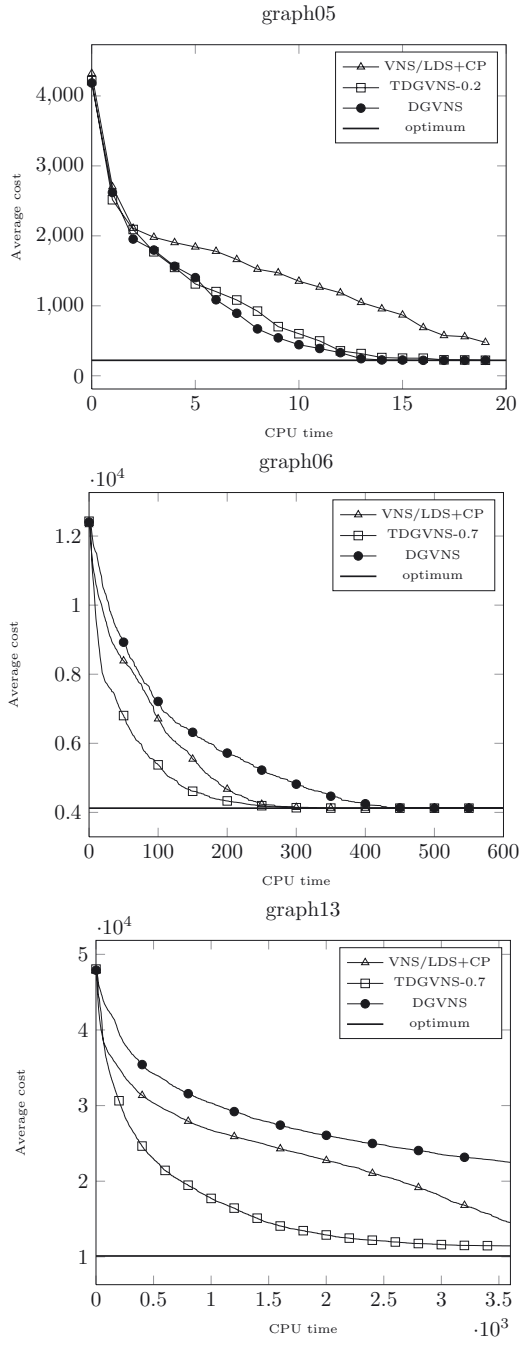


FIGURE 19. Comparison of performance profiles on GRAPH instances.

References

[1] `http://www-sop.inria.fr/coprin/neveu/incop/presentation-incop.html`.

[2] S. Arnborg, D.G. Corneil and A. Proskurowski, Complexity of finding embeddings in a k–tree. *SIAM J. Algebraic Discrete Methods* **8** (1987) 277–284.

[3] E. Balas and J. Xue, Weighted and unweighted maximum clique algorithms with upper bounds from fractional colorings. *Algorithmica* **15** (1996) 397–412.

[4] E. Bensana, M. Lemaître and G. Verfaillie, Earth observation satellite management. *Constraints* **4** (1999) 293–299.

[5] E. Bensana, G. Verfaillie, J.C. Agnèse, N. Bataille and D. Blumstein, Exact and approximate methods for the daily management of an earth observation satellite, in *Proc. of the 4th International Symposium on Space Mission Operations and Ground Data Systems (SpaceOps-9-)* (1996).

[6] C. Boutilier, editor. *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, Pasadena, California, USA, July 11-17, 2009 (2009).

[7] B. Cabon, S. de Givry, L. Lobjois, T. Schiex and J.P. Warners, Radio link frequency assignment. *Constraints* **4** (1999) 79–89.

[8] L.R. Cardon and G.R. Abecasis, Using haplotype blocks to map human complex trait loci. *Trends Genetics* **19** (2003) 135–140.

[9] M. C. Cooper, S. de Givry, M. Sánchez, T. Schiex, M. Zytnicki and T. Werner, Soft arc consistency revisited. *Artif. Intell.* **174** (2010) 449–478.

[10] S. de Givry, F. Heras, M. Zytnicki and J. Larrosa, Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. *IJCAI* (2005) 84–89.

[11] S. de Givry, T. Schiex and G. Verfaillie, Exploiting tree decomposition and soft local consistency in weighted CSP. *AAAI Press* (2006) 22–27.

[12] Simon. de Givry, *Rapport d'habilitation à diriger les recherches.* INRA UBIA Toulouse (2011).

[13] R. Dechter and J. Pearl, Tree clustering for constraint networks. *Artif. Intell.* **38** (1989) 353–366.

[14] C.S. Carlson et al. Selecting a maximally informative set of single-nucleotide polymorphisms for association analyses using linkage disequilibrium. *Am. J. Hum. Genet.* **74** (2004) 106–120.

[15] M. Fontaine, S. Loudni and P. Boizumault, Guiding VNS with tree decomposition. *ICTAI.* IEEE (2011) 505–512.

[16] F. Gavri, The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Comb. Theory Ser. B* **16** (1974) 47–56.

[17] G. Gottlob, S.T. Lee and G. Valiant, Size and treewidth bounds for conjunctive queries, in edited by Jan Paredaens and Jianwen Su. *PODS ACM* (2009) 45–54.

[18] G. Gottlob, Z. Miklós and T. Schwentick, Generalized hypertree decompositions: Np–hardness and tractable variants. *J. ACM* **56** (2009).

[19] G. Gottlob, R. Pichler and F. Wei, Tractable database design through bounded treewidth, in *PODS*, edited by Stijn Vansummeren. ACM (2006) 124–133.

[20] P. Hansen, N. Mladenovic and D. Perez-Brito, Variable neighborhood decomposition search. *J. Heuristics* **7** (2001) 335–350.

[21] W.D. Harvey and M.L. Ginsberg, Limited discrepancy search, in *IJCAI (1)*. Morgan Kaufmann (1995) 607–615.

[22] P. Jégou, S. Ndiaye and C. Terrioux, Computing and exploiting tree–decompositions for solving constraint networks, in *CP*, edited by Peter van Beek. Springer. *Lect. Notes Comput. Sci.* **3709** (2005) 777–781.

[23] Ph. Jégou, S. Ndiaye and C. Terrioux, Strategies and heuristics for exploiting tree–decompositions of constraint networks, in *Inference methods based on graphical structures of knowledge (WIGSK'06)* (2006) 13–18.

[24] M. Kitching and F. Bacchus, Exploiting decomposition on constraint problems with high tree–width, in Boutilier [6] 525–531.
[25] A.M.C.A. Koster, H.L. Bodlaender and S.P.M. van Hoesel, Treewidth: Computational experiments. *Electr. Notes Discrete Math.* **8** (2001) 54–57.
[26] J. Larrosa and T. Schiex, In the quest of the best form of local consistency for weighted CSP, in *IJCAI* (2003) 239–244.
[27] J. Larrosa and T. Schiex, Solving weighted CSP by maintaining arc consistency. *Artif. Intell.* **159** (2004) 1–26.
[28] S. Loudni and P. Boizumault, Combining VNS with constraint programming for solving anytime optimization problems. *EJOR* **191** (2008) 705–735.
[29] R. Marinescu and R. Dechter, AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artif. Intell.* **173** (2009) 1457–1491.
[30] N. Mladenovic and P. Hansen. Variable neighborhood search. *Comput. Oper. Res.* **24** (1997) 1097–1100.
[31] B. Neveu, G. Trombettoni and F. Glover, ID Walk: A candidate list strategy with a simple diversification device. in Wallace [45] 423–437.
[32] C. Papadimitriou and M. Yannakakis, Optimization, approximation and complexity classes. *J. Comput. Syst. Sci.* **43** (1991) 425–440.
[33] J. Pearl, Probabilistic inference in intelligent systems, in *Networks of Plausible Inference*. Morgan Kaufmann (1998).
[34] L. Perron, P. Shaw and V. Furnon, Propagation guided large neighborhood search, in Wallace [45] 468–481.
[35] Z.S. Qin, S. Gopalakrishnan and G.R. Abecasis, An efficient comprehensive search algorithm for tagsnp selection using linkage disequilibrium criteria. *Bioinformatics* **22** (2006) 220–225.
[36] I. Rish and R. Dechter, Resolution versus search: Two strategies for sat. *J. Autom. Reason.* **24** (2000) 225–275.
[37] N. Robertson and P.D. Seymour, Graph minors. ii. algorithmic aspects of tree–width. *J. Algorithms* **7** (1986) 309–322.
[38] M. Sánchez, D. Allouche, S. de Givry and T. Schiex, Russian doll search with tree decomposition, in Boutilier [6] 603–608.
[39] M. Sánchez, S. de Givry and T. Schiex, Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints* **13** (2008) 130–154.
[40] T. Sandholm, An algorithm for optimal winner determination in combinatorial auctions, in *IJCAI'99* (1999) 342–347.
[41] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in *CP*, edited by M.J. Maher and J.-F. Puget. Springer. *Lect. Notes Comput. Sci.* **1520** (1998) 417–431.
[42] R.E. 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** (1984) 566–579.
[43] C. Terrioux and P. Jégou, Hybrid backtracking bounded by tree–decomposition of constraint networks. *Artif. Intell.* **146** (2003) 43–75.
[44] H. van Benthem, GRAPH: Generating radiolink frequency assignment problems heuristically. Master Thesis, Delft Univ. Technol, The Netherlands (1995).
[45] M. Wallace, editor. Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings. Springer. *Lect. Notes Comput. Sci.* **3258** (2004).