Exploitation de la décomposition arborescente pour guider la recherche VNS

Mathieu Fontaine, Samir Loudni, Patrice Boizumault

Exploitation de la décomposition arborescente pour guider la recherche VNS

Mathieu Fontaine **Samir Loudni** **Patrice Boizumault**

Université de Caen Basse-Normandie, UMR 6072 GREYC, F-14032 Caen, France
CNRS, UMR 6072 GREYC, F-14032 Caen, France
{prénom.nom}@unicaen.fr

Résumé

La décomposition arborescente vise à découper un problème en *clusters* formant un graphe acyclique. Il existe des travaux exploitant la décomposition arborescente dans les méthodes complètes. Dans ce papier, nous montrons comment la decomposition arborescente peut être utilisée afin de guider l'exploration de méthodes de recherche locale à voisinages larges comme VNS. Nous présentons la *tightness dependent tree decomposition* (TDTD) qui permet de prendre en compte à la fois la structure du problème et la dureté des contraintes. Les expérimentations menées sur des instance aléatoires (GRAPH) et des instances réelles (CELAR et SPOT5) montrent la pertinence et l'efficacité de notre approche. *Cet article a été publié dans les actes d'ICTAI'2011 [5].*

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 the exploration of local search methods that use large neighborhoods like VNS. We introduce tightness dependent tree decomposition which allows to take advantage of both the structure of the problem and the constraints tightness. Experiments performed on random instances (GRAPH) and real life instances (CELAR and SPOT5) show the appropriateness and the efficiency of our approach. *This paper has been published in the proceedings of ICTAI'2011 [5].*

1 Introduction

Un grand nombre de problèmes réels, tels que les problèmes d'affectation de fréquence [3] ou la planification d'un satellite d'observation de la terre [2], sont de très grande taille et exhibent un graphe de contraintes trés structuré. Exploiter ces propriétés structurelles peut aider à traiter ces problèmes.

La *décomposition arborescente*, proposée par Robertson et Seymour [17], vise à découper un problème en sous-problèmes (les *clusters*) constituant un graphe acyclique. Chaque *cluster* correspond à un sous ensemble de variables fortement connexes. Chaque sous-problème étant plus petit que le problème original, il est plus facile à résoudre. Un certain nombre de travaux exploitent la décomposition arborescente dans des méthodes de recherche complète, comme par exemple RDS-BTD [18], AND/OR graph search [12] et Decomposition Bounding (DB) [9].

Dans les méthodes de recherche locale qui utilisent des voisinages étendus tels que *Large Neighborhood Search* (LNS) [19], *Propagation guided large neighborhood search* (PGLNS) [16] ou encore *Variable Neighborhood Search* (VNS) [13], la définition de structures de voisinage pertinentes est une étape cruciale, car elle permet d'intensifier et de diversifier la recherche dans différentes régions de l'espace de recherche. À notre connaissance, il n'existe aucun travail exploitant la décomposition arborescente dans de telles méthodes.

Le cadre des *problèmes de satisfaction de contraintes pondérées* ($WCSP$ [10]) est un formalisme générique utilisé pour la modélisation et la résolution de problèmes d'optimisation sous contraintes. Il permet de traiter les problèmes sur-contraints et de modéliser les préférences entre solutions. Les $WCSP$ peuvent être résolus par des méthodes de recherche locale, hybride ou arborescente.

Dans ce papier, nous montrons comment la décomposition arborescente peut être utilisée afin de guider l'exploration de VNS. De plus, nous proposons la *tightness dependent tree decomposition* (TDTD) qui permet de prendre en compte à la fois la structure du

problème et la dureté des contraintes. Les expérimentations menées sur des instances aléatoires (GRAPH) et des instances réelles (CELAR et SPOT5) montrent que l'exploitation de la décomposition arborescente procure des performances bien meilleures comparé à VNS/LDS+CP [11] ou à ID-Walk [15]. À notre connaissance, nos travaux constituent la première tentative visant à utiliser la décomposition arborescente afin de guider une recherche locale de type VNS.

La section 2 présente le contexte de nos travaux. La section 3 montre comment exploiter la décomposition arborescente dans VNS. La TDTD est présentée dans la section 4. Les sections 5 et 6 présentent les résultats expérimentaux. Enfin, nous concluons et présentons des perspectives.

2 Définitions et notations

2.1 Problème de satisfaction de containtes pondérées (WCSP)

Un $WCSP$ est défini par un quadruplet $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{S}(k_\top))$, où $\mathcal{X}=\{x_1, ..., x_n\}$ est un ensemble de n *variables*, $\mathcal{D}=\{d_1, ..., d_n\}$ sont leurs *domaines finis respectifs*, $\mathcal{C}$ est un ensemble de *contraintes* sur $\mathcal{X}$, et $\mathcal{S}(k_\top)$ est une *structure de valuation*. $\mathcal{S}(k_\top)$ est un triplet $([[0,1,...,k_\top], \oplus, \geq)$ où : $k_\top$ est un entier naturel dans $[1, .., \infty]$, $\oplus$ est défini par : $a \oplus b = min(k_\top, a+b)$ et $\geq$ est l'opérateur d'ordre standard sur les entiers naturels. Chaque contrainte $c \in \mathcal{C}$ est défini par i) son *scope* $\mathcal{X}_c \subseteq \mathcal{X}$ (ensemble de variables sur lesquelles porte c), ii) sa *relation* $R_c = \prod_{x_i \in \mathcal{X}_c} d_i$ qui définit un ensemble d'affectations (ou tuples) de $\mathcal{X}_c$ et iii) sa *fonction de coût* $f_c : R_c \mapsto [0, k_\top]$. L'affectation de x_i à une valeur $a \in d_i$ est notée $(x_i{=}a)$. Une affectation *complète* $\mathcal{A}=(a_1, ..., a_n)$ est une affectation de toutes les variables ; dans le cas contraire, on l'appelle affectation *partielle*. $\mathcal{A}_{\downarrow \mathcal{X}_c}$ désigne une affectation partielle obtenue en projetant l'affectation complète $\mathcal{A}$ sur les variables de la contrainte c. Pour une affectation complète $\mathcal{A}$, si $f_c(\mathcal{A}_{\downarrow \mathcal{X}_c}) = 0$, c est dite satisfaite, sinon elle est violée. Le coût d'une affectation complète $\mathcal{A}=(a_1, ..., a_n)$ est défini par $f(A) = \sum_{c \in \mathcal{C}} f_c(\mathcal{A}_{\downarrow \mathcal{X}_c})$. L'objectif est de trouver une affectation complète de coût minimal : $min_{\mathcal{A} \in d_1 \times d_2 \times ... \times d_n} f(\mathcal{A})$.

2.2 Décomposition arborescente

Le graphe de contraintes d'un $WCSP$ est un graphe $G=(\mathcal{X}, E)$ composé d'un sommet par variable et d'une arête (u, v) pour chaque contrainte $c \in \mathcal{C}$ telle que $u, v \in \mathcal{X}_c$.

Definition 1 *Une décomposition arborescente [17] de $G=(\mathcal{X}, E)$ est un couple (C_T, T) où :*

- *$T = (I, A)$ est un arbre avec pour ensemble de nœuds I et pour ensemble d'arêtes A,*
- *$C_T = \{C_i \mid i \in I\}$ est une famille de sous-ensembles de $\mathcal{X}$ (appelés clusters) telle que :*
  - *$\cup_{i \in I} C_i = \mathcal{X}$,*
  - *$\forall (u, v) \in E$, $\exists C_i \in C_T$ t.q. $u, v \in C_i$,*
  - *$\forall i, j, k \in I$, si j est sur le chemin de i à k dans T, alors $C_i \cap C_k \subseteq C_j$.*

Definition 2 *L'intersection entre deux clusters est appelée séparateur, et est notée $sep(C_i, C_j)$. Deux clusters sont adjacents s'ils partagent au moins une variable. Le voisinage d'un cluster C_i dans T est $\mathcal{N}(C_i)=\{C_j \mid j \in I, sep(C_i, C_j) \neq \emptyset\}$. De la même façon, pour un ensemble C_s de clusters dans T, $\mathcal{N}(C_s)=\cup_{C_i \in C_s} \mathcal{N}(C_i)$. Les variables appartenant à un et un seul cluster sont appelées variables propres.*

La largeur d'une décomposition arborescente $T=(I, A)$ est définie par $w(T)=\max_{i \in I}(|C_i|-1)$. La largeur de décomposition $tw(G)$ d'un graphe G est la plus petite largeur de toutes les décompositions arborescentes possibles de G. Calculer la largeur de décomposition d'un graphe est un problème NP-complet [1]. Cependant, des heuristiques reposant sur la notion de *triangulation* de graphe [1] permettent le calcul de décompositions approchées. Elles fournissent un majorant de la largeur de décomposition.

Dans cet article, nous utilisons l'heuristique *maximum cardinality search* (MCS) [20] qui constitue un bon compromis entre la largeur de la décomposition obtenue et le temps nécessaire à son calcul [8].

La Fig. 1 illustre les trois étapes nécessaires au calcul d'une décomposition d'un graphe G (a). Tout d'abord, le graphe G est triangulé par ajout d'arêtes (b). Ensuite, on calcule les cliques maximales afin de constituer le graphe de *clusters* (c). Enfin, on obtient la décomposition arborescente de G par calcul de l'arbre de jointure (d).

2.3 VNS/LDS+CP

VNS/LDS+CP [11] est une méthode de recherche locale qui exploite le principe de VNDS (*Variable Neighborhood Decomposition Search*) [6]. Les voisinages sont obtenus en désaffectant une partie de la solution courante selon une heuristique de choix de voisinage. Ensuite, l'exploration de l'espace de recherche est réalisée par une recherche arborescente partielle LDS (*Limited discrepancy search*, [7]) aidée par une propagation des contraintes (CP) basée sur un calcul de minorants.

L'algorithme 1 présente le pseudo-code de VNS/LDS+CP. Soit $\mathcal{X}$ l'ensemble des variables et

1. Un graphe est cordal ou triangulé si et seulement si tout ses cycles de taille supérieure à quatre ont une corde (une arête connectant deux sommets non-adjacents du cycle).

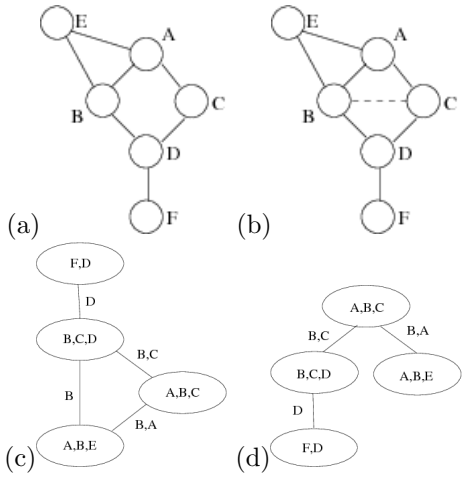


FIGURE 1 – (a) Graphe initial G. (b) Exemple de triangulation de G. (c) Cliques maximales du graphe triangulé (*graphe de clusters*). (d) Décomposition arborescente de G de largeur 2.

Algorithm 1: VNS/LDS+CP

```
   fonction VNS/LDS+CP(X, C, 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' ← LDS+CP(A, X_un, δ_max, f(S), S) ;
7          if f(S') < f(S) then
8              S ← S' ;
9              k ← k_init ;
10         else k ← k + 1 ;
11     return S ;
   end
```

N_k la structure de voisinage de dimension k ; N_k correspond à l'ensemble des sous-ensembles de k variables de $\mathcal{X}$. VNS/LDS+CP part d'une solution initiale S aléatoire (ligne 1). Un sous-ensemble de k variables est sélectionné dans $\mathcal{X}$ par l'heuristique de choix de voisinage Hneighborhood (ligne 4). Une affectation partielle $\mathcal{A}$ est générée à partir de la solution courante S en désaffectant les k variables sélectionnées ; Les $(n-k)$ variables restantes gardent leur valeur dans S (ligne 5). Ensuite, la solution est reconstruite (ligne 6) par une recherche arborescente partielle (LDS) aidée par une propagation des contraintes (CP) basée sur le calcul de minorants. Si cette recherche trouve une solution de meilleure qualité S' dans le voisinage de S (ligne 7), S' devient la solution courante et k est réinitialisé à k_{init} (lignes 8-9). Sinon, on cherche de nouvelles améliorations dans N_{k+1} (structure de voisinage de taille $k+1$)(ligne 10)). La recherche s'arrête quand elle a atteint k_{max} la taille maximale de voisinage ou la limite de temps (ligne 3).

2.4 Heuristique de choix de voisinage

L'heuristique de choix de voisinage, utilisée pour sélectionner les variables à désaffecter, guide VNDS dans l'exploration de l'espace de recherche afin de trouver des solutions de meilleure qualité. ConflictVar est une des heuristiques les plus simples : pour une dimension de voisinage k, ConflictVar sélectionne aléatoirement k variables à désaffecter parmi celles en conflit. Une telle heuristique, basée sur des choix aléatoires, permet de diversifier la recherche et de sortir rapidement des optima locaux.

3 Intensification/Diversification exploitant la décomposition arborescente

Dans cette section, nous présentons la première contribution de notre article : DGVNS (Decomposition Guided VNS) qui utilise le graphe de *clusters* afin de construire des structures de voisinage permettant une meilleure intensification et une meilleure diversification que VNS/LDS+CP. Tout comme VNS/LDS+CP, DGVNS utilise une recherche arborescente partielle (LDS+CP) pour la phase de reconstruction, mais les voisinages sont gérés différemment afin de tirer parti du graphe de *clusters*.

Au lieu d'utiliser les structures de voisinage N_k de VNS/LDS+CP (voir section 2.3), DGVNS utilise des structures de voisinage $N_{k,i}$, où k est la dimension du voisinage et C_i désigne le *cluster* dans lequel les variables à désaffecter vont être sélectionnées. L'algorithme 2 décrit le pseudo-code de DGVNS.

Nous favorisons les mouvements dans des régions fortement liées. Le *cluster* est une structure qui permet d'identifer ces régions, du fait de sa taille (plus petite que le problème original), et du lien fort entre les variables qu'il contient. Nous sélectionnons tout d'abord les k variables à désaffecter dans un même *cluster* C_i. Si $(k > |C_i|)$, alors l'ensemble des variables potentielles C_s est complété en ajoutant les variables des *clusters* C_j adjacents à C_i afin de prendre en compte la topologie du graphe de *clusters*. Ce traitement est accompli par la fonction CompleteCluster(C_i) (ligne 7). La structure de voisinage $N_{k,i}$ est constituée de l'ensemble des sous-ensembles de k variables de C_s (ligne 8).

le but de la diversification est de traiter un grand nombre de régions différentes, afin d'assurer que l'espace de recherche est correctement exploré, et d'essayer de localiser la région contenant l'optimum global.

Algorithm 2: DGVNS

```
   fonction DGVNS(X, C, k_init, k_max, δ_max);
   begin
1      Soit G le graphe de contraintes de P ;
2      Soit (C_T, T) une décomposition arborescente de G ;
       Soit 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) ;
8          X_un ← Hneighborhood(C_s, N_{k,i}, S) ;
9          A ← S\{(x_i = a) | x_i ∈ X_un} ;
10         S' ← LDS+CP(A, X_un, δ_max, f(S), S) ;
11         if f(S') < f(S) then
12             S ← S' ;
13             k ← k_init ;
14             i ← succ(i) ;
15         else k ← k + 1 ; i ← succ(i) ;
16     return S ;
   end
```

Afin d'améliorer la diversification, nous considérons successivement tous les C_i.

Soit p le nombre total de *clusters*, *succ* est la fonction qui associe à un *cluster* i son successeur [2], et $N_{k,i}$ la structure de voisinage courante : si LDS+CP trouve une solution de meilleure qualité S' dans le voisinage de S (ligne 11), alors S' devient la solution courante (ligne 12), k est réinitialisé à k_{init} (ligne 13), et le *cluster* suivant est considéré (ligne 14). Sinon, on cherche des améliorations dans $N_{(k+1),succ(i)}$ (structure de voisinage de taille $(k+1)$) (ligne 15). La recherche s'arrête quand elle atteint k_{max} la taille maximale de voisinage ou la limite de temps (ligne 6).

Tout d'abord, la diversification réalisée par le déplacement du *cluster* C_i vers le *cluster* $C_{succ(i)}$ est nécessaire. Les expérimentations que nous avons menées montrent que rester dans un même *cluster* donne de moins bons résultats : sélectionner un nouveau *cluster* permet d'améliorer la qualité de la solution en visitant de nouvelles parties de l'espace de recherche.

Puis, quand un minimum local est atteint, passer de k à $(k+1)$ permet de renforcer la diversification par l'élargissement de la taille du voisinage.

Le but de l'intensification est de trouver la meilleure solution dans une petite région de l'espace de recherche. Ceci est réalisé par la reconstruction de la solution partielle avec LDS+CP et par la réduction de la taille de voisinage à k_{init} à chaque amélioration.

2. si $i < p$ alors $succ(i) = i + 1$ sinon $succ(p) = 1$.

4 Tightness Dependent Tree Decomposition

Les problèmes d'optimisation contiennent souvent des sous-problèmes plus durs à résoudre que les autres en raison de la *dureté* de leur contraintes. Ces sous-problèmes restent cachés dans la décomposition. Pour identifier ces sous-parties difficiles, on utilise *la dureté de la contrainte*. Soit c une contrainte et R_c sa relation. La dureté de la contrainte c est définie par [3] :

$$t(c) = \frac{|\{t \,|\, f_c(t) > 0, t \in R_c\}|}{|\{t \in R_c\}|} \quad (1)$$

Notre approche procède en deux temps :

1. Toutes les contraintes de dureté inférieure à un seuil (noté λ) sont retirées du graphe initial.
2. Le graphe ainsi obtenu est décomposé.

La fig. 2 montre l'influence du seuil λ sur la décomposition de l'instance Scen06. La col. 1 donne les différentes valeurs de λ. La col. 2 indique le pourcentage de contraintes retirées. La col. 3 présente le nombre total de *clusters*. Les col. 4-12 reportent respectivement, pour chacun des trois paramètres (*taille des clusters*, *degré du cluster* et *taille des séparateurs*), leurs valeurs minimale, moyenne et maximale. Enfin la dernière colonne indique le nombre de séparateurs. La décomposition arborescente sur le graphe initial correspond au seuil λ=0.

Pour $0,2 \leq \lambda \leq 0,5$, la TDTD donne le plus souvent un grand nombre de *clusters* de petite taille, tout en conservant les contraintes les plus importantes du problème initial. De plus, les *clusters* ont (en moyenne) un plus haut degré (une connexité plus grande) que ceux obtenus avec $\lambda = 0$. Ceci montre l'intérêt et l'importance de la TDTD. Cependant, pour des valeurs élevées de λ ($\lambda \geq 0,6$), elle n'est pas pertinente car trop de contraintes sont retirées, conduisant à des *clusters* isolés de taille négligeable. Les résultats obtenus sur les autres instances du CELAR ne sont pas reportés dans cet article car ils sont très similaires. Sur les instances SPOT5, la TDTD est pertinente pour $0,1 \leq \lambda \leq 0,3$ (voir sect. 6.3 pour plus de détails).

5 Jeux de test

Les expérimentations ont été menées sur les instances de trois problèmes différents :

3. Notre définition courante de la dureté d'une contrainte ne prend pas en compte les fonctions de coût (voir les perspectives, section 7).

λ	% dropped	$\lvert C_T \rvert$	Taille des *clusters*			Degré des *clusters*			Taille des séparateurs			
	c		min.	moy.	max.	min.	moy.	max.	min.	moy.	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 2 – TDTD de l'instance Scen06.

Instance	Méthode	Succ.	Temps	Moy.
Scen06	DGVNS	50/50	112	3389
$n = 100$	SDGVNS-0, 2	**50/50**	**58**	**3389**
$e = 1222$	VNS/LDS+CP	15/50	83	3399
$S^* = 3389$	ID-Walk	NA	840	3447 (3389)
Scen07	DGVNS	40/50	317	345614
$n = 200$	SDGVNS-0, 4	**49/50**	**221**	**343600**
$e = 2665$	VNS/LDS+CP	1/50	461	355982
$S^* = 343592$	ID-Walk	NA	360	373334 (343998)
Scen08	DGVNS	3/50	1811	275
$n = 458$	SDGNVS-0, 5	**9/50**	**442**	**272**
$e = 5286$	VNS/LDS+CP	0/50	-	394 (357)
$S^* = 262$	ID-Walk	NA	3000	291 (267)

FIGURE 3 – Comparaison entre DGVNS, VNS/LDS+CP et ID-Walk sur les instances RLFAP.

Instances RLFAP : Le CELAR (Centre d'électronique de l'Armement) a mis à disposition un ensemble d'instances du problème d'affectation de fréquences radio (RLFAP) [3]. L'objectif est d'assigner un nombre limité de fréquences à un ensemble de liens radios entre des paires de sites, afin de minimiser les interférences dues à la réutilisation des fréquences. Nous reportons les résultats sur les instances les plus difficiles : Scen06, Scen07 et Scen08.

Instances GRAPH : Le générateur GRAPH (Generating Radio link frequency Assignment Problems Heuristically) a été développé par le projet CALMA [21] afin de proposer des instances aléatoires ayant une structure proche des instances RLFAP.

Instances SPOT5 : La planification quotidienne d'un satellite d'observation de la terre, SPOT5, consiste à sélectionner les prises de vue à effectuer dans la journée en prenant en compte les limites matérielles du satellite tout en maximisant l'importance des photographies sélectionnées [2]. Nous avons réalisé des expérimentations sur les instances SPOT5 afin de confirmer les conclusions tirées à partir des instances RLFAP. Nous reportons les résultats sur six instances sans contraintes dures de capacité.

6 Expérimentations

6.1 Protocole expérimental

Chaque méthode a été appliquée sur chaque instance, avec une *discrepancy* de 3 pour LDS, ce qui correspond à la meilleure valeur trouvée sur les instances RLFAP [11]. Les valeurs de k_{min} et k_{max} ont été respectivement fixées à 4 et n (le nombre total de variables), et le temps de calcul maximal à 3600 secondes. Un ensemble de 50 essais par instance a été réalisé sur un AMD opteron 2,1 GHz et 256 Go de RAM. Toutes les méthodes ont été implantées en C++ en utilisant la librairie toulbar2 [4].Pour ID-Walk, Nous avons utilisé la librairie INCOP [14]. Pour chaque instance et chaque méthode, nous reportons le nombre d'essais réussis (c'est-à-dire ayant atteint l'optimum), le temps de calcul moyen pour atteindre l'optimum ainsi que le coût moyen des solutions trouvées sur les 50 essais et, si nécessaire, le meilleur coût obtenu (noté entre parenthèse) dans le cas où l'optimum n'a pas été atteint. Nous présentons aussi les *profils de performance moyens* sur les 50 essais montrant l'évolution de la qualité des solutions en fonction du temps.

Dans un premier temps, nous comparons DGVNS avec VNS/LDS+CP et ID-Walk [15], une des méthodes de recherche locale les plus efficaces sur les instances RLFAP (section 6.2). Dans un second temps, nous comparons DGVNS avec la version utilisant la TDTD (SDGVNS-λ). Nous présentons les résultats pour différentes valeurs du seuil λ (section 6.3). Ensuite, nous étudions l'influence de la largeur de décomposition sur notre approche (section 6.4). Il est à noter que la comparaison des temps de calcul entre notre méthode et des méthodes de recherche complète serait peu pertinente. En effet, ces dernières vont à la fois trouver la ou les solutions optimales et prouver leur optimalité. Ces opérations peuvent prendre quelques jours sur ces instances [18].

4. http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP

Instance	Méthode	Succ.	Temps	Moy.
408	DGVNS	49/50	117	6228
$n = 200$	SDGVNS-0, 1	**50/50**	**72**	**6228**
$e = 2232$	VNS/LDS+CP	26/50	149	6228
$S^* = 6228$	ID-Walk	**50/50**	**3**	**6228**
412	DGVNS	36/50	84	32381
$n = 300$	SDGVNS-0, 1	**38/50**	**71**	**32381**
$e = 4348$	VNS/LDS+CP	32/50	130	32381
$S^* = 32381$	ID-Walk	10/50	2102	3238
414	DGVNS	38/50	554	38478
$n = 364$	SDGNVS-0, 1	**42/50**	**430**	**38478**
$e = 10108$	VNS/LDS+CP	12/50	434	38481
$S^* = 38478$	ID-Walk	0/50	-	38481
505	DGVNS	**50/50**	**63**	**21253**
$n = 240$	SDGVNS-0, 1	48/50	92	21253
$e = 2242$	VNS/LDS+CP	41/50	143	21253
$S^* = 21253$	ID-Walk	**50/50**	**358**	**21253**
507	DGVNS	33/50	71	27390
$n = 311$	SDGVNS-0, 1	**45/50**	**62**	**27390**
$e = 5732$	VNS/LDS+CP	11/50	232	27391
$S^* = 27390$	ID-Walk	7/50	1862	27391
509	DGVNS	**40/50**	**265**	**36446**
$n = 348$	SDGVNS-0, 2	**39/50**	**313**	**36446**
$e = 8624$	VNS/LDS+CP	12/50	598	36448
$S^* = 36446$	ID-Walk	0/50	-	36450

FIGURE 4 – Comparaison entre DGVNS, VNS/LDS+CP et ID-Walk sur les instances SPOT5.

Nos expérimentations montrent :

1. l'efficacité de notre approche comparée à VNS/LDS+CP et ID-Walk sur des instances structurées telles que RLFAP et SPOT5 ;
2. l'utilité de la TDTD pour créer des structures de voisinage pertinentes.

6.2 Apport de la décomposition arborescente

Sur les instances RLFAP, DGVNS surclasse VNS/LDS+CP (fig. 3). DGVNS atteint l'optimum sur 100% des essais pour la Scen06, 80% pour la Scen07 et 6% pour la scen08. VNS/LDS+CP n'obtient l'optimum que dans de rares cas sur les deux premières instances et ne l'obtient pas pour la Scen08 : la meilleure solution trouvée a un coût de 357. Cette tendance se confirme sur les instances SPOT5 (fig. 4) pour lesquelles DGVNS devance VNS/LDS+CP, en terme de taux de réussite (avec un gain de 40% en moyenne) et en terme de temps de calcul. En effet, DGVNS atteint l'optimum sur les 50 essais sur deux instances (408 et 505). Pour les autres instances (sauf la 507), le taux de réussite est au moins de 70%.

De plus, DGVNS dépasse nettement les performances d'ID-Walk [5], notamment sur les deux instances les plus

Instance	λ	Succ.	Temps	Moy.
Scen06	0, 2	**50/50**	**58**	**3389**
	0, 3	**50/50**	**61**	**3389**
	0, 4	50/50	110	3389
	0, 5	49/50	122	3389
Scen07	0, 2	45/50	271	344603
	0, 3	**47/50**	**229**	**344198**
	0, 4	**49/50**	**221**	**343600**
	0, 5	45/50	244	344603
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**	**6228**
	0, 2	**44/50**	**90**	**6228**
	0, 3	10/50	105	6229
412	0, 1	**38/50**	**71**	**32381**
	0, 2	32/50	135	32384
	0, 3	8/50	1337	32384
414	0, 1	**42/50**	**430**	**38478**
	0, 2	**38/50**	**471**	**38478**
	0, 3	4/50	704	38480
505	0, 1	**48/50**	**92**	**21253**
	0, 2	**18/50**	**148**	**21253**
	0, 3	4/50	319	21256
507	0, 1	**45/50**	**62**	**27390**
	0, 2	**28/50**	**134**	**27390**
	0, 3	3/50	418	27391
509	0, 1	**36/50**	**286**	**36446**
	0, 2	**39/50**	**313**	**36446**
	0, 3	2/50	583	36448

FIGURE 5 – Influence du seuil λ sur la TDTD

difficiles, Scen07 et Scen08 (Fig. 3). Pour la Scen07 (resp. Scen08), DGVNS obtient des solutions ayant une *déviation moyenne* (pourcentage de déviation par rapport à l'optimum) de 0, 55% (resp. 5%), tandis qu'ID-Walk obtient des solutions avec une déviation moyenne de 8% (resp. 11%).

Sur les instances SPOT5 (sauf la 408), DGVNS devance ID-Walk (Fig. 4), particulièrement sur les instances 414 et 509, où ID-Walk ne trouve jamais l'optimum. Pour ces deux instances, les meilleures solutions trouvées ont un coût respectif de 38479 et 36447.

Les fig. 10 et 11 comparent les profiles de performance de DGVNS et VNS/LDS+CP. DGVNS surpasse VNS/LDS+CP sur les problème RLFAP et SPOT5 (sauf pour l'instance 505). D'un point de vue *anytime*, on observe deux phénomènes importants. Premièrement, la courbe associée au comportement de DGVNS présente une plus forte pente. De plus, la décélération de la courbe associée au comportement de VNS/LDS+CP est beaucoup plus forte que celle de DGVNS. Cela confirme la pertinence de l'utilisation de la décomposition arborescente pour orienter l'exploration dans VNS.

5. Les résultats pour les instances RLFAP sont tirés de [15]. Le nombre de réussites et les temps de calcul ne sont pas disponibles (NA Fig. 3), Nous ne donnons donc que le temps par essai, le coût moyen obtenu sur 10 essais et le meilleur coût (entre parenthèses). Pour les autres instances (SPOT5 et GRAPH), les résultats ont été obtenus avec le protocole expérimental décrit précédemment.

6.3 Apport de la *tightness dependent tree decomposition* (TDTD)

Les fig. 3 et 4 comparent également SDGVNS et DGVNS. On reporte ici les valeurs associées aux seuils λ ayant donné les meilleurs résultats.

Sur les instances RLFAP, l'apport de la TDTD est très important, particulièrement sur les instances Scen07 et Scen08, pour lesquelles on observe de fortes améliorations en comparaison des résultats de DGVNS. Pour la Scen07, SDGVNS améliore de 18% le taux de réussite (de 80% à 98%) et réduit la déviation moyenne à l'optimum (de $0,55\%$ à $0,002\%$). Pour la Scen08, le taux de réussite est amélioré de 12% (de 6% à 18%), la déviation moyenne descend de 5% à $3,80\%$ et SDGVNS est 4 fois plus rapide. Pour la Scen06, SDGVNS divise par deux le temps de calcul. Une fois encore, cette tendance se confirme sur les instances SPOT5, sur lesquelles SDGVNS se montre très efficace. Sur les grandes instances (412, 414 et 507), SDGVNS améliore le nombre moyen de réussites ainsi que le temps de calcul de 12% et 25%. Sur les autres instances, les deux méthodes montrent des performances similaires. Cependant, SDGVNS est au moins $1,5$ fois plus rapide que DGVNS. Du point de vue des profils de performance (fig. 10 et 11), on observe que SDGVNS a systématiquement un meilleur comportement *anytime* que DGVNS.

La fig. 5 montre l'impact du seuil λ sur la TDTD. La meilleure valeur pour le seuil dépend de l'instance considérée. Les expérimentations montrent que pour les instances RLFAP, une valeur de λ comprise entre $0,2$ et $0,5$ donne les meilleurs résultats. Pour les instances SPOT5, on obtient les meilleures performances avec un seuil fixé entre $0,1$ et $0,3$. En effet, ces instances sont très contraintes car seulement un petit nombre de prises de vue peuvent être sélectionnées dans une solution optimale. Un seuil de $0,3$ (resp. $0,1$) donne des résultats plus homogènes sur les instances RLFAP (resp. SPOT5). Enfin, pour les instances RLFAP (resp. SPOT5), un seuil fixé au-delà de $0,6$ (resp. $0,4$) ne donne pas de bons résultats car trop de contraintes sont retirées. Nous ne reportons donc pas les résultats correspondants dans cet article.

6.4 Impact de la largeur de décomposition

Comme nous l'avons précisé dans l'introduction, les problèmes réels présentent souvent une structure particulière, notamment des sous-structures peu connectées entre elles (fig. 6). Ces structures reflètent des propriétés topologiques induites par la décomposition arborescente et se caractérisent par trois mesures : la largeur de la décomposition, la taille des séparateurs, et le degré des *clusters*. La largeur de décomposition donne une bonne indication sur la taille des

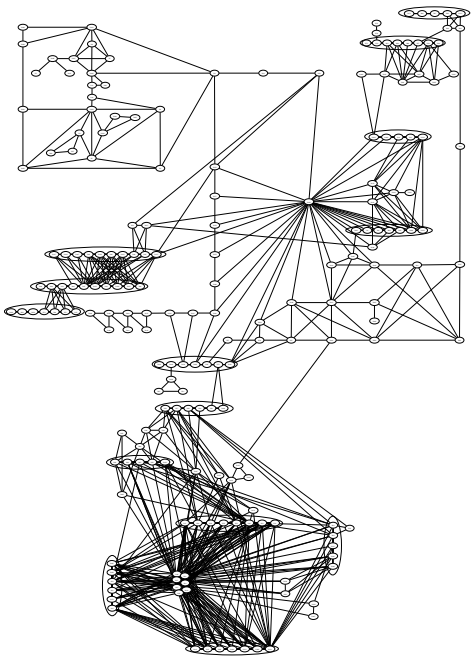


FIGURE 6 – Le graphe de contrainte de l'instance Scen07. Les variables contenues dans une ellipse forment une clique.

Instance	n	w^-	mc	$\frac{w^-}{n}$	$\frac{mc}{n}$
Scen06	100	11	10	0,11	0,10
Scen07	200	18	10	0,09	0,05
Scen08	458	18	10	0,03	0,02
Graph05	100	28	8	0,28	0,08
Graph06	200	54	8	0,27	0,04
Graph11	340	102	7	0,3	0,02
Graph13	458	136	6	0,29	0,01

FIGURE 7 – Plus petite largeur de décomposition, taille et ratios pour les instances RLFAP et GRAPH.

sous-problèmes, tandis que les deux autres mesures permettent d'évaluer la connexité entre *clusters*. En effet, plus ces valeurs sont basses, moins les *clusters* sont connectés. Comme il est indiqué dans [4], le calcul d'une décomposition arborescente optimisant ces trois mesures est une question ouverte. Nous avons sélectionné l'heuristique MCS [20] car celle-ci présente un bon compromis entre la qualité de la décomposition et le temps nécessaire pour son calcul [8].

Pour chaque instance *inst* RLFAP, SPOT5 et GRAPH, nous avons calculé toutes les décompositions fournies par MCS. Pour une instance *inst*, soit $w^-(inst)$ la plus petite largeur de décomposition obtenue par MCS. Le tableau 7 indique pour chaque instance : la valeur de w^-, la taille de la plus grande clique dans le graphe

de contraintes (mc), la taille du problème (n), et les ratios ($\frac{w^-}{n}$) et ($\frac{mc}{n}$) pour les instances RLFAP et GRAPH. Les résultats pour les instances SPOT5 sont très similaires à ceux des instances RLFAP, ils ne sont donc pas reportés ici. ($mc-1$) est un minorant de la largeur de décomposition. Elle donne une bonne indication sur la qualité de la décomposition obtenue.

Sur les instances RLFAP (fig. 7), la plus petite largeur w^- augmente avec la taille du problème (n), tandis que le ratio ($\frac{w^-}{n}$) diminue. Ce qui signifie que pour les plus grandes instances, les *clusters* sont plus dispersés. Au contraire, pour les instances GRAPH, les valeurs de w^- ainsi que leurs ratios sont très élevés comparés aux instances RLFAP, particulièrement pour les instances Graph11 et Graph13. Ce tableau montre aussi de grands écarts (> 100) entre les minorants et les majorants de la largeur de décomposition sur les grandes instances GRAPH, alors que cet écart est au plus de 8 pour les instances RLFAP. Les mauvais résultats sur les instances Graph11 et Graph13 montrent l'impossibilité pour l'heuristique MCS d'exhiber précisément une structure pertinente pour ces problèmes. Cette différence notable entre les instances les plus grandes des problèmes RLFAP et GRAPH semble provenir de l'origine de ces deux jeux de test : les instances RLFAP sont issues de problèmes réels, tandis que les instances GRAPH sont générées aléatoirement.

Comme nous l'avons montré dans les sections 6.2 et 6.3, DGVNS et SDGVNS surpassent VNS/LDS+CP et ID-Walk sur les instances ayant de faibles valeurs de w^- (instances RLFAP et SPOT5). Ceci se confirme sur les instances Graph05 et Graph06 en terme de taux de réussite et de qualité de solution (fig. 8) mais aussi au niveau des profils de performance (fig. 12).

Cependant, sur les instances Graph11 et Graph13, VNS/LDS+CP montre de meilleures performances que DGVNS et SDGVNS (fig. 8). Cela s'explique par le fait que les *clusters* sont de très grande taille et fortement connexes. Ainsi, l'impact de notre méthode n'est pas aussi fort car les *clusters* se recouvrent fortement et sont donc très semblables. On peut néanmoins remarquer que la comparaison des profils de performance (fig. 12) montre que le comportement de SDGVNS est bien meilleur au niveau de l'évolution de la qualité de la solution au fil du temps que celui de VNS/LDS+CP. Des travaux futurs devront être menés afin de tirer des conclusions plus précises.

Enfin, la fig. 9 décrit l'impact du seuil λ sur la TDTD pour les instances GRAPH. Les meilleurs résultats sont obtenus avec des valeurs de λ élevées $(0,7)$. En effet, les *clusters* étant relativement denses, un grand nombre de contraintes doivent être retirées pour obtenir une décomposition pertinente.

Instance	Méthode	Succ.	Temps	Moy.
Graph05	DGVNS	50/50	10	221
$n = 100$	SDGVNS-0,7	**50/50**	**8**	**221**
$e = 1034$	VNS/LDS+CP	50/50	17	221
$s* = 221$	ID-Walk	0/50	-	10584 (2391)
Graph06	DGVNS	50/50	367	4123
$n = 200$	SDGVNS-0,7	50/50	261	4123
$e = 1970$	VNS/LDS+CP	**50/50**	**218**	4123
$s* = 4123$	ID-Walk	0/50	-	18949 (13001)
Graph11	DGVNS	8/50	3046	4234
$n = 340$	SDGVNS-0,5	12/50	2818	3268
$e = 3417$	VNS/LDS+CP	**44/50**	2403	3090
$s* = 3080$	ID-Walk	0/50	-	41604 (27894)
Graph13	DGVNS	0/50	-	22489 (18639)
$n = 458$	SDGVNS-0,7	0/50	-	11449 (10268)
$e = 4915$	VNS/LDS+CP	**3/50**	3477	14522
$s* = 10110$	ID-Walk	0/50	-	58590 (47201)

FIGURE 8 – Comparaison de DGVNS, VNS/LDS+CP et ID-Walk sur les instances GRAPH.

Instance	λ	Succ.	Temps	Moy.
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	4123
	0,6	49 / 50	259	4123
	0,7	**50 / 50**	**261**	**4123**
Graph11	0,5	**12 / 50**	**2818**	**3643**
	0,6	**7 / 50**	**2446**	**3123**
	0,7	5 / 50	1417	3223
Graph13	0,5	0 / 50	-	24637 (18833)
	0,6	0 / 50	-	19340 (14318)
	0,7	**0 / 50**	**-**	**11449 (10268)**

FIGURE 9 – Impact du seuil λ sur la TDTD.

7 Conclusion

Nous avons montré que la *décomposition arborescente* permet de guider efficacement l'exploration de l'espace de recherche. De plus, nous avons proposé la *tightness dependent tree decomposition* qui permet de prendre en compte à la fois la structure du problème et la dureté des contraintes. Notre méthode d'intensification/diversification est générique et peut être appliqué à d'autre méthodes de recherche locale telle que LNS ou PGLNS. Enfin, nous avons montré expérimentalement que l'exploitation des décompositions arborescentes permet de dépasser les performances de VNS/LDS+CP et d'ID-Walk.

Nous travaillons actuellement sur trois pistes : tirer parti des séparateurs et des variables propres des *clusters* pour orienter la recherche, paralléliser l'exploration des *clusters*, et étudier l'apport de la méthode *min-fill* pour améliorer la qualité de la décomposition. Il serait intéressant d'étudier la pertinence de la prise en compte des coûts, comme proposé dans [9], pour guider la décomposition.

Ce travail a été soutenu par l'Agence Nationale de la Recherche, référence ANR-10-BLA-0214.

Références

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

[2] E. Bensana, M. Lemaître, and G. Verfaillie. Earth observation satellite management. *Constraints*, 4(3) :293–299, 1999.

[3] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints*, 4(1) :79–89, 1999.

[4] S. de Givry. Rapport de hdr, INRA UBIA Toulouse. 2011.

[5] M. Fontaine, S. Loudni, and P. Boizumault. Guiding vns with tree decomposition. In *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*, pages 505–512. IEEE, 2011.

[6] P. Hansen, N. Mladenovic, and D. Perez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4) :335–350, 2001.

[7] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *IJCAI'95*, pages 607–615, 1995.

[8] P. Jégou, S. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *CP*, pages 777–781, 2005.

[9] M. Kitching and F. Bacchus. Exploiting decomposition on constraint problems with high tree-width. In *IJCAI*, pages 525–531, 2009.

[10] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *IJCAI*, pages 239–244, 2003.

[11] S. Loudni and P. Boizumault. Combining VNS with constraint programming for solving anytime optimization problems. *EJOR*, 191 :705–735, 2008.

[12] R. Marinescu and R. Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artif. Intell.*, 173(16-17) :1457–1491, 2009.

[13] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers And Operations Research*, 24 :1097–1100, 1997.

[14] B. Neveu and G. Trombettoni. Incop : An open library for incomplete combinatorial optimization. In *Principles and Practice of Constraint Programming–CP 2003*, pages 909–913. Springer, 2003.

[15] B. Neveu, G. Trombettoni, and F. Glover. ID Walk : A candidate list strategy with a simple diversification device. In *CP'04, LNCS 3258*, pages 423–437, 2004.

[16] L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. *Principles and Practice of Constraint Programming–CP 2004*, pages 468–481, 2004.

[17] N. Robertson and P.D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3) :309–322, 1986.

[18] M. Sánchez, D. Allouche, S. de Givry, and T. Schiex. Russian doll search with tree decomposition. In *IJCAI*, pages 603–608, 2009.

[19] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP-98*.

[20] 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(3) :566–579, 1984.

[21] H. van Benthem. Graph : Generating radiolink frequency assignment problems heuristically, 1995.

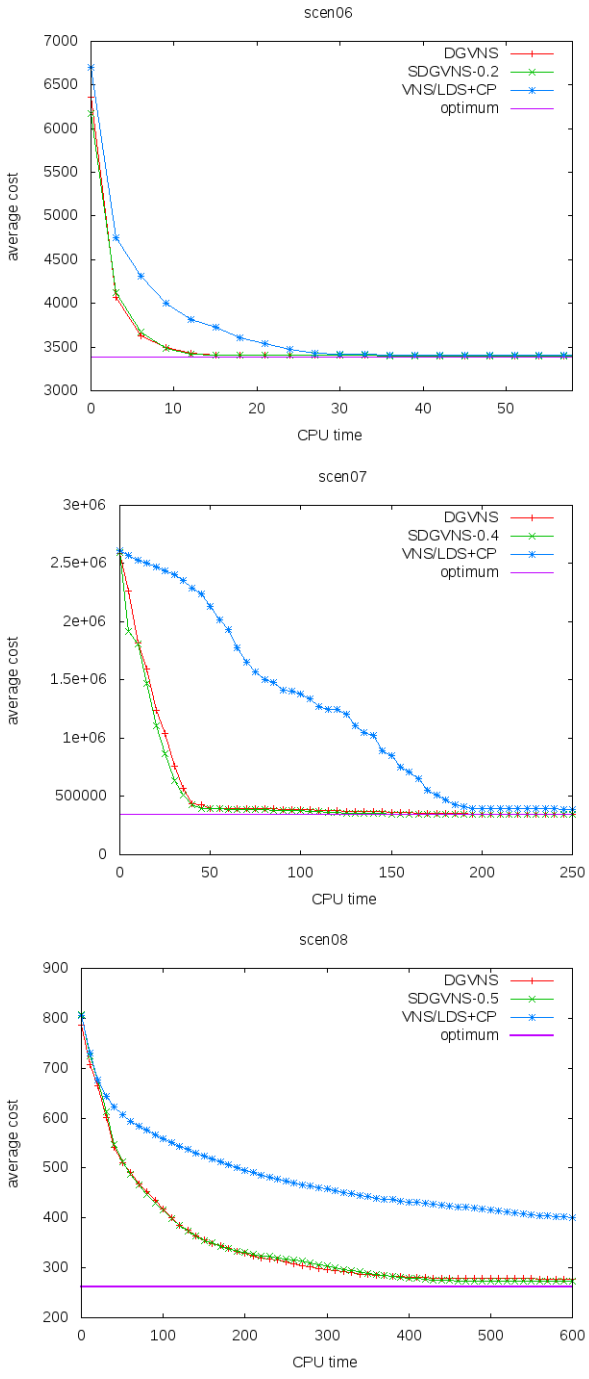
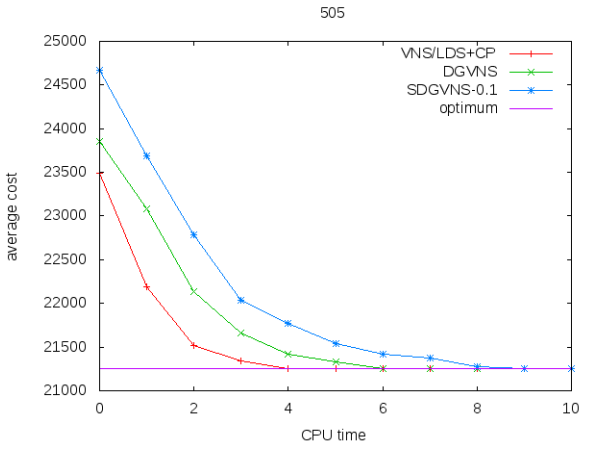
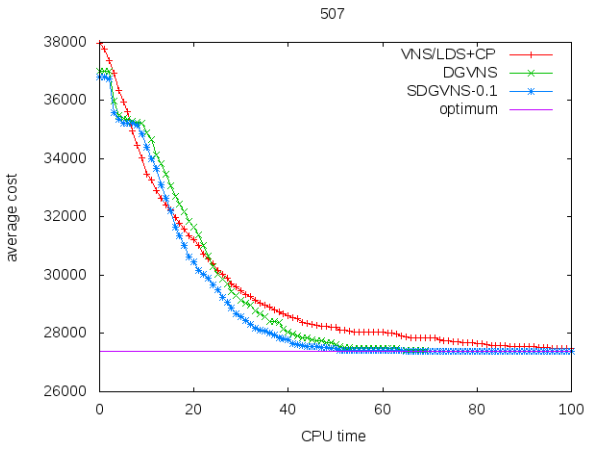
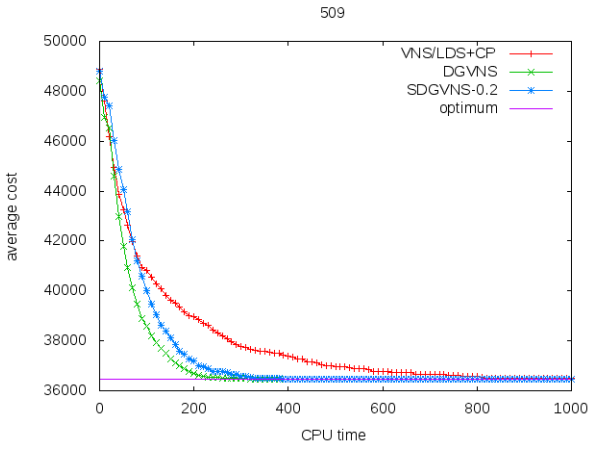
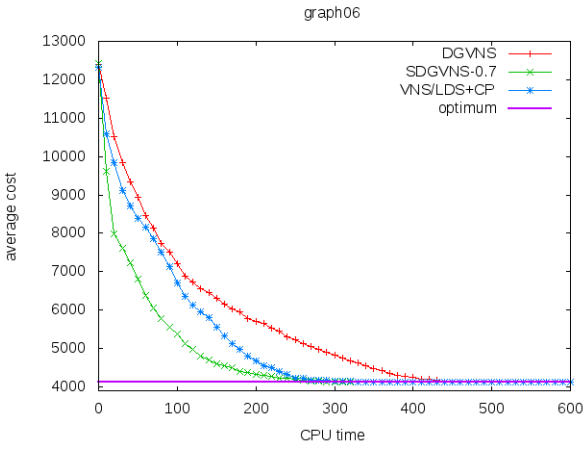


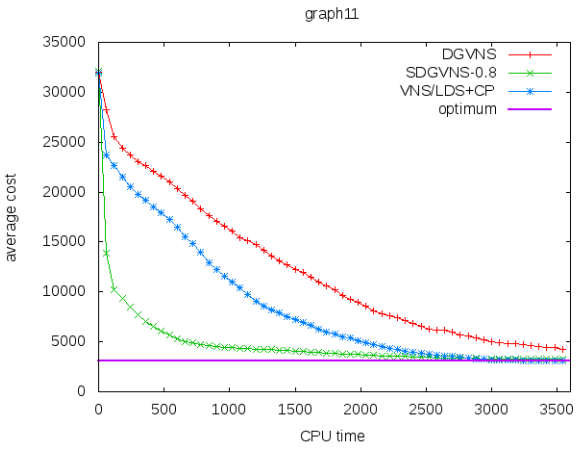
FIGURE 10 – Comparaison des profils de performance sur les instances `RLFAP`.











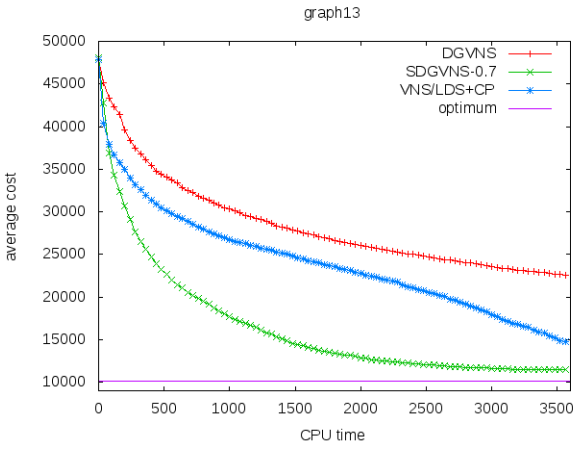


FIGURE 11 – Comparaison des profils de performance sur les instances SPOT5.

FIGURE 12 – Comparaison des profils de performance sur les instances GRAPH.