DGVNS guidée par les séparateurs

Samir Loudni, Mathieu Fontaine, Patrice Boizumault

DGVNS guidée par les séparateurs

Samir Loudni[1,2] **Mathieu Fontaine**[1,2] **Patrice Boizumault**[1,2]

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

Résumé

Dans nos précédents travaux, nous avons proposé la méthode DGVNS (Decomposition Guided VNS) permettant d'exploiter les clusters, issus de la décomposition arborescente du graphe de contraintes, pour guider l'exploration des voisinages dans les méthodes de type VNS. Dans cet article, nous proposons deux extensions de DGVNS permettant d'exploiter à la fois les clusters et les variables partagées entre ces clusters, appelées séparateurs. De telles variables jouent un charnière dans la résolution du problème. En effet, l'affectation de toutes les variables des séparateurs sépare le problème initial en plusieurs sous-problèmes qui peuvent ensuite être résolus indépendamment. Les expérimentations menées sur des instances aléatoires (GRAPH) et des instances réelles (RLFAP, SPOT5 et tagSNP) montrent la pertinence et l'intérêt de notre approche. Cet article est une version étendue de [19].

Abstract

This paper presents two extensions for DGVNS (Decomposition Guided VNS) method, that exploit both the *graph of clusters* and *separators* between these clusters, to efficiently guide the exploration of large neighborhoods in VNS. Experiments conducted on several difficult instances (RLFAP, SPOT5, GRAPHand tagSNP) show the relevance and the interest of our approach. This paper is an extended version of [19].

1 Introduction

La notion de décomposition arborescente, proposée par Robertson et Seymour [22], vise à découper un problème en sous-problèmes (*clusters*) constituant un graphe acyclique. Chaque cluster correspond à un sous-ensemble de variables fortement connectées. Chaque sous-problème, étant plus petit que le problème original, devient plus facile à résoudre. L'intérêt d'exploiter les propriétés structurelles d'un problème a été attestée dans divers domaines : pour déterminer la satisfiabilité en SAT [1, 13, 21], pour résoudre les CSP (CTE [6]), dans les réseaux bayesiens (AND/OR graph search [20]), en bases de données relationnelles [11, 12], pour les problèmes d'optimisation sous contraintes (BTD [25], Lc-BTD$^+$ [5], RDS-BTD [23], DB [17]). Toutes ces propositions exploitent la décomposition arborescente dans des méthodes de recherche complète.

Dans nos précédents travaux [8, 9], nous avons proposé la méthode DGVNS (Decomposition Guided VNS) permettant d'exploiter les clusters issus de la décomposition arborescente du graphe de contraintes, pour guider l'exploration des voisinages dans les méthodes de type VNS. Cependant, DGVNS ne permet pas de prendre en compte les variables partagées entre ces clusters, appelées *séparateurs*. De telles variables jouent un rôle charnière dans la résolution du problème. En effet, l'affectation de toutes les variables des séparateurs sépare le problème initial en plusieurs sous-problèmes qui peuvent ensuite être résolus indépendamment. De plus, les compatibilités entre affectations des différents sous-problèmes dépendent uniquement des séparateurs.

Dans cet article, nous proposons deux extensions de DGVNS, notées SGVNS (Separator-Guided VNS) et ISGVNS (Intensified SGVNS), qui exploitent à la fois le graphe de clusters et les séparateurs entre ces clusters, afin de mieux cibler les régions du problème qui vont être explorées durant la recherche. Notre idée est de tirer parti des affectations des variables des séparateurs pour guider DGVNS vers les clusters qui sont plus susceptibles de conduire à des améliorations importantes. Les expérimentations menées sur des instances aléatoires (GRAPH) et des instances réelles (RLFAP, SPOT5 et tagSNP) montrent la pertinence et l'intérêt de notre approche.

La section 2 présente le contexte de nos travaux. La section 3 détaille comment exploiter les sépara-

teurs dans DGVNS. Les sections 4 et 5 présentent les jeux de test et les résultats expérimentaux. Enfin, nous concluons et présentons des perspectives.

2 Contexte et définitions

2.1 Réseau de fonctions de coût

Un réseau de fonctions de coût (CFN) est un couple (X, W) où $X=\{x_1, \ldots, x_n\}$ est un ensemble de n variables et W est un ensemble de e fonctions de coût. Chaque variable $x_i \in X$ a un domaine fini D_i de valeurs qui peuvent lui être affectées. La taille du plus grand domaine est notée d. Une affectation de x_i à la valeur $a \in D_i$ est notée (x_i, a). Pour un sous-ensemble de variables $S \subseteq X$, on note D^S le produit cartésien des domaines des variables de S. Pour un n-uplet donné t, $t[S]$ représente la projection du n-uplet t sur l'ensemble de variables S. Une affectation *complète* $t=(a_1, ..., a_n)$ est une affectation de toutes les variables ; dans le cas contraire, on l'appelle affectation *partielle*. Une fonction de coût $w_S \in W$, de portée $S \subseteq X$, est une fonction $w_S : D^S \mapsto [0, k_\top]$ où $k_\top$ est un coût entier maximum (fini ou non) utilisé pour représenter les affectations interdites (exprimant des contraintes dures). Pour capturer fidèlement les contraintes dures, les coûts sont combinés par l'addition bornée $\oplus$, définie par $\alpha \oplus \beta = \min(k_\top, \alpha + \beta)$. Le problème consiste à trouver une affectation complète t de l'ensemble des variables minimisant la combinaison des fonctions de coût $\bigoplus_{w_S \in W} w_S(t[S])$.

2.2 Décomposition arborescente

Le graphe de contraintes d'un CFN est un graphe $G=(X,E)$ composé d'un sommet par variable et il existe une arête $\{u, v\} \in E$ ssi $\exists\, w_S \in W, u, v \in S$.

Définition 1 *Une décomposition arborescente [22] de $G=(X, E)$ 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, et $C_T = \{C_i \mid i \in I\}$ est une famille de sous-ensembles de X (appelés clusters) telle que :*

- $\cup_{i \in I} C_i = X$,
- $\forall\, (u, v) \in E$, $\exists\, C_i \in C_T$ *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* $C_i \cap C_k \subseteq C_j$.

Définition 2 *L'intersection entre deux clusters est appelée séparateur et notée $sep(C_i, C_j)$. Deux clusters C_i et C_j sont adjacents ssi $sep(C_i, C_j) \neq \emptyset$. Les variables appartenant à un et un seul cluster sont appelées variables propres.*

Définition 3 *Un graphe de clusters, pour une décomposition arborescente (C_T, T), est un graphe non-orienté $G_T = (C_T, E_T)$ dont les sommets sont les éléments de C_T et il existe une arête $(C_i, C_j) \in E_T$ entre les sommets C_i et C_j ssi $sep(C_i, C_j) \neq \emptyset$.*

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 [2]. 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) [24] qui constitue un bon compromis entre la largeur de la décomposition obtenue et le temps nécessaire à son calcul [16].

2.3 Decomposition Guided VNS

DGVNS (Decomposition Guided VNS) [8, 9] étend le principe de VNDS [14] (Variable Neighborhood Decomposition Search), en exploitant le graphe de clusters pour guider l'exploration de grands voisinages. Les voisinages sont obtenus en désaffectant une partie de la solution courante selon une heuristique de choix de variables. La reconstruction de la solution sur les variables désinstanciées est effectuée par une recherche arborescente partielle, LDS (*Limited discrepancy search*, [15]), aidée par la propagation des contraintes (CP) basée sur un calcul de minorants.

Définition 4 *Soit $G_T=(\mathcal{C}_T, E_T)$ le graphe de clusters associé à G. Soient $C_i \in \mathcal{C}_T$ un cluster de G_T et $k \in [1 \ldots n]$ la dimension du voisinage. La structure de voisinage $N_{k,i}$ désigne l'ensemble de toutes les combinaisons possibles de k variables parmi C_i.*

L'algorithme 1 présente le pseudo-code de DGVNS. Tout d'abord, il construit une décomposition arborescente de G (ligne 4), puis génère aléatoirement une solution initiale S (fonction genInitSol, ligne 6). Afin de favoriser les mouvements dans des régions fortement liées, DGVNS se base sur les structures de voisinages $N_{k,i}$ (cf. Définition 4). En effet, la notion de cluster permet d'exhiber ces régions, de part sa taille (plus petite que le problème initial), et de part la forte connexion entre les variables qu'il contient. Ainsi, l'ensemble de variables candidates C_s à désaffecter est

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

Algorithme 1: Pseudo-code de DGVNS

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

sélectionné à partir du cluster C_i. Si $(k > |C_i|)$, C_s est étendu aux variables des clusters C_j voisins de C_i afin de prendre en compte la topologie du graphe de clusters. Ce traitement est réalisé par la fonction `CompleteCluster`(C_i, k) (ligne 10). De plus, grâce à la forte connexion entre les variables de C_s, l'étape de reconstruction pourra bénéficier d'un plus fort filtrage et d'un meilleur calcul de minorants. Un sous ensemble X_{un} de k variables est sélectionné aléatoirement dans C_s parmi les variables en conflit par l'heuristique de voisinage `Hneighborhood` (ligne 11). Une affectation partielle $\mathcal{A}$ est générée à partir de la solution courante S en désaffectant les k variables de X_{un} (ligne 12). Ensuite, ces variables sont reconstruites (ligne 13) par une recherche arborescente partielle `LDS` [15], aidée par la propagation de contraintes (CP) (voir [18] pour plus de détails). La recherche s'arrête dès que la dimension maximale de voisinage k_{max} ou le *TimeOut* est atteint (ligne 9).

La procédure `NeighborhoodChangeDGVNS` contrôle les mécanismes d'intensification et de diversification de `DGVNS` (cf. Algorithme 1). Soit p le nombre total de clusters, *succ* une fonction de succession [2], et $N_{k,i}$ la structure de voisinage courante. Si `LDS+CP` ne trouve pas de meilleure solution S' dans le voisinage de S, `DGVNS` cherche des améliorations dans $N_{(k+1),succ(i)}$ (la structure de voisinage où $(k+1)$ variables de C_s seront désaffectées) (ligne 22). Tout d'abord, la diversification réalisée par le déplacement du cluster C_i au cluster $C_{succ(i)}$ permet de favoriser l'exploration de nouvelles parties de l'espace de recherche et de rechercher de meilleures solutions dans celles-ci. Ensuite, quand un optimum local est atteint dans le voisinage courant, l'augmentation de la dimension du voisinage k permet aussi de diversifier la recherche en explorant de plus grandes régions.

Quand une solution de meilleure qualité S' est trouvée par `LDS+CP` dans le voisinage $N_{k,i}$, S' devient la solution courante (ligne 19), k est réinitialisé à k_{init} et le prochain cluster est considéré (ligne 21). En effet, rester dans le même cluster rend plus difficile la découverte de nouvelles solutions : se déplacer sur un nouveau cluster permet la diversification de la recherche autour de la nouvelle solution S'.

3 VNS guidée par les séparateurs

Dans cette section, nous présentons les deux extensions de `DGVNS`, notées `SGVNS` et `ISGVNS`, qui exploitent à la fois le graphe de clusters et les séparateurs entre ces clusters, afin de mieux cibler les régions du problème qui vont être explorées durant la recherche.

3.1 Separator-Guided VNS (SGVNS)

`SGVNS` exploite l'évolution de la solution au cours de la recherche afin de guider l'effort de diversification de `DGVNS` vers des clusters contenant, dans leurs séparateurs, **au moins une variable réinstanciée** impliquée dans l'amélioration de la solution courante. L'algorithme 2 présente le pseudo code de `SGVNS`. Son schéma général est similaire à celui de `DGVNS` (cf. algorithme 1). La différence principale, réside dans l'utilisation d'une **liste de prospection**, notée $\mathcal{C}and$, contenant les clusters à visiter en priorité. Cette liste est mise à jour à chaque fois que `LDS+CP` trouve une solution de meilleure qualité.

Soit C_i le cluster courant et S' une solution de meilleure qualité trouvée par `LDS+CP` dans le voisinage de S. Soit V_c l'ensemble des variables qui ont été réinstanciées dans S pour obtenir S' (ligne 23) et C_w l'ensemble des clusters C_j de $\mathcal{C}and$ tel que $sep(C_i, C_j)$ partage au moins une variable avec V_c [3](ligne 24). Au début, $\mathcal{C}and$ est initialisé à l'ensemble des clusters $\mathcal{C}_T$ de la décomposition arborescente (ligne 6).

Contrairement à `DGVNS`, la diversification de `SGVNS` est réalisée en considérant successivement les clusters $C_j \in C_w$. En effet, à chaque fois que `LDS+CP` trouve une solution de meilleure qualité S' dans le voisinage de S (ligne 20), l'ensemble C_w est calculé (lignes

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

3. En effet, comme V_c ne contient que des variables de C_s (cf. ligne 10), si $C_j \cap V_c \neq \emptyset$, alors $sep(C_i, C_j) \cap V_c \neq \emptyset$.

Algorithme 2: Pseudo-code de SGVNS

```
1  Function SGVNS (X, W, k_init, k_max, δ_max);
2  begin
3      let G be the constraints graph of (X, W) ;
4      let (C_T, T) be a tree decomposition of G ;
5      S ← genSolInit();
6      Cand ← C_T;
7      k ← k_init ;
8      C_i ← remove-first(Cand) ;
9      while (k < k_max) ∧ (notTimeOut) do
10         C_s ← CompleteCluster(C_i, k) ;
11         X_un ← Hneighborhood(C_s, N_{k,i}, S) ;
12         A ← S\{(x_i, a) | x_i ∈ X_un} ;
13         S' ← LDS + CP(A, X_un, δ_max, f(S), S) ;
14         NeighbourhoodChangeSGVNS(S, S', k, i) ;
15     return S ;
16 Procedure NeighbourhoodChangeSGVNS(S, S', k, i);
17 begin
18     if Cand = ∅ then
19         Cand ← C_T;
20     if f(S') < f(S) then
21         S ← S' ;
22         k ← k_init;
23         V_c ← {x | S'[x] ≠ S[x]};
24         C_w ← {C_j ∈ Cand | C_j ∩ V_c ≠ ∅};
25         Cand ← Cand\C_w;
26         foreach C_j ∈ C_w do
27             insert-first(Cand, C_j);
28     else
29         k ← k + 1;
30     C_i ← remove-first(Cand);
```

Algorithme 3: Pseudo-code de ISGVNS

```
1  Function ISGVNS (X, W, k_init, k_max, δ_max);
2  begin
3      let G be the constraints graph of (X, W) ;
4      let (C_T, T) be a tree decomposition of G ;
5      S ← genSolInit() ;
6      k ← k_init ;
7      i ← 1 ;
8      P_List ← ∅;
9      while (k < k_max) ∧ (notTimeOut) do
10         C_s ← CompleteCluster(C_i, k) ;
11         X_un ← Hneighborhood(N_{k,i}, C_s, W, S) ;
12         A ← S\{(x_i, a) | x_i ∈ X_un} ;
13         S' ← LDS + CP(A, X_un, δ_max, f(S), S) ;
14         NeighbourhoodChangeISGVNS(S, S', k, i);
15     return S ;
16 Procedure ChangeNeighborISGVNS(S, S', k, i);
17 begin
18     if f(S') < f(S) then
19         S ← S';
20         k ← k_init;
21         V_c ← {x | S'[x] ≠ S[x]};
22         C_w ← {C_j | C_j ∩ V_c ≠ ∅, j = i + 1, ..., | C_T |} ;
23         foreach C_j ∈ C_w do
24             insert-Queue (P_List, C_j) ;
25         make tabu each element x ∈ V_c for L next
           iterations;
26     else
27         k ← k + 1 ;
28     if P_List ≠ ∅ then
29         i ← remove-first(P_List)
30     else
31         i ← succ(i)
```

23-24), les clusters de C_w sont déplacés en début de la liste $\mathcal{C}and$ (lignes 25-27) et le prochain cluster C_i à considérer est retiré de la liste (ligne 30). Comme pour DGVNS, S' devient la solution courante et k est réinitialisé à k_{init} (ligne 22). À l'inverse, si aucune amélioration de la solution courante n'a été trouvée, k est incrémenté de 1 (ligne 29) et SGVNS considère le cluster courant en tête de la liste $Cand$ (ligne 30).

Tout d'abord, la suppression du cluster courant C_i de $\mathcal{C}and$ permet de ne maintenir dans celle-ci que la liste des clusters non encore visités, assurant ainsi une plus large couverture de l'espace de recherche. Ainsi, chaque cluster de $\mathcal{C}_T$ sera visité au moins une fois [4]. De plus, privilégier les clusters contenant, dans leurs séparateurs, au moins une variable réinstanciée permet de guider la recherche vers des régions du problème pouvant mener à de plus larges améliorations de la solution courante. Enfin, cela permet de propager, au travers des séparateurs, les nouvelles affectations de S' vers les clusters $C_j \in C_w$, lors des prochaines itérations de SGVNS.

4. Un cluster pourra être considéré plusieurs fois dès lors que $Cand$ devient vide, car réinitialisée à $\mathcal{C}_T$ (ligne 19).

3.2 Intensified Separator-Guided VNS (ISGVNS)

ISGVNS vise à intensifier l'exploration "autour" des clusters contenant des variables réinstanciées. À cet effet, une *liste de propagation* P_{List} dotée d'une *liste taboue dynamique* T_{List} de taille L sont utilisées. La liste de propagation contient l'ensemble des clusters candidats à examiner après chaque amélioration de la solution courante. La liste taboue assure que les variables impliquées dans la sélection des clusters candidats (i.e. variables de V_c) ne seront pas reconsidérées dans $N_{k,i}$ par la fonction Hneighborhood, lors des L prochaines itérations de ISGVNS.

L'algorithme 3 présente le pseudo-code de ISGVNS. **L'intensification est réalisée en exploitant la liste de propagation**. Comme pour SGVNS, V_c dé-

signe l'ensemble de toutes les variables qui ont été réinstanciées (ligne 21) et C_w est l'ensemble des clusters ayant au moins, dans leurs séparateurs, une variable de V_c (ligne 22). Contrairement à SGVNS, chaque cluster $C_j \in C_w$ est ajouté à P_{List} (ligne 24) et chaque variable $x \in V_c$ est rendue taboue pour les L prochaines itérations (ligne 25). La valeur de L est fixée à la taille de P_{List} afin d'éviter de réaffecter les variables de V_c tant que tous les clusters $C_j \in C_w$ n'ont pas été considérés. Enfin, le prochain cluster à examiner correpond au premier élément de P_{List}, si celle-ci n'est pas vide (ligne 29). Dans le cas contraire, le successeur de C_i dans $\mathcal{C}_T$ est considéré (ligne 31).

ISGVNS permet de renforcer l'équilibre entre intensification et diversification. En effet, tant qu'aucune amélioration n'est effectuée, ISGVNS se comporte comme DGVNS, en considérant successivement tous les clusters C_i. Cependant, dès que LDS+CP améliore la solution courante, ISGVNS bascule vers un schéma d'intensification, jusqu'à ce que tous les clusters de P_{List} aient été examinés.

4 Jeux de test

Les expérimentations ont été menées sur différentes instances de quatre problèmes différents.

- 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) [4]. L'objectif est d'assigner un nombre limité de fréquences à un ensemble de liens radios entre des couples 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 [26] 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 [3]. Nous reportons les résultats sur six instances sans contraintes dures de capacité.

- Instances tagSNP : Un SNP (Single Nucleotide Polymorphism) -ou polymorphisme nucléotidique- est la variation d'une seule paire de nucléotides dans l'ADN de deux individus d'une même espèce ou dans une paire de chromosomes d'un même individu. Les SNP sont des marqueurs biologiques qui peuvent être utilisés pour la prédiction des risques de développement de certaines maladies [7, 10]. Le problème de sélection des tagSNP consiste à choisir un sous-ensemble de SNP, appelé tagSNP, qui permet de capturer le maximum d'information génétique. Ce problème est considéré comme très difficile du fait de sa proximité avec le problème de *set covering* (NP-difficile) [23]. Nous rapportons les résultats des expérimentations menées sur 13 instances issues des données du chromosome1 humain [5] avec r_0=0.5 (instances modélisées sous forme de CFN binaires [23] ayant jusqu'à $n = 1550$ variables, avec des domaines de taille d allant de 30 à 266 valeurs et jusqu'à $e = 250,000$ fonctions de coût). 7 instances sont de taille moyenne et 6 de grande taille.

5 Expérimentations

5.1 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 [18]. Les valeurs de k_{min} et k_{max} ont été respectivement fixées à 4 et n (nombre total de variables). Le $TimeOut$ à été fixé à 3 600 secondes pour les instances RLFAP et SPOT5. Pour les instances tagSNP de taille moyenne (resp. de grande taille), le $TimeOut$ a été fixé à 2 heures (resp. 4 heures). 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 [6].

Pour chaque instance et chaque méthode, nous reportons (i) le nombre de succès (l'optimum est atteint), (ii) le temps de calcul moyen pour atteindre l'optimum, ainsi que (iii) le coût moyen des solutions trouvées sur les 50 essais.

Dans cette section, nous reportons les résultats de nos expérimentations menées sur les instances RLFAP, GRAPH, SPOT5 et tagSNP. Tout d'abord, nous comparons SGVNS et ISGVNS avec DGVNS (cf. sections 5.2 et 5.3), puis nous comparons SGVNS avec ISGVNS (cf. section 5.4).

5.2 Comparaison entre SGVNS et DGVNS

5.2.1 Instances RLFAP

Sur les instances RLFAP (cf. tableau 1), SGVNS surclasse clairement DGVNS sur Scen07, et reste très comparable sur les deux autres instances. Pour la Scen07, SGVNS améliore le taux de succès de DGVNS de 16% (de

5. http://www.costfunction.org/benchmark
6. http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP

Instance	Méthode	Succ.	Temps	Moy.
Scen06, $n = 100$	SGVNS	50/50	111	3,389
$e = 1222$, $d = 44$	ISGVNS	**50/50**	**93**	**3,389**
S^* =3,389	DGVNS	50/50	112	3,389
Scen07, $n = 200$	SGVNS	48/50	652	343,802
$e = 2,665$, $d = 44$	ISGVNS	**49/50**	**806**	**343,794**
S^* =343,592	DGVNS	40/50	317	345,614
scen08, $n = 458$	SGVNS	2/50	532	281
$e = 5,286$, $d = 44$	ISGVNS	**4/50**	**1,408**	**276**
S^* =262	DGVNS	3/50	1,811	275
Graph05, $n = 100$	SGVNS	50/50	19	221
$e = 1,034$, $d = 44$	ISGVNS	50/50	16	221
S^* =221	DGVNS	**50/50**	**10**	**221**
Graph06, $n = 200$	SGVNS	50/50	292	4,123
$e = 1,970$, $d = 44$	ISGVNS	**50/50**	**240**	**4,123**
S^* =4,123	DGVNS	50/50	367	4,123
Graph11, $n = 340$	SGVNS	27/50	3,026	4,238
$e = 3417$, $d = 44$	ISGVNS	**39/50**	**2,762**	**3,349**
S^* =3,080	DGVNS	8/50	3,046	4,234
Graph13, $n = 458$	SGVNS	**6/50**	**3,260**	**14,707**
$e = 4,915$, $d = 44$	ISGVNS	1/50	3,196	17,085
S^* =10,110	DGVNS	0/50	-	22,489 (18,639)

TABLE 1 – Comparaison entre SGVNS, ISGVNS et DGVNS sur les instances RLFAP et GRAPH.

80% à 96%), et réduit la déviation moyenne de l'optimum de (0.58% à 0.06%).

5.2.2 Instances SPOT5

Sur les instances SPOT5 (cf. tableau 2), SGVNS se montre très efficace comparé à DGVNS, particulièrement sur les instances de grande taille (#412, #414, #507 et #509), pour lesquelles l'amélioration est très significative. Pour les instances #507 et #509, SGVNS procure un gain en termes de taux de succès de 12% (de 66% à 78%) et 16% (de 80% à 96%) respectivement. Ces gains atteignent respectivement 24% et 20% pour les instances #412 et #414. Sur les autres instances, les deux méthodes obtiennent des résultats similaires.

5.2.3 Instances GRAPH

La tendance se confirme sur les instances GRAPH (cf. tableau 1). Pour les instances réputées faciles (Graph05 et Graph06), les deux méthodes obtiennent les mêmes taux de succès (i.e. 100%). Toutefois, SGVNS est plus rapide que DGVNS sur Graph06 (gain en temps de calcul d'environ 20%). Pour les instances difficiles, les gains obtenus par SGVNS sont plus importants. Pour l'instance Graph11, SGVNS obtient 3 fois plus de succès que DGVNS (gain de 38% en termes de taux de succès). Pour l'instance Graph13, SGVNS atteint l'optimum (6 essais avec succès), surclassant nettement DGVNS. Ces résultats démontrent clairement la pertinence d'exploiter les séparateurs pour guider l'exploration des voisinages.

Instance	Méthode	Succ.	Temps	Moy.
#408, $n = 200$	SGVNS	**50/50**	**132**	**6,228**
$e = 2,232$, $d = 4$	ISGVNS	50/50	140	6,228
S^* =6,228	DGVNS	49/50	117	6,228
#412, $n = 300$	SGVNS	48/50	351	32,381
$e = 4,348$,$d = 4$	ISGVNS	**49/50**	**434**	**32,381**
S^* =32,381	DGVNS	36/50	84	32,381
#414, $n = 364$	SGVNS	**48/50**	**630**	**38,478**
$e = 2,242$, $d = 4$	ISGVNS	46/50	524	38,478
S^* =38,478	DGVNS	38/50	554	38,478
#505, $n = 240$	SGVNS	50/50	99	21,253
$e = 2,242$, $d = 4$	ISGVNS	50/50	67	21,253
S^* =21,253	DGVNS	**50/50**	**63**	**21,253**
#507, $n = 311$	SGVNS	39/50	576	27,390
$e = 5,732$, $d = 4$	ISGVNS	**45/50**	**694**	**27,390**
S^* =27,390	DGVNS	33/50	71	27,390
#509, $n = 348$	SGVNS	**48/50**	**412**	**36,446**
$e = 8,624$, $d = 4$	ISGVNS	46/50	499	36,446
S^* =36,446	DGVNS	40/50	265	36,446

TABLE 2 – Comparaison entre SGVNS, ISGVNS et DGVNS sur les instances SPOT55.

5.2.4 Instances tagSNP

Pour les instances de taille moyenne (cf. tableau 3), SGVNS et DGVNS atteignent l'optimum sur chacun des 50 essais. Cependant, SGVNS est plus rapide sur trois instances (#4449, #9313 et #16421), plus lent sur trois autres instances (#3792, #8956 et #15757), et très similaire à DGVNS sur l'instance #16706. Pour l'instance #16421, SGVNS améliore de 24% le temps de calcul de DGVNS. Le meilleur résultat est obtenu sur l'instance #9313, où SGVNS améliore le temps de calcul de DGVNS de 36%. Les moins bons résultats sont obtenus sur l'instance #8956, pour laquelle DGVNS est plus rapide.

Instance	Méthode	Succ.	Temps	Moy.
#3792, $n = 528$	DGVNS	50/50	954	6,359,805
$e = 12,084$, $d = 59$	SGVNS	50/50	1,033	6,359,805
$S^* = 6,359,805$	ISGVNS	**50/50**	**853**	**6,359,805**
#4449, $n = 464$	DGVNS	50/50	665	5,094,256
$e = 12,540$, $d = 64$	SGVNS	**50/50**	**661**	**5,094,256**
$S^* = 5,094,256$	ISGVNS	50/50	675	5,094,256
#8956, $n = 486$	DGVNS	50/50	4,911	6,660,308
$e = 20,832$, $d = 106$	SGVNS	50/50	5,483	6,660,308
$S^* = 6,660,308$	ISGVNS	**50/50**	**4,118**	**6,660,309**
#9319, $n = 562$	DGVNS	50/50	788	6,477,229
$e = 14,811$, $d = 58$	SGVNS	**50/50**	**500**	**6,477,229**
$S^* = 6,477,229$	ISGVNS	50/50	672	6,477,229
#15757, $n = 342$	DGVNS	**50/50**	**60**	**2,278,611**
$e = 5,091$, $d = 47$	SGVNS	50/50	104	2,278,611
$S^* = 2,278,611$	ISGVNS	50/50	80	2,278,611
#16421, $n = 404$	DGVNS	50/50	2,673	3,436,849
$e = 12,138$, $d = 75$	SGVNS	**50/50**	**2,025**	**3,436,849**
$S^* = 3,436,849$	ISGVNS	50/50	5,863	3,436,849
#16706, $n = 438$	DGVNS	50/50	153	2,632,310
$e = 6,321$, $d = 30$	SGVNS	50/50	159	2,632,310
$S^* = 2,632,310$	ISGVNS	**50/50**	**89**	**2,632,310**

TABLE 3 – Comparison entre SGVNS, ISGVNS et DGVNS sur les instances tagSNP de taille moyenne.

Instance	Méthode	Succ.	Temps	Moy.
#10442, $n = 908$	DGVNS	**50/50**	**4,552**	**21,591,913**
$e = 28,554$, $d = 76$	SGVNS	50/50	7,153	21,591,913
$S^* = 21,591,913$	ISGVNS	50/50	7,291	21,591,913
#14226, $n = 1,058$	DGVNS	46/50	7,606	25,688,751
$e = 36,801$, $d = 95$	SGVNS	40/50	7,646	25,805,242
$S^* = 25,665,437$	ISGVNS	**50/50**	**9,596**	**25,665,437**
#17034, $n = 1142$	DGVNS	**41/50**	**8,900**	**38,563,232**
$e = 47,967$,$d = 123$	SGVNS	33/50	10,212	38,869,514
$S^* = 38,318,224$	ISGVNS	36/50	10,579	38,746,957
#12976, $n = 920$	DGVNS	0/50	-	24,685,331 (24,483,048)
$e = 44,823$, $d = 128$	SGVNS	0/50	-	25,059,787 (24,483,414)
$S^* = 21,604,644$	ISGVNS	0/50	-	**24,627,760 (24,482,897)**
#13931, $n = 820$	DGVNS	0/50	-	**21,107,259 (21,106,760)**
$e = 38,775$, $d = 145$	SGVNS	0/50	-	21,107,723 (21,106,833)
$S^* = 21,106,731$	ISGVNS	0/50	-	21,331,064 (**21,106,770**)
#14007, $n = 1,554$	DGVNS	0/50	-	57,808,373 (53,753,706)
$e = 54,753$, $d = 100$	SGVNS	0/50	-	58,155,109 (**53,753,161**)
$S^* = 50,290,563$	ISGVNS	0/50	-	**57,565,916** (55,486,138)

TABLE 4 – Comparison entre SGVNS, ISGVNS et DGVNS sur les instances tagSNP de grande taille.

Pour les instances de grande taille (cf. tableau 4), SGVNS se montre moins compétitive en termes de taux de succès et de temps de calcul, en particulier sur les trois instances #10442, #14226 et #17034, où DGVNS est clairement le meilleur. Pour l'instance #17034, DGVNS améliore le taux de succès de SGVNS de 16% (de 66% à 82%), la déviation moyenne de l'optimum est réduite (de 1.43% à 0.63%) et DGVNS est 1.2 fois plus rapide que SGVNS. Pour l'instance #14226, le gain en taux de succès atteint 12% (de 80% à 92%) et DGVNS réduit la déviation moyenne de l'optimum (de 0.54% à 0.09%). Pour les autres instances, DGVNS trouve des solutions de meilleure qualité en moyenne que SGVNS.

5.3 Comparaison entre ISGVNS et DGVNS

5.3.1 Instances RLFAP

ISGVNS domine clairement DGVNS, notamment sur les deux instances réputées difficiles Scen07 et Scen08. Pour la Scen07, ISGVNS améliore le taux de succès de DGVNS de 18% (de 80% à 98%). Pour la Scen08, ISGVNS obtient un succès de plus que DGVNS et réduit le temps de calcul d'environ 22%. Pour la Scen06, ISGVNS est plus rapide que DGVNS.

5.3.2 Instances SPOT5

ISGVNS est meilleur que DGVNS sur la plupart des instances (cf. tableau 2). Pour les instances de grande

taille (#412, #414, #507 et #509), ISGVNS obtient un taux de succès moyen de 93% contre 73.5% pour DGVNS. Pour les autres instances (#408 et #505), les deux méthodes obtiennent les mêmes taux de succès, mais DGVNS est plus rapide.

5.3.3 Instances GRAPH

Une fois de plus, ISGVNS surclasse nettement DGVNS, particulièrement sur les deux instances Graph11 et Graph13 (cf. tableau 1). Pour l'instance Graph11, ISGVNS obtient 5 fois plus de succès que DGVNS. Pour l'instance Graph13, ISGVNS atteint une fois l'optimum et obtient des solutions de meilleure qualité en moyenne. Pour les instances faciles, ISGVNS est plus rapide que DGVNS.

5.3.4 Instances tagSNP

Pour les instances de taille moyenne (cf. tableau 3), ISGVNS est plus rapide que DGVNS sur quatre instances (#3792, #8956, #9313 et #16706), plus lent sur deux instances (#15757 et #16421) et similaire à DGVNS sur l'instance #4449. Pour les instances #8956 et #9313, ISGVNS améliore le temps de calcul de DGVNS d'environ 16% en moyenne. Les meilleurs résultats sont obtenus sur l'instance #16706, pour laquelle le gain est d'environ 41%. À l'inverse, pour l'instance #16421, ISGVNS obtient sa plus mauvaise performance : DGVNS est deux fois plus rapide.

Pour les instances de grande taille, ISGVNS est plus efficace que DGVNS sur l'instance #14226 et est moins compétitive sur l'instance #17034. Les deux méthodes obtiennent les mêmes taux de succès sur l'instance #10442, mais DGVNS est plus rapide. Pour les autres instances, si on compare la qualité moyenne des solutions obtenues, ISGVNS est meilleure sur 2 instances (#12976 et #14007) et reste comparable sur l'instance #13931. Par ailleurs, ISGVNS obtient les meilleurs coûts sur 2 instances parmi 3.

5.4 Comparaison entre SGVNS et ISGVNS

5.4.1 Instances RLFAP

ISGVNS devance SGVNS sur toutes les instances (cf. tableau 1). Pour la Scen06, ISGVNS est plus rapide en moyenne. Pour la Scen07, ISGVNS obtient un succès de plus que SGVNS. Pour la Scen08, ISGVNS obtient **deux fois plus de succès** que SGVNS et des solutions de meilleure qualité en moyenne.

5.4.2 Instances SPOT5

Sur les instances SPOT5, aucune des deux méthodes ne domine l'autre (cf. tableau 2). En effet, SGVNS obtient 2 succès de plus que ISGVNS sur deux instances (#414 et #509) et est plus rapide sur l'instance #408, tandis que ISGVNS devance SGVNS en termes de taux de succès sur deux instances (#412 et #507) et est plus rapide sur l'instance #505.

5.4.3 Instances GRAPH

Sur les instances faciles (Graph05 et Graph06), ISGVNS est légèrement plus rapide que SGVNS (cf. tableau 1). En revanche, sur les deux instances réputées difficiles, aucune méthode ne surclasse nettement l'autre. Pour l'instance Graph11, ISGVNS obtient 24% de succès en plus par rapport à SGVNS et réduit la déviation moyenne de l'optimum de (37% à 8%). Pour l'instance Graph13, SGVNS multiplie le nombre de succès de ISGVNS par 6 et réduit la déviation moyenne de l'optimum de (69% à 45%).

5.4.4 Instances tagSNP

Pour les instances de taille moyenne, SGVNS et ISGVNS obtiennent les mêmes taux de succès. Toutefois, ISGVNS est plus rapide que SGVNS sur quatre instances (#3792, #8956, #15757 et #16706), et est moins rapide sur trois autres instances. Pour les instances de grande taille, ISGVNS surclasse nettement SGVNS. ISGVNS améliore le taux de succès moyen de SGVNS d'environ 13% sur deux instances (#14226 et #17034) et obtient des solutions de meilleure qualité sur les instances #12976 et #14007, tandis que SGVNS est plus rapide sur l'instance #14226 et obtient des solutions de meilleure qualité sur l'instance #13931.

5.5 Bilan sur les apports de SGVNS et ISGVNS

Nous avons montré expérimentalement l'intérêt d'exploiter les séparateurs pour mieux guider DGVNS vers des voisinages susceptibles de conduire à des améliorations plus importantes. Sur la plupart des instances testées, SGVNS et ISGVNS surclassent nettement DGVNS, excepté sur les instances tagSNP de grande taille, où SGVNS est moins efficace que DGVNS. SGVNS reste toutefois plus compétitive sur les instances de taille moyenne.

De ces expérimentations, il ressort également que ISGVNS donne de meilleurs résultats en moyenne que SGVNS. ISGVNS surclasse SGVNS sur 17 instances parmi 26, alors que SGVNS obtient de meilleurs résultats sur 9 instances. Ces résultats montrent l'importance de la liste de propagation pour réaliser un meilleur compromis entre intensification et diversification.

6 Conclusions et perspectives

Nous avons proposé deux extensions de la méthode DGVNS qui tirent parti à la fois du graphe de clusters et des séparateurs entre ces clusters, pour guider efficacement l'exploration des grands voisinages dans DGVNS. Les expérimentations menées sur plusieurs instances difficiles ont montré que SGVNS et ISGVNS sont nettement plus performants que DGVNS, et que ISGVNS est très efficace par rapport à SGVNS. Nous travaillons actuellement sur une version parallèle pour l'exploration des clusters.

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

Références

[1] E. Amir and S. Mcllraith. Solving satisfiability using decomposition and the most constrained subproblem. *Electronic Notes in Discrete Mathematics*, 9 :329–343, 2001.

[2] 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.

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

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

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

[6] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artif. Intell.*, 38(3) :353–366, 1989.

[7] C.S. Carlson et al. Selecting a maximally informative set of single-nucleotide polymorphisms for association analyses using linkage disequilibrium. *Am. J. of Hum. Genetics*, 74(1) :106–120, 2004.

[8] M. Fontaine, S. Loudni, and P. Boizumault. Guiding VNS with tree decomposition. In *ICTAI*, pages 505–512, 2011.

[9] M. Fontaine, S. Loudni, and P. Boizumault. Exploiting tree decomposition for guiding neighborhoods exploration for VNS. *RAIRO Operations Research*, 47(2) :91–123, 2013.

[10] S. Gopalakrishnan and Z. S. Qin. Tagsnp selection based on pairwise ld criteria and power analysis in association studies. In *Pacific Symposium on Biocomputing*, pages 511–522, 2006.

[11] G. Gottlob, S. T. Lee, and G. Valiant. Size and treewidth bounds for conjunctive queries. In Jan Paredaens and Jianwen Su, editors, *PODS*, pages 45–54. ACM, 2009.

[12] G. Gottlob, R. Pichler, and F. Wei. Tractable database design through bounded treewidth. In Stijn Vansummeren, editor, *PODS*, pages 124–133. ACM, 2006.

[13] D. Habet, L. Paris, and C. Terrioux. A tree decomposition based approach to solve structured SAT instances. In *ICTAI*, pages 115–122, 2009.

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

[15] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *IJCAI (1)*, pages 607–615, 1995.

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

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

[18] S. Loudni and P. Boizumault. Combining VNS with constraint programming for solving anytime optimization problems. *EJOR*, 191(3) :705–735, 2008.

[19] S. Loudni, M. Fontaine, and P. Boizumault. Exploiting separators for guiding VNS. *Electronic Notes in Discrete Mathematics*, 39 :265 – 272, 2012.

[20] 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.

[21] I. Rish and R. Dechter. Resolution versus search : Two strategies for SAT. *J. Autom. Reasoning*, 24(1/2) :225–275, 2000.

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

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

[24] 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.

[25] C. Terrioux and P. Jégou. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146(1) :43–75, 2003.

[26] H. van Benthem. GRAPH : Generating radiolink frequency assignment problems heuristically, 1995.