Exploiter les séparateurs : une approche hybride combinant recherche locale et recherche arborescente

Jean-Philippe Metivier, Mathieu Fontaine, Samir Loudni

Exploiter les séparateurs : une approche hybride combinant recherche locale et arborescente *

Jean-Philippe Métivier **Mathieu Fontaine** **Samir Loudni**

Université de Caen Basse-Normandie – GREYC – (CNRS UMR 6072)
Campus II – Côte de Nacre – Boulevard du Maréchal Juin 14000 Caen – France
prenom.nom@unicaen.fr

Résumé

La décomposition d'un graphe de contraintes permet de découper un problème en clusters (sous-ensembles de variables). Les variables partagées par les clusters sont appelées variables séparateurs. Celles-ci jouent un rôle majeur dans la phase de résolution. En effet, si toutes les variables des séparateurs sont instanciées alors chaque cluster peut être traité de façon indépendante. Dans ce papier, nous montrons comment exploiter ce principe dans les méthodes de recherche locale à voisinages larges. Notre approche procède en deux temps. Tout d'abord, on sélectionne un sous-ensemble de variables séparateurs, puis on les réaffectent de façon gloutonne. Ensuite, on reconstruit l'ensemble des clusters contenant ces variables séparateurs à l'aide de méthodes arborescentes. Les expérimentations menées sur des instances réelles (CELAR et SPOT5) montrent la pertinence et l'efficacité de notre approche.

1 Introduction

Un grand nombre de problèmes réels, tels que les problèmes d'affectations de fréquence [2] ou la planification de satellites d'observation [1], sont de très grande taille et exhibent un graphe de contraintes très structuré.

La décomposition, proposée par Robertson et Seymour [12], vise à découper un problème en sous-problèmes (clusters) constituant un graphe acyclique. Chaque cluster représente un ensemble de variables fortement connexes. Chaque sous-problème étant plus petit que le problème d'origine est plus facile à résoudre. Les variables partagées par plusieurs clusters sont appelées séparateurs. Ces séparateurs jouent un rôle charnière dans la résolution des sous-problèmes associés. En effet, toute instanciation d'un séparateur induit plusieurs sous-problèmes indépendants. Cette propriété est au coeur des méthodes complètes exploitant une décomposition arborescente comme `RDS-BTD` [13] ou `AND/OR graph search` [8].

Dans les méthodes de recherche locale qui utilisent de grands voisinages tels que *Large Neighborhood Search* (`LNS`) [14], *Propagation Guided Large Neighborhood Search* (`PGLNS`) [11] ou encore *Variable Neighborhood Search* (`VNS`) [9], la définition de structures de voisinages 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 les séparateurs pour identifier des structures de voisinages.

Dans ce papier, nous montrons comment les séparateurs peuvent être utilisés afin de guider une recherche à grand voisinage comme `VNS`. Les expérimentations menées sur des instances réelles (CELAR et SPOT5) montrent que l'utilisation des séparateurs procure des performances bien meilleures comparé à `VNS/LDS+CP` [7] ou `ID-Walk` [10].

La section 2 présente le contexte de nos travaux. La section 3 montre comment exploiter les séparateurs dans une méthode à grands voisinages. Les sections 4 et 5 présentent les résultats expérimentaux. Enfin, nous concluons et présentons des perspectives.

2 Définition et notations

2.1 Réseau de fonctions de coûts

Un réseau de fonctions de coûts est une paire (X,W) où $X=\{1,..,n\}$ est un ensemble de n variables et de W est un ensemble de fonctions de coûts. Chaque variable

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

Algorithme 1 : VNS/LDS+CP

```
1  fonction VNS/LDS+CP(X,W,k_init,k_max)
2  début
3      S ← genInitSol();
4      k ← k_init;
5      tant que k ≤ k_max faire
6          X_unassign ← Hneighborhood(X,N_k,S);
7          A ← S \ X_unassign;
8          S' ← LDS(A);
9          si f(S') < f(S) alors
10             S ← S';
11             k ← k_init;
12         sinon k ← k + 1;
13     retourner S;
14 fin
```

$X_i \in X$ a un domaine fini D_i de valeurs qui peuvent lui être affectées. 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 t donné, $t[S]$ représente la projection de t sur l'ensemble de variables S. Une fonction de coûts $w_s \in W$, de portée $S \subseteq X$, est une fonction $w_s : D^s \rightarrow [0, k_\top]$ où $k_\top$ est un coût entier maximum représentant les affectations interdites. Les coûts sont combinés par l'addition bornée $\oplus$, définie par $\alpha \oplus \beta = \min(\alpha+\beta, k_\top)$.

Résoudre un réseau de fonctions de coûts consiste à trouver une affectation complète t de l'ensemble des variables minimisant la combinaison des fonctions de coûts $\bigoplus_{w_s \in W} w_s(t[S])$.

2.2 Décomposition

Un réseau de fonctions de coûts peut être vu comme un graphe $G=(X,W)$ composé d'un sommet par variable et d'une arête pour chaque fonction de coûts (reliant les variables de sa portée).

Définition 1 *Une* ***décomposition*** *d'un réseau de fonctions de coûts consiste à former un ensemble de clusters $C=\{C_1,..,C_m\}$ tel que :*
- $\bigcup_{i \in [1..m]} C_i = X$,
- $\forall (u,v) \in W$, $\exists C_i \in C$ *t.q.* $u \in C_i$ *et* $v \in C_i$.

Définition 2 *L'intersection de deux clusters est appelée* ***séparateur****. Sep désignera l'ensemble des variables séparateurs.*

Définition 3 *Les variables d'un cluster contenues dans aucun séparateur sont appelées* ***variables propres****. C^i_{pr} désignera l'ensemble des variables propres du cluster C_i.*

Pour des raisons pratiques, nous utiliserons ici une décomposition arborescente obtenue en utilisant l'heuristique MCS [15].

2.3 VNS/LDS+CP

VNS/LDS+CP [7] est une méthode de recherche locale qui exploite le principe de VNDS [4]. Les voisinages sont obtenus en désaffectant heuristiquement une partie de la solution courante. Ensuite, les variables désinstanciées sont réaffectées à l'aide d'une recherche arborescente partielle (LDS [5]) aidée par des mécanismes de propagation basée sur le calcul de minorants.

L'algorithme 1 présente le pseudo-code de VNS/LDS+CP. N_k désigne l'ensemble des sous-ensembles de k variables de X. Un sous-ensemble de k variables est sélectionné dans X, puis réaffecté avec LDS+CP. Si une solution de meilleure qualité S' est trouvée, k est réinitialisé à k_{init}. Sinon, k est incrémenté de 1. La recherche s'arrête quand elle a atteint k_{max} la taille maximale de voisinage.

3 Guider VNS grâce aux séparateurs

Dans cette section, nous présentons notre contribution : sep-VNS qui utilise les séparateurs pour définir de nouveaux de voisinages.

3.1 Choix des variables à désinstancier

Tout comme VNS/LDS+CP, les voisinages sont obtenus en désaffectant une partie de la solution courante. Toutefois, les variables à désinstancier sont gérés différemment afin de tirer parti des séparateurs.

Les variables à désinstancier sont sélectionnées en deux temps. Tout d'abord, k variables séparateurs sont sélectionnées aléatoirement et désinstanciées (algo. 2, ligne 6). Puis les clusters partageant ces k variables sont, à leur tour, désinstanciés (algo. 2, ligne 7).

Les k variables séparateurs désinstanciés permettent de décrire un sous-espace de recherche intéressant. En effet, ces variables constituent une sorte de « résumé » du problème : lorsqu'elles sont instanciées, les différents clusters sous-jacents peuvent être traités individuellement. De ce fait, il est nécessaire de désinstancier les clusters partageant les k variables sélectionnées car toute nouvelle affectation de ces k variables remettra en question la valuation des clusters associés. Notons que les variables appartenant aux clusters ne contenant pas de variables séparateurs désinstanciées sont conservés dans l'instanciation partielle (algo. 2, ligne 9).

3.2 Reconstruction de la solution partielle

La reconstruction est effectuée en deux temps. Les premières variables à réinstancier sont celles des séparateurs, puis les variables propres de chaque cluster désinstancié.

Algorithme 2 : Sep-VNS

```
1  fonction Sep-VNS(X,W,k_init,k_max,C,Sep)
2  début
3      S ← genInitSol() ;
4      k ← k_init ;
5      tant que k ≤ k_max faire
6          Sep_k ← select(k,Sep) ;
7          X_pr ← {X_i | X_i∈C^j_pr, Cj∩Sep_k≠∅ et j∈[1..m]} ;
8          X_unassign ← Sep_k ∪ X_pr ;
9          A ← S \ X_unassign ;
10         pour chaque X_i ∈ Sep_k faire
11             value ← mincost(D_{X_i}) ;
12             A ← A ∪ {X_i=value} ;
13         pour chaque C_j t.q. Cj ∩ Sep_k ≠ ∅ faire
14             A_j ← A[C_j ∩ Sep_k] ;
15             S_j ← DFBB(C_j,A_j) ;
16             A ← A ∪ S_j ;
17         S' ← A ;
18         si f(S') < f(S) alors
19             S ← S' ;
20             k ← k_init ;
21         sinon k ← k + 1 ;
22     retourner S ;
23 fin
```

Reconstruction des variables séparateurs (algo. 2, lignes 10 à 12) : Chaque variable séparateur désinstanciée va être réinstanciée (de façon gloutonne) à sa valeur engendrant le moins de violations. Afin d'éviter d'être bloqué dans un optimum local, une liste taboue dynamique [3] est utilisée pour diversifier les instanciations des variables séparateurs. Cette liste va interdire une instanciation donnée pour les $(0.1 \times \#conflict + rand(5))$ prochaines iterations.

Reconstruction des clusters désinstanciés (algo. 2, lignes 13 à 16) : La reconstruction des clusters désinstanciés ne commence qu'une fois toutes les variables séparateurs instanciées. Chaque cluster est réinstancié de façon individuelle (en ne considérant que les variables et contraintes de ce cluster) et cela à l'aide d'une recherche arborescente. Lorsque le cluster est suffisament petit (moins de 30 variables) DFBB est utilisé, sinon le cluster est reconstruit en utilisant un LDS.

3.3 Accelérer la reconstruction des clusters

La reconstruction des variables séparateurs peut nous conduire à explorer deux fois le même sous-espace de recherche. En effet, si pour un cluster donné, la reconstruction réinstancie les variables séparateurs à des valeurs déjà explorées précédemment, alors la phase de reconstruction des clusters va explorer le même espace de recherche. Afin d'éviter cela, nous avons mis en place un système de *cacher* [6]. Ces cachers enregistrent, pour chaque cluster et instanciation des séparateurs, la meilleure instanciation des variables propres de ce cluster. Ainsi, lors de la reconstruction des variables d'un cluster, si l'affectation courante des variables séparateurs a été déjà examinée, alors cette reconstruction se fait via le cacher plutôt que par une recherche arborescente.

4 Jeux de test

Les expérimentations ont été menées sur les instances de deux problèmes différents.

Instance RLFAP : Le centre d'électronique de l'armement a mis à disposition un ensemble d'instances du problème d'affectation de fréquences radio [2]. Dans ce problème, le but est d'assigner un nombre limité de fréquences à un ensemble de liens radios, et cela en minimisant les interférences dues à la réutilisation d'une fréquence. Nous reportons ici les résultats sur les instances les plus difficiles : Scen06, Scen07 et Scen08.

Instance SPOT5 : La planification quotidienne de satellite d'observation consiste à sélectionner les prises de vue à effectuer durant une journée en tenant compte des limites matérielles du satellite tout en maximisant l'importance des photographies sélectionnées [1]. Nous reportons les résultats de cinq instances sans contraintes dures de capacité.

5 Expérimentations

5.1 Protocole expérimental

Les différents paramètres de Sep-VNS ont été fixés comme suit : k_{min} à 1, k_{max} au nombre total de variables dans les séparateurs, et la valeur de la *discrepancy* à 3.

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 [1]. Pour ID-Walk, nous avons utilisé la librairie INCOP [10]. Pour chaque instance et chaque méthode, nous reportons le nombre de succès, le temps moyen nécessaire pour atteindre l'optimum (incluant le temps nécessaire à la décomposition), ainsi que le coût moyen des solutions retournées ainsi que le coût de la meilleure solution (entre parenthèses) si l'optimum n'est pas atteint. Nous avons limité le temps d'exécution de chaque méthode à une heure. Les résultats des autres méthodes ont été obtenus dans les mêmes conditions expérimentales que les nôtres.

5.2 Résultats expérimentaux

Le tableau 1 reporte les résultats obtenus sur les instances RLFAP. On peut constater sur ces instances

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

Instance	Méthode	Succés	Temps	Moyenne
Scen06	Sep-VNS	46/50	1026	3390
Opt=3389	VNS/LDS+CP	15/50	83	3399
	ID-Walk	NA	840	3447 (3389)
Scen07	Sep-VNS	2/50	3233	384065
Opt=343592	VNS/LDS+CP	1/50	461	355982
	ID-Walk	NA	360	373334 (343998)
Scen08	Sep-VNS	0/50	–	318 (269)
Opt=262	VNS/LDS+CP	0/50	–	397 (357)
	ID-Walk	NA	–	291 (267)

Table 1 – Résultats sur les instances RLFAP.

Instance	Méthode	Succés	Temps	Moyenne
408	Sep-VNS	38/50	546	6228
Opt=6228	VNS/LDS+CP	26/50	142	6228
	ID-Walk	50/50	3	6228
412	Sep-VNS	34/50	820	32381
Opt=32381	VNS/LDS+CP	32/50	130	32381
	ID-Walk	10/50	2102	32381
414	Sep-VNS	18/50	1761	38479
Opt=38478	VNS/LDS+CP	12/50	434	38481
	ID-Walk	0/50	–	38481
507	Sep-VNS	23/50	925	27390
Opt=27390	VNS/LDS+CP	11/50	232	27391
	ID-Walk	7/50	1862	27391
509	Sep-VNS	19/50	2370	36447
Opt=36446	VNS/LDS+CP	12/50	598	36448
	ID-Walk	0/50	–	36450

Table 2 – Résultats sur les instances SPOT5.

que notre approche surclasse VNS/LDS+CP et ID-Walk sur les instances Scen06 et Scen07. Sur l'instance Scen06, Sep-VNS obtient un taux de réussite de 92% contre 30% pour VNS/LDS+CP. Sur l'instance Scen08, Sep-VNS obtient des performances bien meilleures que VNS/LDS+CP. En effet, la déviation moyenne à l'optimum est réduite de 51.5% à 21.3% et la meilleure solution est très proche de l'optimum.

Une fois encore, cette tendance se confirme sur les instances SPOT5 (tableau 2), sur lesquelles Sep-VNS permet d'améliorer très nettement le taux de réussite (avec un gain de 55% en moyenne par rapport à VNS/LDS+CP).

Notons toutefois, que notre approche a besoin de plus de temps de calcul en moyenne pour obtenir l'optimum, cela à cause de temps de reconstruction plus important.

6 Conclusions et futurs travaux

Dans cet article, nous avons proposé une approche hybride combinant une recherche locale (pour la sélection et la reconstruction des variables des séparateurs) et une recherche arborescente (pour la reconstruction des clusters). Nous avons aussi montré expérimentalement que l'exploitation des séparateurs permet de dépasser les performances de VNS/LDS+CP et ID-Walk. Nous travaillons actuellement sur deux pistes.

i) La première consiste en l'étude de l'influence de la décomposition sur les performances de notre approche. La taille des clusters et le nombre de variables dans l'union des séparateurs semble jouer un rôle majeur pour notre approche (nombre des voisinages, temps de reconstruction, ...). De plus, nous utilisons une décomposition arborescente, alors que la décomposition n'a pas besoin d'être arborescente dans notre cas.

ii) La seconde piste vise à utiliser de manière plus intelligente les cachers. En effet, actuellement ceux-ci sont simplement utilisés pour éviter de re-explorer un sous-espace de recherche. Pourquoi ne pas utiliser ceux-ci pour guider la reconstruction des variables séparateurs.

Références

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

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

[3] F. Glover and M. Laguna. *TABU search*. Kluwer, 1999.

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

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

[6] M. Kitching and F. Bacchus. Symmetric component caching. In *IJCAI*, pages 118–124, 2007.

[7] S. Loudni and P. Boizumault. Combining vns with constraint programming for solving anytime optimization problems. *European Journal of Operational Research*, 191(3) :705–735, 2008.

[8] 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.

[9] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & OR*, 24(11) :1097–1100, 1997.

[10] B. Neveu and G. Trombettoni. Incop : An open library for incomplete combinatorial optimization. In *CP*, pages 909–913, 2003.

[11] L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. In *CP*, pages 468–481, 2004.

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

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

[14] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP*, pages 417–431, 1998.

[15] 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.