Heuristiques Adaptatives Génériques pour la Recherche en Grand Voisinage

Jean-Baptiste Mairy[1], **Pierre Schaus**[2] **et Yves Deville**[1]
[1]Université catholique de Louvain, Belgique
[2]n-Side, Louvain-la-Neuve,Belgique
[1]{jean-baptiste.mairy,yves.deville}@uclouvain.be [2]pierre.schaus@n-side.com

Abstract

Le paradigme de Recherche en Grand Voisinage (LNS) a d'ores et déjà prouvé son efficacité pour la résolution d'une grande variété de Problèmes d'Optimisation Contraints (COP). Le LNS est une métaheuristique de recherche locale (LS) utilisant une recherche exhaustive (telle que CP) pour explorer les voisinages. Ces voisinages sont définis en relaxant un sous-ensemble des variables du problème (appelé fragment) par rapport à la solution courante. Malgré le succès du LNS, il n'existe aucun principe général pour choisir les fragments à relaxer. En l'absence d'une bonne connaissance du problème, ceux-ci sont généralement choisis aléatoirement pour favoriser la diversification. Ce papier [1] propose différentes heuristiques pour un choix automatique et générique de fragments améliorant le choix aléatoire. Ces heuristiques sont testées sur le problème du Car Sequencing pour lequel une modélisation originale est introduite. Ces expériences permettent de montrer que les heuristiques proposées sont plus performantes qu'un choix aléatoire. Deux d'entre elles (impact dynamique proximité et impact dynamique max proximité) sont également significativement meilleures que toutes les autres.

1 Introduction

L'obtention de solutions optimales pour les problèmes d'optimisation combinatoire sous contraintes est un des chalenges majeures pour les sciences informatiques. Le paradigme de Recherche en Grand Voisinage (LNS) s'est montré efficace dans bon nombre de problèmes d'optimisation sous contraintes, en permettant l'obtention rapide de bonnes solutions (par exemple [1, 3, 4, 5, 6]). Cette efficacité vient de la combinaison de l'expressivité de la programmation par contraintes (CP) avec la rapidité de la recherche locale (LS). Le principe de fonctionnement du LNS est de maintenir, durant toute la recherche, une solution (appelée solution courante) qui ne viole aucune contrainte mais qui n'est pas optimale. Cette solution courante correspond à la meilleure solution trouvée depuis le début de la recherche. Les solutions successives sont obtenues en répétant les 2 opérations suivantes jusqu'à arriver à une condition d'arrêt :

1. définition du voisinage : cette étape consiste à choisir le sous-ensemble des variables qui seront relâchées à leur domaine initial, les autres étant affectées à leur valeur dans la solution courante. Le voisinage de la solution courante est, dès lors, défini par l'ensemble des solutions du sous-problème ainsi défini.
2. exploration du voisinage : CP est utilisé dans cette étape pour explorer le sous-problème défini à l'étape 1. Lorsqu'une solution meilleure est trouvée, elle devient la solution courante. Afin d'éviter de passer trop de temps à explorer les voisinages, cette étape est limitée par une borne sur le temps ou le nombre d'échecs de la recherche.

En dépit de la simplicité de cette métaheuristique, la procédure de sélection des fragments doit être choisie précautionneusement. Malheureusement, il n'existe pas de principe général pour réaliser cette tâche. En l'absence d'une connaissance accrue du problème traité, les fragments sont souvent choisis aléatoirement dans le but de favoriser la diversification.

P. Laborie et D. Godard proposent, dans [8], un LNS adaptatif pour les problèmes de scheduling. Les trois stratégies proposées sont combinées en un portfolio. À chaque étape de sélection d'un fragment, une des stratégie est choisie en fonction de probabilités pour les

1. Ce papier a été publié dans sa version anglaise au workshop LSCS de la conférence CP2010

techniques d'être utilisées. Ces probabilités sont ensuite mises à jour en fonction des résultats de l'exploration du voisinage. Malheureusement, les trois stratégies utilisées sont spécifiques au problème du scheduling.

L. Perron et al. introduisent, dans [1], deux heuristiques adaptatives génériques de sélection de fragments. La première stratégie démarre du CSP original (où toutes les variables ont pour domaines leur domaine initial) et sélectionne successivement les variables à fixer pour définir le voisinage. Quand une variable est sélectionnée, elle est fixée à sa valeur dans la solution courante. La propagation de cette assignation est réalisée. L'heuristique sélectionne alors, comme variable suivante, la variable dont le domaine a été le plus réduit par cette assignation. La seconde heuristique sélectionne les variables à relâcher. Le but de cette heuristique est de définir des fragments composés de variables proches. La relation de proximité est calculée par rapport à la propagation ayant eu lieu lors des étapes précédentes d'assignation des variables ne faisant pas partie des fragment.

Ce papier poursuis le travail de Perron et al. en apportant les contributions suivantes :

- De nouvelles heuristiques adaptatives génériques de sélection de fragments
- Une modélisation originale du problème du Car Sequencing comme un problème d'optimisation contraint
- Des résultats expérimentaux montrant l'efficacité de ces heuristiques sur le problème du Car Sequencing

La suite du papier est organisée comme suit : la section 2 définit les heuristiques de sélection de fragments et la section 3 présente les résultats des expérimentations de ces heuristiques sur le problème du Car Sequencing et la section 4 conclut cet article.

2 Heuristiques Adaptives Génériques de sélection de fragments

Cette section se concentre sur les différentes heuristiques développées dans le cadre de ce travail. D'abord, les concepts et les mesures utilisés pour définir les heuristiques sont introduits. Ensuite, les heuristiques simples sont définies. Celles-ci n'utilisent qu'une mesure afin de sélectionner les fragments. Finalement, les heuristiques combinées, utilisant plus d'une mesure, sont présentées.

2.1 Concepts et mesures

Les différents concepts et mesures présentés dans cette section sont : l'impact dynamique, l'impact dynamique moyen, l'impact dynamique max, l'impact adaptatif et la proximité.

Impact dynamique. L'impact dynamique d'une variable tente de mesurer l'impact que la variable aurait sur l'objectif si elle était relaxée. Soit S une solution au problème, l'impact d'une valeur v pour une variable x est défini comme la différence qui est induite sur l'objectif par l'affectation de v à x à la place de sa valeur dans S. Pendant l'évaluation de cet impact, les autres variables sont considérées comme assignées à leur valeur dans S et les contraintes sont ignorées. Pratiquement, pour une fonction objectif O, l'impact dynamique de la valeur v pour la variable x_i par rapport à la solution S se calcule comme :

$$\begin{aligned}\mathcal{I}(x_i, v, S) = \\ & O\Big(S(x_1), \ldots, S(x_{i-1}), v, S(x_{i+1}), \ldots, S(x_n)\Big) \\ & -O\Big(S(x_1), \ldots, S(x_n)\Big),\end{aligned}$$

où $S(x_k)$ représente la valeur de x_k dans S.

L'impact d'une variable x pour une solution S est l'agrégation des $\mathcal{I}(x, v, S)$ pour toutes les valeurs $v \in D_0(x)$ (domaine initial de x). L'impact dynamique défini ici est séparé en deux mesures correspondant chacune à une agrégation différente : l'impact dynamique moyen et l'impact dynamique max.

Impact dynamique moyen. Soit x, une variable, O la fonction objectif du problème et S, une solution. L'impact dynamique moyen de x par rapport à S est défini comme

$$\mathcal{I}_{mean}(x, S) = \frac{1}{|D_0(x)| - 1} \sum_{v \in D_0(x) \setminus \{S(x)\}} \mathcal{I}(x, v, S).$$

Une variable qui, en moyenne, améliore l'objectif lorsqu'on ne change que sa valeur par rapport à la solution S aura un $\mathcal{I}_{mean}$ positif si le problème est une maximisation et négatif si le problème est une minimisation.

Impact dynamique max. Pour les problèmes de maximisation, l'impact dynamique max est défini, pour une variable x, comme la valeur maximale de $\mathcal{I}(x, v, S)$ pour $v \in D_0(x) \setminus \{S(x)\}$. Cela correspond à l'amélioration maximum que la variable pourrait, à elle seule, induire sur l'objectif. Pour une variable x et une solution S, l'impact dynamique max est donc

$$\mathcal{I}_{max}(x, S) = \max_{v \in D_0(x) \setminus \{S(x)\}} \mathcal{I}(x, v, S).$$

Pour les problèmes de minimisation, $\mathcal{I}_{min}$ est défini de manière analogue.

Impact Adaptatif. L'impact adaptatif tente de saisir l'impact des variables sur la fonction objectif en utilisant les informations révélées par la propagation, produites lors des affectations. L'objectif est de refléter l'importance intrinsèque des variables pour l'objectif, indépendamment de la solution actuelle. Plus précisément, l'impact adaptatif mesure l'influence que les affectations des variables ont sur les bornes de l'objectif par le biais de la propagation. Pour les problèmes de maximisation, les bornes utilisées sont les bornes supérieures. Pour les minimisations, ce sont les bornes inférieures. L'impact adaptatif d'une variable x est défini comme la moyenne des variations de la borne d'intérêt sur toutes les affectations de x, depuis le début de la recherche. Supposons que le LNS soit à un point d'avancement p et que $SumImpact(x, p)$ soit la somme des changements de la borne d'intérêt qui ont eu lieu lors des affectations de x jusqu'au point p. Supposons également que $NumTimesBound(x, p)$ représente le nombre de fois où x a été lié jusqu'au point d'avancement p. L'impact adaptatif de x est alors :

$$\mathcal{I}_{adaptive}(x, p) = \frac{SumImpact(x, p)}{NumTimesBound(x, p)}.$$

Cette définition d'impact adaptatif traite chaque variable indépendamment des autres. De futures définitions pourront inclure les interactions entre variables.

Proximité. La proximité est une mesure qui a pour but de caractériser les relations entre les variables. Cette mesure a été inspirée par celle définie dans [1]. La proximité est définie en termes d'interactions que le réseau de contraintes induit entre deux variables. Elle se mesure en inspectant la propagation se produisant lorsque les variables sont liées. Pratiquement, la proximité entre les variables x et y est calculée comme le nombre moyen de valeurs qui sont retirées de $D(y)$ lorsque x est affecté. La seule subtilité est que cette moyenne est calculée uniquement sur les affectations de x où y n'est pas lié. Sans cette subtilité, le calcul inclurait les cas où aucune valeur de y n'est retirée uniquement parce que cette variable a déjà été affectée.

Supposons que le LNS soit à un point p, où $Propag(x, y, p)$ est le nombre total de valeurs supprimées de $D(y)$ par toutes les affectations du x jusqu'à p. Soit également $NumBinds(x, y, p)$ le nombre de fois où x a été lié alors que y n'était pas affecté jusqu'à p. La proximité entre x et y est alors définie comme

$$Proximity(x, y, p) = \frac{Propag(x, y, p)}{NumBinds(x, y, p)}.$$

Si deux variables ont une grande proximité, la relaxation d'une de ces variables sans l'autre retirerait, avant le début de la recherche, un grand nombre de valeurs du domaine de la variable relâchée. Cela réduirait les possibilités de changement de valeur pour la variable relâchée et donc, les possibilités de réoptimisation de l'objectif. Notez que la relation de proximité n'est pas symétrique.

La mesure de proximité définie ici est basée sur la même intuition que la relation de proximité définie dans [1] : les relations entre les variables peuvent être révélées par la propagation. Cependant, la proximité définie dans ce papier est mise à jour chaque fois qu'une variable est liée. Cela inclut le gel des variables ne faisant pas partie des fragments et les points de choix dans les étapes de recherche CP. La mesure de proximité définie dans [1] est seulement mise à jour au cours du gel des variables hors fragments. En outre, comme nous le verrons, la proximité est ici uniquement utilisée en combinaison avec d'autres heuristiques, ce qui n'est pas le cas dans [1].

```
fragment = {}
varSet = X
while (|fragment| < k) do
   Select variable x in varSet according to I_λ
   fragment ⊕ x
   varSet ⊖ x
end while
```

Pseudo-code 2.1 :
Sélection de fragment basée sur l'impact

2.2 Heuristiques simples

Cette section présente les heuristiques simples. Chacune de ces heuristiques utilise une des mesures déjà présentées.

Ces heuristiques ont pour objectif de sélectionner un fragment composé de variables ayant un impact important sur l'objectif. Pour favoriser la diversification, la sélection est randomisée. En effet, les variables à inclure dans les fragments sont sélectionnées *parmi* les variables d'impact maximum. Il existe une heuristique simple pour chaque mesure d'impact définie dans la section 2

Le pseudo-code 2.1 montre le processus de sélection des fragments. Dans ce code, X désigne l'ensemble des variables du problème, k correspond à la taille du fragment à sélectionner, $\mathcal{I}_\lambda$ est la mesure d'impact choisie, $\oplus$ et $\ominus$ sont respectivement l'ajout et la suppression d'un élément dans une liste. La sélection des variables est aléatoire, avec une distribution de probabilité proportionnelle à $\mathcal{I}_\lambda$.

2.3 Heuristiques combinées

Cette section présente les heuristiques combinées. Chacune de ces heuristiques combine une mesure d'impact avec celle de proximité. Il y a donc une heuristique combinée pour chaque notion d'impact définie.

Dans ces heuristiques, le fragment est sélectionné en deux parties, chacune ayant une fonctionnalité distincte. Le but est de sélectionner des fragments composés de variables importantes (1re partie) et des variables qui sont liées avec les plus importantes (2e partie). Cette seconde partie du fragment permet aux variables importantes de changer de valeur pour améliorer la solution courante.

L'impact et la proximité sont ici considérés comme également pertinents. Le fragment est donc séparé en deux parties égales. Dans la seconde partie du fragment, pour déterminer si une variable est liée ou non à une autre, un seuil sur la proximité est utilisé. Une variable x est considérée comme liée à une autre variable y si $proximity(x, y)$ est supérieur au seuil. De plus, la proximité n'étant pas symétrique, elle est utilisée dans les deux sens. Pour une variable x, une première moitié de fragment F_1 et un seuil T, son inclusion dans la seconde moitié du fragment est déterminée par la somme du nombre de variables de F_1 auquel x est lié et du nombre de variables de F_1 qui sont liées à x :

$$NL(x_i, F_1, T) = \big|\{y \in F_1 \mid proximity(x, y) > T\}\big| + \big|\{y \in F_1 \mid proximity(y, x) > T\}\big|$$

L'algorithme pseudo-code 2.2 montre comment les fragments sont sélectionnés par les heuristiques de cette section. Dans ce code, $\mathcal{I}_\lambda$ est la mesure d'impact choisie. La sélection de la première moitié du fragment (F_1) est la même que celle présentée pour les heuristiques basées sur l'impact, sauf pour le nombre de variables. La diversification est assurée dans les deux parties du fragment par le même mécanisme aléatoire que pour les heuristiques précédentes.

```
F1 = {}
varSet = X
while (|F1| < k/2) do
   Select variable x in varSet according to I_λ
   F1 ⊕ x
   varSet ⊖ x
end while
F2 = {}
while (|F2| < k/2) do
   Select variable x in varSet among large NL(x, F1, T)
   F2 ⊕ x
   varSet ⊖ x
end while
fragment = F1 ∪ F2
```

Pseudo-code 2.2 :
Sélection de fragment basée sur l'impact et la proximité

3 Résultats expérimentaux

Cette section présente les résultats expérimentaux. Elle commence par présenter le problème utilisé pour les tests. Ensuite, elle introduit la méthodologie utilisée. Finalement, elle présente les résultats des expériences et leur interprétation.

3.1 Le Car Sequencing comme un problème d'optimisation

Le problème choisi pour tester les différentes heuristiques présentées dans ce papier est le problème du Car Sequencing. Cette section décrit ce problème ainsi que sa modélisation comme un problème d'optimisation sous contraintes.

Le problème du Car Sequencing consiste à ordonner des voitures sur une ligne de production pour satisfaire une demande. Les caractéristiques des voitures à produire sont regroupées en configurations. Une configuration est composée d'options spécifiques (telles que toit ouvrant, climatisation, etc.). Chaque option o est installée par une station qui a une capacité limitée : la station peut installer o sur au plus m_o voitures dans toute séquence de longueur n_o. Le but de ce problème est d'ordonner les voitures à produire de sorte que les capacités des stations ne soient pas dépassées.

Ce problème de faisabilité a été ici modélisé comme un problème d'optimisation sous contraintes en relâchant la contrainte imposant qu'un emplacement de la ligne de production ne puisse contenir qu'une voiture. Plusieurs voitures de la même configuration peuvent donc être situées sur le même emplacement. L'objectif du problème relaxé est de maximiser le nombre d'emplacements utilisés sur la ligne de production (et donc minimiser le nombre d'emplacements avec plu-

sieurs voitures). Une solution pour le problème initial (problème de satisfaction) correspond à une solution du problème relaxé où le nombre d'emplacements utilisés est égal au nombre de voitures. Cette relaxation est différente de la relaxation usuelle (minimisant le nombre de violations des contraintes de capacité) parce que le LNS ne considère que des solutions sans viol de contraintes.

La Figure 1 présente le modèle *Comet* pour ce problème. Les données du problème sont : *demand*, le nombre de voitures à produire pour chaque configuration ; m et n les contraintes de capacité pour chaque option et $configOfOption$, le tableau contenant pour chaque option, l'ensemble des configurations qui la contiennent. Dans ce modèle, $carsWithConfig$ est un tableau contenant les voitures de chaque configuration. Les variables de décision sont contenues dans $placeOfCar$, représentant l'emplacement où chaque voiture est placée sur la ligne de production. Le tableau de variables $configOfCar$ contient la configuration des voitures présentes sur les emplacements de la ligne. Il permet d'utiliser la contrainte *sequence*. Les deux points de vue ($placeOfCar$ et $configOfCar$) sont liés par des contraintes *element*. Il est important de remarquer qu'un emplacement vide (c'est à dire un emplacement non utilisé dans $placeOfCars$) peut contenir n'importe quelle configuration de voiture qui est compatible avec les configurations des voitures environnantes. Cette contrainte permet une réduction importante de l'espace de recherche. Nous pensons donc que cette relaxation est plus forte que celle proposée dans [1]. Dans cette relaxation, des emplacements supplémentaires et des voitures virtuelles ne nécessitant pas d'option sont introduits. L'objectif est alors de minimiser le dernier emplacement avec une voiture réelle. La relaxation présentée ici permet d'utiliser la contrainte globale $atLeastNValue$ [7], qui est une version soft de la contrainte bien connue $AllDifferent$. Cette contrainte lie globalement l'objectif (le nombre d'emplacements non-vides de la ligne de production) et les variables de décision, permettant un filtrage fort. Un autre avantage de $atLeastNValue$ est que le couplage maximal maintenu par son propagateur peut servir d'heuristique de valeur pour affecter les voitures aux emplacements. Nous pouvons retrouver cette heuristique dans l'alternative gauche du bloc *using* du modèle *Comet*. L'inconvénient de ce modèle est qu'il introduit des symétries entre les voitures de même configuration. Ces symétries sont brisées dynamiquement au cours de la recherche dans l'alternative de droite du bloc *using* : en entrant dans l'alternative de droite, il est prouvé que l'emplacement est impossible pour la voiture et est donc supprimé des domaines de toutes les voitures non assignées de même configuration. Chaque fois que la limite sur le nombre d'échecs est atteinte ou que l'arbre de recherche est exploré en entier, le bloc *onRestart* est exécuté, un fragment est choisi et les variables n'appartenant pas à ce fragment sont fixées à leur valeur dans la solution courante (meilleure solution).

3.2 Méthodologie de test

Cette section présente la méthodologie utilisée pour la comparaison des différentes heuristiques présentées dans cet article. Les instances utilisées pour la partie expérimentale proviennent de la CSPLIB [2], seconde série. Toutes ces instances sont satisfaisables, ce qui signifie que, pour chacune d'elles, il existe une solution pour laquelle le nombre d'emplacements non-vides sur la ligne de production est égal au nombre de voitures.

La procédure de test a été conçue pour évaluer les heuristiques de sélection de fragments indépendamment des autres paramètres du LNS. Elle est basée sur la comparaison des résultats des stratégies sur plusieurs exécutions d'une seule étape LNS (sélection de fragment et recherche CP). Afin d'éliminer l'influence des paramètres, les valeurs optimales de ceux-ci sont évaluées pour chaque heuristique. La procédure de test peut être résumée comme suit :

1. Pour chaque instance, cinq solutions de départ sont obtenues à l'aide d'un LNS pour un nombre déterminé d'étapes (le même nombre pour toutes les solutions). Ces solutions, en plus des valeurs des variables, contiennent toutes les quantités qui sont requises par les mesures adaptatives.
2. À partir de ces solutions de départ, la taille optimale du fragment est évaluée pour chaque heuristique.
3. À partir de chaque solution de départ, chaque heuristique est exécutée plusieurs fois pour une unique étape LNS avec les paramètres optimaux.

Pour l'évaluation des paramètres optimaux, les tailles de fragments de 25, 20, 15, 10, 5, 2 pourcents du nombre total de variables sont testées. La limite sur la recherche CP est fixée à 3200 backtracks. Ce dernier paramètre peut être fixé a priori puisque la comparaison ne se fait que sur une étape LNS. Pour chaque heuristique et chaque solution de départ, la taille du fragment utilisée pour les comparaisons est celle qui mène au plus grand nombre d'améliorations sur 10 exécutions.

2. www.csplib.org

```
//Data of the problem
range Options; range Configs; range Slots; range Cars;
int m[Options]; int n[Options]; //capacity m[o] out of n[o] for option o
int requires[Configs,Options];  //1 if config requires option, 0 otherwise
int demand[Configs]; //demand for each configuration c
int config[Cars]; //configuration for each car
set{int} configsOfOption[Options]; //set of cars with option
set{int} carsWithConfig[Configs]; //set of cars with config


Solver<CP> cp();
var<CP>{int} configOfCar[Slots](cp,Configs);//config of the car in the slot
var<CP>{int} placeOfCar[Cars](cp,Slots);//slot of the car
var<CP>{int} atLeast(cp,Cars);//number of slots with a car

AtLeastNValue<CP> atLeastCons;

cp.lnsOnFailure(3200); //LNS Limit

maximize<cp>
  atLeast
subject to{
  forall(o in Options)
    cp.post(sequence(configOfCar,demand,m[o],n[o],configsOfOption[o]));
  forall(c in Cars)
    cp.post(configOfCar[placeOfCar[c]]==config[c]);
  atLeastCons = atLeastNValue(placeOfCar,atLeast);
  cp.post(atLeastCons);
}
using{ //Search Heuristic
 while(!bound(placeOfCar))
    selectMin(c in Cars: !placeOfCar[c].bound())(placeOfCar[c].getSize()){
      int val = placeOfCar[c].getMin();
      if(atLeastCons.hasValInBestAssignment(c))
         val = atLeastCons.getValInBestAssignment(c);
      try<cp>{
         cp.post(placeOfCar[c]==val);
      }|{
        forall(c_ in carsWithConfig[config[c]]:!placeOfCar[c_].bound())
         cp.post(placeOfCar[c_]!=val);
      }
   }
}onRestart{ //LNS Relaxation
   set{int} relaxedCars();
   //.. fragment definition ..
   Solution sol = s.getSolution();
   forall(c in Cars:!relaxedCars.contains(c))
      s.post(placeOfCar[c] == placeOfCar[c].getSnapshot(sol));
}
```

FIGURE 1 – Modèle Comet pour le problème du Car Sequencing

Heuristique	rand	AI	AIP	MDI	MDIP
Taille (%)	15	10	15	15	15

TABLE 1 – taille de fragment optimale pour chaque heuristique (pourcentage du nombre de variables)

Lors de la comparaison, chaque heuristique est exécutée 30 fois par instance (6 fois pour chaque solution de départ). La comparaison des différentes stratégies utilise les informations suivantes :

- Le nombre d'améliorations
- l'amélioration moyenne
- le temps moyen d'exécution

3.3 Résultats expérimentaux

Cette section présente les résultats expérimentaux des heuristiques présentées dans ce papier sur le problème du Car Sequencing. Il faut noter que, sur ce problème, l'impact dynamique moyen et l'impact dynamique max mènent à la même sélection de fragments. En effet, l'objectif étant de maximiser le nombre de valeur différentes affectées aux variables, l'impact dynamique max est proportionnel à l'impact dynamique moyen. Seul l'impact dynamique moyen est donc présenté dans les résultats.

Les solutions de départ ont été obtenues avec une limite de 35 étapes LNS. Le tableau 1 montre les tailles de fragment obtenues avec la procédure de test. Le tableau 2 présente le nombre d'améliorations et le tableau 3, les améliorations moyennes. Dans les deux dernières tables, les meilleurs résultats sont indiqués en gras. La correspondance entre les noms des colonnes et les heuristiques est la suivante :

- rand → *fragment aléatoire*
- AI → *impact adaptatif*
- AIP → *impact adaptatif + proximité*
- MDI → *impact dynamique moyen*
- MDIP → *impact dynamique moyen + proximité*

La première conclusion que l'on peut tirer de ces résultats expérimentaux est que toutes les heuristiques présentées dans ce papier sont plus performantes que la sélection aléatoire de fragments. Cela est sans doute du au manque d'intensification inhérent à la sélection aléatoire. Les meilleures heuristiques sur ce problème sont l'impact dynamique moyen + proximité (MDIP) et l'impact dynamique moyen (MDI). Cela peut être justifié par la diversification induite par l'utilisation du MDI. En effet, cette mesure ne se basant que sur la solution courante, elle est plus variable que les autres et entraine probablement une meilleure diversification. L'ajout de la proximité à cette heuristique semble améliorer la qualité moyenne des fragments. Une inter-

instance	rand	AI	AIP	MDI	MDIP
bench_65_01	4	11	8	12	**13**
bench_65_03	11	14	10	**24**	19
bench_70_01	1	5	6	4	**12**
bench_70_03	3	4	5	**12**	10
bench_70_04	6	16	13	18	**24**
bench_75_01	3	**10**	5	6	6
bench_75_02	8	9	13	12	**15**
bench_75_03	8	13	13	6	**24**
bench_75_04	7	10	6	**24**	23
bench_80_01	8	**19**	13	6	**19**
bench_80_02	9	10	10	**23**	18
bench_80_03	6	11	11	18	**19**
bench_80_04	2	5	**14**	8	12
bench_85_01	6	6	7	**13**	11
bench_85_02	11	15	12	15	**17**
bench_85_04	8	**21**	7	18	8
bench_90_01	4	17	7	**26**	21
bench_90_02	2	10	7	**21**	17
bench_90_03	4	5	11	**19**	17
sum	111	211	178	285	**305**

TABLE 2 – Nombre d'améliorations

instance	rand	AI	AIP	MDI	MDIP
bench_65_01	0.2	0.5	0.4	1.2	**1.5**
bench_65_03	0.4	0.8	0.5	**1.6**	1.5
bench_70_01	0.0	0.3	0.3	0.3	**0.9**
bench_70_03	0.1	0.2	0.2	0.6	**0.9**
bench_70_04	0.2	0.6	0.6	1.0	**1.6**
bench_75_01	0.1	**0.5**	0.2	0.4	0.4
bench_75_02	0.4	0.5	0.6	**1.2**	1.1
bench_75_03	0.4	1.0	0.7	0.8	**2.0**
bench_75_04	0.2	0.4	0.2	**1.4**	1.2
bench_80_01	0.3	0.7	0.5	0.2	**0.9**
bench_80_02	0.3	0.4	0.4	1.0	**1.6**
bench_80_03	0.3	0.4	0.4	1.1	**2.0**
bench_80_04	0.1	0.2	**0.8**	0.3	0.4
bench_85_01	0.2	0.3	0.3	**0.7**	0.6
bench_85_02	0.4	0.9	0.6	**1.6**	1.5
bench_85_04	0.3	**1.0**	0.3	0.9	0.3
bench_90_01	0.2	1.0	0.4	**2.5**	1.7
bench_90_02	0.1	0.5	0.3	**1.4**	1.0
bench_90_03	0.3	0.3	0.6	1.2	**1.4**
global avg	0.2	0.6	0.4	1.0	**1.2**

TABLE 3 – Amélioration moyenne

Heuristic	rand	AI	AIP	MDI	MDIP
Mean time	2.0	2.0	2.3	2.3	2.1

Table 4 – Moyennes des temps d'exécutions en secondes

prétation possible est que la proximité permet d'obtenir des fragments plus structurés. La proximité, étant une mesure adaptative, pourrait également tempérer l'optimisme de l'impact adaptatif moyen. De ces expériences, nous pouvons également voir que l'impact adaptatif (AI) est la troisième meilleure heuristique de l'ensemble. Il semblerait qu'après 35 étapes LNS, les informations que l'impact adaptatif a collectées soient pertinentes pour choisir des fragments. Nous pouvons également remarquer que l'ajout de la proximité à cette heuristique dégrade ses performances. Les temps d'exécution moyen en secondes pour les différentes heuristiques sont fournis dans le tableau 4. Ce sont les moyennes, sur l'ensemble des exécutions, du temps total pris par les heuristiques pour calculer le fragment, geler les variables et effectuer la recherche CP. Le temps moyen d'exécution de l'heuristique aléatoire peut sembler contre-intuitif. En effet, on pourrait penser qu'en l'absence de calculs à réaliser, l'heuristique aléatoire devrait être plus rapide. Cependant, les temps reportés dépendent de la forme du voisinage car ils incluent le temps de la recherche CP. Ces temps d'exécution suggèrent que les voisinages définis par nos heuristiques sont explorés plus rapidement.

4 Conclusion

Ce travail présente diverses heuristiques adaptatives pour le choix des variables à relaxer pour la Recherche en Grand Voisinage. L'impact dynamique moyen, l'impact dynamique max et l'impact adaptatif sont définis pour mesurer l'influence d'une variable sur la valeur de la fonction objectif. La mesure de proximité est une estimation des liens que les contraintes induisent entre les couples de variables. Les heuristiques basées sur ces mesures sont regroupées en deux catégories : les heuristiques simples qui utilisent une mesure et les heuristiques combinées qui associent deux mesures pour choisir les fragments. La catégorie simple compte trois heuristiques, chacune utilisant une des notions d'impact définies. Trois heuristiques combinées ont été définies. Elles sont obtenues en combinant l'une des trois mesures d'impact avec la proximité. Toutes ces heuristiques ont été expérimentées sur une nouvelle relaxation du problème du Car Sequencing. Les résultats ont été obtenus par l'application d'une procédure de test créée pour évaluer les heuristiques indépendamment des autres paramètres du LNS. Cette procédure se base sur la détermination expérimentale, pour des solutions de départ fixées, des paramètres à éliminer. La comparaison porte sur la répétition d'une étape LNS. Les mesures du nombre de total d'améliorations et leurs qualité montrent que toutes nos heuristiques sont plus performantes qu'un choix aléatoire de fragments. Ces test révèlent que les heuristiques utilisant la proximité avec soit l'impact dynamique moyen, soit l'impact dynamique max conduisent au même choix de fragments et sont significativement meilleures que les autres sur le problème du Car Sequencing. Le temps moyen cumulé pour le calcul du fragment, le gel des variables et la recherche CP est comparables pour les heuristiques présentées et pour l'heuristique aléatoire. Il semble que la recherche CP soit plus rapide quand appliquée à des fragments sélectionnés par les heuristiques présentées ici, que quand cette recherche porte sur des fragments aléatoires. La suite de ce travail comprend l'expérimentation des heuristiques sur d'autres problèmes pour évaluer leurs performances en général. Une comparaison étendue des heuristique avec l'état de l'art LNS (LNS générique mais aussi spécifique) ainsi qu'avec d'autres approches incomplètes sera également réalisée. Ces comparaisons seront réalisées sur base de procédures de tests standard. Enfin, nous tenons à comparer la relaxation du Car Sequencing proposée avec celle décrite dans [1].

Remerciements. Nous remercions les reviewers pour leurs commentaires constructifs. Cette recherche est partiellement financée par le programme des Pôles d'Attraction Interuniversitaires (Politique scientifique belge), ainsi que par le projet FRFC 2.4504.10 du Fond National belge de la Recherche Scientifique.

Références

[1] Laurent Perron, Paul Shaw and Vincent Furnon. Propagation Guided Large Neighborhood Search. CP 2004, LNCS 3258 (2004) 468–481.

[2] Nicolas Levasseur, Patrice Boizumault and Samir Loudni. Boosting VNS with Neighborhood Heuristics for Solving Constraint Optimization Problems. LNCS 5296 (2008) 131–145

[3] Daniel Godard, Philippe Laborie and Wim Nuijten. Randomized Large Neighborhood Search for Cumulative Scheduling. AAAI 2005.

[4] Guy Desaulniers, Eric Prescott-Gagnon and Louis-Martin Rousseau. A Large Neighborhood Search Algorithm for the Vehicle Routing Problem with Time Windows. MIC 2007.

[5] Emilie Danna and Laurent Perron. Structured vs. Unstructured Large Neighborhood Search : A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs. CP 2003, LNCS 2833 (2003) 817–821.

[6] Paul Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems CP 1998, LNCS 1520 (1998) 417–431.

[7] T. Petit, J.-C. Régin, and C. Bessière. Specific Filtering Algorithms for Over-Constrained Problems. CP 2001, LNCS 2239 (2001) 451–463.

[8] Philippe Laborie and Daniel Godard. Self-Adapting Large Neighborhood Search : Application to single-mode scheduling problems. Proceedings MISTA-07, Paris, 2007, 276-284.