Université de Nantes — UFR Sciences et Techniques
Master Informatique parcours “Optimisation en Recherche Opérationnelle (ORO)”
Année académique 2021-2022

Rapport du Devoir Maison n°2

Métaheuristiques

Nicolas COMPÈRE – Adrien CALLICO

October 19, 2021

Livrable du devoir maison 2 :
Métaheuristique GRASP, Path-Relinking et extensions

Amélioration par rapport au DM1

Nous n'étions pas satisfaits des temps d'exécution sur les «grosses» instances, notamment lors du 21-exchange. Afin de remédier (au moins partiellement) à cela, nous avons changé de structure de données :

En effet, nous parcourrions systématiquement toute la matrice (pour repérer les mouvements possibles lors des heuristiques de recherche locale, pour vérifier si la solution était admissible, etc.) Le problème étant que certaines matrices sont assez creuses (de densité $< 1\%$), il y a alors beaucoup d'informations «inutiles» puisque ce qui nous intéresse sont les liaisons variables-contraintes (les coefficients égaux à 1).

Nous avons alors créé deux fonctions permettant, à partir d'une matrice, de créer deux vecteurs de vecteurs : l'un de taille nb_variables, dont chaque élément contient les indices des contraintes liées à chaque variable, et l'autre de taille nb_contraintes, dont chaque élément contient les indices des variables liées à chaque contrainte.

Nous avons ainsi observé une nette amélioration des temps d'exécutions :

Glouton :

Instance	z	Temps avant (s)	Temps après (s)
pb_100rnd0100	342	0.014	0.0058
pb_100rnd0300	193	0.003	0.00478
pb_1000rnd0300	507	0.878	0.2525
pb_1000rnd0700	2050	0.543	0.2438
pb_200rnd1300	497	0.013	0.0163
pb_500rnd1100	344	0.036	0.0341
pb_500rnd0500	84	0.032	0.0344
pb_2000rnd0600	7	0.165	0.1880
pb_200rnd0700	945	0.010	0.0106
pb_200rnd0500	162	0.005	0.0113

Recherche locale en plus profonde descente:

Instance	zInit	z	Temps avant (s)	Temps après (s)
pb_100rnd0100	342	350	0.077	0.0167
pb_100rnd0300	193	194	0.019	0.0075
pb_1000rnd0300	507	520	26.221	1.276
pb_1000rnd0700	2050	2057	91.613	5.486
pb_200rnd1300	497	507	0.535	0.094
pb_500rnd1100	344	350	1.019	0.164
pb_500rnd0500	84	84	0.552	0.039
pb_2000rnd0600	7	7	9.960	0.420
pb_200rnd0700	945	956	0.541	0.131
pb_200rnd0500	162	162	0.069	0.0135

Recherche locale en simple descente:

Instance	zInit	z	Temps avant (s)	Temps après (s)
pb_100rnd0100	342	350	0.074	0.0109
pb_100rnd0300	193	194	0.049	0.0064
pb_1000rnd0300	507	513	13.846	0.520
pb_1000rnd0700	2050	2069	4.181	0.2088
pb_200rnd1300	497	510	0.063	0.00834
pb_500rnd1100	344	355	0.259	0.0250
pb_500rnd0500	84	84	0.673	0.0367
pb_2000rnd0600	7	7	8.795	0.2863
pb_200rnd0700	945	956	0.044	0.0119
pb_200rnd0500	162	162	0.079	0.0117

Présentation succincte de GRASP appliqué sur le SPP

GRASP (Greedy Randomized Adaptative Search Procedure) est un algorithme qui fait un compromis entre glouton et aléatoire. En effet, afin d'éviter de se faire piéger dans un optimum local lors d'une recherche locale, GRASP explore plusieurs solutions de départ, qui sont construites de façon à avoir des solutions de bonne facture, tout en choisissant aléatoirement les variables mises à 1. Leur construction est la suivante : On contrôle le côté aléatoire avec un paramètre $\alpha \in [0; 1]$ (0 correspond au «tout aléatoire» et 1 au «tout glouton»). Ce paramètre impose une certaine qualité; pour être choisie, elle doit permettre de dépasser le seuil d'utilité

$$U_{min} + \alpha(U_{max} - U_{min})$$

Une variable permettant de dépasser ce seuil d'utilité est alors choisie aléatoirement, et on répète ce procédé jusqu'à ne plus pouvoir ajouter de nouvelle variable à la solution. Une heuristique de recherche locale essaie d'améliorer la solution, puis on compare la solution avec les autres solutions de départ pour ne garder que la meilleure.

Avantages :

L'aléatoire permet de tester plusieurs solutions initiales différentes. Le paramètre α permet de régler la dose d'aléatoire, afin de tester plusieurs solutions sans pour autant en obtenir d'une trop mauvaise qualité.

Exemple : l'instance didactic.dat

On prend arbitrairement $\alpha = 0.5$. On calcule la limite (3.75) afin de construire la RCL : $[6, 7]$. Au hasard, la 7 est choisie. On l'insère donc dans la solution initiale et on écarte les variables qui font conflit. On recommence : la limite est 4.75 et la RCL est [2]. On choisit (au hasard ?) la 2 et on recommence. La limite est maintenant 3 et la RCL est [1]. On ajoute la variable 1. On ne peut plus ajouter de nouvelle variable à la solution, l'algorithme s'arrête. Vecteur de la solution initiale : $[1, 0, 0, 0, 0, 1, 1, 0, 0]$

Présentation succincte de Path-Relinking appliqué sur le SPP

Le principe du path-relinking est le suivant :

- considérer deux solutions admissibles
- partir de l'une pour rejoindre l'autre en effectuant des mouvements "simples" (add, drop, flip, etc.)
- si, sur le "chemin" ainsi constitué, une solution semble prometteuse (un critère est alors à définir), alors on effectue une recherche locale pour essayer de trouver une meilleure solution.

Nos premiers choix ont été les suivants :

- forward-relinking
- le flip comme mouvement "simple"
- pour passer d'une solution intermédiaire à l'autre, on reste dans l'espace des solutions admissibles, et on choisit au hasard la variable à flip.

Cependant, nous n'avions pas d'idée intéressante sur la façon d'intégrer le path-relinking à GRASP.

En lisant quelques papiers sur le sujet [1], mettre en place des "bassins de solutions d'élite" nous a semblé prometteur :

A chaque itération de GRASP (sauf la première), le path relinking est appliqué à un couple (x, y) de solutions, où x est la solution produite par GRASP à cette itération, et y une solution "d'élite", choisie aléatoirement dans un ensemble contenant au plus **Max_Elite** solutions trouvées tout au long du chemin.

On remplit au fur et à mesure cet ensemble, et s'il est déjà plein, on vérifiera si on peut le mettre à jour afin de garder uniquement les meilleures solutions.

De plus, au lieu de choisir le mouvement de base au hasard, on prend celui qui améliore le plus la valeur de la fonction objective (et on ne garde cette solution que si elle est admissible).

Une idée d'amélioration que nous n'avons pas mis en place serait d'appliquer le path-relinking à toutes les paires de solutions de l'ensemble de solutions d'élite obtenu après le nombre maximum d'itérations.

Exemple :

On considère deux solutions admissibles pour l'instance didactic.dat :

solution de départ : [0, 0, 0, 0, 1, 0, 0, 0, 0] solution target : [1, 0, 0, 0, 0, 1, 1, 0, 0]

Critère d'une solution "prometteuse" : la valeur de la fonction objective est supérieure à celle obtenue avec la solution target.

On commence par déterminer quelles variables sont différentes (cela revient à faire un "ou exclusif") : ce sont les variables 1, 5, 6 et 7. On choisit celle avec qui, en lui appliquant un flip, on obtient la meilleure valeur pour la fonction objectif ; c'est la 6. Cette valeur n'est pas prometteuse, on n'effectue pas de recherche locale. On recommence : c'est maintenant la variable 7 qui est choisie, mais la solution n'est toujours pas "prometteuse". Enfin, ce sera la même observation lorsqu'on choisira les variables 1, puis 5. Cette instance est un peu petite pour avoir des solutions "prometteuses" sur le chemin.

Expérimentation numérique de GRASP

Présentation des instances :

Instance	#Variables	#Contraintes	Densité (%)	Meilleure valeur connue
pb_1000rnd0200	1000	5000	2.59	4
pb_100rnd0100	100	500	2	372
pb_100rnd0300	100	500	2.96	203
pb_100rnd1000	100	300	2	40
pb_200rnd0500	200	1000	2.48	184
pb_200rnd0700	200	200	1.53	1004
pb_200rnd1300	200	600	1.5	571
pb_200rnd1500	200	600	1	926
pb_500rnd0500	500	2500	2.22	122
pb_500rnd0700	500	500	1.2	1141
pb_500rnd0900	500	500	0.9	2236
pb_500rnd1100	500	500	2.26	424

Nos algorithmes ont été implémentés en Julia (version 1.6.0)

L'environnement machine dans lequel nous avons réalisé nos tests présente les caractéristiques suivantes :

Système d'exploitation Ubuntu 20.04

Processeur Intel Core™ i5-7300HQ CPU @ 2.50GHz × 4

Carte Graphique Mesa Intel HD Graphics 630 (KBL GT2)

RAM 8 Go

Le budget de calcul pour les résultats qui suivront est de 200 itérations GRASP (sauf quelques exceptions, auquel cas le budget sera précisé).

Ce tableau dresse les pourcentages de précision par rapport à la valeur optimale de l'instance, pour différentes valeurs de α. En rouge sont notés les pourcentages de précision inférieurs à 95%.

Instance	$\alpha = 0.2$	$\alpha = 0.4$	$\alpha = 0.6$	$\alpha = 0.8$	$\alpha = 0.9$	$\alpha = 0.95$
pb_1000rnd0200	75	75	75	75	75	75
pb_100rnd0100	96	100	100	98	94	94
pb_100rnd0300	100	96	100	96	95	95.5
pb_100rnd1000	97	97.5	95	92	92.5	92.5
pb_200rnd0500	86	92	92	92	88	88
pb_200rnd0700	96	98	98	98	98	96
pb_200rnd1300	87	96	95	94	97	94
pb_200rnd1500	96.6	98.2	99.4	99.7	99.2	98.3
pb_500rnd0500	90	90	98	96	93	88
pb_500rnd0700	95	95	98	98	96	95
pb_500rnd0900	93.7	96.6	97.7	98.2	98.3	97.6
pb_500rnd1100	83	91	96	93	92	87.5

On observe qu'il n'y a pas de tendance générale quant au réglage de α : cela dépend des instances. On peut souligner cependant qu'une valeur proche de 1 pour α (presque "tout glouton") ne donne pas de très bons résultats.

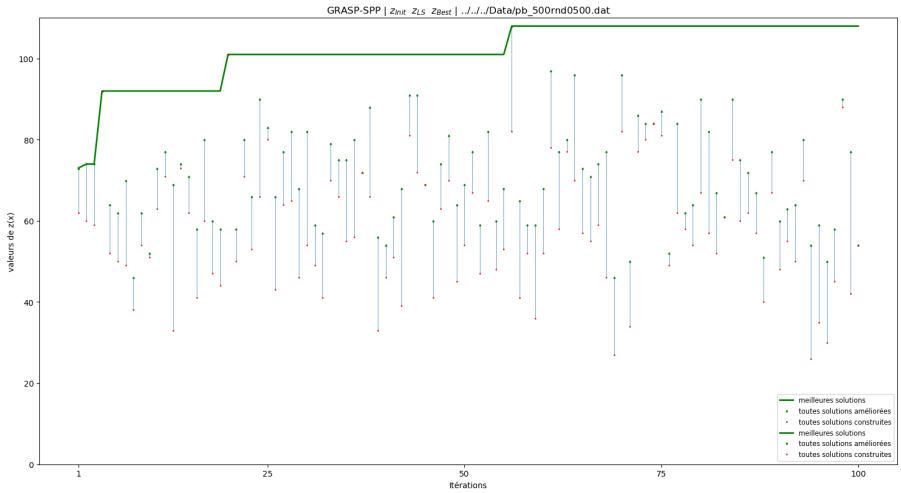

Figure 1: Une run de GRASP avec $\alpha = 0.2$

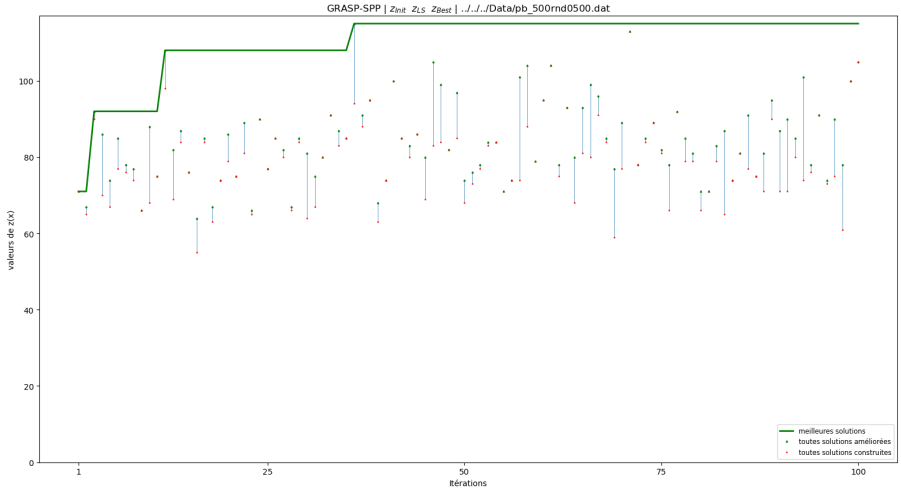

Figure 2: Une run de GRASP avec $\alpha = 0.5$

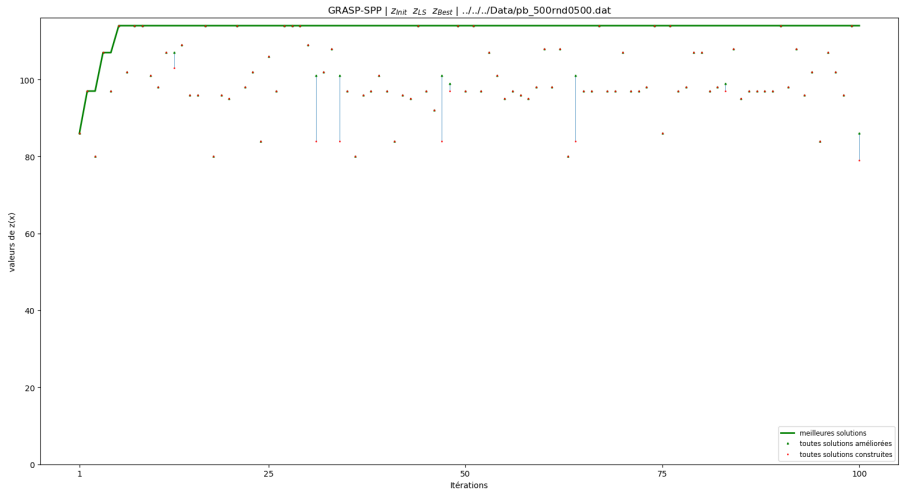

Figure 3: Une run de GRASP avec $\alpha = 0.9$

On observe facilement, avec les figures 1 à 3, l'influence du paramètre α : pour $\alpha = 0.2$ par exemple, les solutions construites sont parfois de mauvaise facture, ce qui est souvent moins le cas avec $\alpha = 0.9$.

Expérimentation numérique de Path-Relinking

Pour les expérimentations de GRASP intensifié avec le Path-relinking, nous avons alloué un budget de 200 itérations GRASP, et nous parcourons tout le chemin entre deux solutions lors du Path-Relinking.

Ce tableau présente l'influence de la taille de l'ensemble des solutions d'élite. Nous avons arbitrairement choisi $\alpha = 0.7$.

Instance	Max_Elite = 3	Max_Elite = 5	Max_Elite = 10
pb_1000rnd0200	75	75	75
pb_100rnd0100	99	98.6	98.9
pb_100rnd0300	96	96	96
pb_100rnd1000	95	95	95
pb_200rnd0500	100	100	92.3
pb_200rnd0700	98	98	98
pb_200rnd1300	96	95	95
pb_200rnd1500	99	99.5	99.6
pb_500rnd0500	96	94	96.7
pb_500rnd0700	98	96	98.5
pb_500rnd0900	97.8	98	98.3
pb_500rnd1100	92.5	92	96

Observations : GRASP intensifié avec le Path-Relinking donne de très bons résultats, mais on n'observe pas de différences majeures lorsqu'on utilise différentes valeurs pour **Max_Elite**. Intuitivement, nous pensons que ce paramètre a plus d'influence sur de plus grosses instances.

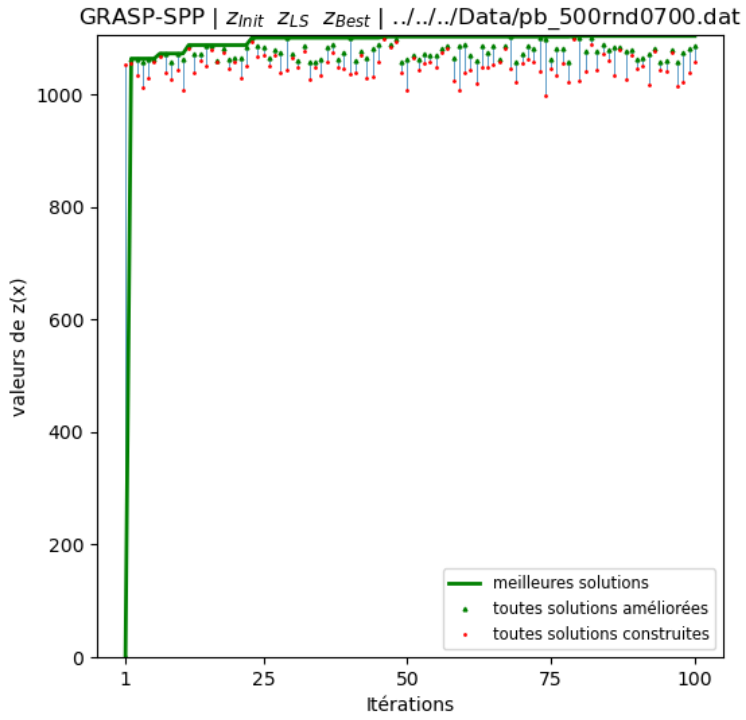

Figure 4: Une run de GRASP, intensifié avec le Path-Relinking ($\alpha = 0.7$)

Eléments de contribution au bonus

Lorsqu'on ne connaît pas bien les instances, il peut être difficile de régler le paramètre α. L'idée de Reactive Grasp est de le régler "automatiquement" en introduisant une phase d'apprentissage : on discrétise les valeurs possibles de α, et on leur attribue à chacun une probabilité égale. Au bout d'un certain nombre d'itérations N_α (paramètre) on recalcule ces probabilités en fonction de la qualité des solutions qu'ils ont produit.

200 itérations, $N_\alpha = 20$

Instance	α dominant	Précision (%)
pb_1000rnd0200	0.95	75
pb_100rnd0100	0.5	99
pb_100rnd0300	0.95	97
pb_100rnd1000	0.35	95
pb_200rnd0500	0.95	91
pb_200rnd0700	0.5	98
pb_200rnd1300	0.95	95
pb_200rnd1500	0.95	99.5
pb_500rnd0500	0.8	95
pb_500rnd0700	0.8	97
pb_500rnd0900	0.95	97.7
pb_500rnd1100	0.8	92

Voici l'évolution de la répartition des probabilités de chaque valeur de α, présentée en trois graphiques : au bout de 50, 200 et 500 itérations de GRASP.

L'instance est pb_500rnd0900.

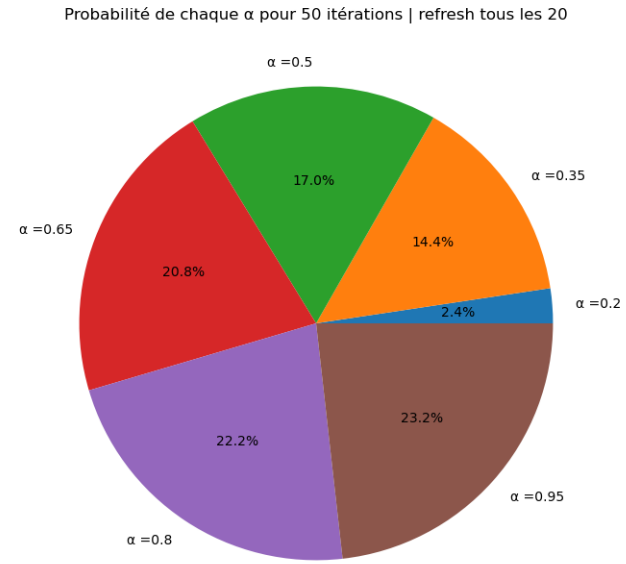

Figure 5: Probabilités des α de Reactive GRASP pour 50 itérations

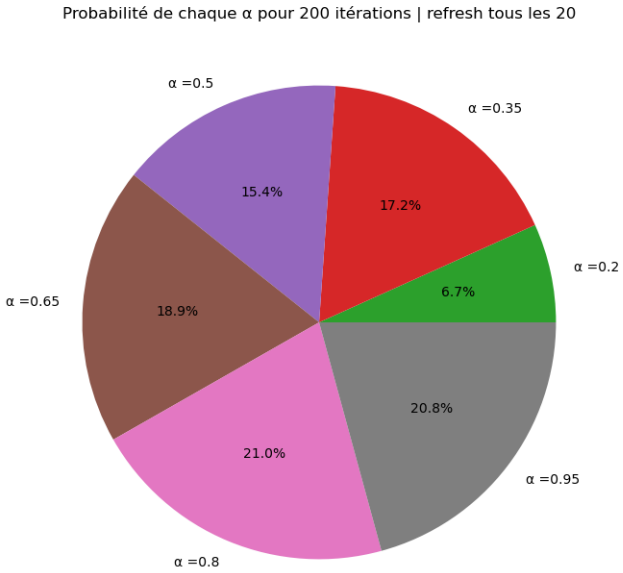

Figure 6: Probabilités des α de Reactive GRASP pour 200 itérations

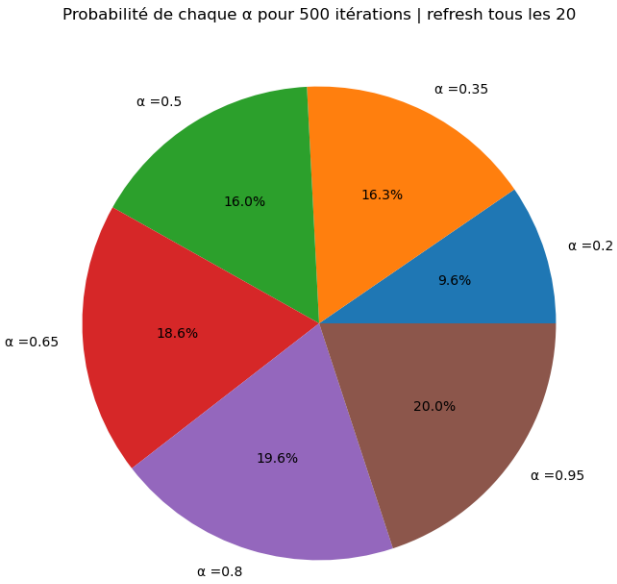

Figure 7: Probabilités des α de Reactive GRASP pour 500 itérations

Discussion

On observe que les résultats de GRASP sont globalement meilleurs que ceux du glouton seul. L'ajout d'une dimension stochastique, si elle est bien dosée, est une importante amélioration du glouton. Cela nécessite certes de régler un paramètre, ce qui peut ne pas être évident au premier abord, mais la composante Réactive-Grasp permet de pallier à ce problème.

L'emploi du Path-Relinking comme stratégie d'intensification permet d'améliorer encore nettement les solutions de GRASP, déjà de très bonne facture.

Un bon réglage des paramètres est essentiel pour les metaheuristiques qui en nécessitent, ce qui peut être fait en connaissant les instances, de nombreux tests ou par un apprentissage automatisé.

References

[1] Mauricio G.C. Resendel et Celso C. Ribeiro : *GRASP with Path-Relinking: Recent Advances and Applications*, pages 29–63. Springer US, Boston, MA, 2005.