Université de Nantes — UFR Sciences et Techniques
Master informatique parcours “optimisation en recherche opérationnelle (ORO)”
Année académique 2018-2019

Dossier Devoirs Maison

Métaheursitiques

Marie Humbert–Ropers[1] – Arthur Gontier[2]

19 octobre 2018

Livrable du devoir maison 2 : Métaheuristique GRASP, ReactiveGRASP et extensions

Les codes sont réalisés sous Julia 0.6.4 en mode mutlicoeur avec la commande : julia -p <nombres de coeurs>

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. Contrairement à une descente simple, il ne se limite pas à un optimum local mais explore plusieurs solutions de départ pour des heuristiques d'amélioration. Pour ceci, la construction des solutions de départ sont faites pour avoir de bonnes solutions tout en choisissant de manière aléatoire les variables mises à 1. Ensuite, on relance plusieurs fois les constructions et améliorations pour visiter un plus grand espace de solutions. Plus précisèment, la construction des solutions se fait de la façon suivante :
L'aléatoire dans l'heuristique de construction est controlée par un paramètre α. Ce paramètre limite la qualité des nouvelles variables qui seront choisies pour entrer dans la solution de départ, les obligeant à dépasser le seuil d'utilité :

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

Une variable entrante dépassant ce seuil d'utilité est donc choisie aléatoirement et le processus se répète jusqu'à ce que la solution de départ ne puisse plus accueillir de nouvelles variables à 1 sans contredire les contraintes. Ensuite, une heuristique de recherche locale améliore la solution. Enfin, on compare avec les autres résultats venant des différentes solutions de départ pour ne garder que la meilleure solution.

La méthode GRASP possède plusieurs avantages par rapport à la méthode de recherche locale et du glouton simple :

- L'aléatoire permet de tester des solutions initiales différentes (multi-start)
- Le paramètre α permet de régler la qualité de l'aléatoire et d'ainsi contrôler un peu cet aléatoire, c'est-à-dire d'essayer d'éviter de se laisser la possibilité de sortir d'un optimum local sans non plus choisir une solution de départ de très mauvaise qualité.
- L'heuristique d'amélioration est lancée sur plusieurs solutions différentes et la meilleure seulement est retenue : cela diminue les chances que cette heuristique se bloque sur un cas particulier de solution initiale.

En reprenant l'exemple didactique, pour construire la solution initiale, un α de 0.8 est donnée. Il permet de calculer la limite et de repérer les variables qui ont une utilité au-dessus de celle-ci : le 6,7,4. Au hasard, la 7 est choisie, elle est donc mise à 1 et les autres variables présentes dans les mêmes contraintes que la 7 sont écartées. On répète la même opérations sur les variables restantes, la limite est recalculée et parmi les variables les plus utiles, la 6 est choisie. Ensuite, il ne reste plus que la variable 4 qui est rajoutée à la solution. Il y a ensuite une recherche en profondeur sur cette solution. Dans ce cas, la solution construite est tombée sur la bonne solution donc on ne voit pas d'amélioration. Cette étape est répétée plusieurs fois, pour ne garder que la meilleur solution, puisqu'il est possible selon le choix aléatoire et la limite que l'on ne trouve pas une aussi bonne solution du premier coup.

Présentation succincte de ReactiveGRASP appliqué sur le SPP

Dans le cas d'un Grasp, le choix du paramètre α est complètememt arbitraire. Pour le choisir plus intelligemment, l'objectif est d'essayer de le régler sur les données en introduisant une phase d'apprentissage durant laquelle on choisira un α qui correspond le mieux possible aux données. Pour ce faire, on choisit de discrétiser les valeurs possibles de α et de leur attribuer une probabilité égale (loi uniforme). Après avoir laissé l'algorithme trouver quelques solutions, la probabilité de chaque α est recalculée en fonction de la qualité des solutions qu'ils ont engendrés et l'algorithme recommence l'étape précédente afin de trouver d'autres solutions avec cette nouvelle distribution. Puis, on recommence ce cycle de modifications de probabilités puis de construction et recherche locale de solutions jusqu'à un temps/nombre d'itérations limite. Ainsi, le paramètre du Grasp apprend et correspond de mieux en mieux aux données sur lesquelles il est appliqué.

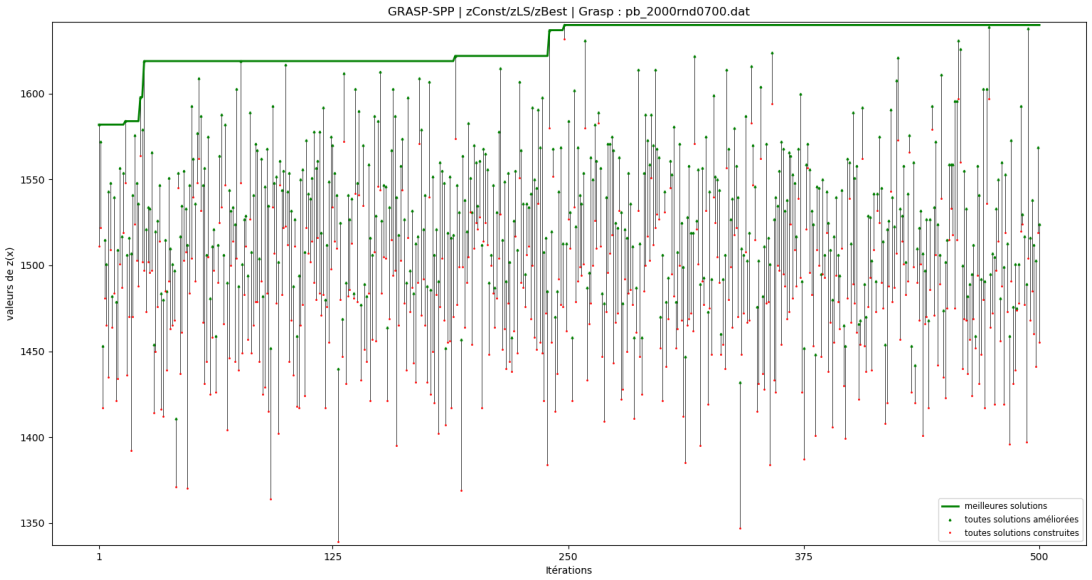
Cette méthode est très intéressante car elle nous permet de ne plus avoir à choisir arbitrairement le paramètre α mais elle introduit aussi deux autres "paramètres" arbitraires :

- N_α : le nombre d'itérations de Grasp avant qu'on ne modifie les probabilités est à première vue arbitraire mais il est possible de discuter du fait de le régler en fonction du nombre de variables.
- La discrétisation des α : En effet, on discrétise α entre 0 et 1 mais on le fait arbitrairement. C'est pourquoi une représentation plus souple se rapprochant d'un α continu pourrait être intéressante à étudier. Pour prendre en compte des valeurs d'α entre 0 et 1, il serait possible de choisir aléatoirement un α entre 0 et 1 selon des probabilités qui seraient liés à un intervalle de valeurs qui serait réglable selon les qualités des solutions.

En reprenant l'exemple didactique, pour construire la solution initiale, un α est choisi parmi le vecteur d'α résultant de la discrétisation du paramètre. On reprend ensuite la même démarche que dans le GRASP pendant N_α itérations. Une fois ce nombre d'itération atteint, si certains α ont contribué à de meilleurs solutions, leurs propabilités d'apparitions sont augmentées. Cette étape est répétée plusieurs fois, ce qui permet de détecter les α les plus efficaces et de les favoriser pour les itérations suivantes. Une fois le critère d'arrêt atteint, on ne garde que la meilleure solution trouvée.

Expérimentation numérique de GRASP

Exemple 1 *Une éxecution de Grasp sur le Pb 2000rnd0700 avec* $\alpha = 0.5$



Pourcentage de précision de Grasp sur 10 instances pour différents α pendant 10 secondes

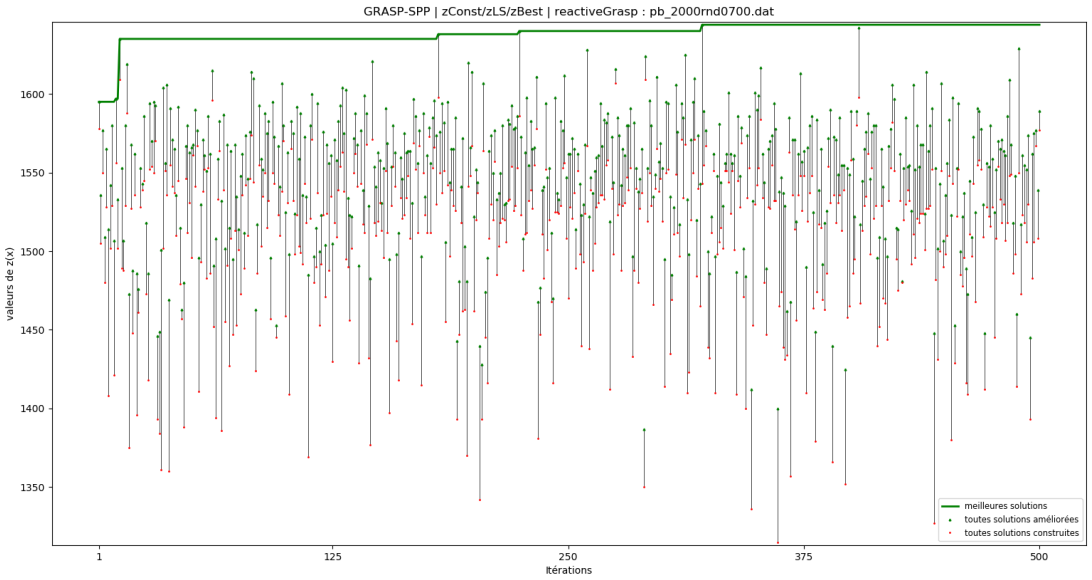
nom de l'instance	$\alpha = 0.2$	$\alpha = 0.4$	$\alpha = 0.5$	$\alpha = 0.6$	$\alpha = 0.7$	$\alpha = 0.8$	$\alpha = 0.9$	$\alpha = 0.95$
pb1000rnd0300	76.25	82.9	83.36	86.23	86.69	89.41	87.29	88.2
pb1000rnd0700	93.81	94.07	96.24	95.18	95.13	95.04	93.63	93.01
pb100rnd0500	100	100	100	100	99.06	98.12	98.12	98.12
pb2000rnd0700	78.63	88.63	88.4	88.63	90.78	90.56	89.56	88.07
pb200rnd0100	99.28	98.8	97.84	100	96.88	96.88	95.91	95.19
pb200rnd0300	96.03	96.58	96.99	96.17	96.85	96.58	96.31	95.76
pb200rnd0700	98.9	99.7	99.5	98.71	98.8	98.21	97.71	96.41
pb200rnd0900	99.62	100	99.85	99.85	99.85	99.62	99.62	99.55
pb500rnd0700	97.63	97.72	98.6	99.21	99.74	98.69	96.67	96.67
pb500rnd0900	97.54	98.75	98.88	98.66	98.93	98.97	98.88	98.52

Remarques :

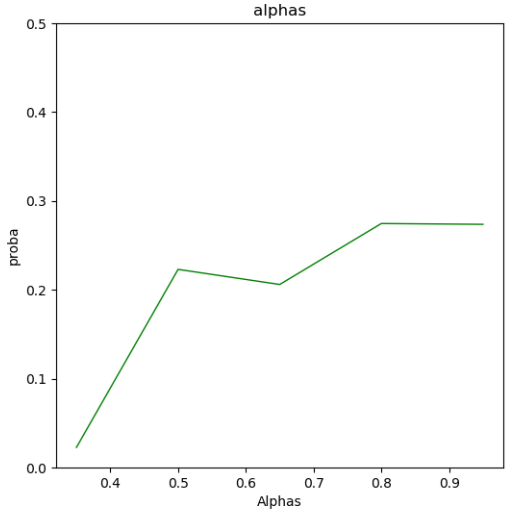
- On observe que le choix du paramètre est important et peut entrainer un écart de précision de plus de 10%.
- On remarque aussi que les grandes instances et les instances particulières qui étaient lentes sur les k-p échanges font moins d'itérations de Grasp que les rapides, et donc qu'il est important de régler le N_α en conséquence.

Expérimentation numérique de ReactiveGRASP

Exemple 2 *Une éxecution du réactive Grasp sur le Pb 2000rnd0700*



Distribution finale d'α :



Pourcentage de précision de Réactive Grasp sur 10 instances pour α dans $[0.35, 0.5, 0.65, 0.8, 0.95]$. Avec $N_\alpha = 100$ sur 500 itérations.

nom de l'instance	α dominant	précision
pb1000rnd0300	0.8	90.92%
pb1000rnd0700	0.65	96.19%
pb100rnd0500	0.5	100%
pb2000rnd0700	0.8	90.78%
pb200rnd0100	0.95	98.8%
pb200rnd0300	0.95	96.31%
pb200rnd0700	0.65	98.41%
pb200rnd0900	0.95	100%
pb500rnd0700	0.95	98.95%
pb500rnd0900	0.95	99.02%

Remarques :

- On retrouve sensiblement les mêmes précisions qu'avec les meilleurs Grasp et les α dominants correspondent, sauf sur les instances où le paramètre n'avait pas beaucoup d'influence (par exemple le pb 200rnd0300)
- On a donc réussi à régler le paramètre α mais au prix d'une discrétisation arbitraire et d'un nouveau paramètre N_α.

Eléments de contribution au bonus

Path-relinking

Le principe du path-relinking[1] est de pousser plus loin les recherches de solutions de bonne qualité à travers l'espace des solutions. Nous partons de solutions trouvées par plusieurs lancés du reactiveGrasp sur une même instance. Nous essayons de partir d'une solution pour rejoindre l'autre en réalisant un swap aléatoire sur les variables qui ont des valeurs différentes dans les deux solutions, tout en restant dans l'espace de solutions admissibles. Sur ce parcours entre les deux, dès qu'une solution trouvée par un swap semble prometteuse (défini arbitrairement), nous réalisons une descente profonde pour rechercher s'il n'y a pas un optimimum, au moins local, plus intéressant.

Observations : On remarque d'une part, peu d'améliorations sur les petites instance et un path relinking qui boucle sur certaines combinaisons de solutions de départ. Mais d'autre part des améliorations conséquentes (jusqu'à 5%) sur certaines grosses instances.

Nous avons essayé deux définitions de solution prometteuse, soit on considérait toutes les instances admissibles trouvées prometteuses sans distinction soit on ne prenait que les solutions dont la fonction objectif était supérieur à la solution de départ.

Parallélisme

Le parallélisme consiste à envoyer plusieurs parties de nos calculs aux différents coeurs d'une machine et de mutualiser les résultats pour gagner du temps. Cette méthode se révèle très pratique pour une application du path relinking. On calcule en parallèle deux solutions avec reactive Grasp et on les utilise ensuite pour faire un path relinking. En julia on utilise la fonction @parallel pour paralléliser les appels de reactive Grasp et la fonciton append ! comme réducteur pour concaténer nos solutions dans un vecteur de solutions.

Nous n'avons utilisé le parralélisme seulement pour le path-relinking mais il serait possible de le mettre en oeuvre avec le GRASP et le ReactiveGRASP pour réduire les temps de calculs.

Discussion

Le problème du Grasp est que son paramètre est arbitraire et peut beaucoup jouer sur la qualité de la solution. Le Réactive grasp résoud ce problème en apprenant sur les données le α le mieux adapté à celles-ci. Néanmoins, cet apprentissage a besoin d'être réglé aussi et une fois fait, certaines instances restent peu propices à cet apprentissage. Le path-relinking et le parallélisme permettent d'améliorer les solutions en peu de temps de calculs sur les grandes instances.

Remarques :

- On observe que nos heuristiques sont globalement très mauvaises en temps (par rapport au simplexe) sur les données qui ont des fonctions objectifs uniformes (où toutes les variables ont le même poids)
- On remarque aussi que la meilleure solution est parfois trouvée par les deux résolutions parallèles. Cela rend le path relinking inutile. On peut en conclure que limiter les grasp par leur nombre d'itérations est plus sécurisant quant à l'efficacité de l'heuristique. Malheureusement, la limite réelle est souvent temporelle.

Pour les heuristiques à paramètres, il faut être conscient qu'un bon réglage des paramètres est essentiel pour un bon fonctionnement des heuristiques. Ce réglage peut être fait par l'observation et les tests ou par un apprentissage automatisé.

Références

[1] Xavier Delorme, Xavier Gandibleux, and Joaquin Rodriguez. Grasp for set packing problems. *European Journal of Operational Research*, 153(3) :564–580, 2004.