Université de Nantes — UFR Sciences et Techniques
Master informatique parcours “optimisation en recherche opérationnelle (ORO)”
Année académique 2020-2021

Dossier Devoirs Maison

Méta-heuristiques

Duc Anh LE[1] – Daoud OUSMAN[2]

19 octobre 2021

Livrable du devoir maison 2 :
Méta-heuristique GRASP, ReactiveGRASP et extensions

Présentation succincte de GRASP appliqué sur le SPP

Algorithm 1: greedyRandomizedConstruction

Input: Le vecteur des coefficients C, la matrice des contraintes A, le paramètre GRASP a
Output: x, z, $Einit$

function *greedyRandomizedConstruction(C, A, a)*
  $x \leftarrow [0, .., 0]$
  $S \leftarrow \{1, .., n\}$ l'ensemble d'indice de variable disponible
  $M \leftarrow \{1, .., m\}$ l'ensemble d'indice de contrainte disponible
  E_i l'ensemble d'indice de variable actuellement en conflit avec x_i, E l'ensemble de $[E_i]$
  E_{init} l'ensemble E au problème initial
  F_i l'ensemble d'indice de contrainte contient x_i
  U_i le profit actuel de variable x_i (pour évaluer x_i)
  $firstloop \leftarrow true$
  while $S \neq \emptyset$ **do**
    if *length(S) = 1* **then**
      $i_{sel} \leftarrow S_i$
    else
      Mettre à jour E et F en fonction des variables disponible dans S et des contraintes disponibles dans M
      Pour chaque variables disponible $x_{i \in S}$, calculer son profit $U_i \leftarrow C_i - \sum_{v \in E_i} C_v$ et obtenir U_{max}, U_{min}
      $limit \leftarrow U_{min} + a(U_{max} - U_{min})$
      $RCL \leftarrow \{i \in S \mid U_i \geq limit\}$
      Choisir i_{sel} aléatoire dans RCL
    if $firstloop$ **then**
      $E_{init} \leftarrow E$
      $firstloop \leftarrow false$
    $x_{isel} \leftarrow 1$
    Mettre à jour les variables disponibles $S \leftarrow S \backslash (E_{isel} \cup i_{sel})$
    Mettre à jour les contraintes disponibles $M \leftarrow M \backslash F_{isel}$
  return x, z, E_{init}

Exemple 1 :

$$
\begin{array}{rccccccl}
max\ z = & 2x_1 & + 4x_2 & + 3x_3 & + 7x_4 & + 3x_5 & & \\
s.c & x_1 + & & x_3 + & & x_5 & \leq 1 \\
 & & x_2 + & & x_4 + & x_5 & \leq 1 \\
 & x_1 + & x_2 & & & & \leq 1 \\
 & x_1 & & & & & \leq 1 \\
 & & & x_3 & & & \leq 1 \\
 & & & & x_4 + & x_5 & \leq 1
\end{array}
$$

$$x_i \in \{0,1\},\ \alpha = 0.7$$

Première boucle dans *greedyRandomizedConstruction*, on calculer les ensembles E, F, U et la valeur *limit* pour trouver les candidats :

x = [0,0,0,0,0] ; S = [1,2,3,4,5] ; M = [1,2,3,4,5,6]

E = E_{init} = [[2,3,5], [1,4,5], [1,5], [2,5], [1,2,3,4]]

F = [[1,3,4], [2,3], [1,5], [2,6], [1,2,6]]

U = [-8,-8,-2,0,-13] ; $U_{max} = 0$; $U_{min} = -13$; $limit = -13 + 0.7(0 - (-13)) = -3.9$

Donc on trouve RCL = [3,4] ; choisir aléatoirement $i_{sel} \in RCL$; $i_{sel} = 3$; x = [0,0,1,0,0]

On obtient le nouvel problème pour la deuxième boucle dans *greedyRandomizedConstruction* :

X = [0,0,1,0,0], S = $S \backslash (E_3 \cup \{3\})$ = [2,4], M = $M \backslash F_3$ = [2,3,4,6]

On re-initie et recalculer les ensembles E, F et U :

E = [[], [4], [], [2], []]

F = [[], [2,3], [], [2,6], []]

U = [$-\infty$, -3, $-\infty$, 3, $-\infty$] ; $U_{max} = 3$; $U_{min} = -3$; $limit = -3 + 0.7(3 - (-3)) = 1.2$

Donc on trouve RCL = [4] ; un seul candidat à prendre $i_{sel} = 4 \in RCL$; x = [0,0,1,1,0]

S = $S \backslash (E_4 \cup \{4\})$ = []

S est vide donc on sortir de la boucle, la fonction *greedyRandomizedConstruction* retourne x = [0,0,1,1,0], z = 10 et l'ensemble E_{init} pour entrer dans la fonction d'amélioration.

Algorithm 2: GRASP

Input: Le vecteur des coefficients C, la matrice des contraintes A, le paramètre GRASP a, le temps d'exécution $finish$

Output: x_{max}, z_{max}

function *GRASP(C, A, a, finish)*
    $z_{max} \leftarrow 0$
    $x_{max} \leftarrow [0, .., 0]$
    $t \leftarrow 0$ le temps d'exécution actuel
    while $t \leq finish$ **do**
        x_{init}, z_{init}, $E_{init} \leftarrow$ *greedyRandomizedConstruction(C, A, a)*
        x, $z \leftarrow$ *GreedyImprovement(C, A, xInit, zInit, Einit)* (méthode de plus profonde descente, portée de DM1)
        $t' \leftarrow$ temps d'exécution les 2 fonction au dessus
        $t \leftarrow t + t'$
        if $z > z_{max}$ **then**
            $z_{max} \leftarrow z$
            $x_{max} \leftarrow x$
    return x_{max}, z_{max}

Exemple 2 :

$$\begin{array}{lccccccl}
\multicolumn{8}{c}{max\ z = 2x_1 + 4x_2 + 3x_3 + 7x_4 + 3x_5} \\
s.c & x_1 + & & x_3 + & & x_5 & \leq 1 \\
 & & x_2 + & & x_4 + & x_5 & \leq 1 \\
 & x_1 + & x_2 & & & & \leq 1 \\
 & x_1 & & & & & \leq 1 \\
 & & & x_3 & & & \leq 1 \\
 & & & & x_4 + & x_5 & \leq 1 \\
\multicolumn{8}{c}{x_i \in \{0,1\},\ \alpha = 0.7,\ finish = 10}
\end{array}$$

$z_{max} \leftarrow 0$; $x_{max} \leftarrow [0, 0, 0, 0, 0]$; $t \leftarrow 0$

$t = 0 < finish = 10$ donc on entre dans la boucle.

On appelle la fonction *greedyRandomizedConstruction* et avoir ses retours $x_{init} = [0, 0, 1, 1, 0]$, $z_{init} = 10$, l'ensemble E_{init} comme les entrées dans la fonction d'amélioration *GreedyImprovement* (méthode de plus profonde descente, portée de DM1). Cette fonction d'amélioration retourne toujours x = [0,0,1,1,0] et z = 10.

On trouve $z = 10 > z_{max} = 0$ donc $z_{max} = 10$ et $x_{max} = [0, 0, 1, 1, 0]$.

$t = t + t' = 0 + t' = t'$ (t' le temps d'exécution de *greedyRandomizedConstruction* et *GreedyImprovement*). On répète la boucle avec ces 2 fonctions jusqu'à $t > finish = 10$ et retourne z_{max}, x_{max}.

Présentation succincte de ReactiveGRASP appliqué sur le SPP

Algorithm 3: ReactiveGRASP

Input: Le vecteur des coefficients C, la matrice des contraintes A, le vecteur de m paramètre GRASP a, le temps d'exécution $finish$, le nombre d'itération effectué pour changer les poids N

Output: x_{max}, z_{max}

function *reactiveGRASP(C, A, a, finish, N)*
- $z_{max} \leftarrow 0$
- $z_{min} \leftarrow 0$
- $z_{moy} \leftarrow [0, .., 0]$ l'ensemble de k élément contient la valeur z moyenne obtenir avec a_k
- $x_{max} \leftarrow [0, .., 0]$
- $t \leftarrow 0$ le temps d'exécution d'itération GRASP cumulé
- $p \leftarrow [\frac{1}{m}, .., \frac{1}{m}]$ vecteur de poids
- $nbIter \leftarrow 1$ nombre d'itération GRASP actuel
- **while** $t \leq finish$ **do**
  - **if** *(nbIter - 1) mod N = 0 et nbIter* $\neq$ *1* **then**
    - **if** $z_{max} = z_{min}$ **then**
      - Cas spécial, solution optimale trouvée en première itération GRASP, sortir de la boucle
    - **else**
      - Calculer $q_k \leftarrow \frac{z_{moyk} - z_{min}}{z_{max} - z_{min}}$ et $p_k \leftarrow \frac{q_k}{\sum_{k=1}^{m} q_k}$
  - Tirer a_k
  - x_{init}, z_{init}, $E_{init} \leftarrow$ *greedyRandomizedConstruction(C, A, a_k)*
  - x, $z \leftarrow$ *GreedyImprovement(C, A, xInit, zInit, Einit)* (méthode de plus profonde descente, portée de DM1)
  - $t' \leftarrow$ temps d'exécution la itération GRASP (les 2 fonction au dessus)
  - $t \leftarrow t + t'$
  - **if** *nbIter = 1* **then**
    - $z_{max} \leftarrow z_{min} \leftarrow z$ et $x_{max} \leftarrow x$
  - **else if** $z > z_{max}$ **then**
    - $z_{max} \leftarrow z$ et $x_{max} \leftarrow x$
  - **else if** $z < z_{min}$ **then**
    - $z_{min} \leftarrow z$
  - Mettre à jour la valeur de z_{moyk}
  - $nbIter \leftarrow nbIter + 1$
- **return** x_{max}, z_{max}

Exemple 3 :

$$
\begin{array}{llllllll}
max\ z = 2x_1 + 4x_2 + 3x_3 + 7x_4 + 3x_5 \\
s.c & x_1 + & & x_3 + & & x_5 & \leq 1 \\
& & x_2 + & & x_4 + & x_5 & \leq 1 \\
& x_1 + & x_2 & & & & \leq 1 \\
& x_1 & & & & & \leq 1 \\
& & & x_3 & & & \leq 1 \\
& & & & x_4 + & x_5 & \leq 1
\end{array}
$$

$$x_i \in \{0,1\},\ \alpha = [0.2, 0.7],\ finish = 10,\ N = 3$$

$z_{max} = z_{min} = 0$; $z_{moy} = [0,0]$; $x_{max} = [0,0,0,0,0]$; $t = 0$; $p = [\frac{1}{2}, \frac{1}{2}]$; $nbIter = 1$
$t = 0 < finish = 10$ donc on entre dans la boucle.

$nbIter = 1$ donc on n'entre pas dans la condition et tire $a_k = 0.2$.

On appelle la fonction *greedyRandomizedConstruction* avec $a = a_k = 0.2$ et avoir ses retours $x_{init} = [1,0,0,1,0]$, $z_{init} = 9$, l'ensemble E_{init} comme les entrées dans la fonction d'amélioration *GreedyImprovement* (méthode de plus profonde descente, portée de DM1). Cette fonction d'amélioration retourne x = [0,0,1,1,0] (échanger valeur de x_1 avec x_3) et z = 10.
C'est la première itération donc $z_{max} = z_{min} = z = 10$ et $x_{max} = [0,0,1,1,0]$.
On cumule $t = t'$ et $z_{moy} = [10,0]$. Si $t' > 10$ (normalement impossible) on sort de la boucle, sinon on continue la prochaine itération GRASP.

$nbIter = 2$ donc on n'entre pas dans la condition et tire $a_k = 0.2$ toujours.

On appelle la fonction *greedyRandomizedConstruction* avec $a = a_k = 0.2$ et avoir ses retours $x_{init} = [0,1,1,0,0]$, $z_{init} = 7$, l'ensemble E_{init}. La fonction d'amélioration *GreedyImprovement* retourne aussi x = [0,0,1,1,0] (échanger valeur de x_2 avec x_4) et z = 10.
$z = z_{max} = z_{min} = 10$ donc on ne modifie pas les valeurs de z_{max} et z_{min}.
On cumule $t = t''$ et $z_{moy} = [10,0]$. Si $t'' > 10$ (normalement impossible) on sort de la boucle, sinon on continue la prochaine itération GRASP.

$nbIter = 3$ donc on n'entre pas dans la condition et tire $a_k = 0.7$.

On appelle la fonction *greedyRandomizedConstruction* avec $a = a_k = 0.7$ et la fonction d'amélioration *GreedyImprovement* pour obtenir les valeurs x = [0,0,1,1,0] et z = 10 comme dans l'exemple 1 ci-dessus.
$z = z_{max} = z_{min} = 10$ toujours donc on ne modifie pas les valeurs de z_{max} et z_{min}.
On cumule $t = t'''$ et $z_{moy} = [10,10]$. Si $t''' > 10$ (normalement impossible) on sort de la boucle, sinon on continue la prochaine itération GRASP.

$nbIter = 4$ donc on entre dans la condition car (4-1) mod 3 = 0.

$z_{max} = z_{min}$ donc on est dans le cas spécial : la solution optimale est trouvée en première itération GRASP. On sort de la boucle et retourner $x_{max} = [0,0,1,1,0]$, $z_{max} = 10$.

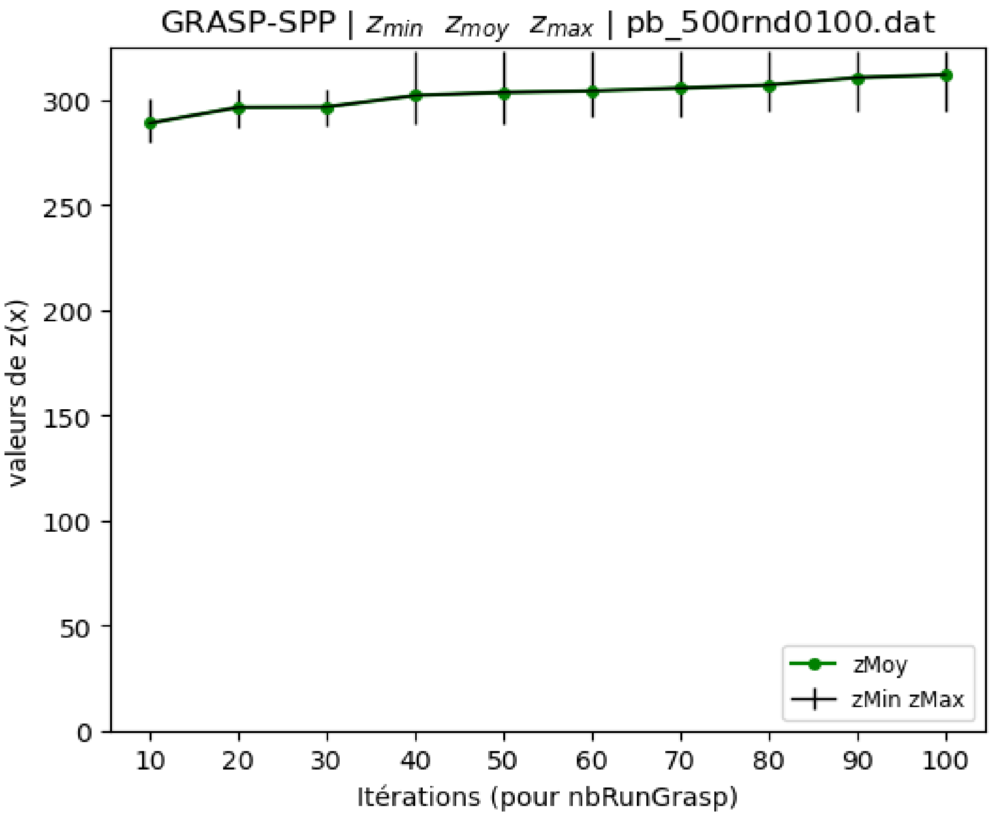
Expérimentation numérique de GRASP

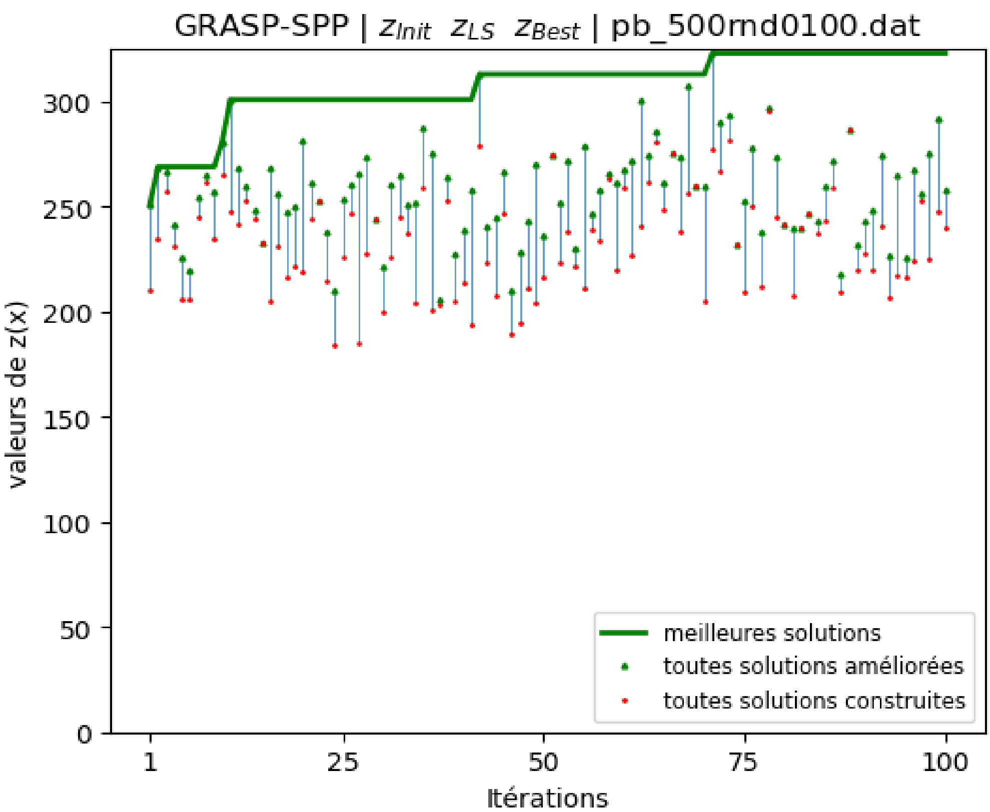
L'environnement machine sur lequel les algorithmes vont tourner (référence) :
CPU : AMD Ryzen 7 5700U, 8 cœurs 16 threads, fréquence 1.8 - 4.3 GHz
RAM : 8 Go, fréquence 3200 Mhz
Julia version 1.5.3

L'expérimentation graphique (compiler "experiment.jl") :
Nombre de fois que la résolution GRASP est répétée : 10
Nombre d'itération que compte une résolution GRASP : 100
Nombre de division que compte une résolution GRASP : 10
L'instance sélectionnées :

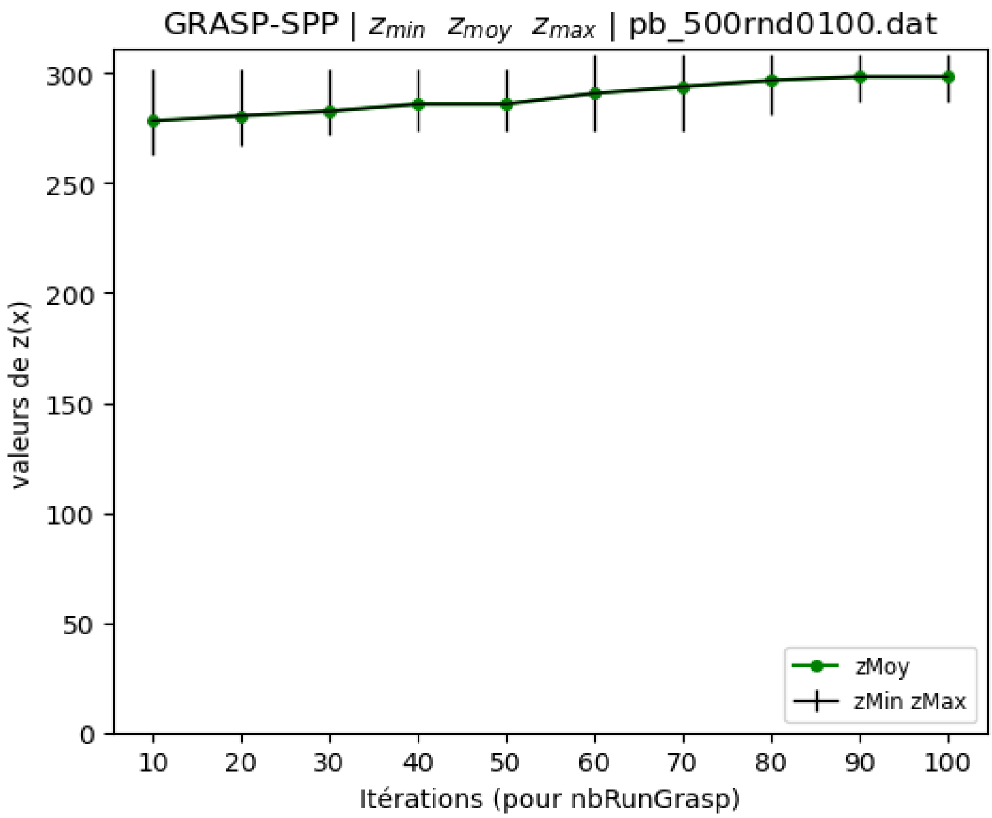
Instance	Nb variables	Nb contraintes	Densite	Max-Uns	Meilleur valeur connue
pb_500rnd0100.dat	500	2500	1,23	10	323

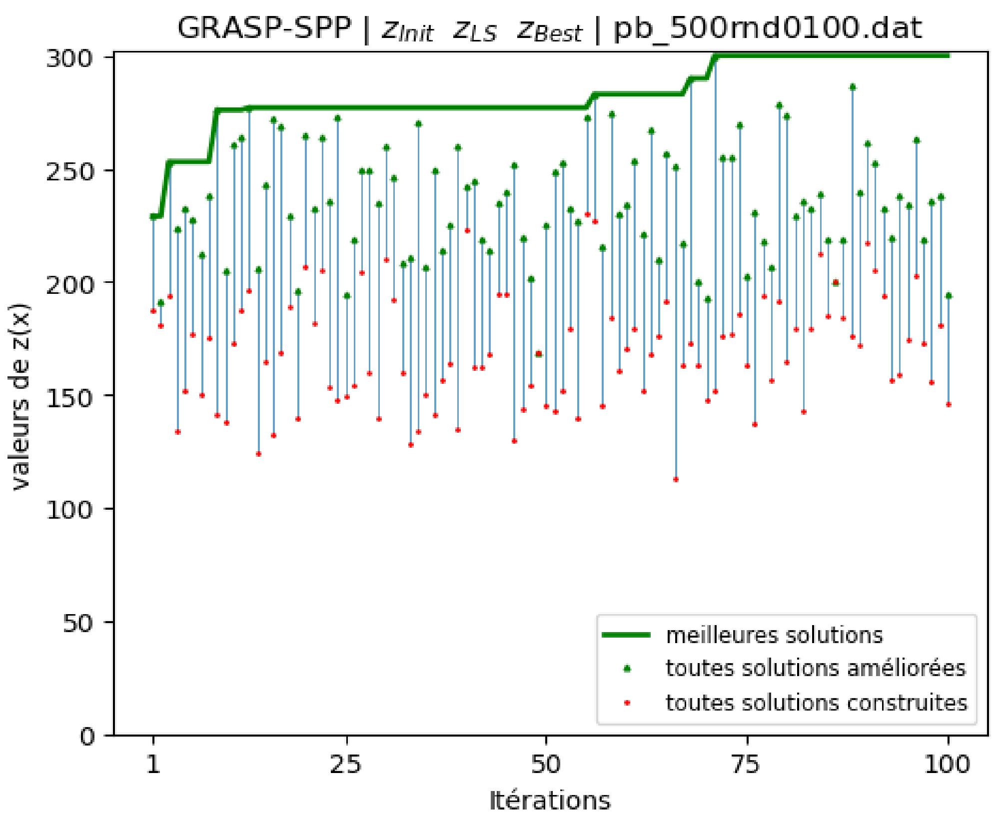
Pour $\alpha = 0.75$, on obtient les deux graphiques suivants avec $CPUt_{moyenne} \approx 59.4$ seconds :





Pour $\alpha = 0.25$, on obtient les deux graphiques suivants avec $CPUt_{moyenne} \approx 47.2$ seconds :





L'influence du paramètre α :

On peut observer qu'avec α est petit, le temps d'exécution moyenne $CPUt_{moyenne}$ est plus petit qu'avec α est grande. Par contre, le nombre d'itération nécessaire pour trouver une bonne solution est plus grande et le domaine d'oscillation de z est beaucoup plus grande qu'avec α est grande. Cela veut dire qu'avec α est petit, la qualité de solution est plus mauvais. On peut confirmer cette affirmation par observer et comparer les valeurs de $z_{moyenne}$ entre $\alpha = 0.25$ et $\alpha = 0.75$: les valeurs de $z_{moyenne}$ avec $\alpha = 0.25$ se montent plus lent et elles sont plus petites.

Les résultats numériques obtenus pour les 10 instances sélectionnées dans DM 1 (compiler "dm2.jl") :

Budget de calcul : 10 fois
Condition(s) d'arrêt pour chaque itération : 120 seconds
Réglage des paramètres : $\alpha = 0.7$

Instance	Meilleure valeur connue	zBest en DM1	zMin	zMax	zMoyenne
pb_1000rnd0100.dat	67	39	53	57	54.5
pb_1000rnd0400.dat	48	43	43	44	43.5
pb_100rnd0100.dat	372	372	372	372	372.0
pb_100rnd1100.dat	306	300	306	306	306.0
pb_2000rnd0100.dat	40	30	33	40	38.0
pb_200rnd0100.dat	416	396	415	416	415.8
pb_200rnd0600.dat	14	11	14	14	14.0
pb_500rnd0100.dat	323	280	306	323	313.4
pb_500rnd0900.dat	2236	2180	2208	2225	2215.1
pb_500rnd1200.dat	33	32	33	33	33.0

Expérimentation numérique de ReactiveGRASP

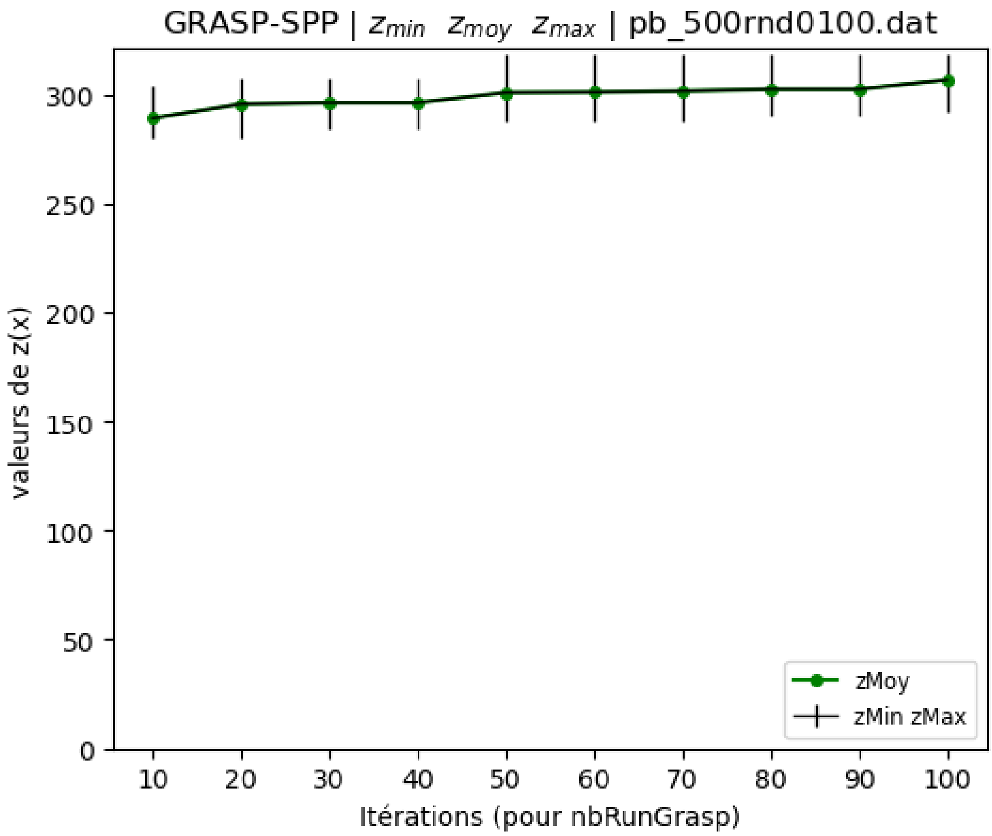
L'environnement machine sur lequel les algorithmes vont tourner (référence) :

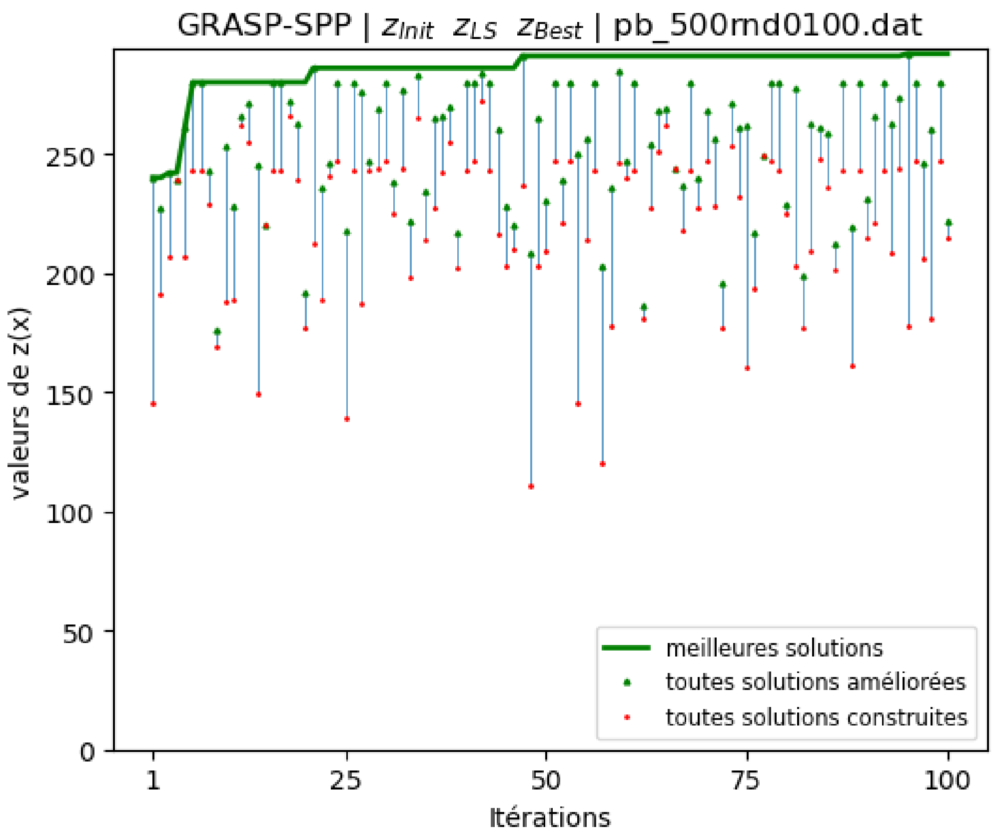
CPU : AMD Ryzen 7 5700U, 8 cœurs 16 threads, fréquence 1.8 - 4.3 GHz
RAM : 8 Go, fréquence 3200 Mhz
Julia version 1.5.3

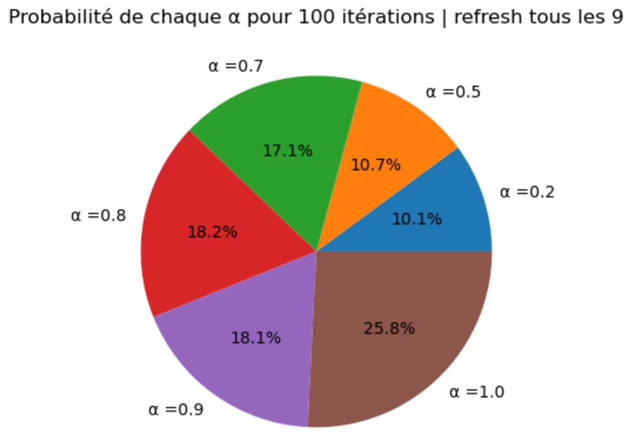
L'expérimentation graphique (compiler "experiment.jl") :

Nombre de fois que la résolution ReactiveGRASP est répétée : 10
Nombre d'itération que compte une résolution ReactiveGRASP : 100
Nombre de division que compte une résolution ReactiveGRASP : 10
Réglage des paramètres : $\alpha \in [0.2, 0.5, 0.7, 0.8, 0.9, 1]$ et $N = 9$
L'instance sélectionnées :

Instance	Nb variables	Nb contraintes	Densite	Max-Uns	Meilleur valeur connue
pb_500rnd0100.dat	500	2500	1,23	10	323


GRASP-SPP | z_{min} z_{moy} z_{max} | pb_500rnd0100.dat
valeurs de z(x)
Itérations (pour nbRunGrasp)
zMoy
zMin zMax


GRASP-SPP | z_{Init} z_{LS} z_{Best} | pb_500rnd0100.dat
valeurs de z(x)
Itérations
meilleures solutions
toutes solutions améliorées
toutes solutions construites



Avec $CPUt_{moyenne} \approx 60.7$ seconds, on peut conclure que $CPUt_{moyenneReGRASP} \approx CPUt_{moyenneGRASP}$ avec α est grande.

Le domaine d'oscillation est très grande mais la valeur de z_{best} se monte assez vite.

On peut observer que les probabilités de α sont plus enclines à les grandes α même si elles sont égales à l'initial. Cela veut dire que le méta-heuristique ReactiveGRASP a automatiquement réglé ces valeurs en fonction de la qualité de solution pour choisir le meilleur α que possible.

Les résultats numériques obtenus pour les 10 instances sélectionnées dans DM 1 (compiler "dm2.jl") :

Budget de calcul : 10 fois

Condition(s) d'arrêt pour chaque itération : 120 seconds

Réglage des paramètres : $\alpha \in [0.2, 0.5, 0.7, 0.8, 0.9, 1]$ et $N = 17$

Instance	Meilleure valeur connue	zBest en DM1	zMin	zMax	zMoyenne
pb_1000rnd0100.dat	67	39	52	57	54.5
pb_1000rnd0400.dat	48	43	43	46	44.6
pb_100rnd0100.dat	372	372	372	372	372.0
pb_100rnd1100.dat	306	300	306	306	306.0
pb_2000rnd0100.dat	40	30	32	40	39.2
pb_200rnd0100.dat	416	396	413	416	415.7
pb_200rnd0600.dat	14	11	14	14	14.0
pb_500rnd0100.dat	323	280	301	323	311.2
pb_500rnd0900.dat	2236	2180	2220	2230	2225.6
pb_500rnd1200.dat	33	32	33	33	33.0

Éléments de contribution au bonus

Discussion

Le GRASP marche très bien si on connaît la bonne valeur de α. Le ReactiveGRASP a un très grande domaine d'oscillation de solution donc il peut être plus lent, mais après un nombre précis d'itération, il peut régler les probabilités de α afin de choisir une de ses bonnes valeurs. Donc pour la conclusion, on peut totalement utiliser ReactiveGRASP en premier temps pour trouver une bonne valeur de α et le mettre en œuvre avec GRASP afin d'obtenir les avantages de tous ces deux variantes de méta-heuristique.