Université de Nantes

Département d'Informatique

Livrable de l'Exercice d'Implémentation 2

Métaheuristique GRASP, ReactiveGRASP et extensions

Étudiants : Sèdjro Amos GANDONOU
Florias TOKOTCHI

Enseignant : Pr. Xavier Gandibleux

Formation : Master 1 – ATAL (Apprentissage et Traitement Automatique de la Langue)

Année académique : 2025–2025

Nantes, le 19 novembre 2025

Livrable de l'exercice d'implémentation 2 : Métaheuristique GRASP, ReactiveGRASP et extensions

Présentation succincte de GRASP appliqué sur le SPP

0.1 Présentation de l'algorithme GRASP

La méthode **GRASP** (Greeedy Randomized Adaptive Search Procedure) est une métaheuristique qui allie les heuristiques gloutonne et d'amélioration en ajoutant à la première une compose aléatoire. C'est donc un algorithme glouton mais avec le fort mérite de proposer des solutions très variables.

Algorithm 1: Métaheuristique GRASP

Data: Instance du SPP : matrice A, vecteur de coûts C
Result: Meilleure solution x^* et sa valeur Z^*
$\alpha \in [0, 1]$: paramètre de randomisation du GRASP
x^* : meilleure solution

while *critère d'arrêt non atteint* **do**
    Générer une solution x par l'heuristique gloutonne aléatoire avec α
    $x \leftarrow greedyRandomizedConstruction(A, C, \alpha)$
    Générer une meilleur solution avec l'heuristique d'amélioration ;
     $x' \leftarrow localSearchImprovement(x)$
    Mise à jour de la solution
    $updateSolution(x^*, x')$
end
return x^*, Z^*

Algorithm 2: Heuristique gloutonne GRASP (greedyRandomizedConstruction)

Data: Instance du SPP : matrice A, vecteur de coûts C
Ensemble des candidats Q, Sous ensembles des candidats choisis S
Result: Solution gloutonne S

$\alpha \in [0, 1]$: paramètre de randomisation du GRASP
$S \leftarrow \varnothing$
while $(Q \neq 0)$ **do**
  Evaluer l'utilité de chaque variable $q \in Q$
  $u_{limit} \leftarrow min_{q \in Q} + \alpha \times (max_{q \in Q} - min_{q \in Q})$
  Construire liste restreinte de candidats (RCL)
  $RCL \leftarrow \{q \in Q, u(q) \geq u_{limit}\}$
  Tirage aléatoire d'une solution de la RCL
  $q \leftarrow RandomSelect(RCL)$
  Mise à jour des solutions
  $S \leftarrow S \cup q$
  Mise à jour de l'ensemble des candidats
  $Q \leftarrow Q \setminus conflict(\{q\})$
end
return S

Algorithm 3: localSearchImprovement

while *(S non localement optimal)* **do**
  Trouver $S' \in \mathcal{N}(S) \mid Z_{S'} > Z_S$
  $S \leftarrow S'$
end
return S

0.2 Illustration de l'algorithme

Nous reprenons notre exemple de l'exercice d'implémentation 1 avec $\alpha = 0.5$

$$j = \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix}$$

$$c_j = \begin{matrix} 6 & 5 & 4 & 7 & 3 \end{matrix}$$

$$a_{ij} = \begin{matrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}$$

0.2.1 Heuristique gloutonnne GRASP

Itération 1 :

$$\begin{aligned}
\sum_{i\in I} a_{ij} = &\quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \\
u(x_j) = &\quad 6 \quad 5 \quad 4 \quad \tfrac{7}{2} \quad \tfrac{3}{2} \qquad \Rightarrow u_{\text{limit}} = 3.75 \\
RCL_1 = &\quad 1 \quad 2 \quad 3 \qquad\qquad\quad \Rightarrow j^* = 3
\end{aligned}$$

Les lignes utilisées sont donc : $Used = \{2\}$

Itération 2 :

$$\begin{aligned}
\sum_{i\in I} a_{ij} = &\quad 1 \quad 0 \quad \times \quad 2 \quad 2 \\
u(x_j) = &\quad 6 \quad \times \quad \times \quad \tfrac{7}{2} \quad \tfrac{3}{2} \qquad \Rightarrow u_{\text{limit}} = 3.75 \\
RCL_2 = &\quad 1 \qquad\qquad\quad \Rightarrow j^* = 1
\end{aligned}$$

Les lignes utilisées sont donc : $Used = \{2, 1\}$

Itération 3 :

$$\begin{aligned}
\sum_{i\in I} a_{ij} = &\quad \times \quad 0 \quad \times \quad 2 \quad 1 \\
u(x_j) = &\quad \times \quad \times \quad \times \quad \tfrac{7}{2} \quad 3 \qquad \Rightarrow u_{\text{limit}} = 3.25 \\
RCL_3 = &\quad 4 \qquad\qquad\quad \Rightarrow j^* = 4
\end{aligned}$$

Les lignes utilisées sont donc : $Used = \{2, 1, 3, 4\}$
La solution initiale trouvée avec l'heuristique gloutonne est donc :

$$x = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \end{bmatrix} \quad avec\ Z = 17$$

0.2.2 Heuristique d'amélioration

Soit le kp-exchange, l'heuristique d'amélioration.
Dans le voisinage $kp - echange$ avec $k = 1$ et $p = 1$ nous obtenons six (06) solutions voisines :

$$x_k|Z = \left[\begin{array}{ccccc|c} 0 & 1 & 1 & 1 & 0 & 16 \\ 0 & 0 & 1 & 1 & 1 & 14 \\ 1 & 1 & 0 & 1 & 0 & 18 \\ 1 & 0 & 0 & 1 & 1 & 16 \\ 1 & 1 & 1 & 0 & 0 & 15 \\ 1 & 0 & 1 & 0 & 1 & 13 \end{array}\right]$$

La solution optimale est donc :

$$x = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \end{bmatrix} \text{ avec } Z = 18$$

La vérification de la satisfaction des contraintes par celle se prouve ci-dessous :

$$Ax = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Présentation succincte de ReactiveGRASP appliqué sur le SPP

0.3 Présentation de l'algorithme ReactiveGRASP

L'algorithme GRASP repose sur une composante $\alpha \in [0, 1]$. On peut se poser la question : ***Pour quelle valeur de*** α ***obtient on une meilleur solution ?*** C'est de cette inquiétude que naît l'algorithme ReactiveGRASP proposé par **Marcelo Prais** et **Celso C. Ribeiro**.

Principe de l'algorithme

1. Définir m valeurs de α avec les mêmes probabilités de départ $p_k = \frac{1}{m}$
2. Effectuer N_α itération GRASP pour chaque valeur α_k
3. Déterminer parmi les $m \times N_\alpha$ coûts obtenus le meilleur Z_{best} et le pire Z_{worst}
4. Calculer pour chaque α_k, son $q_k = \frac{Z_{avgK} - Z_{worst}}{Z_{best} - Z_{worst}}$
5. Mettre à jour les probabilités des $\alpha_k \mid p_k = \frac{q_k}{\sum_{i=1}^{m} qi}$
6. Reprendre n fois les étapes précédentes.
7. Selectionner α_j ayant la plus grande probabilité de sélection p_j
8. Retourner le meilleur coût Z_j obtenu avec α_j

0.4 Illustration du fonctionnement du ReactiveGRASP

Nous considérons l'instance précédent du *Set Packing Problem* :

$$C = \begin{array}{ccccc} 6 & 5 & 4 & 7 & 3 \end{array}, \qquad A = \begin{array}{ccccc} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{array}$$

Prenons trois valeurs du paramètre α à savoir : $\alpha_1 = 0.25$, $\alpha_2 = 0.50$, et $\alpha_3 = 0.75$, avec des probabilités initiales égales $p_k^{(0)} = 1/3$. Six itérations du GRASP sont effectuées et les probabilités sont réactualisées toutes les trois itérations.

Nous présentons ci-dessous deux situations possibles :

— le **Cas 1** où les résultats sont stables : l'aléatoire ne modifie pas les moyennes,

— le **Cas 2** où la variabilité des coûts entraîne une réelle évolution des probabilités.

Cas 1 : stabilité des résultats

Dans ce premier scénario, toutes les valeurs de α produisent des coûts Z très proches, ce qui conduit à des probabilités constantes d'un bloc à l'autre.

TABLEAU 1 – Cas 1 : valeurs Z obtenues à chaque itération.

Itération	$\alpha = 0.25$	$\alpha = 0.50$	$\alpha = 0.75$
1	18	18	18
2	18	18	18
3	17	17	18
4	18	18	18
5	17	17	18
6	18	18	18

TABLEAU 2 – Cas 1 : mises à jour des probabilités (identiques sur les deux blocs).

α	Z du bloc			Z_{avg}	$(Z_{\text{best}}, Z_{\text{worst}})$	q_k	p_k
0.25	18	18	17	17.67	(18, 17)	0.67	0.29
0.50	18	18	17	17.67	(18, 17)	0.67	0.29
0.75	18	18	18	18.00	(18, 17)	1.00	0.43

Dans ce cas, l'instance étant très simple, les solutions gloutonnes atteignent rapidement la même qualité. Les probabilités associées aux valeurs de α restent donc stables. Nous obtenons une solution optimale $Z^* = 18$ pour $\alpha = 0.75$

Cas 2 : évolution des probabilités sous l'effet de l'aléatoire

Dans ce second scénario, des tirages aléatoires différents conduisent à des moyennes Z légèrement distinctes entre blocs. L'algorithme ajuste alors ses probabilités pour favoriser les paramètres les plus performants.

TABLEAU 3 – Cas 2 : résultats du bloc 1 et première mise à jour des probabilités.

α	Z du bloc			Z_{avg}	$(Z_{\text{best}}, Z_{\text{worst}})$	q_k	p_k
0.25	17	18	17	17.33	(18, 17)	0.33	0.23
0.50	18	18	18	18.00	(18, 17)	1.00	0.45
0.75	18	17	18	17.67	(18, 17)	0.67	0.32

TABLEAU 4 – Cas 2 : résultats du bloc 2 et deuxième mise à jour des probabilités.

α	Z du bloc			Z_{avg}	$(Z_{\text{best}}, Z_{\text{worst}})$	q_k	p_k
0.25	17	17	17	17.00	(18, 17)	0.00	0.00
0.50	18	18	18	18.00	(18, 17)	1.00	0.60
0.75	18	18	17	17.67	(18, 17)	0.67	0.40

Dans ce second cas, la valeur $\alpha = 0.5$ s'avère la plus robuste et voit sa probabilité de sélection croître au fil des mises à jour avec $Z^* = 18$.

Conclusion. Ces deux expériences illustrent le rôle de l'aléatoire dans le comportement du *ReactiveGRASP*. Dans des cas stables, les probabilités demeurent constantes. En revanche, lorsque la variabilité des solutions influe sur les moyennes de coût, le mécanisme adaptatif oriente la recherche vers les valeurs de α les plus performantes.

(Scénarios illustratifs du principe du ReactiveGRASP)

Expérimentation numérique de GRASP

0.5 Environnement d'implémentation

L'implémentation des heuristiques sur les dix (10) instances a été réalisé sur un ordinateur HP dont voici les caractéristiques :

- **Processeur :** Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.59 GHz
- **Mémoire RAM installée :** 8,00 Go (7,89 Go utilisable)
- **Carte graphique :** Intel(R) HD Graphics 520 (128 MB)
- **Type du système :** Système d'exploitation 64 bits, processeur x64
- **Édition :** Windows 10 Famille
- **Version :** 22H2

L'Environnement de développement intégré utilisé est Visual Studio Code version **1.105.1** pour l'exécution des différents programmes julia dans sa version **1.11.7**.

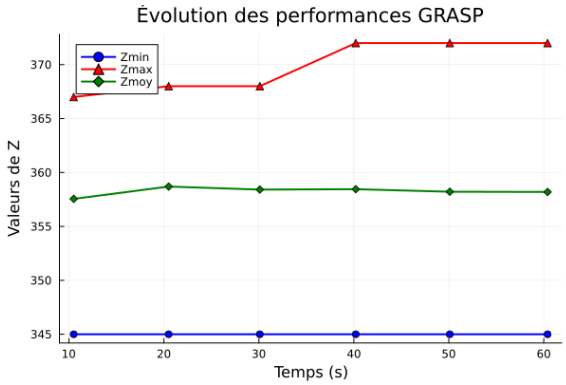
0.6 Résultats obtenus avec GRASP

On peut obtenir de meilleurs résultats avec l'algorithme GRASP sur nos 10 instances. Toute fois, des efforts extrêmes pour pousser aussi loin que possible la recherche de solutions meilleures aboutit sur une consommation énorme de temps.

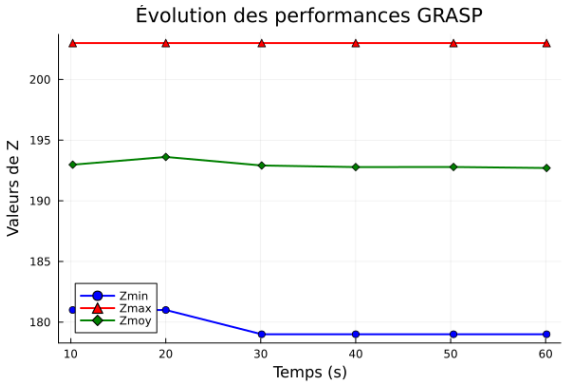
Nous avons donc choisi le temps comme critère d'arrêt de la recherche de solution avec GRASP. Soit $\Delta T = 60s$

0.6.1 Résultats de GRASP sur des intervalles de temps réguliers.

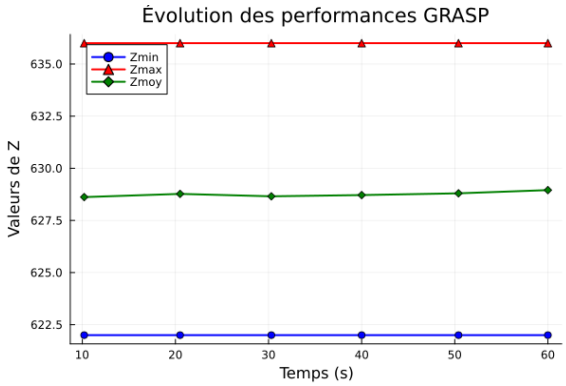
La figure 1 en desssous présentent, les valeurs moyenne $\hat{z}_{moy}$, minimale $\hat{z}_{min}$ et maximale $\hat{z}_{max}$ obtenue avec l'algorithme GRASP($\alpha = 0.6$) sur nos 10 instances par pas de 10 secondes. Il ressort que généralement après 30s maximum de recherche, GRASP se retrouve dans un optimum local.



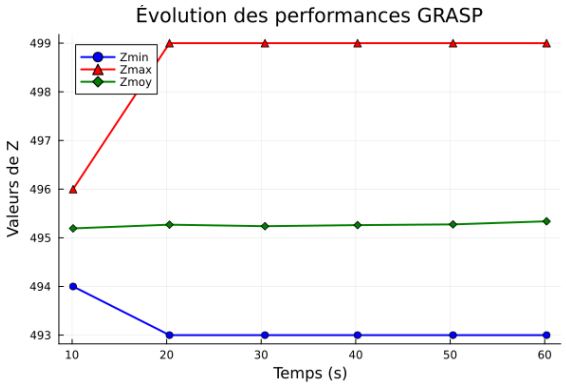
(a) $pb\_100rnd0100$



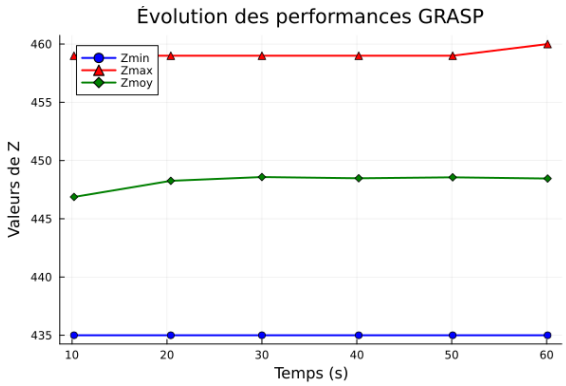
(b) pb_100rnd0300



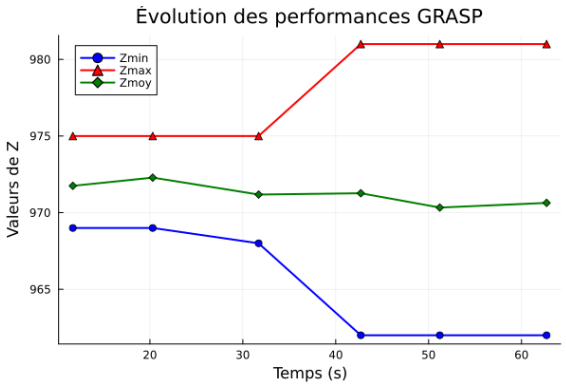
(c) pb_100rnd0500



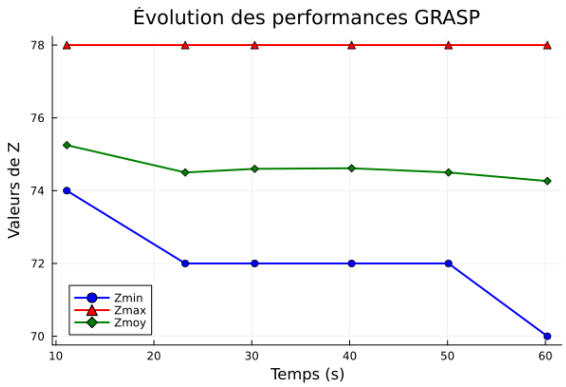
(d) pb_100rnd0700



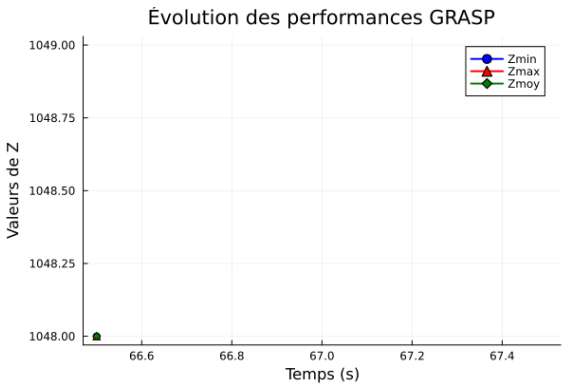
(e) pb_100rnd0900



(f) pb_200rnd0700



(g) pb_200rnd0800



(h) pb_500rnd1500

FIGURE 1 – Évolution de la qualité des solutions GRASP ($Z_{\min}$, $Z_{\max}$ et Z_{moy})

0.6.2 Influence du paramètre α

En analysant le tableau 5 on remarque que sur de petites instances, nous avons les meilleures solutions avec $\alpha = 0.6$. Mais sur de grandes instances, c'est $\alpha = 0.8$ qui l'emporte. Cela nous permet de conclure généralement qu'avec l'algorithme GRASP nous avons de bon résultat au fur et à mesure que le paramètre aléatoire se rapproche de 1 et que lorsque la taille des données et le nombre de contraintes augmentent, il faut prioriser des valeurs élevé du paramètre aléatoire α.

TABLEAU 5 – Comparaison des résultats de GRASP selon α pour 10 instances.

Instance	Z				t (s)			
	0.2	0.4	0.6	0.8	0.2	0.4	0.6	0.8
pb_100rnd0100	359	368	372	370	60.5	60.4	60.3	60.3
pb_100rnd0300	194	203	203	203	60.1	60.1	60.1	60.4
pb_100rnd0500	638	639	639	631	60.1	60.4	60.1	60.5
pb_100rnd0700	503	503	499	495	60.1	60.1	60.3	60.0
pb_100rnd0900	440	459	463	453	60.3	60.1	60.0	60.4
pb_200rnd0700	961	985	985	981	62.5	62.1	60.7	61.5
pb_200rnd0800	77	75	77	79	60.2	61.6	60.0	60.7
pb_500rnd1500	887	1023	1048	1075	100.1	61.2	62.1	65.3
pb_1000rnd0300	381	382	494	526	171.4	203.9	226.8	239.5
pb_2000rnd0700	1188	1526	1543	1589	1060.3	1640.9	2241.8	1723.8

Expérimentation numérique de ReactiveGRASP

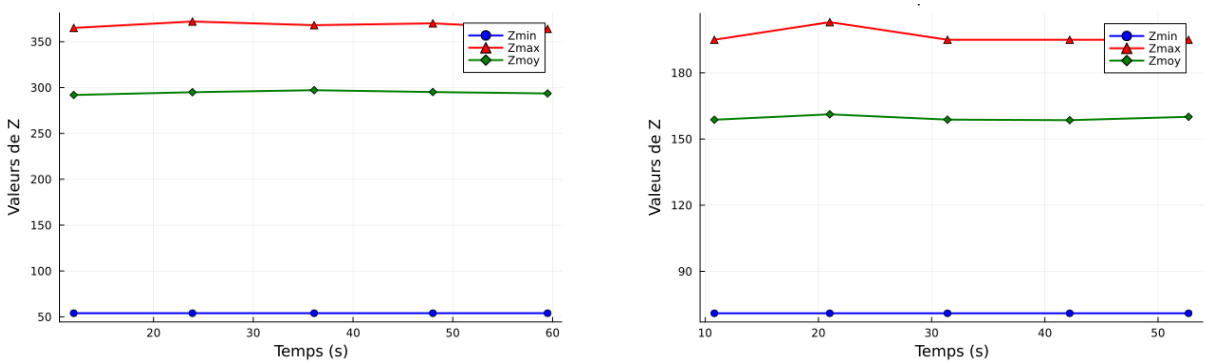
Nous avons implémenter l'algorithme ReactiveGRASP avec le même matériel. Par contre nous avons utilisé deux critères d'arrêt différents, l'un pour l'algorithme GRASP implémenté dans l'algorithme ReactiveGRASP, l'autre pour l'algorithme ReactiveGRASP lui-même.

Critères d'arrêt

1. Critère GRASP : Le nombre de runs $NRuns$ que nous avons fixé à 10 pour une exploration plus intensive à ce niveau de l'implémentation du ReactiveGRASP.
2. Critère ReactiveGRASP : Le temps $\Delta T = 60s$

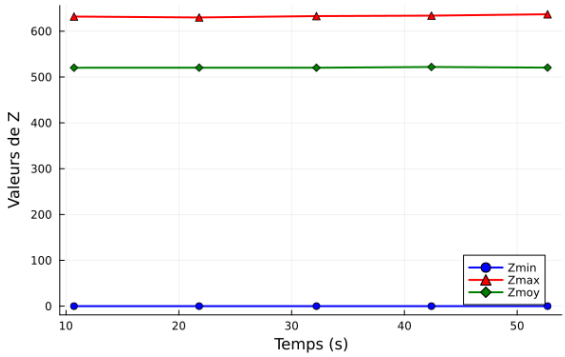
0.6.3 Résultats de ReactiveGRASP sur des intervalles de temps réguliers.

Les figures 2 et 3 en desssous présentent, les valeurs moyenne $\hat{z}_{moy}$, minimale $\hat{z}_{min}$ et maximale $\hat{z}_{max}$ obtenue avec l'algorithme ReactiveGRASP sur nos 10 instances par pas de 10 secondes. Il ressort qu'en 10s en général, ReactiveGRASP trouve un bon optimum local qu'il n'arrive pas à améliorer dans les temps qui lui sont impartis.

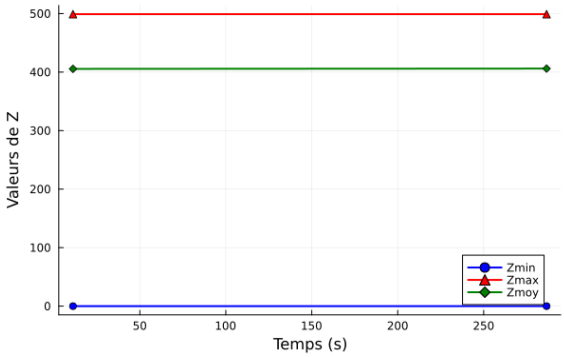


(a) pb_100rnd0100

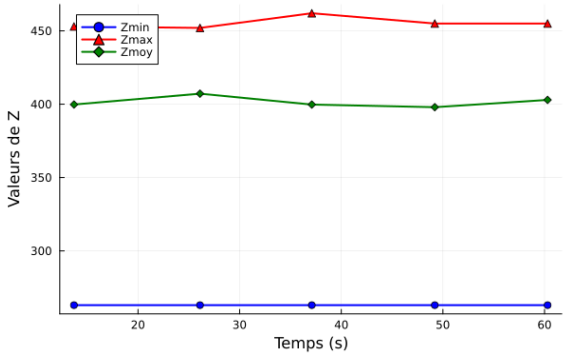
(b) pb_100rnd0300



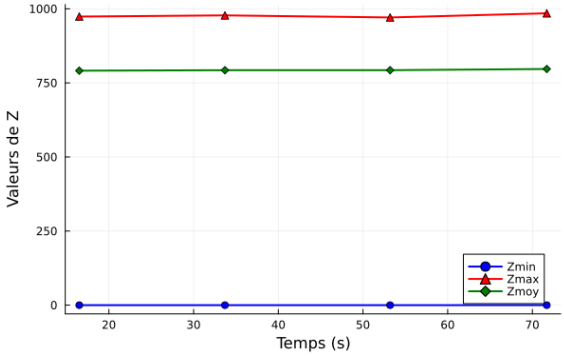
(c) pb_100rnd0500



(d) pb_100rnd0700

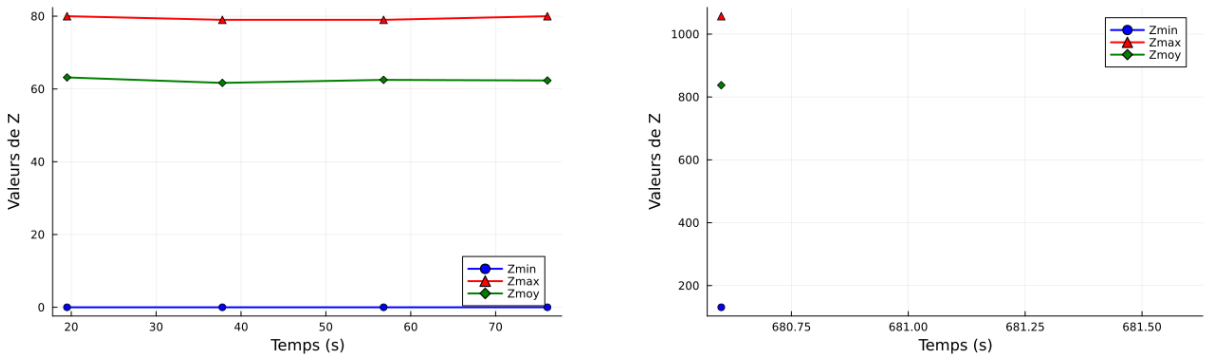


(e) pb_100rnd0900



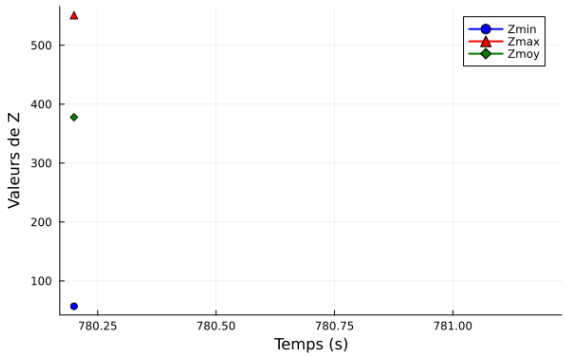
(f) pb_200rnd0700

FIGURE 2 – Évolution de la qualité des solutions ReactiveGRASP ($Z_{\min}$, $Z_{\max}$ et Z_{moy})

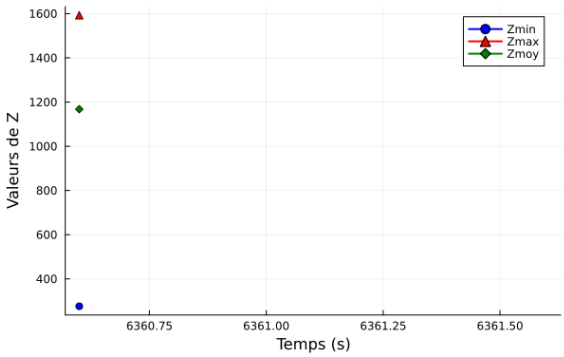


(a) pb_200rnd0800

(b) pb_500rnd1500



(c) pb_1000rnd0300



(d) pb_2000rnd0700

FIGURE 3 – Évolution de la qualité des solutions ReactiveGRASP (suite)

Présenter sous forme de tableau les résultats finaux obtenus pour les 10 instances sélectionnées.

0.6.4 Résumé des résultats de ReactiveGRASP

Le tableau 6 présente les résultats de l'apprentissage du paramètre α et les valeurs Z obtenus. On note que bien que l'algorithme ReactiveGRASP choisit $\alpha = 0.8\,(Z_{0.8} = 195)$, une solution meilleure est trouvée avec $\alpha = 0.6\,(Z_{0.6} = 203)$ pour l'instance `pb_100rnd0300`. Nous avons un résultat similaire avec d'autres instances telles que `pb_100rnd0500` et `pb_100rnd0900`. On conclut que plus α est grand, mieux est la solution. Cependant, cela ne garantit pas forcément d'avoir la meilleur des solutions.

TABLEAU 6 – Comparaison entre le α sélectionné par la probabilité P_k et celui associé à la meilleure valeur $Z_{\max}$

Instance	$\alpha^*_{(Pk)}$	$Z_{\max}(\alpha^*_{(Pk)})$	$\alpha^*_{(Z_{\max})}$	$Z_{\max}(\alpha^*_{(Z_{\max})})$	**Temps (s)**
pb_100rnd0100	0.6	372.0	0.8	368.0	62.1
pb_100rnd0300	0.8	195.0	0.6	203.0	60.3
pb_100rnd0500	0.6	632.0	0.6	637.0	60.7
pb_100rnd0700	0.4	499.0	0.4	499.0	286.7
pb_100rnd0900	0.8	453.0	0.6	462.0	60.3
pb_200rnd0700	0.6	985.0	0.8	978.0	71.7
pb_200rnd0800	1.0	80.0	0.8	79.0	76.1
pb_500rnd1500	0.8	1057.0	0.8	1057.0	680.6
pb_1000rnd0300	0.8	551.0	0.8	551.0	780.6
pb_1000rnd0300	0.8	1593	0.8	1593	6360.6

0.6.5 Influence du paramètre $N\alpha$ sur la stabilité des résultats

Le paramètre $N\alpha$ est gage de stabilité comme le révèle le tableau 7, il permet de converger vers des valeurs fixent de α, meilleurs candidats pour des solutions optimales. Cet exemple pourrait bien être illustré pour des valeurs de α telles que choisies précédemment.

TABLEAU 7 – Influence du paramètre $N_\alpha = 10$ sur la stabilité du α sélectionné et la performance du ReactiveGRASP (100 itérations).

Instance	$\alpha^*_{\text{dominant}}$	**Fréquence** (%)	$Z_{\max}$	**Variabilité de** α^*	**Temps (s)**
pb_100rnd0100	0.75	100	367.0	Stable	23.6
pb_100rnd0300	0.75 / 0.5	60 / 40	195.0	Faible	10.0
pb_100rnd0500	0.75 / 0.5	70 / 30	633.0	Faible	16.4
pb_100rnd0700	0.75	100	495.0	Stable	12.1
pb_100rnd0900	0.75 / 0.5	60 / 40	459.0	Modérée	22.6
pb_200rnd0700	0.75 / 0.5	90 / 10	983.0	Très stable	99.8
pb_200rnd0800	1.0	100	81.0	Stable	107.5
pb_500rnd1500	0.75	100	1079.0	Stable	2326.4
pb_1000rnd0300	0.75	100	556.0	Stable	6808.5
pb_2000rnd0700	0.75	100	1590.0	Stable	–

Discussion

Tel que le montre le tableau 8 l'heuristique GRASP l'emporte sur les autres heuristiques la plus part du temps. Le ReactiveGRASP à le mérite d'explorer un plus large éventaille de possibilité par la sélection du meilleur paramètre aléatoire. Cependant, il peut passer à côter de quelques solutions meilleures que celui du meilleur paramètre α.

Les instances `pb_1000rnd0300` et `pb_2000rnd0700` sont très grandes et nécessite du temps et de la puissance de calcul. Et pour celles-ci heuristiques ReactiveGRASP propose les meilleures solutions. Ce qui nous amène à dire, que ReactiveGRASP est plus percutant que GRASP pour des instances très grandes.

TABLEAU 8 – Résumé des résultats des heuristiques gloutonnes, GRASP et ReactiveGRASP

Instances	Z_{greedy}	*GRASP*		*ReactiveGRASP*	
		α	Z	α	Z
`pb_100rnd0100`	342	0.6	372	0.6	372
`pb_100rnd0300`	193	0.6	203	0.8	195
`pb_100rnd0500`	621	0.6	639	0.6	632
`pb_100rnd0700`	495	0.2	503	0.4	499
`pb_100rnd0900`	442	0.6	463	0.8	453
`pb_200rnd0700`	945	0.6	985	0.6	985
`pb_200rnd0800`	75	0.8	79	1.0	80
`pb_500rnd1500`	1059	0.8	1075	0.8	1057
`pb_1000rnd0300`	507	0.8	526	0.8	551
`pb_2000rnd0700`	1524	0.8	1589	0.8	1593