

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

Présentation succincte de GRASP appliqué sur le SPP

Présenter l'algorithme mis en oeuvre. Illustrer sur un exemple didactique (poursuivre avec l'exemple pris en DM1). Présenter vos choix de mise en oeuvre.

Le grasp que j'ai réalisé fonctionne de la manière suivante :

1. **Construction gloutonne randomisée** : On construit une solution initiale en sélectionnant itérativement des éléments à ajouter à la solution. À chaque étape, on crée une RCL. Un élément qui respecte les contraintes et qui a une valeur d'utilité supérieur au U_{limit} calculé comme ceci : $U_{limit} = u_{min} + \alpha \cdot (u_{max} - u_{min})$. Ensuite, on sélectionne aléatoirement un élément dans la RCL à ajouter à la solution.
2. **Amélioration locale** : Une fois la solution initiale construite, on applique notre descente locale avec un voisinage 1-1 pour améliorer la solution.
3. **Itérations** : Les étapes de construction et d'amélioration sont répétées un certain nombre de fois (ou jusqu'à une condition d'arrêt), et la meilleure solution trouvée au cours de ces itérations est retenue comme solution finale.

Présentation succincte de ReactiveGRASP appliqué sur le SPP

Présenter l'algorithme mis en oeuvre. Illustrer sur un exemple didactique (poursuivre avec l'exemple pris en DM1). Présenter vos choix de mise en oeuvre.

Mon ReactiveGRASP est une extension de GRASP classique qui ajuste dynamiquement les α au cours de l'exécution.

1. **Initialisation** : On définit un ensemble de valeurs de α à tester (dans notre cas : $\{0.1, 0.3, 0.5, 0.7, 0.9\}$). Chaque α se voit attribuer une probabilité initiale uniforme ($p_i = 0.2$ pour 5 valeurs).
2. **Sélection adaptative** : À chaque itération, on sélectionne un α selon une distribution de probabilités. Plus un α a produit de bonnes solutions, plus sa probabilité d'être sélectionné augmente.
3. **L'amélioration locale et la construction GRASP** : Garde le même fonctionnement que GRASP classique.
4. **Mise à jour des probabilités** : Tous les N itérations (dans notre cas, $N = 20$), on recalcule les probabilités selon la formule :

$$q_i = \left(\frac{\bar{z}_i}{z^*} \right)^k$$

où \bar{z}_i est la qualité moyenne des solutions obtenues avec α_i , z^* est la meilleure solution globale trouvée, et k est un paramètre d'intensification (ici $k = 5$).

Les probabilités sont ensuite normalisées : $p_i = \frac{q_i}{\sum_j q_j}$.

Expérimentation numérique de GRASP

Présenter le protocole d'expérimentation (environnement matériel; budget de calcul; condition(s) d'arrêt; réglage des paramètres).

1 Protocole d'expérimentation

1.1 Environnement matériel

Les algorithmes ont tourné sur la machine suivante :

- **Système d'exploitation** : Windows 11
- **Processeur** : AMD RYZEN 7 5700X 8-Core
- **Mémoire** : 16Go

1.2 Budget de calcul et conditions d'arrêt

L'expérimentation s'est déroulée en deux phases :

Premier temps Un test a été réalisé avec une condition d'arrêt basée sur le nombre d'itérations.

- **Condition d'arrêt** : 200 itérations.
- **Répétitions** : 3 fois par instance avec un alpha différent à chaque fois.
- **Instances** : 10 instances différentes.

Second temps Un test a été réalisé avec une condition d'arrêt basée sur le temps d'exécution.

- **Condition d'arrêt** : 60 secondes.
- **Répétitions** : 3 fois par instance avec un alpha différent à chaque fois.
- **Instances** : 10 instances différentes.

Rapporter graphiquement vos résultats selon \hat{z}_{min} , \hat{z}_{max} , \hat{z}_{moy} mesurés à intervalles réguliers (exemple de pas de 10 secondes).

Les résultats sont obtenus avec 3 runs avec chacun 3 alpha différents et 200 itérations par instances ($\alpha = \{0.1, 0.5, 0.9\}$).

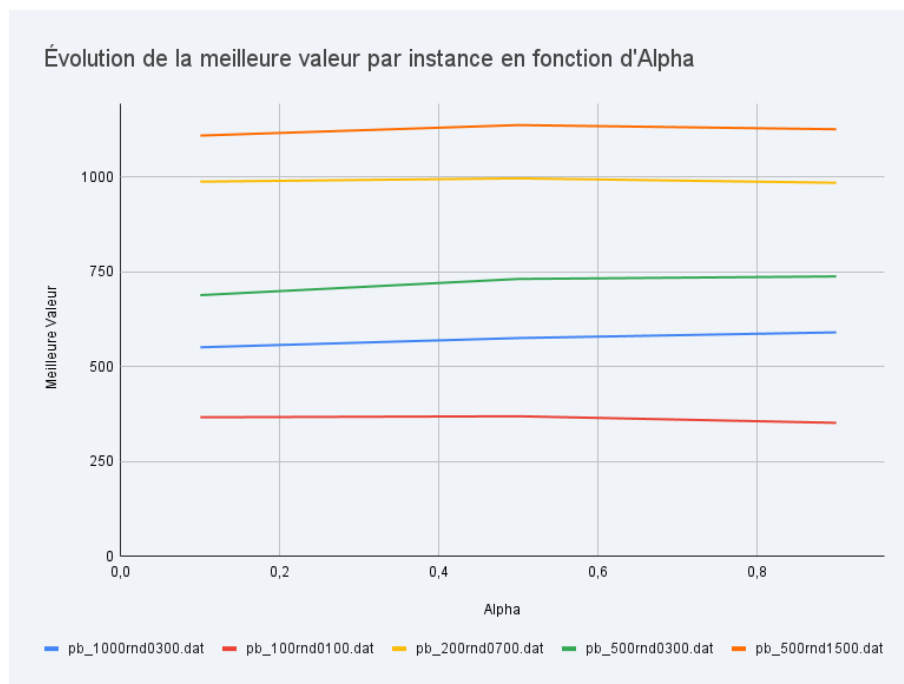


Figure 1: Z_{mean} .

Rapporter l'étude de l'influence du paramètre α .

Dans notre expérimentation, nous avons testé trois valeurs de α : 0.1, 0.5 et 0.9. Voici les observations principales :
Présenter sous forme de tableau les résultats finaux obtenus pour les 10 instances sélectionnées.

Table 1: Résultats finaux pour les 10 instances (200 itérations)

Instance	α	Val. best	CPU (s)
pb_100rnd0100.dat	0.1	368	0.66
	0.5	370	0.40
	0.9	352	0.40
pb_200rnd0100.dat	0.1	403	3.45
	0.5	408	2.38
	0.9	386	2.20
pb_500rnd0300.dat	0.1	718	53.13
	0.5	715	22.86
	0.9	736	15.19
pb_500rnd1500.dat	0.1	1093	78.01
	0.5	1135	41.52
	0.9	1122	30.15
pb_500rnd1700.dat	0.1	166	1.92
	0.5	170	1.47
	0.9	165	1.14
pb_200rnd0400.dat	0.1	59	1.81
	0.5	59	1.98
	0.9	57	2.04
pb_200rnd0700.dat	0.1	992	7.13
	0.5	992	2.17
	0.9	981	1.71
pb_1000rnd0100.dat	0.1	57	3.01
	0.5	67	3.14
	0.9	57	2.59
pb_1000rnd0300.dat	0.1	554	128.72
	0.5	590	64.70
	0.9	594	45.91
dat/didactic.dat	0.1	30	0.00
	0.5	30	0.00
	0.9	30	0.00

Table 2: Résultats finaux pour les 10 instances (60 secondes)

Instance	α	Val. best	CPU (s)	Itérations
pb_100rnd0100.dat	0.1	370	60.0	18640
	0.5	372	60.0	32237
	0.9	352	60.0	28761
pb_200rnd0100.dat	0.1	414	60.01	3724
	0.5	415	60.01	7362
	0.9	386	60.0	7483
pb_500rnd0300.dat	0.1	697	60.06	224
	0.5	724	60.08	481
	0.9	739	60.02	786
pb_500rnd1500.dat	0.1	1080	60.0	163
	0.5	1145	60.11	285
	0.9	1134	60.06	414
pb_500rnd1700.dat	0.1	180	60.0	8089
	0.5	192	60.0	11527
	0.9	165	60.0	11727
pb_200rnd0400.dat	0.1	61	60.0	8472
	0.5	60	60.0	6516
	0.9	57	60.0	6754
pb_200rnd0700.dat	0.1	991	60.03	2168
	0.5	1003	60.0	6888
	0.9	992	60.0	8420
pb_1000rnd0100.dat	0.1	67	60.0	5409
	0.5	67	60.01	5217
	0.9	57	60.01	5563
pb_1000rnd0300.dat	0.1	546	60.04	102
	0.5	573	60.37	228
	0.9	587	60.13	363
dat/didactic.dat	0.1	30	60.0	18693182
	0.5	30	60.0	29640773
	0.9	30	60.0	29826432

Expérimentation numérique de ReactiveGRASP

Présenter le protocole d'expérimentation (env. matériel; budget de calcul; condition(s) d'arrêt).

2 Protocole d'expérimentation

2.1 Environnement matériel

Les algorithmes ont tourné sur la machine suivante :

- **Système d'exploitation** : Windows 11
- **Processeur** : AMD RYZEN 7 5700X 8-Core
- **Mémoire** : 16Go

2.2 Budget de calcul et conditions d'arrêt

L'expérimentation s'est déroulée en deux phases :

Premier temps Un test a été réalisé avec une condition d'arrêt basée sur le nombre d'itérations.

- **Condition d'arrêt** : 200 itérations.
- **Répétitions** : 3 fois par instance.
- **Instances** : 10 instances différentes.

Second temps Un test a été réalisé avec une condition d'arrêt basée sur le temps d'exécution.

- **Condition d'arrêt** : 60 secondes.
- **Répétitions** : 3 fois par instance.
- **Instances** : 10 instances différentes.

Rapporter graphiquement vos résultats selon \hat{z}_{min} , \hat{z}_{max} , \hat{z}_{moy} mesurés à intervalles réguliers (exemple de pas de 10 secondes).

Les résultats sont obtenus avec 3 runs et les paramètre "classique" défini précédemment avec 200 itérations par instances.

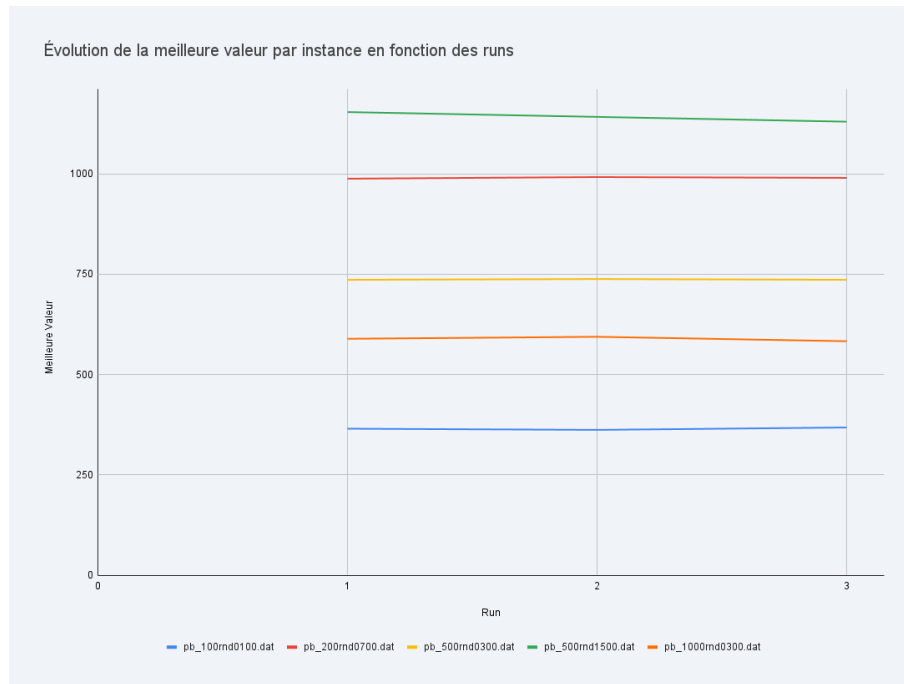


Figure 2: Z_mean.

Rapporter l'apprentissage du paramètre α réalisé par ReactiveGRASP, les valeurs saillantes établies.

Les valeurs de α ont été ajustées dynamiquement par ReactiveGRASP en fonction des performances observées. Voici un résumé des tendances observées :

- Les valeurs de α entre 0.1 et 0.3 ne donnent dans mon expérimentation jamais la meilleur solution.
- Les valeurs de α entre 0.5 et 0.7 peuvent être efficaces dans certains cas, dans nos données elles le sont surtout sur des instances de petite taille.
- Les α de 0.9 sont dans notre expérimentation les plus performantes sur la majorité des instances et sont sans concurrence sur les grandes instances.

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

Table 3: Résultats finaux pour les 10 instances (200 itérations)

Instance	Run	Val. Best	CPU (s)	α -Best	Val. Best Known
pb_100rnd0100.dat	1	368	0.58	0.7	372*
	2	366	0.44	0.9	372*
	3	368	0.40	0.7	372*
pb_200rnd0100.dat	1	399	1.84	0.7	416*
	2	403	1.98	0.5	416*
	3	406	2.04	0.5	416*
pb_500rnd0300.dat	1	739	22.41	0.9	776
	2	736	23.58	0.9	776
	3	736	21.47	0.9	776
pb_500rnd1500.dat	1	1129	35.66	0.9	1196
	2	1136	32.75	0.9	1196
	3	1130	34.60	0.9	1196
pb_500rnd1700.dat	1	167	1.23	0.9	192
	2	172	1.01	0.9	192
	3	173	1.03	0.9	192
pb_200rnd0400.dat	1	59	1.43	0.7	64*
	2	58	1.46	0.7	64*
	3	60	1.44	0.7	64*
pb_200rnd0700.dat	1	994	2.36	0.5	1004*
	2	988	2.27	0.5	1004*
	3	992	2.44	0.5	1004*
pb_1000rnd0100.dat	1	56	1.94	0.9	67*
	2	59	1.93	0.9	67*
	3	57	1.97	0.9	67*
pb_1000rnd0300.dat	1	584	39.12	0.9	661
	2	583	52.87	0.9	661
	3	603	52.80	0.9	661
didactic.dat	1	30	0.00	0.5	30
	2	30	0.00	0.5	30
	3	30	0.00	0.5	30

Table 4: Résultats de l'expérimentation avec une condition d'arrêt de 60 secondes.

Instance	Run	Val. Best	CPU (s)	Itér.	α -Best	Val. Best Known
pb_100rnd0100.dat	1	372	60.00	27385	0.7	372*
	2	372	60.00	25738	0.7	372*
	3	372	60.00	25736	0.7	372*
pb_200rnd0100.dat	1	414	60.00	5368	0.5	416*
	2	415	60.01	5720	0.5	416*
	3	414	60.02	5425	0.5	416*
pb_500rnd0300.dat	1	739	60.15	499	0.9	776
	2	739	60.15	456	0.9	776
	3	738	60.03	483	0.9	776
pb_500rnd1500.dat	1	1138	60.18	305	0.9	1196
	2	1139	60.26	308	0.9	1196
	3	1136	60.18	293	0.9	1196
pb_500rnd1700.dat	1	188	60.00	9333	0.9	192
	2	189	60.00	10278	0.9	192
	3	186	60.00	9451	0.9	192
pb_200rnd0400.dat	1	61	60.00	5973	0.7	64*
	2	61	60.01	6998	0.7	64*
	3	60	60.01	6284	0.9	64*
pb_200rnd0700.dat	1	1001	60.02	4634	0.5	1004*
	2	997	60.03	5102	0.5	1004*
	3	1001	60.02	4541	0.5	1004*
pb_1000rnd0100.dat	1	67	60.00	5429	0.9	67*
	2	67	60.00	5397	0.9	67*
	3	67	60.01	5359	0.9	67*
pb_1000rnd0300.dat	1	590	60.02	224	0.9	661
	2	594	60.14	244	0.9	661
	3	592	60.07	233	0.9	661
didactic.dat	1	30	60.00	31001331	0.5	30
	2	30	60.00	31299926	0.5	30
	3	30	60.00	30727593	0.5	30

Eléments de contribution au bonus

Présenter vos contributions aux aspects proposés en bonus.

Discussion

Tirer des conclusions en comparant les résultats collectés avec vos deux variantes de métaheuristiques.

En comparant les résultats de GRASP et ReactiveGRASP j'ai pu constater que ReactiveGRASP offre globalement de meilleurs résultats. J'ai aussi remarqué sur le paramétrage que les résultats obtenus avec une limite de temps sont meilleurs que ceux obtenus par un nombre d'itération fixe, ce qui n'est pas réellement surprenant car il réalise souvent plus d'itérations que celle fixée dans ma première expérimentation surtout quand les instances sont petites. Pour finir j'ai pu constater que les valeurs de α les plus élevées sont celles qui donnent les meilleurs résultats dans la majorité des cas.

Je recommande l'utilisation de ReactiveGRASP. Son implémentation est facile à mettre en place à partir du GRASP classique et produit des résultats meilleurs ou au moins équivalents dans tous les cas. L'auto-ajustement du paramètre α est particulièrement intéressant, car il offre un diagnostic sur l'efficacité de notre heuristique de construction. En effet, le fait que les meilleurs résultats soient obtenus avec un α petit ou grand indique si notre fonction gloutonne est performante ou si elle est souvent piégée dans des optima locaux, ce qui a été notre cas.