Sur l'apprentissage des valeurs du paramètre α de GRASP.

Master Informatique – Projet de recherche

Etudiants : Axel Verneuil et Athanaël Jousselin
Encadrants : Xavier Gandibleux et Celso Ribeiro

Table des matières

1 Introduction

1.1 Problématique de la recherche

Ce travail se penche sur la composante Reactive-GRASP, pour lequel des résultats préliminaires montrent un comportement sur l'évolution des probabilités liées à la procédure d'apprentissage qui questionne. En effet, la mise en œuvre de ce composant sur le problème d'optimisation combinatoire dit set packing problem (SPP) ne fait pas apparaître une installation des probabilités qui permettrait d'en déduire une forme de convergence claire de l'apprentissage.

Au contraire, on peut observer une quasi équiprobabilité des valeurs à chaque mise à jour des probabilités de répartition quand on se base sur le Reactive-GRASP originalement proposé en 2000[1]. Une première tentative a été proposée en 2019 par Xavier Gandibleux qui, elle, montre un grand brassage.

Ce comportement est-il inhérent aux instances numériques utilisées pour les expérimentations, au problème d'optimisation support (le set packing), au composant Reactive-GRASP ou un autre facteur ?

1.2 Questions étudiées

Nous cherchons à obtenir une convergence dans l'apprentissage qui permettrait de faire sortir un ou plusieurs α. Nous commencerons par nous pencher sur la littérature afin de voir s'il y a déjà eu des propositions. Puis nous poserons trois problèmes différents qui ont des rapports variés à GRASP afin de pouvoir généraliser nos observations.

Nous proposerons deux variations du Reactive-GRASP de 2000 dans l'objectif d'obtenir une convergence tout en gardant la simplicité et l'efficacité de Reactive-GRASP, nous les testerons sur les trois problèmes.

Dans un second temps, nous changerons de composant pour un Bandit-Manchot, qui reste très simple et efficace comme Reactive-GRASP. Nous le testerons aussi sur les trois problèmes.

Nous finirons par comparer toutes les variations et composant sur les solutions qu'ils donnent pour vérifier si cette recherche de la convergence n'a pas érodé la qualité des solutions trouvées par Reactive-GRASP.

2 Aperçu de la littérature

2.1 Description des algorithmes

2.1.1 Aperçu de GRASP

L'algorithme GRASP (Greedy Randomized Adaptative Search Procedure) est une métaheuristique pour les problèmes d'optimisation discrets qui a été introduit par Feo et Resende en 1989 [2].

Dans la première phase de GRASP, un algorithme glouton produit une solution initiale (ligne 1.3). L'algorithme est un mélange entre deux stratégies :
- D'une part une stratégie gloutonne se basant sur une fonction d'utilité. L'utilité est une fonction mathématique qui va mettre en place un tri selon une valeur de préférence pour chaque élément d'un ensemble. Par conséquent, cette stratégie retourne un classement parmi les éléments candidats à une solution réalisable.
- D'autre part, une stratégie aléatoire qui utilise un paramètre α. Ce paramètre permet de limiter les choix parmi les éléments candidats, afin d'appliquer de l'aléatoire dans la construction des solutions. L'ajout de l'aléatoire fait en sorte que l'algorithme ne soit plus déterministe.

La seconde est une phase de recherche locale (ligne 1.4), l'algorithme prend la solution construite et explore le voisinage afin de pouvoir trouver une meilleure solution. Plusieurs algorithmes tels que $add(), drop(), kp-exchange()$ permettent de modifier la structure d'une solution afin d'obtenir leurs voisins. Enfin, une règle d'arrêt est ajoutée en paramètre afin de définir le nombre d'itérations de ces deux phases (ligne 1.6).

Paramètres :
Soit le paramètre $\alpha \in [0; 1]$ représentant le compromis entre la stratégie gloutonne et aléatoire.
Soit le paramètre *stoppingCondition* représentant la condition d'arrêt pour le lancement des deux phases.

Algorithm 1 Métaheuristique GRASP

1: $Solutions \leftarrow \emptyset$
2: **repeat**
3: $\quad SolutionInitiale \leftarrow GreedyRandomizedConstruction(problem, \alpha)$
4: $\quad SolutionAmelioree \leftarrow LocalSearchImprovement(SolutionInitiale)$
5: $\quad Solutions \leftarrow Solutions \cup SolutionAmelioree$
6: **until** ($stoppingCondition$)
7: **return** best(Solutions)

2.1.2 Aperçu de Reactive-GRASP

Reactive-GRASP est une composante supplémentaire introduite par Preis et Ribeiro [1]. Le but de cet algorithme est de régler automatiquement le paramètre α de GRASP suivant la qualité des solutions rencontrées lors des itérations des deux phases de la métaheuristiques.
Le principe est le suivant : a chaque itération de GRASP, un α sera sélectionné parmi un tableau de plusieurs valeurs (ligne 2.4). elles possèdent une probabilité égale d'être choisie à l'initialisation (ligne 2.2). Au terme d'un nombre d'itération de GRASP défini par un paramètre N_α (ligne 2.8), Les probabilités seront recalculées (ligne 2.9-10) . Les formules prennent en compte la qualité de solution afin de "choisir" les α retournant de bonnes solutions.

Algorithm 2 Composante Reactive-GRASP

1: $Solutions \leftarrow \emptyset$
2: $proba_\alpha \leftarrow 1/|EnsembleAlpha|, \quad \forall \alpha \in EnsembleAlpha$
3: **repeat**
4: $\quad \alpha\_Itt \leftarrow SelectionAleatoire(EnsembleAlpha, proba)$
5: $\quad SolutionInitiale \leftarrow GreedyRandomizedConstruction(problem, \alpha\_Itt)$
6: $\quad SolutionAmelioree \leftarrow LocalSearchImprovement(SolutionInitiale)$
7: $\quad Solutions \leftarrow Solutions \cup SolutionAmelioree$
8: $\quad$ **if** MiseAJourProba(N_α) is True **then**
9: $\quad\quad q_\alpha \leftarrow \frac{zAVG_\alpha - zWorst}{zBest - zWorst}, \quad \forall \alpha \in EnsembleAlpha$
10: $\quad\quad p_\alpha \leftarrow \frac{q_\alpha}{\sum_{i=1}^{EnsembleAlpha} q_i}, \quad \forall \alpha \in EnsembleAlpha$
11: $\quad$ **end if**
12: **until** ($stoppingCondition$)
13: **return** best(Solutions)

2.2 Implémentation du Reactive-GRASP sur différents problèmes d'optimisation

L'implémentation utilisera le même algorithme pour tous les problèmes, les seules différences sont l'algorithme de construction (sur lequel GRASP s'appuiera), l'algorithme de descente et une différence pour un problème en maximisation ou en minimisation.

2.3 Littérature

2.3.1 Support de base en lien avec notre recherche :

Pour commencer, notre étude de la métaheuristique GRASP avec la composante Reactive-GRASP se base sur l'article d'Alvarez et Ribeiro [3]. Notre premier objectif était d'essayer de trouver un article qui parlait de l'apprentissage de la valeur α avec Reactive-GRASP. Après un temps de recherche, nous en avons conclu qu'il n'existait pas de travaux existant sur l'heuristique de construction, seulement des modifications sur la recherche locale. De plus, tous les articles sur Reactive-GRASP se basent sur celui d'Alvarez et Ribeiro [3] sans grandes modifications de celui-ci. Dans un second temps, pour la composante Bandit-Manchot. Nous avons eu à disposition l'article [4] afin de comprendre la partie mathématique derrière l'idée de ce problème.

2.3.2 Support utilisé dans le cadre de notre recherche :

Pour l'implémentation de Reactive-GRASP dans le problème du SPP, nous nous sommes basés sur l'article de Delorme [5]. Il traite de l'algorithme GRASP et d'une variation de la composante Reactive-GRASP. Pour ce qui est du Strip-Packing, nous avons utilisé l'heuristique présentée en détail dans l'article [6]. Toutes les heuristiques de recherche locale se basent sur plusieurs articles différents. Les heuristiques de recherche locale du SPP et du TSP se basent sur le support du cours de Xavier Gandibleux nommé "Métaheuristiques". Pour la seconde partie sur le problème des Bandit-Manchot, nous utilisons l'article [7] par Sutton et Barto. Il introduit une définition du problème et présente les différents algorithmes dans la littérature actuelle.

3 Formulation Mathématiques des trois problèmes choisis

Nous voulions généraliser quelque peu nos résultats, ainsi le choix a été fait de prendre trois problèmes qui marche plus ou moins bien avec GRASP. Le Set Packing Problem est supposé neutre par rapport à GRASP. Le Strip Packing Problem est supposé très favorable par rapport à GRASP et le Traveling Salesman Problem est supposé plutôt défavorable par rapport à GRASP.

3.1 Set Packing Problem (SPP) [5]

3.1.1 Contexte

Le problème de couplage généralisé est un problème de maximisation. Étant donné un vecteur profit c pour J activités et une matrice a indiquant quelles activités consomment les I ressources. Il faut maximiser le profit en sélectionnant des activités sans qu'une ressource ne soit utilisée plus d'une fois.

3.1.2 Modèle

Posons les variables du problème.

$$x_j = \begin{cases} 1 & \text{si on sélectionne l'activité } j \\ 0 & \text{sinon} \end{cases} \quad \forall j \in J$$

La fonction objectif est la somme des profits des activités sélectionnées. Nous voulons maximiser ce profit.

$$\max\ z \quad = \quad \sum_{j \in J} c_j x_j \tag{1}$$

Pour chaque ressource i, au plus une activité qui l'utilise peut-être sélectionnée.

$$\sum_{j \in J} a_{ij} x_j \leq 1 \qquad \forall i \in I \tag{2}$$

3.1.3 Heuristique de construction Gloutonne

- On initialise une liste de fonction d'utilité qui est calculée pour chaque activité comme le profit divisé par le nombre de ressource consommé, la liste est ordonnée dans l'ordre décroissant.
- L'heuristique choisit la première activité dans la liste de fonction d'utilité, donc celle avec le plus grand ratio. L'activité est ajoutée à la solution et est retirée de la liste de fonction d'utilité.
- On retire de la liste de fonction d'utilité toutes les activités qui ont au moins une ressource commune à l'activité choisie.
- On retourne au choix de la première activité jusqu'à ce que la liste soit vide.

3.1.4 Heuristique de descente Destroy and Repair

L'heuristique est divisée en deux parties et est limitée car très lente, elle ne serait pas utilisable dans une implémentation efficace, nous l'utilisons ici pour avoir une bonne descente.

La première partie est exécutée au plus 2 fois.

- Pour chaque activité dans la solution, on retire l'activité de la solution.
  - On calcule une liste de fonction d'utilité pour toutes les activités qui ne consomme que des ressources qui ne sont pas consommées par les activités qu'il reste dans la solution.

- L'heuristique choisit la première activité dans la liste de fonction d'utilité, donc celle avec le plus grand ratio. L'activité est ajoutée à la solution et est retirée de la liste de fonction d'utilité.
- On retire de la liste de fonction d'utilité toutes les activités qui ont au moins une ressource commune à l'activité choisie.
- On retourne au choix de la première activité jusqu'à ce que la liste soit vide.

— Dès qu'on trouve une solution améliorante on s'arrête et on ré-exécute la première partie si possible. Si on ne trouve aucune solution améliorante alors on s'arrête là et on passe à la deuxième partie.

Une fois qu'on a exécuté 2 fois la première partie ou que l'on a trouvé aucune solution améliorante on passe à la deuxième partie.

La deuxième partie est exécutée au plus 2 fois.

— Pour chaque couple d'activité dans la solution, on retire le couple d'activité de la solution.
  - On calcule une liste de fonction d'utilité pour toutes les activités qui ne consomme que des ressources qui ne sont pas consommées par les activités qu'il reste dans la solution.
  - ...

— Dès qu'on trouve une solution améliorante on s'arrête et on ré-exécute la deuxième partie si possible. Si on ne trouve aucune solution améliorante alors on s'arrête là.

3.2 Strip Packing Problem (StPP)

3.2.1 Contexte

Le problème de compactage en bande est un problème de minimisation géométrique bidimensionnel. Etant donné un ensemble I de n rectangles et une bande de largeur fixée W, le but est de déterminer une hauteur entière minimale H de la bande en la remplissant de tous les carrés. Les n rectangles doivent être placés sans chevauchement ni rotation possible. On commence par poser le coin bas-gauche de la bande comme origine de notre plan à deux dimensions. L'axe X représente la largeur W de la bande, l'axe Y représente la hauteur H à minimiser. Chaque rectangle appartenant à I est désigné par la coordonnée (x_i, y_i) représentant le coin bas-gauche du rectangle. On pose également l'ensemble $\pi = \{(x_i, y_i) | r_i \in I\}$ des coordonnées qui est appelé placement de I. Ce modèle du Strip packing Problem ci-dessous se base sur l'article de Arahori, Imamichi et Nagamochi [8].

3.2.2 Modèle

On pose les variables $(x_i, y_i),\ \forall\ r_i \in I$ représentant les coordonnées du carré i.
On a les données W la largeur de la bande et $h_i,\ w_i \forall r_i \in I$ représentant la hauteur et la largeur du carré i.

$$\min\ z \quad = \quad H \tag{3}$$

Les deux premières contraintes font en sorte de placer tous les rectangles de I dans la bande.

$$x_i + w_i \leq W \qquad\qquad r_i \in I \tag{4}$$

$$y_i + h_i \leq H \qquad\qquad r_i \in I \tag{5}$$

La troisième contrainte évite le chevauchement entre chaque rectangle de I.

$$(x_i + w_i \leq x_j\ ou\ x_j + w_j \leq x_i)\ \ ou\ (y_i + h_i \leq y_j\ ou\ y_j + h_j \leq y_i) \qquad\qquad r_i, r_j \in I, i \neq j \tag{6}$$

$$x_i, y_i \geq 0 \qquad\qquad r_i \in I \tag{7}$$

Un placement π est admissible sans rotation pour n'importe quelle instance s'il satisfait les 4 contraintes. Autrement, ce placement n'est pas possible à la coordonnée actuelle.

3.2.3 Heuristique de construction gloutonne

L'heuristique de construction gloutonne que nous avons utilisé se base sur celle présente dans l'article de Alvarez-Valdes, Parreño et Tamarit [3]. Leur heuristique est composée de cinq étapes :

— Initialisation, on pose un ensemble de rectangles vides $\lambda = \{S\}$, l'ensemble de pièces à placer $P = \{p_1, p_2, ..., p_m\}$ (ordonnée par la largeur et la longueur), Q_i le nombre de pièces de type i à placer. Enfin, C l'ensemble de pièces de type i qui pour lesquelles on a Q_i copies de placées.
— Choix du rectangle dans λ. On doit choisir le rectangle le plus gros pour lesquelles les autres pièces rentrent dedans.
— Choix de la pièce à placer. L'heuristique prend en compte plusieurs critères afin de choisir la pièce à placer. Le but est de choisir la pièce la plus importante en faisant en sorte de ne pas augmenter la hauteur de la bande le plus possible.
— Choix de la position dans la bande pour placer la pièce. La pièce sera forcément le plus bas possible de la bande, mais il reste à déterminer si elle va à gauche ou à droite.
— Mise à jour de la liste.

3.2.4 Heuristique de descente

Selon l'article Alvarez-Valdes, Parreño et Tamarit [3], GRASP est un algorithme très bien adapté pour le problème de compactage. Nous avons donc émis l'hypothèse qu'un algorithme de recherche locale n'améliorerait pas ou peu les solutions construites par l'algorithme glouton. C'est pour cela que l'implémentation ne possède pas d'heuristique de descente.

3.3 Traveling Salesman Problem (TSP)

3.3.1 Contexte

Le problème du voyageur de commerce est un problème de minimisation. Étant donné n villes (notées $N = \{1, 2, \dots, n\}$) et la distance entre chacune de ces villes (notée c_{ij} la distance entre la ville i et la ville j). Nous devons trouver le trajet le plus court passant exactement une fois dans chacune des villes. Le modèle du Traveling Salesman Problem retenu se base sur l'article de Miller, Tucker et Zemlin [9].

3.3.2 Modèle

Posons les variables du problème.

$$x_{ij} = \begin{cases} 1 & \text{si la route qui va de la ville } i \text{ à la ville } j \text{ est emprunté} \\ 0 & \text{sinon} \end{cases} \qquad \forall i, j \in N, i \neq j$$

$t_i \geq 0$, $i \in N \backslash \{1\}$ la date à laquelle la ville i est visitée. On exclut la ville 1 car cette ville commence et termine le voyage.
La fonction objectif est la somme du coût des routes empruntées, c'est à dire la distance parcourue. Nous voulons minimiser ce coût pour trouver le trajet le plus court.

$$\min \; z \quad = \quad \sum_{\substack{i,j \in N \\ (i \neq j)}} c_{ij} x_{ij} \tag{8}$$

Il est possible de partir d'une ville qu'une et une seule fois.

$$\sum_{\substack{j \in N \\ (j \neq i)}} x_{ij} = 1 \qquad \forall i \in N \tag{9}$$

Il est possible de rentrer dans une ville qu'une et une seule fois.

$$\sum_{\substack{i \in N \\ (i \neq j)}} x_{ij} = 1 \qquad \forall j \in N \tag{10}$$

Il faudrait s'assurer que si $(x_{ij} = 1)$ alors $(t_i < t_j)$ c'est-à-dire que la ville i précède la ville j puisque la route qui va de la ville i à la ville j a été emprunté. Cependant en programmation linéaire les inégalités strictes sont interdite ce qui nous donne la reformulation suivante, si $(x_{ij} = 1)$ alors $(t_i + 1 \leq t_j)$. De manière équivalente si $(x_{ij} = 1)$ alors $(t_i - t_j + 1 \leq 0)$. Cela se traduit par la contrainte suivante.

$$\begin{aligned} & t_i - t_j + 1 \quad \leq \quad M(1 - x_{ij}) && \forall i, j \in N \backslash \{1\}, i \neq j \\ \Leftrightarrow \quad & t_i - t_j + M x_{ij} \quad \leq \quad M - 1 && \forall i, j \in N \backslash \{1\}, i \neq j \end{aligned}$$

Où M est un grand nombre, qui peut être fixé à n car bien que les t_i soient continus ils ne dépasseront pas n puisqu'il y a n villes à visiter en n temps. Ce qui donne donc cette contrainte.

$$t_i - t_j + n x_{ij} \leq n - 1 \qquad \forall i, j \in N \backslash \{1\}, i \neq j \tag{11}$$

3.3.3 Heuristique de construction Nearest Neighbour

— Nearest Neighbour initialise toutes les villes comme non visité puis choisit une ville comme étant la ville de départ. La ville de départ est marquée comme visité.
— On calcule la liste de fonction d'utilité comme le coût entre la ville actuelle et toutes les villes non visitées, dans l'ordre croissant.
— L'heuristique choisit la première ville dans la liste de fonction d'utilité, donc avec le plus petit coût. La ville choisie est ajoutée à la solution, est marquée comme visitée et devient la ville actuelle.
— On retourne au calcul de la liste de fonction d'utilité jusqu'à ce qu'il n'y ait plus de ville non visitée. On finit par rajouter le coût entre la ville finale et la ville de départ.

3.3.4 Heuristique de descente 2-opt [10]

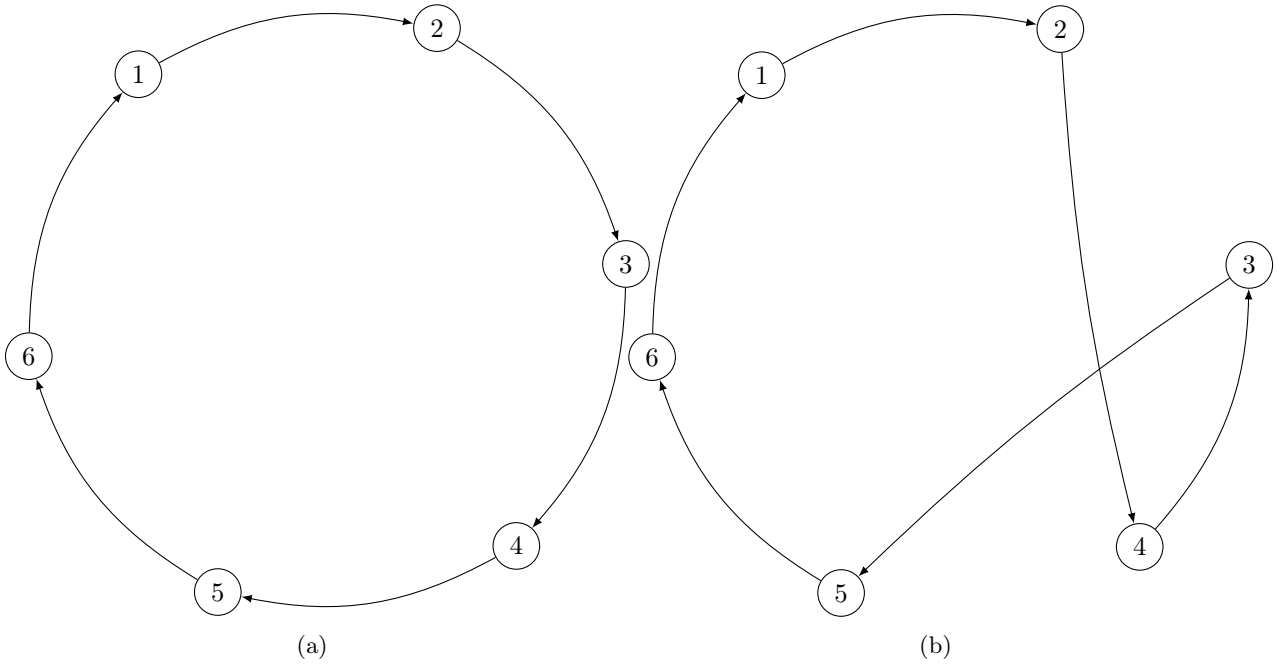

FIGURE 1 – Explication heuristique 2-opt.

Dans un premier temps l'heuristique choisit un couple d'arcs non-consécutifs, par exemple sur la figure 1a elle prend l'arc(2, 3) et l'arc(4, 5). Elle va interdire ces arcs et reconstruire une solution admissible. Il n'y a qu'une solution préservant les autres arcs du cycle, elle est donnée par la figure 1b. Ainsi l'heuristique essaye pour chaque couple d'arcs non-consécutifs jusqu'à trouver une solution améliorante. Une fois fait elle recommence jusqu'à ce qu'aucune paire ne permette d'améliorer notre solution.

Après avoir choisi les trois problèmes d'études, nous avons décidé de proposer trois tentatives de variation de Reactive-GRASP. Ces trois variations ont été chacune construite avec un composant différent dans le but d'obtenir une convergence vers la meilleure classe α.

4 Tentative de convergence

4.1 Reactive-GRASP & évaluation/récompense

Cette première tentative a été mise en place en amont de notre étude par Xavier Gandibleux en 2019. Cette implémentation est un outil constituant la base de notre travail pour la recherche d'un apprentissage. L'idée est la suivante. On évalue systématiquement toutes les classes présente avec 5 itérations par classe afin de laisser une chance à toutes les classes. Suite à cela, on introduit un mécanisme d'amplification sur la classe ayant trouvé le plus de solutions améliorantes. En effet, on compte le nombre de fois qu'une meilleure solution est trouvée. La probabilité va être biaisée par ce nombre et elle sera normalisée à la fin. Cette tentative avait pour but un premier jet de convergence pour retourner la classe avec la probabilité la plus haute.

4.2 Reactive-GRASP & Tabou

Cette tentative a pour objectif d'introduire de l'inertie et de récompenser la classe ayant trouvé la meilleure solution connue.
Au moment des mises à jour des probabilités, toutes les N_α itérations, on vérifie pour chaque classe α si son q_α augmente ou diminue (ligne 3.12). Si il augmente alors la classe est marqué comme tabou pour y mise à jours (ligne 3.17). Si il diminue, on regarde si la classe α est marqué comme tabou (ligne 3.13), si la classe est tabou alors on l'empêche de diminuer en lui donnant son q_α précédent (ligne 3.14).
La classe qui a donné la meilleure solution est noté comme $\alpha\_aspiration$, on lui donne systématiquement $z\%$ de probabilité (ligne 3.21).

Algorithm 3 Tentative 2, Reactive-GRASP & Tabou

1: $Solutions \leftarrow \emptyset$
2: $proba_\alpha \leftarrow 1/|EnsembleAlpha|, \quad \forall \alpha \in EnsembleAlpha$
3: **repeat**
4: $\quad \alpha\_Itt \leftarrow SelectionAleatoire(EnsembleAlpha, proba)$
5: $\quad SolutionInitiale \leftarrow GreedyRandomizedConstruction(problem, \alpha\_Itt)$
6: $\quad SolutionAmelioree \leftarrow LocalSearchImprovement(SolutionInitiale)$
7: $\quad Solutions \leftarrow Solutions \cup SolutionAmelioree$
8: $\quad$ **if** MiseAJourProba is True **then**
9: $\quad\quad q_\alpha Precedent \leftarrow q_\alpha, \quad \forall \alpha \in EnsembleAlpha$
10: $\quad\quad q_\alpha \leftarrow \frac{zAVG_\alpha - zWorst}{zBest - zWorst}, \quad \forall \alpha \in EnsembleAlpha$
11: $\quad\quad$ **for** α *in* $EnsembleAlpha$ **do**
12: $\quad\quad\quad$ **if** $q_\alpha \leq q_\alpha Precedent$ **then**
13: $\quad\quad\quad\quad$ **if** $tabou(\alpha)$ **then**
14: $\quad\quad\quad\quad\quad q_\alpha \leftarrow q_\alpha Precedent$
15: $\quad\quad\quad\quad$ **end if**
16: $\quad\quad\quad$ **else**
17: $\quad\quad\quad\quad tabou(\alpha) \leftarrow True$ pour y mise à jour des alphas
18: $\quad\quad\quad$ **end if**
19: $\quad\quad$ **end for**
20: $\quad\quad p_\alpha \leftarrow \frac{q_\alpha}{\sum_{i=1}^{EnsembleAlpha} q_i}, \quad \forall \alpha \in EnsembleAlpha$
21: $\quad\quad p_{\alpha\_aspiration} \leftarrow z\%$
22: $\quad\quad normaliser(p_\alpha), \quad \forall \alpha \in EnsembleAlpha$
23: $\quad$ **end if**
24: **until** $(stoppingCondition)$
25: **return** best(Solutions)

4.3 Reactive-GRASP & Seuil

Le but de cette variante est de mettre en place une sélection par classement par l'intermédiaire d'une récompense importante, de pénalité, ainsi que deux seuils de probabilités.
Pour chaque N_α itérations, la classe α ayant donné de bonnes solutions au cours des itérations précédentes voit sa probabilité actuelle multipliée par 4 (ligne 4-15). La valeur de ce multiplicateur a pour but de récompenser grandement la classe. Dans le cas contraire, si la classe n'a pas donné de bonnes solutions (ligne 4-12), sa probabilité est divisée par 2 (ligne 4-13). Cette valeur permet de pénaliser de moitié la récompense attribuée auparavant à certaine classe qui ne trouve pas de bonnes solutions. Par la suite, nous avons mis en place le système de seuils, un premier à 15% (ligne 4-17) et le deuxième à 30% (ligne 4-20).
Le premier seuil de 15% permet de sélectionner les classes intéressantes afin de procéder à une amplification (ligne 4-18). En effet, les probabilités de ces classes ne pourront descendre en dessous de ce seuil.
Lorsque le seuil de 30% est atteint pour la première fois par la probabilité d'une classe α, on met en valeur celui-ci en faisant en sorte qu'il ne puisse pas descendre en dessous de ce seuil (ligne 4-22). De plus, chaque seuil atteint par les probabilités des autres classes sont débloquées (ligne 4-21), elles peuvent donc redescendre en dessous de celle-ci. On termine la tentative en normalisant les probabilités (ligne 4-25).

Algorithm 4 Tentative 3, Reactive-GRASP & Seuil

$Solutions \leftarrow \emptyset$
$proba_\alpha \leftarrow 1/|EnsembleAlpha|, \quad \forall \alpha \in EnsembleAlpha$
repeat
  $\alpha\_Itt \leftarrow SelectionAleatoire(EnsembleAlpha, proba)$
  $SolutionInitiale \leftarrow GreedyRandomizedConstruction(problem, \alpha\_Itt)$
  $SolutionAmelioree \leftarrow LocalSearchImprovement(SolutionInitiale)$
  $Solutions \leftarrow Solutions \cup SolutionAmelioree$
  if MiseAJourProba is True **then**
    $q_\alpha Precedent \leftarrow q_\alpha, \quad \forall \alpha \in EnsembleAlpha$
    $q_\alpha \leftarrow \frac{zAVG_\alpha - zWorst}{zBest - zWorst}, \quad \forall \alpha \in EnsembleAlpha$
    for α *in* $EnsembleAlpha$ **do**
      if $q_\alpha \leq q_\alpha Precedent$ **then**
        $q_\alpha \leftarrow q_\alpha / 2$
      else
        $q_\alpha \leftarrow q_\alpha * 4$
      end if
      if $q_\alpha \geq 15\%$ **then**
        $q_\alpha \leftarrow BloqueSeuil(q_\alpha)$
      end if
      if $q_\alpha \geq 30\%$ **then**
        $q_\alpha \leftarrow DebloqueSeuil(q_\alpha), \quad \forall \alpha \in EnsembleAlpha$
        $q_\alpha \leftarrow BloqueSeuil(q_\alpha)$
      end if
    end for
    $p_\alpha \leftarrow \frac{q_\alpha}{\sum_{i=1}^{\#EnsembleAlpha} q_i}, \quad \forall \alpha \in EnsembleAlpha$
  end if
until (*stoppingCondition*)
return best(Solutions)

4.4 GRASP et le principe d'une roulette

Pour comparer les différentes tentatives nous utiliserons une implémentation la plus simple possible, sans apprentissage, en guise de témoin.

Nous utilisons un simple principe de roulette, les alphas sont sélectionnés dans l'ordre un par un.

Algorithm 5 Roulette

$Solutions \leftarrow \emptyset$
$proba_\alpha \leftarrow 1/|EnsembleAlpha|, \quad \forall \alpha \in EnsembleAlpha$
repeat
  for $\alpha \in EnsembleAlpha$ **do**
    $SolutionInitiale \leftarrow GreedyRandomizedConstruction(problem, \alpha)$
    $SolutionAmelioree \leftarrow LocalSearchImprovement(SolutionInitiale)$
    $Solutions \leftarrow Solutions \cup SolutionAmelioree$
  end for
until (*stoppingCondition*)
return best(Solutions)

5 Analyse des données

Instances utilisées

SPP

Le lecteur pourra retrouver les instances utilisées à l'adresse suivante :
https://www.emse.fr/~delorme/SetPackingFr.html (visité le 22/04/2024)

StPP

Le lecteur pourra retrouver les instances utilisées à l'adresse suivante dans le fichier nommé "strip1.txt" :
https://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/ (visité le 22/04/2024)

TSP

Le lecteur pourra retrouver les instances utilisées à l'adresse suivante :
http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/ (visité le 22/04/2024)

Protocole expérimental

Les paramètres sont :
- liste alpha = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 et 0.9
- mise à jour des alphas = 50
- itérations = 1000

Pour tabou nous rajoutons :
- y = 3
- z = 33%

Pour seuil nous rajoutons :
- Premier_Seuil = 15%
- Deuxième_Seuil = 30%

Explication des graphiques

Cette partie étudie la répartition des probabilités des classes sur nos tentatives. L'objectif est toujours de faire ressortir une classe α parmi celle proposée. L'étude de la qualité des solutions sera traitée dans la partie Analyse Expérimentale (7).

Les graphiques montrent l'évolution des pourcentages de répartition de chaque classe pour le choix d'un paramètre α du bas vers le haut. Plus la probabilité est élevée plus la classe aura de chance d'être sélectionnée. La répartition initiale est équiprobable.

5.1 Etude sur le SPP

Reactive-GRASP

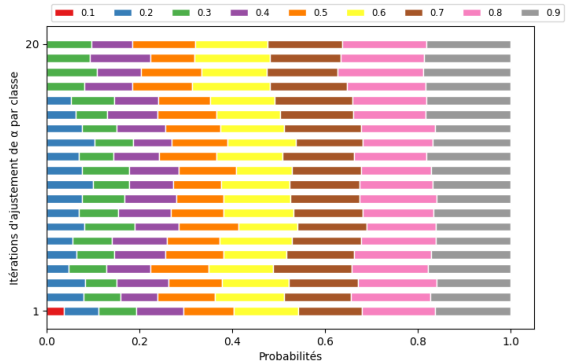

FIGURE 2 – pb_500rnd0100.dat Reactive-GRASP

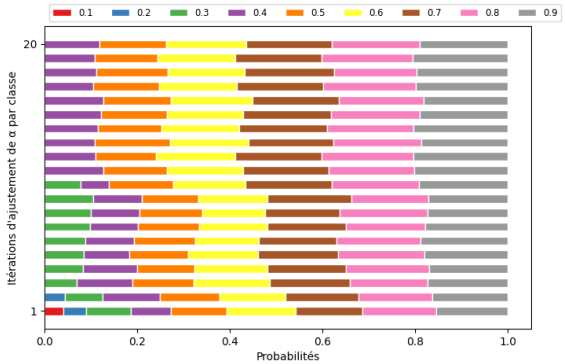

FIGURE 3 – pb_1000rnd0300.dat Reactive-GRASP

On observe sur les figures (2) et (3) que les plus petites classes tombent à 0% de probabilité. On peut l'expliquer par la structure du SPP. GRASP, avec un alpha proche de 0, choisira presque aléatoirement quelle activité sélectionner. Dans certains cas, cette action va prendre beaucoup de ressources pour très peu de profits, donnant ainsi de mauvaises solutions.

On peut noter que c'est un comportement que nous ne retrouvons pas dans les autres problèmes, pour StPP, peu importe la pièce choisie en 1ère, l'heuristique va toujours essayer de combler l'espace disponible ce qui limite les mauvais choix et pour le TSP, l'heuristique de descente 2-opt va grandement réparer les mauvais choix qui pourraient être faits.

Tabou

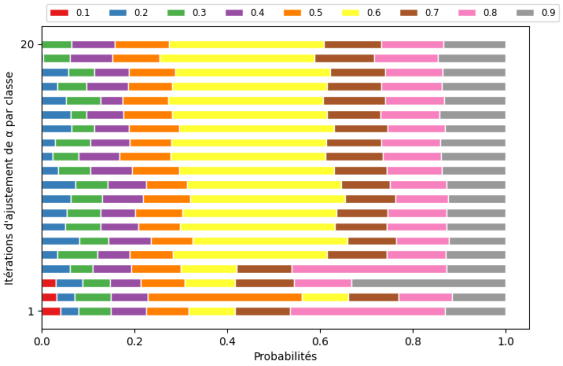

FIGURE 4 – pb_500rnd0100.dat Tabou

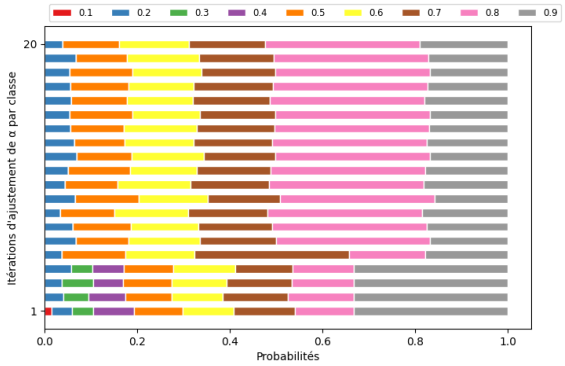

FIGURE 5 – pb_1000rnd0300.dat Tabou

On remarque sur la figure (4) que la classe 0.8 obtient $\sim 1/3$ des probabilités, puis elle repasse à $\sim 1/9$ alors que c'est la classe 0.5 qui prend $\sim 1/3$. On voit ce même échange avec la classe 0.9 et 0.6 avant une stabilisation sur la classe 0.6. On explique ces mouvements dans les probabilités par le critère d'aspiration qui change plusieurs fois pendant les premières mises à jour avant de se stabiliser sur la classe 0.6. C'est-à-dire que la meilleure solution est découverte par la classe 0.8, puis une solution améliorante est découverte pas la classe 0.5 et ainsi de suite. Le même comportement est vu sur la figure (5) avec les classes 0.9, 0.7 et 0.8.

On remarque également que sur les figures (4) et (5) que les classes les plus proches de 0 ont beaucoup moins de chance d'être sélectionné et tombe vers 0%. On pense, comme précédemment, que c'est dû à la structure du SPP.

Par rapport à notre objectif d'introduire de l'inertie, on voit que les classes stagnent. Aucune inertie n'est visible à part une légère sélection. Les classes proches de 0 tombent à 0% mais on remarquait déjà cette sélection avec Reactive-GRASP. Par contre nous avons bien réussi à donner plus d'importance et de chance d'être choisit à la classe qui nous donne la meilleure solution connue.

Seuil

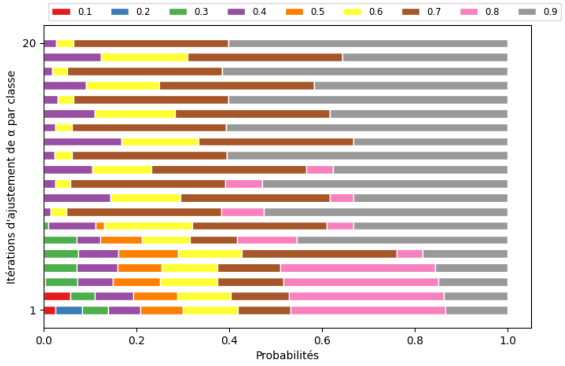

FIGURE 6 – pb_500rnd0100.dat Seuil

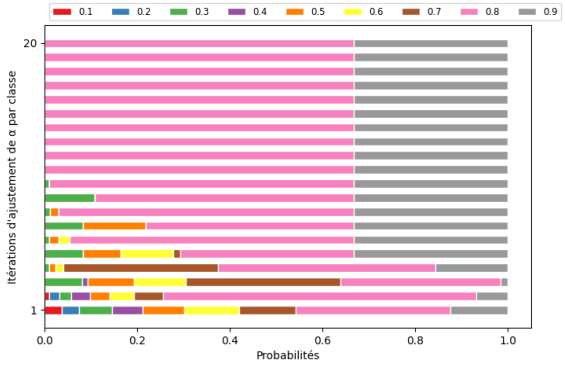

FIGURE 7 – pb_1000rnd0300.dat Seuil

Nous observons pour les deux images une sélection de deux classes. En effet, pour la figure (6), la classe 0.7 et 0.9 dépassent le deuxième seuil de 30% assez rapidement. Les autres classes sont soit totalement supprimées, soit de probabilités très faibles comme 0.4 et 0.6. Pour la figure (7), la sélection est immédiate. Toutes les classes inférieures à 0.8 sont supprimées. La classe 0.8 possède un peu moins de 70% des probabilités de répartition de α. Malheureusement, cette sélection ne retourne pas un α constant pour tous les problèmes du SPP.

5.2 Etude sur le StPP

Reactive-GRASP

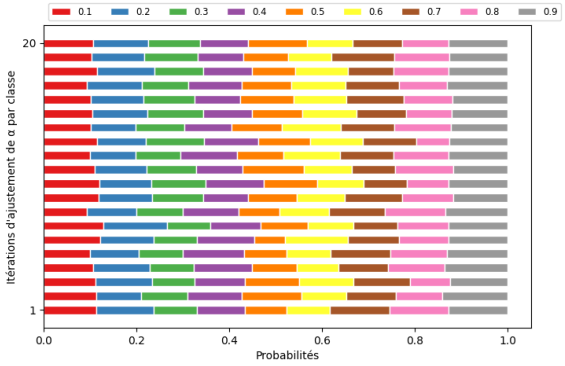

FIGURE 8 – strip80_1 Reactive-GRASP

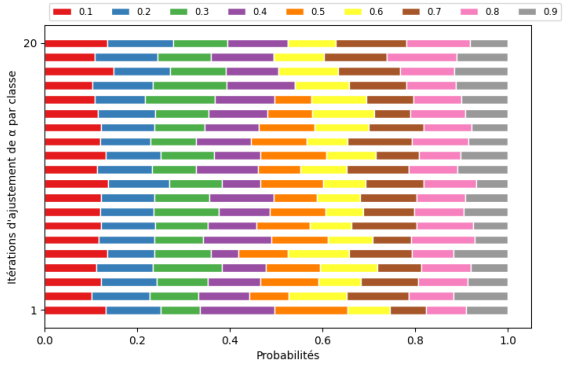

FIGURE 9 – strip160_1 Reactive-GRASP

Comme on peut l'observer sur les figures (8) et (9), à l'exception de la classe 0.5 sur la figure (9), il n'y a aucune convergence, toutes les classes ont des probabilités proches.

On peut l'expliquer par le fonctionnement de l'heuristique de construction, elle cherche à remplir les espaces vides. Ainsi même si on fait des mauvais choix, l'heuristique finira par combler le plus possible les espaces vides et obtiendra une solution compacte. Par conséquent, on observe très peu de différences entre les plus mauvaises solutions créées par GRASP et les solutions moyennes.

On peut aussi noter que plus le problème à de rectangles, plus les rectangles auront tendance à se compenser les uns et autres. Il y a donc beaucoup de solutions avec la même valeur objectif.

Tabou

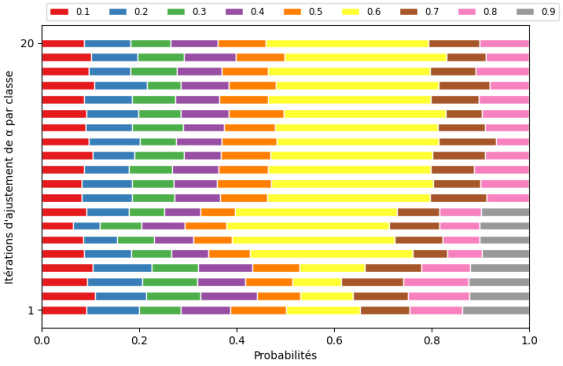

FIGURE 10 – strip80_1 Tabou

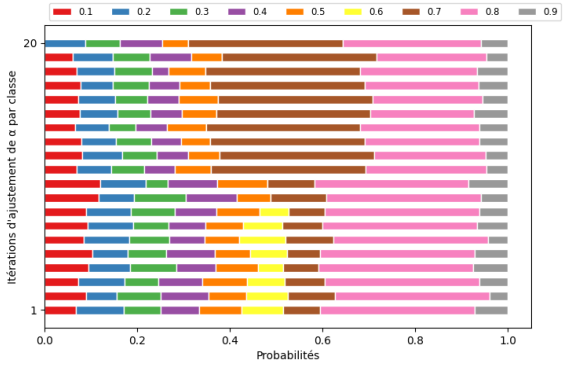

FIGURE 11 – strip160_1 Tabou

On peut expliquer la figure (10) par le fait que la meilleure solution a été trouvée par notre solution initiale jusqu'à la 5eme mise à jour où la classe 0.6 a trouvé une solution améliorante. Pour la figure (11) la classe 0.8 trouve une meilleure solution jusqu'à ce que la classe 0.7 en trouve une meilleure.

On voit bien que l'on donne plus de budget de calcul à la classe qui nous donne la meilleure solution. Il y a une légère sélection avec la classe 0.9 qui tombe à 0% dans la figure (10) et les classes 0.6 et 0.1 dans la figure (11). Cependant, on ne voit pas vraiment d'inertie.

Seuil

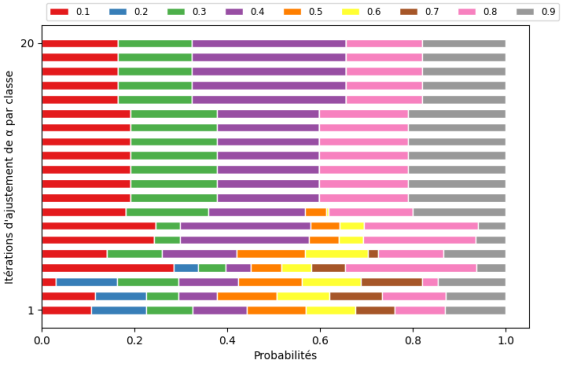

FIGURE 12 – strip80_1 Seuil

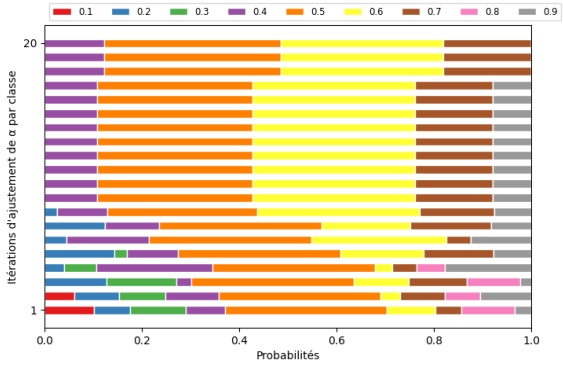

FIGURE 13 – strip160_1 Seuil

En observant les deux images, on peut voir que la tentative de seuil se solde par une sélection de différents α. En effet, dans la figure (12), la classe 0.2, 0.5, 0.7 et 0.6 se fait totalement supprimer pour laisser place aux 5 dernières classes. Le même processus est observé dans la figure (13) mais sur des classes différentes. Avec la première instance, c'est 0.4 qui atteint le deuxième seuil alors que c'est 0.5 et 0.6 dans la deuxième. En conclusion, malgré un processus de sélection fonctionnel, il n'y a pas vraiment de convergence vers un seul et même α.

5.3 Etude sur le TSP

Reactive-GRASP

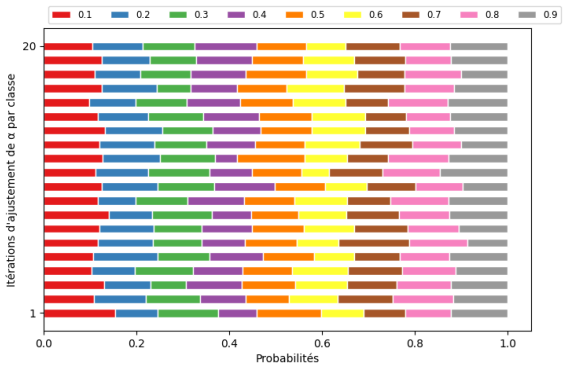

FIGURE 14 – pa561.tsp Reactive-GRASP

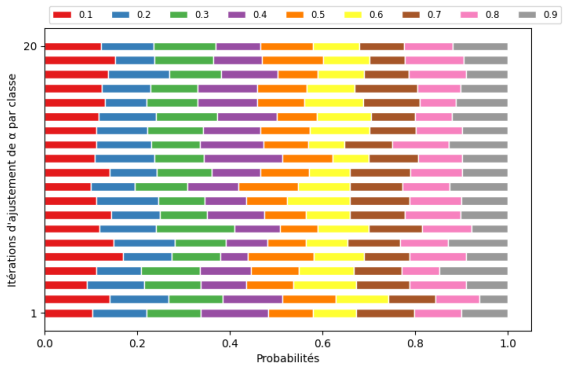

FIGURE 15 – si1032.tsp Reactive-GRASP

On observe sur les figures (14) et (15) qu'il n'y a aucune convergence, toutes les classes ont des probabilités similaires d'être choisies.

On l'explique par l'efficacité de l'heuristique de descente 2-opt. En effet, l'heuristique permet de grandement améliorer les solutions et les harmonises comme on peut le voir sur le tableau (1). La première ligne indique quel alpha est utilisé, la seconde indique la solution obtenue par GRASP avec l'aide de l'heuristique Nearest Neighbour et la dernière ligne indique la solution obtenue après l'application de l'heuristique 2-opt.

alpha utilisé	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Solution GRASP avant descente	357005	354983	316302	282995	281434	277007	269540	269643	254532
Solution après descente 2-opt	93646	94138	94093	93663	94166	93740	93511	93914	94163

TABLE 1 – Démonstration de l'efficacité de 2-opt sur l'instance si1032.tsp.

Tabou

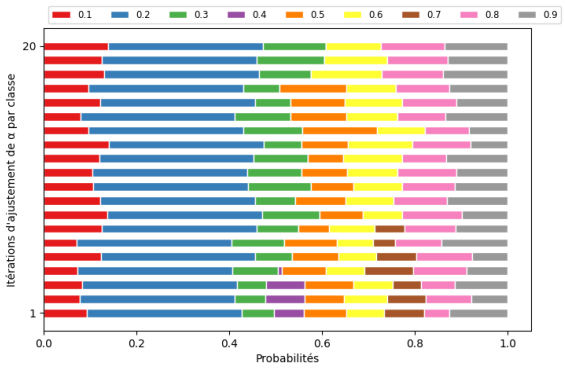

FIGURE 16 – pa561.tsp Tabou

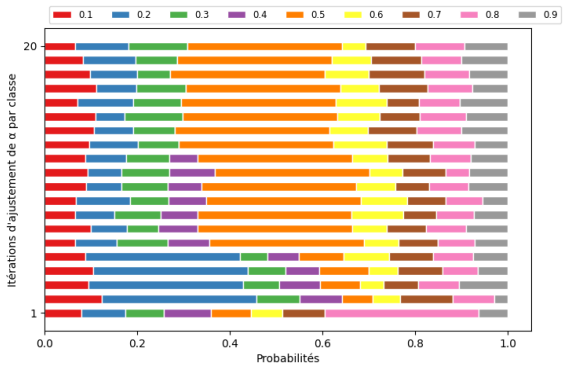

FIGURE 17 – si1032.tsp Tabou

Comme le présente la figure (16), la classe 0.2 a donné les meilleures solutions et a donc dominé grâce au critère d'aspiration. On voit aussi qu'une certaine sélection a été faite, les classes 0.4, 0.5 et 0.7 sont passés à 0% de probabilité. Pour la figure (17), on observe le critère d'aspiration s'échange entre les classes 0.8, 0.2 et 0.5.

On donne plus de chance d'être choisi à la classe qui trouve la meilleure solution, il y a une certaine sélection avec des classes qui passe à 0% mais il n'y a pas d'inertie créée.

Seuil

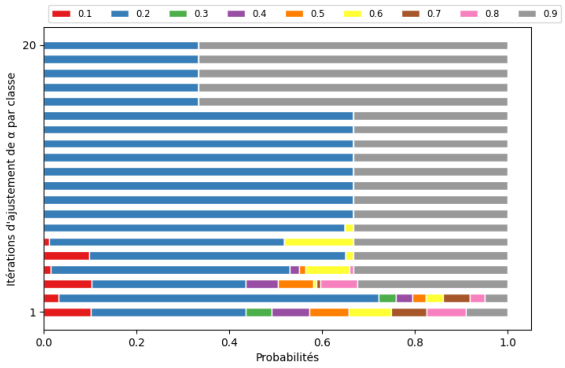

FIGURE 18 – pa561.tsp Seuil

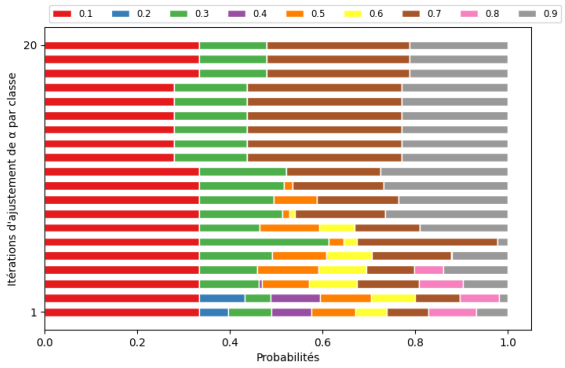

FIGURE 19 – si1032.tsp Seuil

Pour ce qui est de la tentative de seuil sur le TSP, on observe sur la figure (18) que le processus de sélection a fonctionné. En effet, les classes 0.2 et 0.9 sont mises en valeur avec la disparition des classes restantes. La classe 0.9 est a 70% au niveau du pourcentage de sélection. Une convergence est donc observée sur cet α. Cependant, en vue de la figure (19), Le processus de sélection est strictement différent. Par conséquent, il n'y a aucune convergence constante vers un α qui retourne toujours la meilleure solution.

5.4 Conclusion

Après avoir analysé la composante Reactive-GRASP sous les trois problèmes d'optimisation sélectionnés, nous n'observons pas de convergence constante vers un α. L'apprentissage ne nous retourne pas une classe d'α propre à chaque problème, voire même à chaque instance. Cela est vrai pour toutes les tentatives que nous avons mises en place. Nous en concluons que le caractère aléatoire et agressif de GRASP permet d'obtenir de très bonnes solutions pour n'importe quelle classe α.

6 Algorithme du Bandit Manchot

6.1 Discussion

Pour la deuxième partie de notre recherche, nous devions orienter nos travaux vers un composant alternatif à Reactive-GRASP afin d'obtenir un apprentissage automatique du paramètre α. L'étude du problème des Bandit Manchot nous a été proposé afin de mener une étude approfondie sur le réglage avec l'algorithme GRASP. En effet, Reactive-GRASP et les Bandit-Manchot présentent une technique similaire, celle de récompenser par l'obtention de bonnes solutions.

6.2 Définition

Le problème du Bandit Manchot est défini par la littérature de la façon suivante. On considère un agent (a learner) qui se trouve face à des machines à sous. A chaque itération, il doit décider quelle action effectuer, c'est à dire, sur quelle machine il doit jouer. Chacune de ses machines lui rapporte une récompense. Son but est de maximiser la somme des récompenses qu'il a obtenu (le gain cumulé). Il doit donc trouver l'action optimale. En général, les performances des bandits manchot sont mesurées avec du regret. Il faut qu'il soit le plus petit possible. Pour cette deuxième phase de recherche, nous avons sélectionné l'algorithme considéré comme la stratégie la plus simple de la littérature sur les problèmes de bandit.

6.3 Algorithme ϵ-greedy

Pour l'utilisation de cette algorithme nous nous basons sur une partie du chapitre 2 : "Multi-Arm Bandits" du livre de Sutton et Barto [7]. ϵ-greedy algorithm fonctionne de la façon suivante. On pose un paramètre $\epsilon \in [0; 1]$. Ce paramètre sert à déterminer la proportion d'exploration et d'exploitation que l'on veut par rapport à un nombre n d'actions.
En effet, a chaque itération on va définir un random entre 0 et 1. Si le random est inférieur à ϵ (ligne 6.5), on lance la phase d'exploration (ligne 6.6, on choisit une action au hasard). Si le random est supérieur, on lance la phase d'exploitation (ligne 6.8, on choisit l'action qui la possède récompense estimée la plus élevée). En fonction, de la récompense obtenue grâce à l'action, on met à jour la liste des valeurs estimée (ligne 6.2). Cette mise à jour se base sur un principe "d'estimation Valeur-Action"[7]. La méthode de base consiste à diviser pour chaque actions, la somme des récompenses obtenues au cours des précédentes itérations, par le nombre de fois que l'action a été choisie. La valeur obtenue par cette opération est ajoutée à l'estimation.

Les paramètres sont :
- $\epsilon \in [0; 1]$
- $n\_steps = nbIteration$
- $Tab\_actions$

Algorithm 6 ϵ-greedy algorithm

```
1:  actionOptimale ← []
2:  count ← []
3:  values ← []
4:  for i in 1 :n_steps do
5:      if Random ≤ ε then
6:          action ← ActionRandom(Tab_actions)
7:      else
8:          action ← ActionMaxReward(Tab_actions)
9:      end if
10:     reward ← GetRecompense(action)
11:     count[action] ← count[action] + 1
12:     values[action] ← reward + values[action]/counts[action]
13:     if action = ActionMaxReward(Tab_action) then
14:         actionOptimale[i] ← Vrai
15:     end if
16: end for
        return actionOptimale, reward, count
```

6.4 Implémentation avec GRASP

Pour relier cette composante à l'algorithme GRASP, nous avons choisi de poser un tableau de n valeurs α comme à la manière de Reactive-GRASP. Chaque valeur va représenter une action que l'algorithme choisira à chaque itération afin de pouvoir lancer GRASP par la suite.
Après avoir choisi l'action pour cette itération, on lance les deux phases de GRASP qui nous retourne une valeur de la solution (ligne 7.12). Pour la mise à jour des probabilités, nous avons modifié la stratégie expliquée dans la partie précédente.
En effet, nous avons pris en compte la qualité des valeurs des solutions que GRASP nous retourne à chaque itération. Nous prenons la valeur de la solution retournée par GRASP, que l'on va soustraire à la somme des récompenses cumulées de l'action. Cette valeur va être divisée par le nombre de fois que l'action a été choisie (ligne 7.14).

Paramètres :
- $\epsilon \in [0; 1]$
- $n\_steps = nbIteration$
- $Tab\_alpha = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$

Algorithm 7 ϵ-greedy avec GRASP (construction + recherche locale)

```
actionOptimale ← []
best_alpha ← Tab_alpha[1]
best_reward ← INF
count ← []
values ← []
for i in 1 :n_steps do
    if Random ≤ ϵ then
        action ← ActionRandom(Tab_alpha)
    else
        action ← ActionMaxReward(Tab_alpha)
    end if
    current_reward ← GRASP(Tab_alpha[action])
    count[action] ← count[action] + 1
    values[action] ← values[action] + (current_reward + values[action])/counts[action]
    if action = ActionMaxReward(Tab_alpha) then
        actionOptimale[i] ← Vrai
    end if
    if current_reward ≤ best_reward then
        best_reward ← current_reward
        best_alpha ← Tab_alpha[action]
    end if
end for
    return best_alpha, actionOptimale, best_reward, count
```

6.5 Explication de l'implémentation

Après avoir implémenté la composante ϵ-greedy avec GRASP, nous avons décidé de faire une partie expérimentation avec l'obtention de moyennes sur plusieurs runs de l'algorithme. En effet, On pose un paramètre supplémentaire "nb_run" pour déterminer le nombre de fois que l'on va lancer l'algorithme pour une seule instance. Cet algorithme retourne 4 valeurs par rapport aux tableaux d'alpha passé en paramètre :

- Un tableau de pourcentage de l'α qui retourne la meilleure solution avec GRASP en moyenne.
- Un tableau de pourcentage de l'α que l'algorithme ϵ-greedy à le plus choisi (Celui qu'il considère comme action optimale).
- Le pourcentage de choix d'actions optimales en moyenne, c'est à dire le budget de calcule alloué à l'exploitation.
- La valeur de la solution trouvée en moyenne.

Voici les résultats de l'étude portée sur les trois problèmes.

Pour chaque problème, on lance la fonction $experimentation(nb\_run, \epsilon, nb\_Iteration)$
Les deux phases de GRASP restent les mêmes que celles utilisées avec la composante Reactive-GRASP

6.6 Etude sur le SPP

Protocole expérimental

Les paramètres sont :
- liste alpha = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 et 0.9
- itérations = 1000
- epsilon = 0.5

pb_500rnd0100.dat									
nombre de run : 30									
valeur epsilon : 0.5									
nombre d'itération par run : 1000									
Valeurs alpha	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Alpha qui retourne la meilleure solution en moyenne	0%	0%	0%	0%	0%	3%	7%	10%	80%
Alpha qui a été choisi le plus par epsilon-greedy en moyenne	0%	0%	0%	0%	0%	0%	0%	0%	100%
Pourcentage d'actions optimales choisies en moyenne : 0.561									
Valeur de la solution trouvée en moyenne : 315.3									

FIGURE 20 – Bandit_Manchot pb_500rnd0100.dat

pb_1000rnd0300.dat									
nombre de run : 30									
valeur epsilon : 0.5									
nombre d'itération par run : 1000									
Valeurs alpha	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Alpha qui retourne la meilleure solution en moyenne	0%	0%	0%	0%	0%	3%	7%	40%	50%
Alpha qui a été choisi le plus par epsilon-greedy en moyenne	0%	0%	0%	0%	0%	0%	0%	3%	97%
Pourcentage d'actions optimales choisies en moyenne : 0.559									
Valeur de la solution trouvée en moyenne : 603.3									

FIGURE 21 – Bandit_Manchot pb_1000rnd0300.dat

Pour le problème du SPP, on peut observer que c'est l'α 0.9 qui retourne le plus fréquemment la meilleure solution et qui est choisie le plus par l'ϵ-greedy pour les deux figures (20) et (21). La solution retournée n'est pas la valeur optimale selon les tableaux en Annexe (8). On note aussi que 56% du budget de calcule est alloué à l'exploitation.

6.7 Etude sur le StPP

strip80_1									
nombre de run : 30									
valeur epsilon : 0.5									
nombre d'itération par run : 1000									
Valeurs alpha	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Alpha qui retourne la meilleure solution en moyenne	0%	0%	3%	0%	7%	13%	17%	27%	33%
Alpha qui a été choisi le plus par epsilon-greedy en moyenne	0%	0%	0%	0%	0%	0%	0%	0%	100%
Pourcentage d'actions optimales choisies en moyenne : 0.542									
Valeur de la solution trouvée en moyenne : 127.3									

FIGURE 22 – *Bandit_Manchot* sur strip80_1

strip160_1									
nombre de run : 30									
valeur epsilon : 0.5									
nombre d'itération par run : 1000									
Valeurs alpha	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Alpha qui retourne la meilleure solution en moyenne	0%	0%	0%	0%	10%	17%	37%	33%	3%
Alpha qui a été choisi le plus par epsilon-greedy en moyenne	27%	27%	13%	27%	0%	0%	6%	0%	0%
Pourcentage d'actions optimales choisies en moyenne : 0.516									
Valeur de la solution trouvée en moyenne : 254.4									

FIGURE 23 – *Bandit_Manchot* sur strip160_1

Pour le problème du StPP, on peut observer que c'est l'α 0.9 qui retourne la meilleure solution avec une proportion de 33% (pourcentage le plus élevé) pour la figure (22). 0.9 a été constamment choisies par l'ϵ-greedy. Avec 54% du budget de calcule alloué à l'exploitation.
Pour ce qui est de la figure (23), les α qui retourne la meilleure solution sont 0.5, 0.6, 0.7, 0.8 et 0.9, pourtant l'ϵ-greedy choisit les α 0.1, 0.2, 0.3 et 0.4. On note aussi que le budget de calcule alloué à l'exploitation est plus bas, à 51% (sachant que sa valeur planché est 50%).
Cette fois-ci, la convergence vers un α n'est pas totale et dépend grandement de l'instance. On l'explique par l'heuristique de construction qui est très puissante sur le problème du StPP ainsi il est plus difficile d'identifié la classe offrant la meilleure récompense.

6.8 Etude sur le TSP

pa561.tsp									
nombre de run : 30									
valeur epsilon : 0.5									
nombre d'itération par run : 1000									
Valeurs alpha	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Alpha qui retourne la meilleure solution en moyenne	13%	17%	10%	10%	7%	20%	17%	3%	3%
Alpha qui a été choisi le plus par epsilon-greedy en moyenne	13%	17%	20%	17%	0%	13%	10%	3%	7%
Pourcentage d'actions optimales choisies en moyenne : 0.508									
Valeur de la solution trouvée en moyenne : 3007.5									

FIGURE 24 – Bandit_Manchot pa561.tsp

si1032.tsp									
nombre de run : 30									
valeur epsilon : 0.5									
nombre d'itération par run : 1000									
Valeurs alpha	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Alpha qui retourne la meilleure solution en moyenne	10%	13%	13%	16%	20%	7%	10%	3%	7%
Alpha qui a été choisi le plus par epsilon-greedy en moyenne	7%	13%	10%	23%	17%	7%	7%	10%	7%
Pourcentage d'actions optimales choisies en moyenne : 0.514									
Valeur de la solution trouvée en moyenne : 93164									

FIGURE 25 – Bandit_Manchot si1032.tsp

Pour le problème du TSP, on peut observer qu'il n'y a pas vraiment de sélection d'un meilleur α, que ça soit sur un retour d'une meilleure solution ou bien sur la valeur choisie par ϵ-greedy, ce qui peut être expliqué par le tableau (1). Par conséquent, l'observation n'est pas concluante sur le problème du TSP.

6.9 Conclusion

Après analyse de la composante ϵ-greedy avec le GRASP sur les trois problèmes d'optimisations sélectionnés, on observe que celle-ci nous donne de meilleurs résultats que la composante Reactive-GRASP. En effet, l'apprentissage par renforcement développé par l'algorithme choisi la plupart du temps une classe α avec une proportion très élevée. De plus, la classe qui retourne le plus la meilleure solution en moyenne est le même α. Cependant, pour un nombre de run donnée, la valeur de la solution en moyenne n'est pas la solution dite "optimale" selon l'annexe (8). Par conséquent, l'apprentissage par renforcement des Bandit-Manchots est une piste plus intéressante que la composante Reactive-GRASP en terme de convergence vers une seule classe α. Il nous reste à comparer la qualité des solutions retournées pour tous les problèmes choisis.

7 Analyse Expérimentale

Protocole expérimental

Les paramètres sont :

— liste alpha = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 et 0.9
— mise à jour des alphas = 50
— itérations = 1000

Pour tabou nous rajoutons :
— y = 3
— z = 33%

Pour seuil nous rajoutons :
— Premier_Seuil = 15%
— Deuxième_Seuil = 30%

Pour Bandit-Manchot nous rajoutons :
— epsilon = 0.5

7.1 Tableaux

Nous avons pour chaque instance la valeur moyenne sur une trentaine d'expérimentations, le lecteur trouvera en annexe davantage d'informations comme les médianes, les solutions minimales et maximales.

SPP

"*" indique que ce n'est pas la valeur optimale mais la meilleure connue.

instances	rg	tabou	seuil	roulette	bandit	optimal
Data/pb_100rnd0100.dat	372.0	372.0	372.0	372.0	372.0	372
Data/pb_100rnd0300.dat	203.0	203.0	203.0	203.0	203.0	203
Data/pb_100rnd0900.dat	463.0	463.0	463.0	463.0	463.0	463
Data/pb_200rnd0100.dat	**410.97**	410.86	410.62	408.24	408.14	416
Data/pb_200rnd0300.dat	714.34	715.66	**716.59**	712.69	710.72	731
Data/pb_200rnd0500.dat	183.28	182.41	182.59	**183.83**	182.86	184
Data/pb_500rnd0100.dat	**316.66**	316.62	316.55	315.28	315.41	323*
Data/pb_500rnd0300.dat	744.62	744.76	745.24	742.97	**746.34**	776*
Data/pb_500rnd0500.dat	**122.0**	**122.0**	**122.0**	**122.0**	121.86	122
Data/pb_1000rnd0300.dat	605.28	**607.76**	605.45	604.0	603.34	661*
Data/pb_1000rnd0500.dat	217.93	217.93	217.86	216.21	**218.48**	222*
Data/pb_2000rnd0300.dat	440.69	443.59	441.97	439.21	**445.97**	478*
Data/pb_2000rnd0500.dat	133.52	133.21	**134.59**	133.31	131.79	140*

Table 2 – Expérimentation numérique SPP.

Tous les algorithmes obtiennent des résultats selon les instances comme on le voit sur le tableau 2. Aucun ne se démarque particulièrement, ainsi on pourrait se demander si Reactive-GRASP est plus intéressant qu'une simple roulette.

instances	rg	roulette	optimal
Data/pb_100rnd0100.dat	372.0	372.0	372
Data/pb_100rnd0300.dat	203.0	203.0	203
Data/pb_100rnd0900.dat	463.0	463.0	463
Data/pb_200rnd0100.dat	**410.97**	408.24	416
Data/pb_200rnd0300.dat	**714.34**	712.69	731
Data/pb_200rnd0500.dat	183.28	**183.83**	184
Data/pb_500rnd0100.dat	**316.66**	315.28	323*
Data/pb_500rnd0300.dat	**744.62**	742.97	776*
Data/pb_500rnd0500.dat	122.0	122.0	122
Data/pb_1000rnd0300.dat	**605.28**	604.0	661*
Data/pb_1000rnd0500.dat	**217.93**	216.21	222*
Data/pb_2000rnd0300.dat	**440.69**	439.21	478*
Data/pb_2000rnd0500.dat	**133.52**	133.31	140*

TABLE 3 – Expérimentation numérique SPP, comparatif Reactive-GRASP et roulette.

Quand on se concentre sur Reactive-GRASP et la roulette on voit clairement, sur le tableau 3, que Reactive-GRASP est meilleurs sur le SPP.

StPP

Le lecteur trouvera, en annexe, une explication de l'absence des valeurs optimales pour ce problème.

instances	rg	tabou	seuil	roulette	bandit
Data/strip20_1	20.03	20.07	20.07	**20.0**	**20.0**
Data/strip20_2	21.59	**21.38**	21.52	21.48	21.45
Data/strip20_3	**20.1**	**20.1**	20.17	20.17	20.31
Data/strip40_1	16.0	16.0	16.0	16.0	**15.97**
Data/strip40_2	16.0	16.0	16.0	16.0	16.0
Data/strip40_3	16.0	15.97	16.0	**15.93**	16.0
Data/strip60_1	31.86	31.83	31.76	**31.62**	31.69
Data/strip60_2	32.21	32.21	32.24	32.17	**32.07**
Data/strip60_3	**31.83**	31.97	31.9	31.93	**31.83**
Data/strip80_1	127.41	**127.17**	127.45	127.34	127.34
Data/strip80_2	**127.72**	127.93	127.79	128.07	**127.72**
Data/strip80_3	128.1	128.14	**127.97**	128.03	127.69
Data/strip160_1	253.76	**253.34**	**253.34**	253.93	254.55
Data/strip160_2	253.79	253.41	**252.62**	253.55	253.21
Data/strip160_3	253.9	253.41	**253.1**	253.21	254.17

TABLE 4 – Expérimentation numérique StPP.

Comme prévu, GRASP est bon sur le problème ainsi la différence entre la différente implémentation est très petite. On observe sur le tableau 4 que la différence entre meilleur et la pire implémentation se joue à des décimales ainsi il est dur d'affirmer qu'une implémentation est clairement meilleure à une autre.

instances	rg	roulette
Data/strip20_1	20.03	**20.0**
Data/strip20_2	21.59	**21.48**
Data/strip20_3	**20.1**	20.17
Data/strip40_1	16.0	16.0
Data/strip40_2	16.0	16.0
Data/strip40_3	16.0	**15.93**
Data/strip60_1	31.86	**31.62**
Data/strip60_2	32.21	**32.17**
Data/strip60_3	**31.83**	31.93
Data/strip80_1	127.41	**127.34**
Data/strip80_2	**127.72**	128.07
Data/strip80_3	128.1	**128.03**
Data/strip160_1	**253.76**	253.93
Data/strip160_2	253.79	**253.55**
Data/strip160_3	253.9	**253.21**

TABLE 5 – Expérimentation numérique StPP, comparatif Reactive-GRASP et roulette.

En se concentrant sur Reactive-GRASP et la roulette dans le tableau 5, il apparaît que la roulette remporte plus d'instance que Reactive-GRASP. Cependant, comme les écarts se jouent à des décimales, il est difficile de tirer des conclusions sur le StPP.

TSP

instances	rg	tabou	seuil	roulette	bandit	optimal
Data/bayg29.tsp	1610.0	1610.0	1610.0	1610.0	1610.0	1610
Data/bays29.tsp	2020.0	2020.0	2020.0	2020.0	2020.0	2020
Data/brazil58.tsp	**25395.17**	25395.52	25398.28	25400.86	**25395.17**	25395
Data/dantzig42.tsp	699.0	699.0	699.0	699.0	699.0	699
Data/fri26.tsp	937.0	937.0	937.0	937.0	937.0	937
Data/gr17.tsp	2085.72	2085.41	2086.34	2085.41	**2086.14**	2085
Data/gr21.tsp	2707.0	2707.0	2707.0	2707.0	2707.0	2707
Data/gr24.tsp	1272.0	1272.0	1272.0	1272.0	1272.0	1272
Data/gr48.tsp	5065.52	5066.93	5069.41	**5063.07**	5068.34	5046
Data/gr120.tsp	7133.24	7130.14	7133.34	7133.14	**7123.31**	6942
Data/hk48.tsp	11546.79	**11525.76**	11530.1	11535.41	11527.69	11461
Data/pa561.tsp	3005.07	3010.59	3005.07	**3004.97**	3007.28	2763
Data/si175.tsp	**21486.72**	21489.93	21492.93	21489.72	21491.21	21407
Data/si535.tsp	48760.07	48764.72	48760.03	48770.03	**48754.0**	48450
Data/si1032.tsp	93168.24	93181.66	**93160.38**	93168.24	93181.07	92650
Data/swiss42.tsp	**1273.0**	1273.03	1273.03	1273.03	**1273.0**	1273

TABLE 6 – Expérimentation numérique TSP.

On observe sur le tableau 6 que les différentes implémentations que nous avons proposées ne domine pas particulièrement Reactive-GRASP bien que le bandit, particulièrement, trouve le plus de meilleures solutions.

instances	rg	roulette	optimal
Data/bayg29.tsp	1610.0	1610.0	1610
Data/bays29.tsp	2020.0	2020.0	2020
Data/brazil58.tsp	**25395.17**	25400.86	25395
Data/dantzig42.tsp	699.0	699.0	699
Data/fri26.tsp	937.0	937.0	937
Data/gr17.tsp	2085.72	**2085.41**	2085
Data/gr21.tsp	2707.0	2707.0	2707
Data/gr24.tsp	1272.0	1272.0	1272
Data/gr48.tsp	5065.52	**5063.07**	5046
Data/gr120.tsp	7133.24	**7133.14**	6942
Data/hk48.tsp	11546.79	**11535.41**	11461
Data/pa561.tsp	3005.07	**3004.97**	2763
Data/si175.tsp	**21486.72**	21489.72	21407
Data/si535.tsp	**48760.07**	48770.03	48450
Data/si1032.tsp	93168.24	93168.24	92650
Data/swiss42.tsp	**1273.0**	1273.03	1273

TABLE 7 – Expérimentation numérique TSP, comparatif Reactive-GRASP et roulette.

Sur le tableau 7 comparant Reactive-GRASP et roulette on voit que la roulette est meilleur que Reactive-GRASP sur plus d'instance. Ainsi il semblerait que, sur le TSP, Reactive-GRASP n'ait pas d'intérêt. Sur un problème où GRASP à du mal il donc peut être intéressant de développer d'autres algorithmes le manipulant comme le Bandit-Manchot, si l'on souhaite absolument utiliser GRASP.

8 Conclusion Finale

Pour conclure, l'analyse expérimentale que nous avons détaillé relève que la qualité des solutions de l'algorithme ϵ-greedy est supérieur ou équivalente en moyenne à toutes les variations de la composante Reactive-GRASP sur les trois problèmes d'optimisation (7). De plus, le type d'apprentissage proposé par le problème du Bandit-Manchot donne de résultats plus concluants avec la sélection d'un α qui ne change pas dans la plupart des instances des problèmes (6). A contrario, l'étude de la composante Reactive-GRASP ne nous a pas donné de résultats constants. En effet, il a une dépendance beaucoup trop forte envers la solution que GRASP retourne, car plus la solution est bonne, plus la classe sera récompensée. Le caractère aléatoire de GRASP fait donc en sorte que n'importe quelle classe peut retourner une bonne solution. C'est donc pour cela que l'apprentissage par renforcement offert par les Bandit-Manchot est de meilleure qualité. L'algorithme ϵ-greedy étant le plus simple de la littérature de ce problème, il pourrait être intéressant de continuer cette piste en implémentant une variante plus complexe afin de voir si une convergence plus agressive en ressort.

Références

[1] Marcelo PRAIS et Celso C RIBEIRO. "Reactive GRASP : An application to a matrix decomposition problem in TDMA traffic assignment". In : *INFORMS Journal on Computing* 12.3 (2000), p. 164-176.

[2] Thomas A FEO et Mauricio GC RESENDE. "A probabilistic heuristic for a computationally difficult set covering problem". In : *Operations research letters* 8.2 (1989), p. 67-71.

[3] Ramón ALVAREZ-VALDÉS, Francisco PARREÑO et José Manuel TAMARIT. "Reactive GRASP for the strip-packing problem". In : *Computers & Operations Research* 35.4 (2008), p. 1065-1083.

[4] Tor LATTIMORE et Csaba SZEPESVÁRI. *Bandit algorithms*. Cambridge University Press, 2020.

[5] Xavier DELORME, Xavier GANDIBLEUX et Joaquin RODRIGUEZ. "GRASP for set packing problems". In : *European Journal of Operational Research* 153.3 (2004), p. 564-580.

[6] Igor VASILYEV et al. "Generalized multiple strip packing problem : Formulations, applications, and solution algorithms". In : *Computers & Industrial Engineering* 178 (2023), p. 109096.

[7] Richard S SUTTON et Andrew G BARTO. *Reinforcement learning : An introduction*. 2018.

[8] Yohei Arahori, Takashi Imamichi et Hiroshi Nagamochi. “An exact strip packing algorithm based on canonical forms”. In : *Computers & Operations Research* 39.12 (2012), p. 2991-3011.

[9] A.W. Tucker et R.A. Zemlin C.E. Miller. “Integer Programming Formulations and Traveling Salesman Problems”. In : *Journal of the Association for Computing Machinery* 7 (1960), p. 326-329.

[10] G.A. Croes. “A Method for Solving Traveling-Salesman Problems”. In : *Operation Research* 6 (1958), p. 791-812.

Annexe

Expérimentation numérique complète

Pour chaque instance la valeur moyenne sur une trentaine d'expérimentations en haut puis la valeur médiane au milieu et la valeur min et max trouvé.
"*" indique que ce n'est pas la valeur optimale mais la meilleure connue.

instances	rg	tabou	seuil	roulette	bandit	optimal
Data/pb_100rnd0100.dat	372.0 372 372|372	372.0 372 372|372	372.0 372 372|372	372.0 372 372|372	372.0 372 372|372	372
Data/pb_100rnd0300.dat	203.0 203 203|203	203.0 203 203|203	203.0 203 203|203	203.0 203 203|203	203.0 203 203|203	203
Data/pb_100rnd0900.dat	463.0 463 463|463	463.0 463 463|463	463.0 463 463|463	463.0 463 463|463	463.0 463 463|463	463
Data/pb_200rnd0100.dat	410.97 411 404|416	410.86 411 406|414	410.62 411 404|416	408.24 408 402|414	408.14 407 404|414	416
Data/pb_200rnd0300.dat	714.34 715 706|722	715.66 716 702|723	716.59 718 708|722	712.69 712 706|720	710.72 710 703|723	731
Data/pb_200rnd0500.dat	183.28 184 177|184	182.41 184 176|184	182.59 184 173|184	183.83 184 183|184	182.86 184 173|184	184
Data/pb_500rnd0100.dat	316.66 317 310|319	316.62 316 315|320	316.55 317 311|319	315.28 315 310|318	315.41 315 315|319	323*
Data/pb_500rnd0300.dat	744.62 746 735|748	744.76 746 736|748	745.24 746 736|751	742.97 746 735|752	746.34 747 741|751	776*
Data/pb_500rnd0500.dat	122.0 122 122|122	122.0 122 122|122	122.0 122 122|122	122.0 122 122|122	121.86 122 118|122	122
Data/pb_1000rnd0300.dat	605.28 604 594|629	607.76 607 594|633	605.45 605 596|619	604.0 602 592|621	603.34 602 594|626	661*
Data/pb_1000rnd0500.dat	217.93 222 212|222	217.93 222 212|222	217.86 222 211|222	216.21 214 212|222	218.48 222 212|222	222*
Data/pb_2000rnd0300.dat	440.69 441 430|456	443.59 441 432|459	441.97 441 433|462	439.21 437 423|459	445.97 443 436|465	478*
Data/pb_2000rnd0500.dat	133.52 131 131|140	133.21 131 130|140	134.59 132 130|140	133.31 131 131|140	131.79 131 130|140	140*

TABLE 8 – Expérimentation numérique SPP complète.

instances	rg	tabou	seuil	roulette	bandit
Data/strip20_1	20.03 20 20|21	20.07 20 20|21	20.07 20 20|21	20.0 20 20|20	20.0 20 20|20
Data/strip20_2	21.59 22 21|22	21.38 21 21|22	21.52 22 21|22	21.48 21 21|22	21.45 21 21|22
Data/strip20_3	20.1 20 20|21	20.1 20 20|21	20.17 20 20|21	20.17 20 20|21	20.31 20 20|21
Data/strip40_1	16.0 16 16|16	16.0 16 16|16	16.0 16 16|16	16.0 16 16|16	15.97 16 15|16
Data/strip40_2	16.0 16 16|16	16.0 16 16|16	16.0 16 16|16	16.0 16 16|16	16.0 16 16|16
Data/strip40_3	16.0 16 16|16	15.97 16 15|16	16.0 16 16|16	15.93 16 15|16	16.0 16 16|16
Data/strip60_1	31.86 32 31|32	31.83 32 31|32	31.76 32 31|32	31.62 32 31|32	31.69 32 31|32
Data/strip60_2	32.21 32 31|33	32.21 32 31|33	32.24 32 32|33	32.17 32 32|33	32.07 32 32|33
Data/strip60_3	31.83 32 31|32	31.97 32 31|32	31.9 32 31|32	31.93 32 31|33	31.83 32 31|32
Data/strip80_1	127.41 128 125|128	127.17 127 125|128	127.45 128 126|128	127.34 127 125|128	127.34 128 126|128
Data/strip80_2	127.72 128 125|129	127.93 128 125|129	127.79 128 126|130	128.07 128 126|130	127.72 128 126|130
Data/strip80_3	128.1 128 126|129	128.14 128 126|129	127.97 128 126|129	128.03 128 126|129	127.69 128 126|129
Data/strip160_1	253.76 254 251|256	253.34 254 251|255	253.34 253 251|257	253.93 254 250|256	254.55 254 252|258
Data/strip160_2	253.79 254 250|256	253.41 254 251|256	252.62 253 248|255	253.55 254 250|257	253.21 253 250|257
Data/strip160_3	253.9 254 251|256	253.41 254 248|255	253.1 253 250|257	253.21 253 249|256	254.17 254 252|257

TABLE 9 – Expérimentation numérique StPP complète.

instances	rg	tabou	seuil	roulette	bandit	optimal
Data/bayg29.tsp	1610.0 1610 1610|1610	1610.0 1610 1610|1610	1610.0 1610 1610|1610	1610.0 1610 1610|1610	1610.0 1610 1610|1610	1610
Data/bays29.tsp	2020.0 2020 2020|2020	2020.0 2020 2020|2020	2020.0 2020 2020|2020	2020.0 2020 2020|2020	2020.0 2020 2020|2020	2020
Data/brazil58.tsp	25395.17 25395 25395|25400	25395.52 25395 25395|25400	25398.28 25395 25395|25480	25400.86 25395 25395|25475	25395.17 25395 25395|25400	25395
Data/dantzig42.tsp	699.0 699 699|699	699.0 699 699|699	699.0 699 699|699	699.0 699 699|699	699.0 699 699|699	699
Data/fri26.tsp	937.0 937 937|937	937.0 937 937|937	937.0 937 937|937	937.0 937 937|937	937.0 937 937|937	937
Data/gr17.tsp	2085.72 2085 2085|2088	2085.41 2085 2085|2088	2086.34 2085 2085|2088	2085.41 2085 2085|2088	2086.14 2085 2085|2088	2085
Data/gr21.tsp	2707.0 2707 2707|2707	2707.0 2707 2707|2707	2707.0 2707 2707|2707	2707.0 2707 2707|2707	2707.0 2707 2707|2707	2707
Data/gr24.tsp	1272.0 1272 1272|1272	1272.0 1272 1272|1272	1272.0 1272 1272|1272	1272.0 1272 1272|1272	1272.0 1272 1272|1272	1272
Data/gr48.tsp	5065.52 5066 5049|5080	5066.93 5066 5055|5080	5069.41 5074 5046|5080	5063.07 5063 5046|5080	5068.34 5074 5046|5078	5046
Data/gr120.tsp	7133.24 7133 7088|7187	7130.14 7136 7060|7194	7133.34 7139 7063|7187	7133.14 7136 7041|7175	7123.31 7131 7058|7169	6942
Data/hk48.tsp	11546.79 11545 11461|11624	11525.76 11511 11470|11611	11530.1 11532 11461|11603	11535.41 11532 11461|11614	11527.69 11532 11461|11617	11461
Data/pa561.tsp	3005.07 3006 2979|3028	3010.59 3012 2973|3031	3005.07 3007 2971|3024	3004.97 3007 2984|3029	3007.28 3008 2976|3024	2763
Data/si175.tsp	21486.72 21485 21465|21511	21489.93 21491 21454|21521	21492.93 21495 21456|21519	21489.72 21492 21456|21522	21491.21 21495 21454|21519	21407
Data/si535.tsp	48760.07 48765 48725|48785	48764.72 48769 48708|48806	48760.03 48761 48707|48811	48770.03 48774 48716|48810	48754.0 48760 48688|48792	48450
Data/si1032.tsp	93168.24 93169 93022|93308	93181.66 93193 93032|93346	93160.38 93170 92983|93285	93168.24 93160 93039|93300	93181.07 93211 92996|93269	92650
Data/swiss42.tsp	1273.0 1273 1273|1273	1273.03 1273 1273|1274	1273.03 1273 1273|1274	1273.03 1273 1273|1274	1273.0 1273 1273|1273	1273

TABLE 10 – Expérimentation numérique TSP complète.

Explication absence valeurs optimales StPP

Les instances que nous avons utilisées sont fournies avec les valeurs optimales pour le problème de compactage en bande où les rotations sont autorisées. Or notre problème n'autorise pas les rotations.

Nous aurions donc dû résoudre les instances avec le modèle que nous avons présenté. Cependant les contraintes (6) qui utilisent des "ou" créent un grand nombre de variables binaires, ce qui ralentit énormément la résolution.

Par manque de temps nous ne pouvions pas résoudre les instances avec ce modèle et n'avons pas eu le temps de chercher d'autres modèles ou algorithmes permettant de résoudre ces instances à l'optimalité.