1 Introduction

Le but de ce projet est de trouver le meilleur ordonnancement de photos qui maximize le score des transitions entre des diapositives de photos, de tel sorte que deux diapositives consécutives aient suffisament d'étiquettes en commun pour que la transition soit naturelle mais qu'elles aient en même temps suffisament d'étiquettes différentes pour que la transition ne soit pas monotone. Une diapositive est composée soit d'une photo horizontale, soit de deux photos verticales. Le Score de transition S entre deux diapositives A et B contenant respectivemet les étiquettes E_A et E_B se calcule de la manière suivante:

$$S = Min(E_A \cap E_B; E_A \setminus E_B; E_B \setminus E_A)$$

Nous allons tester différentes méthodes et comparer leurs performances sur des instances fournies lors de la compétition Google Hash Code 2019.

2 Complexité du problème

Le problème d'optimisation de qui est de trouver l'ordonnancement des diapositves maximisant le score de transitions est NP-difficile si et seulement si sa version décison est NP-complète. Le probleme de décision de l'ordonnancement des diapositives est le suivant: existe-il un ordonnancement de photos dont la somme des transitions et supérieure à K. Ce problème est-il NP-complet ? Ce problème appartient bien à P car le certificat étant l'ordonnancement des photos est vérifiable en temps polinomiale, en l'occurence il suffit de parcourir la chaine de photos et faire la somme des transitions et de vérifier qu'elle soit bien supérieur à K, ce qui est fait en temps linéaire en la taille de l'instance. On peut démontrer que ce problème est aussi NP-complet en trouvant une réduction polynomiale du probleme de décison de la chaine hamiltonniene vers le problème de décision de l'ordonnancement de diapositives contenant uniquement des photos horizontales. Le probleme de décision de la chaine hamiltoniene étant NP-complet, le problème de décision de l'odonnancement des diapositives horizontales le sera alors aussi. L'ordonnancement des diapositives contenant uniquement des photos horizontales étant une relaxation du problème contenant des photos horizontales et verticales, si le premier est NP-difficile alors le second le sera aussi.

Réduction

Pour réduire le probleme de décision de la chaine hamiltonienne à celui de décision de l'ordonnancement de photos, on part d'un graphe $G = (V, E)$ complet non orienté avec des poids entier négatifs sur toutes les arêtes $w(e) <= 0, \forall e \in E$. On associe à ce graph un ensemble de photos avec étiquettes de la manière suivante:

- on crée un nouveau graphe $G' = (V, E')$, en gardant les même sommets V.
- On garde les même arrêtes en changeant les poids par l'opposé $E' = \{e' | w(e') := -w(e), e \in E\}$
- Chaque sommet $v \in V$ représene une photo.

- Pour chaque arête e' on ajoute $w(e')$ mêmes étiquettes aux sommets aux extrémités. Pour chaque arête on créé de nouvelles étiquettes. De telle sorte que le nombre d'étiquettes en commun entre chaque paire de photos est égale au poids de l'arête reliant les sommets associés $w(e')$. Le fait de créer de nouveaaxu tags à chaque fois permet de s'assurer que des photos n'aient pas plus de tags en commun que le poids reliant les sommets associés.
- Pour chaque arrête reliant les sommet v_1 et v_2 on compte le nombre d'étiquettes d_1 présentes dans v_1 mais pas dans v_2 et si $d_1 < w(e')$ on ajoute $w(e') - d_1$ nouvelles étiquettes à v_1 de sorte à ce qu'il y ait au moins ($w(e')$ étiquette dans v_1 non présente dans v_2. et on fait de meme pour v_2.

On obtient donc une instance pour le problème d'ordonnancement de photos dans laquelle le score des transitions entre chaque paire de photos est égale au nombre d'étiquettes en commun entre ces deux photos, le nombre d'étiquettes en supplément d'une photo à l'autre étant toujours supérieur au nombre d'étiquettes en commun entre ces deux photos. Le problème de trouver une chaine hamiltoniene de longueur inférieure à K ($K <= 0$ et entier) est bien réductible polynomialement au problème de trouver un ordonnancement de photos étiquettés dont le score de transition est supérieur à $-K$.

3 Documentation utilisateur

Pour compiler le programme il suffit de taper la commande suivante à la racine du projet, en s'étant assuré que la variable d'environnement GUROBI_HOME contienne bien le chemin au répertoire contenant gurobi.

```
make
```

L'exécutables est généré dans le répertoire `bin/`.

3.1 Méthodes gloutonnes

La première méthode implémentée est une méthode gloutonne naive qui partant d'une diapositive aléatoire ajoute à la solution la diapositttive maximisant la transition parmis toutes les diapositives possibles avec les photos restantes. Complexité en $O(n^2)$.

Elle s'utilise de la manière suivante:

```
bin/Slideshow naive_greedy instances/<instance_file> <pourcentage>
solutions/<output_file>.sol
```

`<pourcentage>` étant le pourcentage de l'instance sur lequel executer le programme.

La secondes est une méthode gloutonne fonctionnant de la même manière, mais ne regardant pas parmis toutes les diapositives possibles mais seulement l diapositives. Cela permet d'être plus rapide pour les grosses instances. Complexité en $O(n \times l)$.

Elle s'utilise de la manière suivante:

```
bin/Slideshow greedy instances/<instance_file> <pourcentage>
solutions/<output_file>.sol -l <length>
```

`-l <length>` étant le nombre de diapositives que l'algorithme va regarder à chaque nouvel ajout de diapositive, défaut : 1000.

3.2 Méthode de descente stochastique

Partant d'une solution aléatoire ou d'une solution initiale donnée. On selectionne une diapositive (ou une photo verticale d'une diapositive), et on essaye de l'échanger avec de nombreuses autres diapositives (ou d'e photos verticales d'autres diapositives) pour voir lequelle de ces échanges permet de maximiser le gain de score de transitions.

Elle s'utilise de la manière suivante:

```
bin/Slideshow stoch_descent instances/<instance_file> <pourcentage>
solutions/<output_file>.sol -n <nb_neighbours> -i <max_iter>
-t <time_limit> -s <starting solution file>
```

`-n <nb_neighbours>` étant la taille du voisinage regardé pour la déscente, défaut : 1000
`-i <max_iter>` étant le nombre maximal d'itérations de l'algorithme, défaut : 10000
`-t <time_limit>` étant le temps maximal d'exécution de l'algorithme, défaut : infini
`-s <starting_solution_file>` une solution existante à partir de laquelle exécuter l'algorithme.

3.3 Méthode de recuit simulé

Partant d'une solution aléatoire, On selectionne deux diapositives (ou deux photos verticales de deux diapositives différentes) et on les échanges si le score de transition est meilleur, ou si le coefficient de température l'autorise l'échange même si le score n'augmente pas. Avec un coefficient de température qui diminue au file du temps.

Elle s'utilise de la manière suivante:

```
bin/Slideshow annealing instances/<instance_file> <pourcentage>
solutions/<output_file>.sol -i <max_iter>
-t <time_limit> -s <starting_solution_file>
```

`-i <max_iter>` étant le nombre maximal d'itérations de l'algorithme, défaut : 10000
`-t <time_limit>` étant le temps maximal d'exécution de l'algorithme, défaut : infini
`-s <starting_solution_file>` une solution existante à partir de laquelle exécuter l'algorithme.

3.4 Algorithme génétique

Partant d'une population de solutions aléatoire. On selectionne les individus ayant les meilleurs scores et on les croises pour former une nouvelle génération, puis on applique des mutations sur les nouveaux individus. Ce la dans le but de voir émerger des solution performantes par un processus evolutionnaire.

Elle s'utilise de la manière suivante:

```
bin/Slideshow genetic instances/<instance_file> <pourcentage>
solutions/<output_file>.sol -n <population_size>
-i <max_iter> -t <time_limit>
```

`-n <population_size>` étant la taille de la population, défaut : 1000
`-i <max_iter>` étant le nombre maximal d'itérations de l'algorithme : défaut 10000
`-t <time_limit>` étant le temps maximal d'exécution de l'algorithme, défaut : infini

3.5 Programmation linéaire en nombre entier (PLNE)

Implémentation du PLNE donnée en annexe du sujet du projet, en l'adaptant à une maximisation de l'objectif à la place d'une minimisation, et en supprimant l'arrête de score minimal dans le cycle retoruné par le programme. Les trois techniques de suppression des sous-tour qui sont "MTZ" "flots" et "flots désagrégés" sont implémentés.

Elle s'utilise de la manière suivante:

```
bin/Slideshow ilp instances/<instance_file> <pourcentage>
solutions/<output_file>.sol -f <sous_tour>
```

`-f <sous_tour>` étant la techinque de suppression des sous tour à utiliser parmis: "MTZ", "flow" et "desaggregated flow", défaut : "flow".

3.6 Méthode d'arrondie

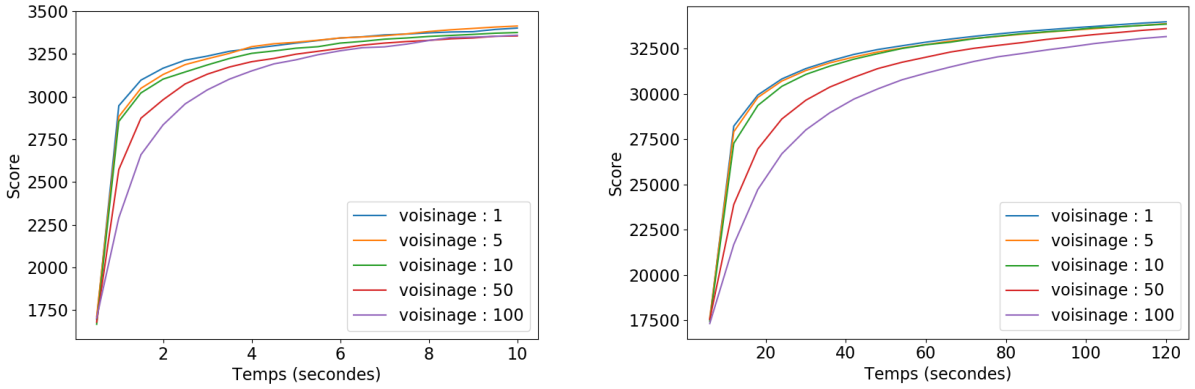
Relaxation du PLNE précédent en programme linéaire (PL) à nombre réels. Puis construction de la solution en ajoutant les arrêtes par ordre décroissant suggéré pas le PL sans former de cycle. Les trois techniques de suppression des sous-tour qui sont "MTZ", "flots" et "flots désagrés" sont implémentés, il est aussi possible de ne pas utiliser de suppression des sous tours lors de la résolution du PL mais le résoudre lors de la construction de la solution après résolution du PL.

Elle s'utilise de la manière suivante:

```
bin/Slideshow round instances/<instance_file> <pourcentage>
solutions/<output_file>.sol -f <sous_tour>
```

4 Analyze expérimentale

4.1 Descente stochastique



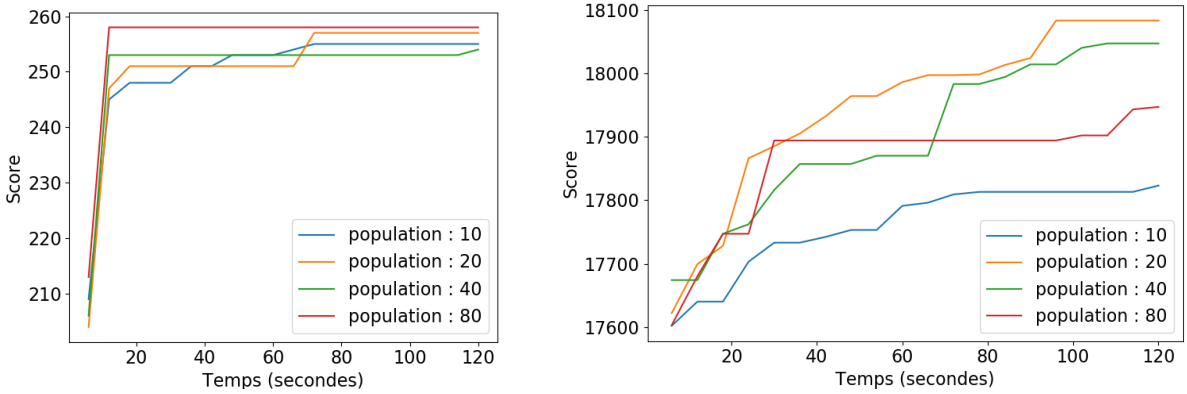
(a) 1% de l'instance d_pet_pictures, soit 900 photos

(b) 10% de l'instance d_pet_pictures, soit 9000 photos

Figure 1: algorithme de descente stochastique pour différentes tailles d'instances

Le nombre de voisin observé ne semble pas avoir un impact significatif sur le résulat. Néamoins il semble qu'observer un unique voisin à chaque itération et choisir ce voisin s'il est meilleur que la solution courante converge un peut plus rapidement, mais toutes les versions convergent vers le meme résultat.

4.2 Algorithme génétique



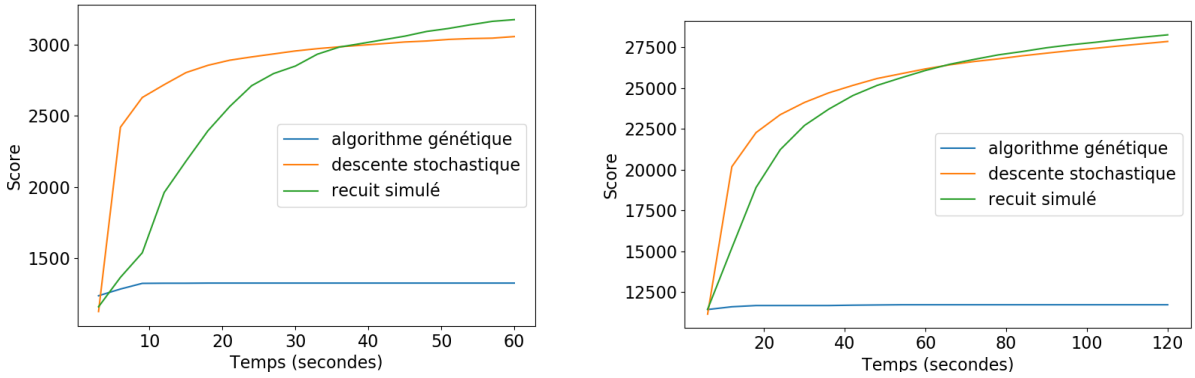
(a) 0.1% de l'instance d_pet_pictures, soit 90 photos

(b) 10% de l'instance d_pet_pictures, soit 9000 photos

Figure 2: algorithme génétique pour différentes tailles d'instances

Il ne semble pas y avoir de déffirence significative selon la taille de la population, ce qui peut etre expliqué par : une petite population aura du mal à générer de nouvelles solution efficaces, il y aura de nombreuses générations mais sans grand diversité, contrairement aux grandes population qui ont une rgande diversité mais qui sont longues à calculer et donc peu de générations successives peuvent être crées en un lapse de temps donné.

4.3 Descente stochastique, recuit simulé et algorithme génétique



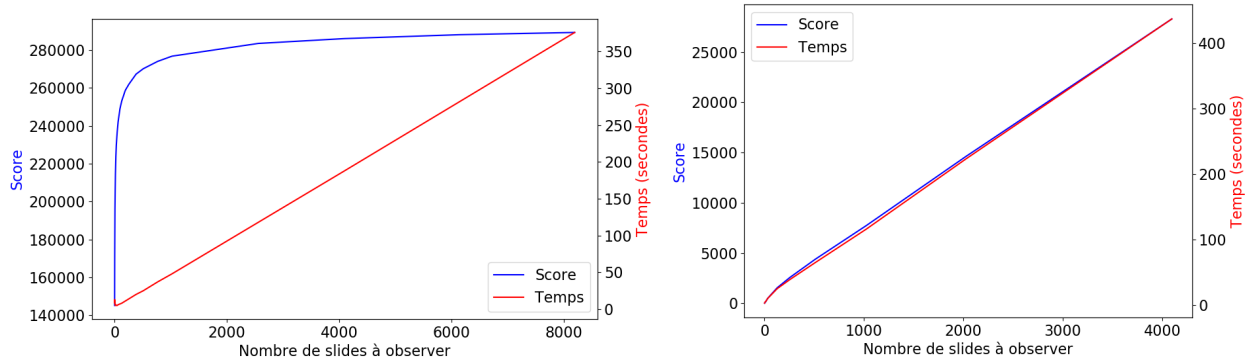
(a) 1% de l'instance d_pet_pictures, soit 900 photos

(b) 10% de l'instance d_pet_pictures, soit 9000 photos

Figure 3: algorithme stochastique, recuit simulé et génétique pour différentes tailles d'instances

On observe une réelle différence entre d'une part l'algorithme génétique et d'autre part les algorithmes de descente stochastique et de recuit simulé. Notre implémentation de l'algorithme génétique ne semble pas dutout adapter à la résolution de ce problème et n'obtient pas de bons résultats. On observe aussi clairement que pour n'importe quelle taille d'instance la méthode du recuit simulé est la méthode qui donne les meilleurs résultas résultats.

4.4 Méthode gloutonne



(a) Instance d_pet_pictures, soit 90000 photos

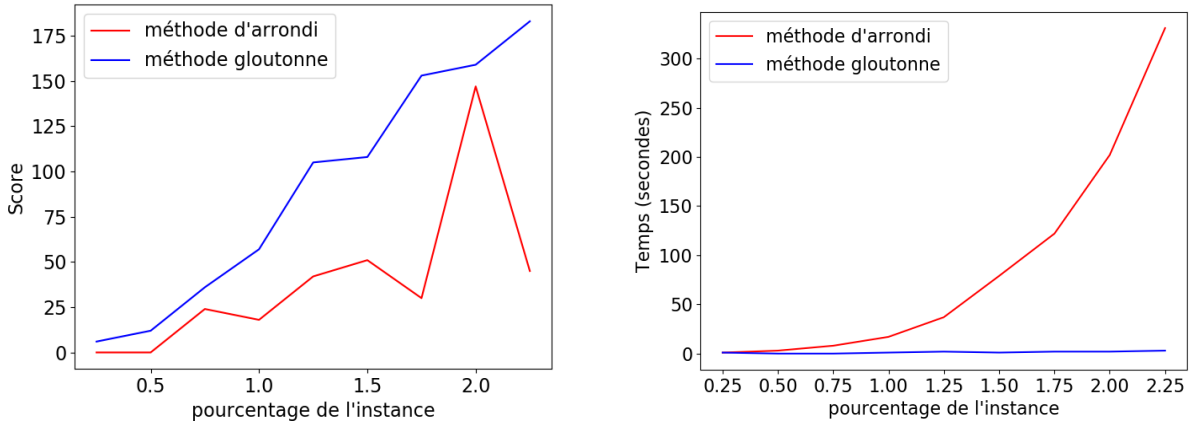
(b) Instance b_lovely_landscapes, soit 80000 photos

Figure 4: algorithme glouton pour différentes instances

On observe sans surprise que plus l'algorithme observe de transitions plus la solution gloutonne sera efficace mais plus le temps de calcul sera long. Sur l'instance d_pet_pictures le score augmente très fortement jusqu'a environ 1000 transitions observés, puis cette progression ralenti très fortement au dela, alors que le temps de calcul augmente de facon linéaire avec le nombre de transitions observés. Contrairement à l'instance b_lovely_landscapes ou le score tout comme les temps de calcul augmente de facon linéaire avec le nombre de transitions observés.

Pour l'instance d_pet_pictures l'algorithme glouton prend 350 secondes à s'executer pour un nombre de transitions observés de 8000, si l'on considère le temps de calcul en fonction du nombre de transitions observés linéaire, par un rapide calcul on peut en déduire que sur cette instance l'algorithme glouton naif observerai de l'ordre de 90000 transitions à chaque transition i, et s'executerai donc en plus de 3900 secondes pour n'obtenir un score qui ne sera possible pas beaucoup mieux que le score obtenue avec 1000 transisions observés. Alors que sur l'instance le score comme le temps de calcul augmentant de facon linéaire, il peut être utile d'executer l'algorithme glouton naif pout tenter d'obtenir un bon score.

4.5 Glouton vs arrondi

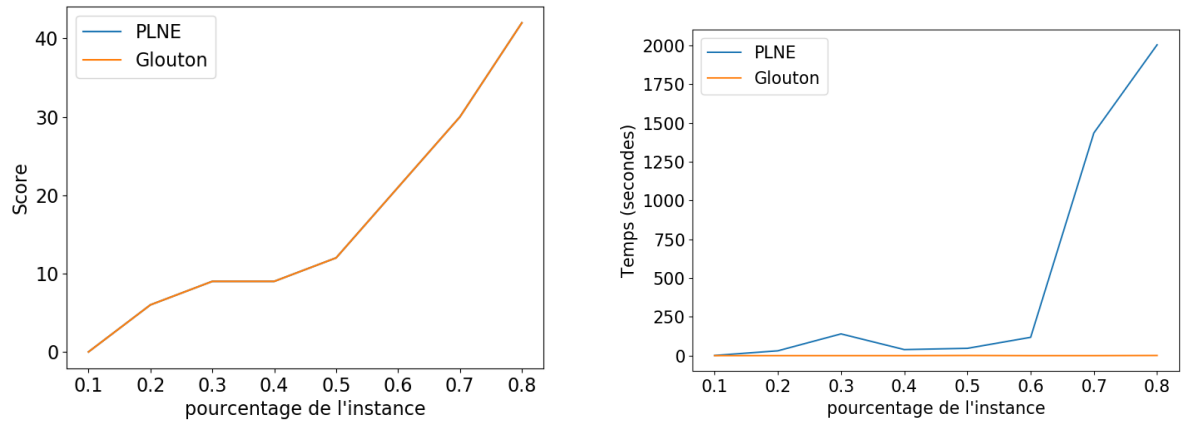


(a) score en fonction de la taille de l'instance (b) temps de calcul en fonction de la taille de l'instance

Figure 5: algorithme glouton pour différents pourcentages de l'instances b_lovely_landscapes qui contient au total 80000 photos

On observe que l'algorithme glouton est bien plus performant que la méthode d'arrondi en temps de calcul, le temps de calcul de la méthode d'arrondi augmentant de manière exponentielle avec la taille des instances. Et les solution générée par l'algorithme glouton sont meilleurs. Par exemple pour 2% de l'instance soit 1600 photos, la m"thode de l'arrondi nécessite 5 minutes de temps de calcul alors que la méthode gloutonne est quasiment instantanée.

4.6 Glouton vs PLNE



(a) score en fonction de la taille de l'instance (b) temps de calcul en fonction de la taille de l'instance

Figure 6: algorithme glouton pour différents pourcentages de l'instances b_lovely_landscapes qui contient au total 80000 photos

Le PLNE utilisé ici est la version avec les sous tours supprimé par la méthode des flots, les méthodes MTZ et de flots désagrégés ne permettant pas de trouver la solution optimale pour 0.1% de l'instance donnée en moins de 15 minutes et leurs temps de calcul augmentant de facon exponentiel avec la taille de l'instance. On observe que le PLNE et l'algorithme glouton obtiennent éxactement les même résultats en terme de score, alors que le temps de calcul explose littéralement avec la taille de l'instance pour le PLNE avec méthode des flots. Il est sûr que pour des instances plus grandes il y aura une différence de score entre les deux algorithmes étant donné que le PLNE donne le résultat optimal alors le l'algorithme glouton non, mais on ne peut pas vérifier cela, les temps de calcul du PLNE étant trop élévés.

5 Meilleurs résultats obtenus

La méthode la plus efficace pour maximiser les scores de transitions entre les slides pour la majorité des instances est d'utiliser un algorithme de recuit simulé partant d'une solution initiale aléatoire. C'est cette méthode que nous avons utilisés pour les instances c_memorable_moments, d_pet_pictures et e_shiny_selfies. La seule instance pour laquelle cette méthode n'est pas efficace est l'instance b_lovely_landscapes pour laquelle nous avons obtenu les meilleurs résultats avec la méthode gloutonne. Voici les meilleurs scores que nous avons obtenus :

Instance	Score
b_lovely_landscapes	202605
c_memorable_moments	1609
d_pet_pictures	320607
e_shiny_selfies	243332

6 Conclusion

Pour toutes les instances du Google HashCode 2019 et quelque soit la taille d'instance on observe que les méthodes de PLNE et d'arrondis ne donnent pas de résultats satisfaisant en terme de temps de calcul qui croit de facon exponentiel en la taille des instances. L'algorithme génétique n'a pas permis d'obtenir de bon résultats, nous pensons que le problème vient de notre implémentation, car même sur de petites instances les résultats obtenus sont très faible surtout comparé aux autres méthodes. La méthode Gloutonne quand à elle est très efficace en temps de calcul et donne des résultats plutot bons en moyenne, même si comme toujours cela dépend énormément de la nature de l'instance. Une des deux méthodes les plus efficaces que nous avons testés est la méthode de descente stochastique, qui est d'autant plus efficace quand elle est intialisé à une solution obtenue par algorithme glouton. La deuxième méthode qui s'est révélé être très efficace est l'algorithme du recuit simulé, qui sur certaines instance donne approximativement le même résulats que la méthode déscente stochastique alors que sur d'autre instances les résultats sont nettement meilleurs.