AC - TP4

Théo Le Goc **Robin Gaignoux**

1. Introduction

Dans ce TP, nous aborderons les algorithmes génétiques appliqués aux problèmes du sac à dos et du voyageur de commerce. Les algorithmes génétiques nous permettent de se rapprocher d'une solution idéale à un problème en un temps plus raisonnable : en effet, certains problèmes (comme ceux que nous allons essayer de résoudre) ont une solution de complexité conséquente, souvent de l'ordre de $\theta(n^2)$.

L'idée générale de ce TP sera donc de générer une population initiale de solutions, puis de les améliorer au fur et à mesure à travers des opérations de sélection, croisement et mutation, afin d'obtenir la meilleure solution au problème donné.

2. Problème du sac à dos

Le problème du sac à dos vise à optimiser le choix d'objets à placer dans un sac en maximisant la somme de leurs valeurs, tout en respectant la contrainte que la somme totale de leurs poids ne dépasse pas la capacité du sac.

Pour représenter un individu associé au problème du sac à dos, nous utilisons 3 attributs :

- un tableau représentant les poids de chacun des objets du sac
- une capacité maximale
- un tableau de booléens qui indique les objets contenus dans le sac de l'individu (0 : non contenu, 1 : contenu)

Chaque individu créé dans la population d'origine possède un tableau de booléens aléatoire.

Chaque itération va sélectionner aléatoirement (de façon pondérée afin d'avoir en priorité les meilleurs individus) 2 parents et effectuer un croisement pour obtenir 2 individus enfants. Pour cela, on sélectionne un point de coupure dans le tableau représentant les objets choisis du premier conjoint, puis on intervertit une des moitiés par l'autre moitié correspondante chez le second conjoint. Il y aura ensuite une passe de mutation pour chaque enfant, puis l'algorithme peut passer à la génération suivante si le résultat n'est pas trouvé.

Nous pouvons maintenant générer les populations successives en se faisant reproduire les individus précédents, selon une certaine probabilité de mutation. Nous testons notre algorithme génétique sur plusieurs fichiers de poids, en testant avec différentes probabilité de mutation :

Pour un fichier contant 28 objets, nous souhaitons obtenir un sac ayant une capacité se rapprochant le plus possible de 1581. Nous obtenons les résultats suivants :

```
[121.0 380.0 56.0 22.0 52.0 7.0 29.0 26.0 676.0 8.0 188.0 16.0 ]
```

Résultats sur 1000 tests pour chaque variations de paramètres

Iterations max	100	100	1000	1000
Nombre d'individus	100	100	100	100
Probabilité de mutation	0.01	0.1	0.01	0.1
Nombre moyen d'itérations	10	8.1	88.4	8.7
% de résultats corrects	57.7%	100%	81.3%	100%

Pour un fichier contant 70 objets, nous souhaitons obtenir un sac ayant une capacité se rapprochant le plus possible de 350. Nous obtenons les résultats suivants :

Résultats sur 1000 tests pour chaque variations de paramètres

Iterations max	1000	1000
Nombre d'individus	100	100
Probabilité de mutation	0.01	0.1
Nombre moyen d'itérations	30.5	84.1
% de résultats corrects	100%	100%

Après ces tests, nous pouvons nous apercevoir que les paramètres d'initialisation vont considérablement changer la performance et le résultat sortant de l'algorithme. Dans le premier cas, nous remarquons qu'une probabilité de mutation plus élevée permet d'atteindre un résultat beaucoup plus rapidement; cependant, dans le test suivant, l'inverse se produit.

On notera que la précision de notre algorithme est aussi liée à la limite d'itérations imposées. Il serait judicieux de mettre une limite très haute voire pas de limite, au risque d'avoir un algorithme qui ne s'arrête jamais.

3. Problème du voyageur de commerce

Le problème du voyageur de commerce a pour objectif de déterminer le plus court circuit passant une seule fois par chaque ville parmi un ensemble donné de villes, et revenant au point de départ.

Pour représenter un individu associé au problème du voyageur de commerce, nous utilisons 3 attributs :

- un tableau représentant les coordonnées des villes, sur l'axe horizontal
- un tableau représentant les coordonnées des villes, sur l'axe vertical
- un tableau représentant le circuit utilisé par l'individu pour traverser les villes

De manière similaire au problème précèdent, nous générons des populations de façon successive, pour obtenir l'individu possédant la meilleure solution au problème, c'est-à-dire le chemin le plus court.

La reproduction des parents va cependant fonctionner différemment du problème SAD. En effet, on ne peut pas simplement intervertir une partie de tableau, au risque d'avoir certaines villes visitées plusieurs fois. Nous procédons donc en 3 étapes :

1. Choix d'un point de coupure et copie de chaque première partie de tableau dans les enfants
2. Copie des villes de la suite du tableau pour chaque conjoint uniquement si la ville n'a pas été déjà visitée
3. Ajout successif des villes non rencontrées dans le 2nd parent à la fin de chaque tableau pour les enfants.

Pour ce problème, il n'est malheureusement pas possible de poser un point d'arrêt, étant donné qu'on ne connais pas l'objectif à l'avance. Nous testons notre algorithme sur différents nombres de villes.

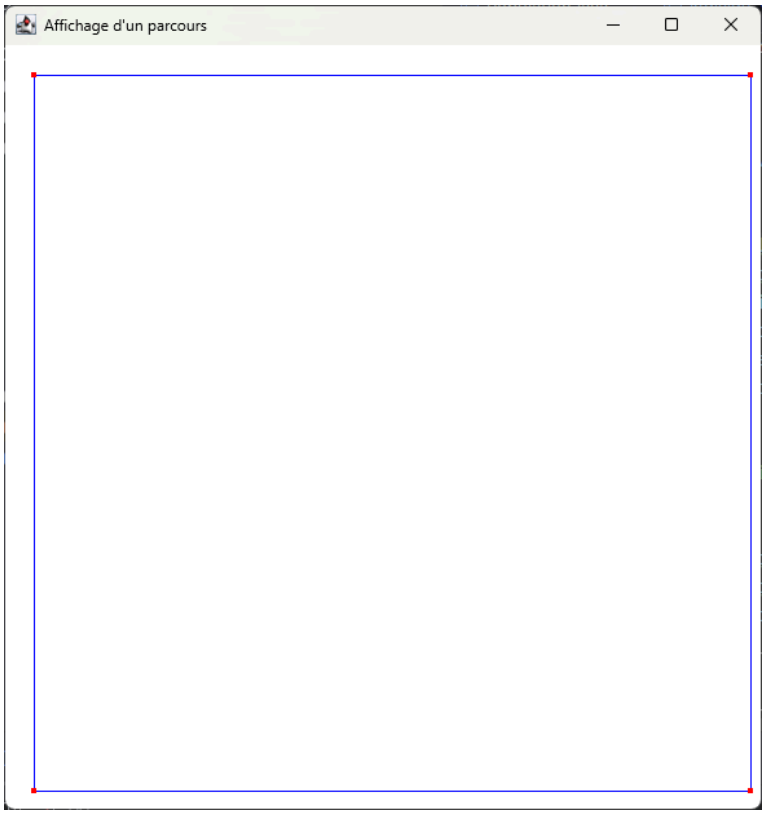

Figure 1: Chemin pour 4 villes, 1000 individus, probabilité de mutation de 0.01 et 1000 itérations max. Distance: 4

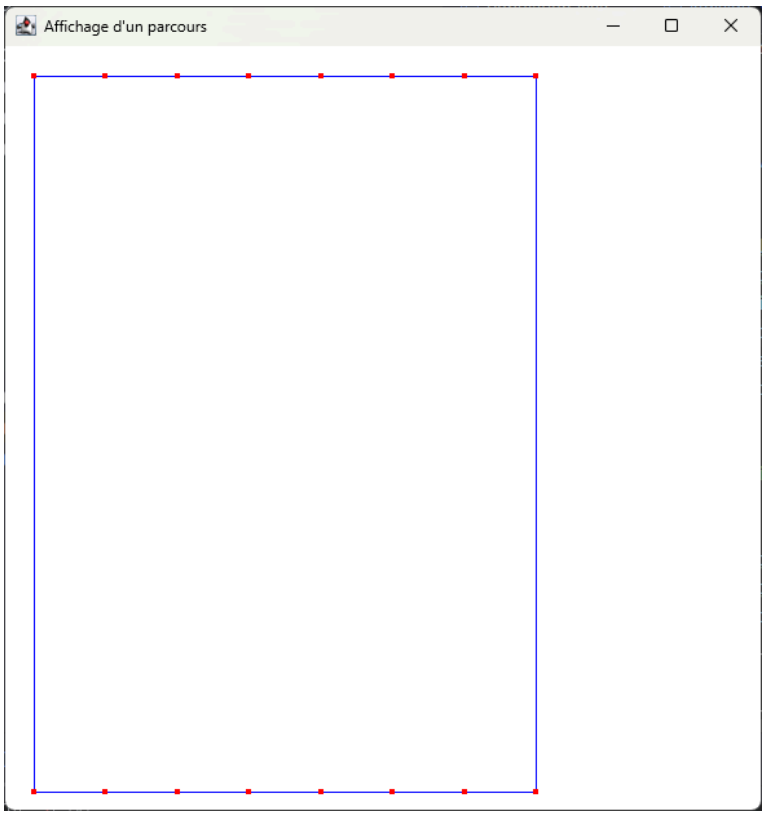

Figure 2: Chemin pour 64 villes, 1000 individus, probabilité de mutation de 0.01 et 1000 itérations max. Distance: 34

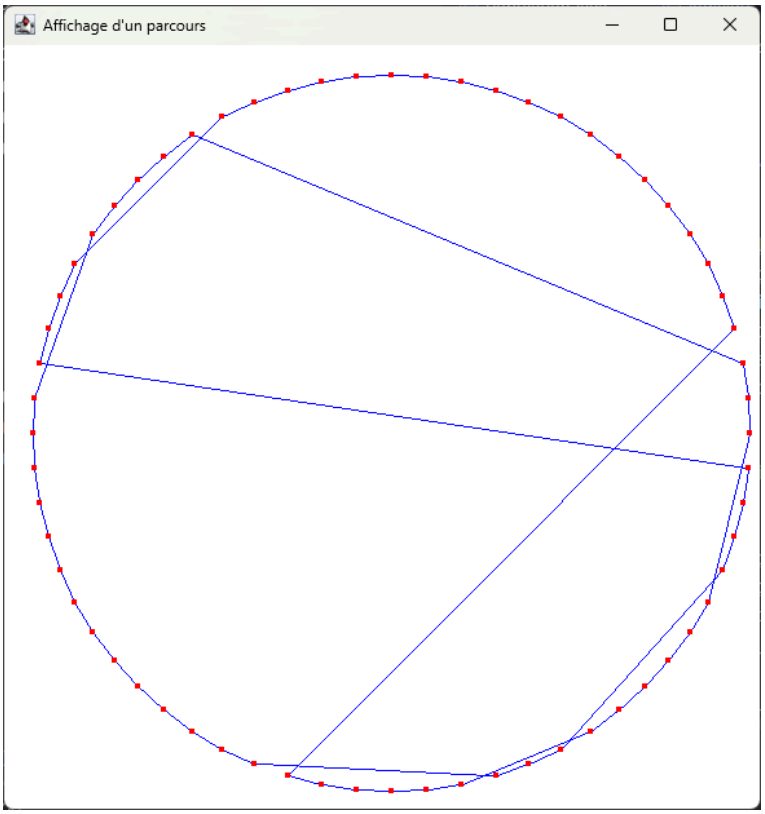

Figure 3: Chemin pour 64 villes, 1000 individus, probabilité de mutation de 0.01 et 1000 itérations max. Distance: 14.11

Pour un nombre de villes assez faible, le résultat semble être le plus optimal. Cependant, dès que le nombre de villes augmente, le résultat semble moins cohérent. Cela est probablement dû à notre fonction de croisement, qui ne prend pas en compte les meilleurs résultats et se contente de fusionner bêtement les chemins des 2 parents.

Il faut tout de même noter que le résultat est obtenu beaucoup plus rapidement en comparaison à un algorithme standard : dans notre cas, nous avons 1000 itérations, alors qu'avec une approche naïve, nous aurions n! itérations, soit $1.2e^{89}$ itérations pour le cas à 64 villes.

4. Conclusion

Pour conclure, nous retenons que les algorithmes génétiques permettent de résoudre des problèmes normalement "infaisables" en un temps raisonnable, parfois au prix de la précision. Notre algorithme ne fournit pas le résultat "parfait", notamment dans le cas du VDC. Il peut donc être amélioré, même si les résultats obtenus sont déjà très satisfaisants.