Algorithme d'optimisation appliqué au problème du voyageur du commerce

Sacha Ployet, Telo Philippe, Yente Poirot

23 juin 2021

Résumé

Le but de ce TIPE est de réaliser un algorithme donnant une solution satisfaisante au problème du voyageur du commerce en temps polynomial à l'aide du modèle des colonies de fourmis (*Ant Colony Optimization*)

Table des matières

1 Problème du voyageur de commerce

On considère un ensemble de points sur un plan euclidien. Le problème est de déterminer le chemin le plus court passant par tous les points, et une seule fois par chaque point, à la manière d'un marchand voulant visiter toutes les villes d'une région en limitant au maximum ses déplacements. Ce problème d'apparence simple est très difficile à résoudre de manière exacte.

2 Algorithme de recherche exhaustive

2.1 Fonctionnement

Une première approche consiste à générer tous les chemins possibles du marchand à travers le réseau de points et ainsi en déduire le parcours de longueur minimale. Cette méthode simple à l'avantage d'être exacte. On peut implémenter cette méthode par récursivité sur les différents points du réseau.

2.2 Résultats

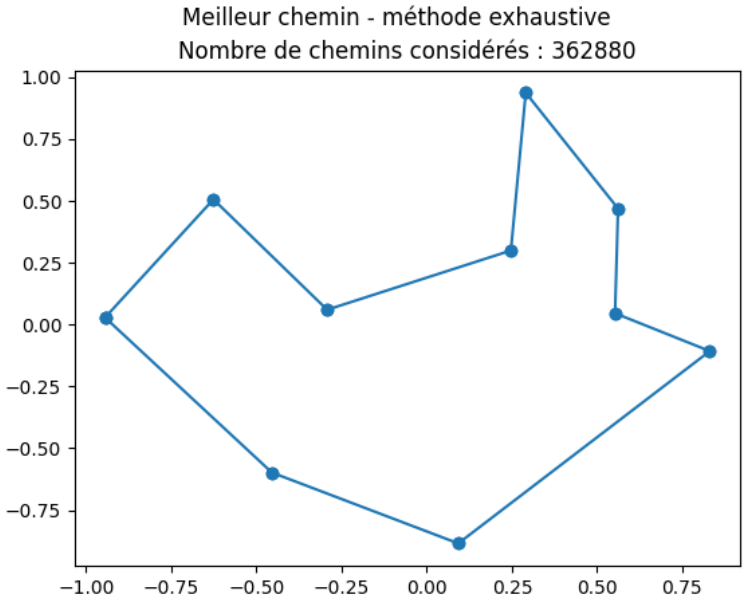

FIGURE 1 – Résultats de l'algorithme sur 10 points

2.3 Limites

Bien que garantissant la meilleure solution indépendamment du nombre de points, cet algorithme est très peu efficace : la complexité est en $O((n-1)!)$ pour n points. Cette approche nécessite donc un temps de calcul déraisonnable pour des réseaux comportant un nombre de points important.

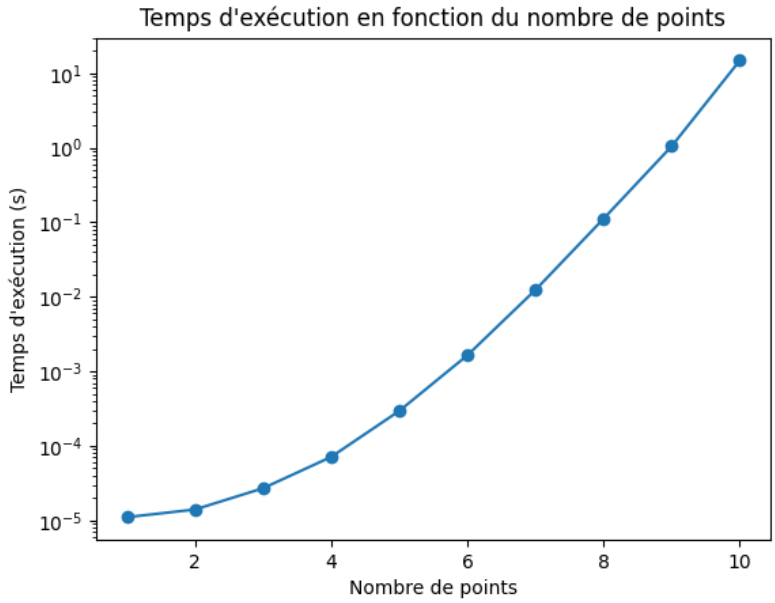

FIGURE 2 – Explosion du temps de calcul

3 Algorithme de colonies de fourmis

3.1 Motivation : les heuristiques

Pour un nombre de points élevé, il n'est donc pas envisageable d'utiliser la méthode exhaustive. On cherche alors une méthode qui permet de limiter considérablement le nombre de chemins considérés, pour obtenir un résultat approché en un temps raisonnable. Des algorithmes itératifs se basant sur l'aléatoire, appelés heuristiques, permettent d'obtenir ce type de résultat. Parmi cette famille d'algorithmes, le principe d'optimisation par colonie de fourmis est particulièrement adapté à la résolution ce problème.

3.2 Fonctionnement

3.2.1 Description qualitative

L'algorithme se décompose en une série de générations, une génération correspondant à une unique fourmi réalisant une suite d'actions prédéfinies.

1. La fourmi construit un des trajets possibles en choisissant de proche en proche les différents points qui le compose. Cette sélection se fait suivant une loi de probabilité.
2. Lorsqu'une fourmi a choisi un nouveau chemin, elle y dépose une piste de phéromones d'intensité inversement proportionnelle à la longueur de ce parcours.
3. On répète plusieurs fois ce procédé, et après plusieurs itérations, chaque arrête du meilleur chemin est renforcée par rapport aux autres, donc les fourmis ont plus de chances d'emprunter ce chemin.
4. On en déduit le meilleur chemin trouvé par les fourmis.

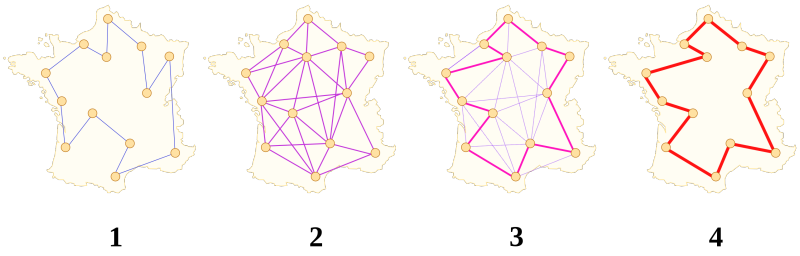

FIGURE 3 – Différentes étapes de l'algorithme

3.2.2 Description plus formelle

On définit $\rho_{m_1m_2}$ la quantité de phéromones présente sur le segment m_1m_2 et $\delta_{m_1m_2}$ la visibilitée entre deux points m_1 et m_2 telle que :

$$\delta_{m_1m_2} = \frac{1}{\|\overrightarrow{m_1m_2}\|}$$

Soit P_r l'ensemble des points qui ne composent pas le trajet déjà construit par la fourmi, la probabilité que la fourmi se trouvant en m_1 décide d'aller ensuite en m_2 est :

$$p_{m_1m_2} = \frac{(\rho_{m_1m_2})^\alpha \cdot (\delta_{m_1m_2})^\beta}{\sum_{m_i \in P_r} (\rho_{m_1m_i})^\alpha \cdot (\delta_{m_1m_i})^\beta} \in [0,1]$$

avec α et β deux coefficients de contrôle.

A partir de ces formules de probabilités, la fourmi détermine un chemin, sous la forme d'un n-uplet :

$$\begin{aligned} C &= (c_1, c_2, ..., c_p) \\ &= (m_{\sigma(1)}, m_{\sigma(2)}, ..., m_{\sigma(p)}) \end{aligned}$$

avec $\sigma \in S_p$.

Enfin, soit

$$l = \|\overrightarrow{c_p c_1}\| + \sum_{k=1}^{p-1} \|\overrightarrow{c_k c_{k+1}}\|$$

la longueur de ce chemin, pour tout $k \in [\![1, p-1]\!]$, on ajoute $1/l$ à $\rho_{c_k c_{k+1}}$, ainsi qu'à $\rho_{c_p c_1}$.

On peut donc faire une étude de complexité : si on note n le nombre de générations et p le nombre de points, la fonction $fourmi$ est appelée n fois. On détermine alors la complexité de la fonction fourmi :

Pour k allant de 1 à p, la fonction *gen_probas* est appelée avec une complexité en $O(k)$. Donc la fonction fourmi a une complexité en $O(p^2)$, et l'algorithme complet a une complexité en $O(np^2)$.

3.3 Implémentation en Python

Soit p le nombre de points et n le nombre de générations.

1. A l'initialisation, on crée une matrice numpy $p \times p$ représentant les phéromones (où $pheromones[i, j]$ est la quantité de phéromones sur le chemin $c_i c_j$). On initialise de même une matrice des distances, afin de limiter l'utilisation de la racine carrée.
2. Pour la fonction fourmi : on fixe le premier point du chemin au premier point de la liste, car le point initial n'a pas d'importance.Ensuite, on calcule pour chaque autre point sa probabilité d'être choisi, en utilisant la formule donnée précédemment, puis on utilise *numpy.random.choice* sur ces probabilités afin de déterminer le point suivant. On répète ensuite ce procédé, en ajoutant ces points successifs dans une liste python, jusqu'à obtenir le chemin complet. De plus, la longueur du chemin est calculée en parallèle de la construction du chemin, en ajoutant à chaque nouveau point la distance du point précédant à ce point.
3. Ensuite, dans une boucle for allant de 0 à $n-1$, on appelle cette fonction $fourmi$ qui renvoie un chemin et sa longueur, on ajoute l'inverse de cette longueur à tout les coefficients de la matrice de phéromones concernés. On garde de plus en mémoire le meilleur chemin calculé jusqu'alors, afin de l'afficher à la fin de l'exécution.
4. Enfin, on affiche les différent graphiques et le chemin en utilisant les fonctions de *matplotlib*.

3.4 Résultats

On obtient très rapidement la solution optimale (même solution qu'avec le programme exhaustif) avec cet algorithme :

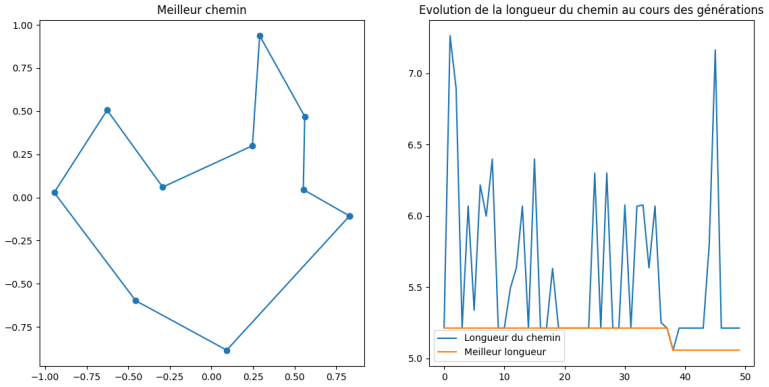

FIGURE 4 – Résultats de l'algorithme sur 10 points et 50 générations

On obtient également un *bon* résultat pour un ensemble de 80 points :

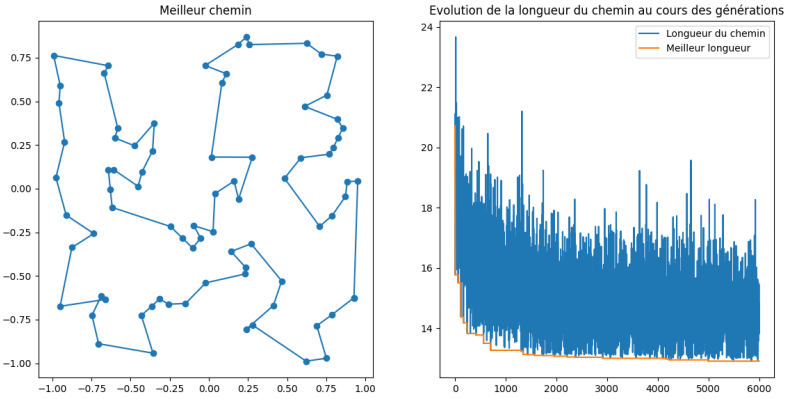

FIGURE 5 – Résultats de l'algorithme sur 80 points et 6000 générations

3.5 Limites et points d'amélioration

Même si l'algorithme renvoie des solutions qui semblent s'approcher de la solution exacte, il n'existe aucun moyen de comparer avec la solution exacte pour estimer l'erreur.

Ensuite, même si nous ne considérons que deux paramètres α et β dans notre implémentation, d'autres parties de l'algorithme peuvent être paramétrées, comme

par exemple un coefficient en exposant des phéromones ajoutées, ou un facteur d'évaporation. Tous ces paramètres sont arbitraires, et nécessitent beaucoup de tests pour pouvoir être optimaux.

Enfin, plus le nombre de générations est grand, meilleur est le résultat. Donc pour un grand nombre de points on peut théoriquement obtenir le meilleur chemin après un grand nombre de générations, mais il est difficile de savoir combien de générations sont nécessaires.

Références

[1] Algorithme de colonies de fourmis, Wikipedia

[2] Métaheuristique, Wikipedia