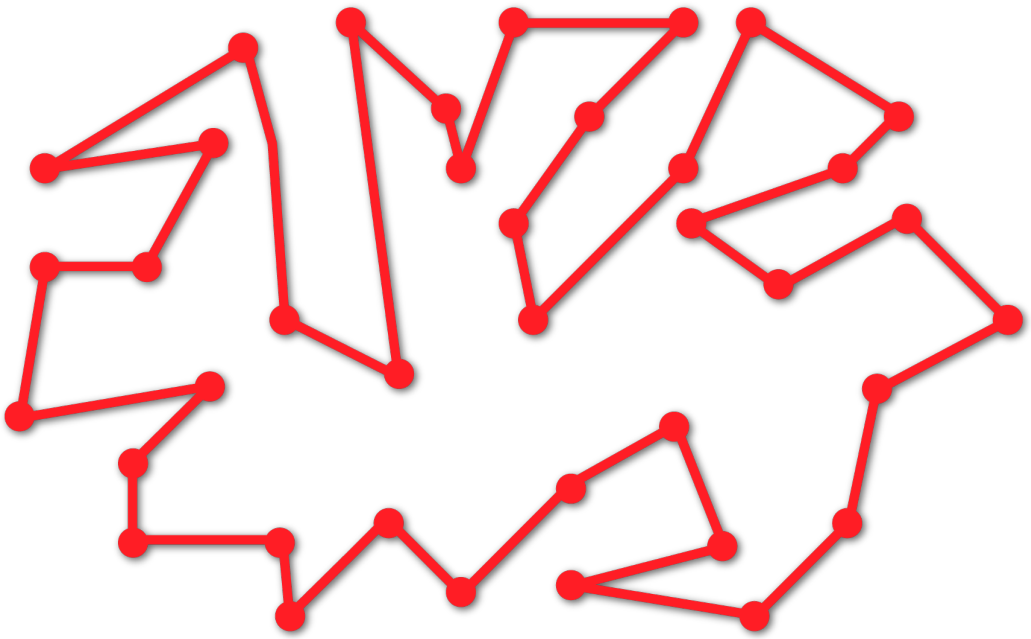
ecpm
École européenne d'ingénieurs
de **chimie, polymères** et **matériaux**
Université de Strasbourg

Problème du voyageur de commerce

Caroline Solon, Floréal Rouxel, Alexis Lemoine

22 Octobre 2023



Sommaire :

Introduction au problème du voyageur du commerce……3

Application du problème à la chimie……4

Algorithme du recuit simulé……5

Algorithme génétique……10

Algorithme de colonie de Fourmis……14

Comparaison des résultats……18

Bibliographie ……20

Introduction au problème du voyageur du commerce :

Le problème du voyageur de commerce est un problème classique d'optimisation. L'objectif, pour une liste données de villes, est de trouver le chemin le plus court qui relie les villes entre elles en passant qu'une seule fois par ville, c'est à dire construire un cycle hamiltonien. C'est un problème classique d'optimisation et faisant partie des problème de la catégorie "NP-hard", c'est à dire les problèmes pour lesquelles il n'y pas de solutions simple, c'est à dire en temps "polynomial" ni façon simple de vérifier la solution, il est également un des problèmes les plus étudiés en informatique théorique.

Ce problème est également transposable à toute une variété d'autres problématiques comme la logistique, le séquençage de génomes, la conception de circuits électroniques. Le problème du voyageur de commerce est capital dans le domaine de la logistique.

Ces problèmes doivent tout de même être résolus pour des applications de la vie réelle.

Les méthodes heuristiques/métaheuristiques:

La méthode **heuristique** offre une solution rapide, basée sur l'intuition et l'expérience afin de guider la recherche de solution. Elle est une méthode spécifique à un problème donné, et ne garantit pas de trouver la solution optimale. Dans le cas du TSP, elle pourrait s'apparenter à la recherche de plus proches voisins.

La méthode **métaheuristique** quant à elle, est une méthode d'optimisation de haut niveau qui s'appuie sur des algorithmes efficients en s'appuyant sur une heuristique sous-jacente afin de trouver la solution optimale. C'est une méthode générale donc non spécifique qui ne garantit pas forcément de trouver la solution optimale mais qui s'en rapproche grandement grâce à ses algorithmes combinatoires.

La notion de "NP-hard":

Comme le problème du voyageur de commerce fait partie de la catégorie des problèmes "NP-Hard", des méthodes métaheuristiques sont employées pour les résoudre de façon satisfaisante dans des temps limités.
En effet le terme "NP" désigne la classe de problèmes avec laquelle on peut trouver très rapidement (temps polynomial) une solution.

Ainsi, le terme “NP-hard” désigne des classes de problèmes aussi voire plus difficiles que des problèmes “NP” . La seule différence est que nous ne sommes pas obligés dans ce cas là de vérifier une solution pour que le problème soit “NP-hard”

Ainsi, le TSP peut être qualifié de “NP-hard” et “NP” car nous pouvons vérifier et résoudre rapidement si le chemin obtenu répond aux contraintes du TSP.

Dans notre cas, nous nous limitons aux 100 villes les plus peuplées du monde.

La façon la plus simple de résoudre ce problème serait tout simplement d’explorer naïvement toutes les permutations entre les villes et de calculer la distance du parcours à chaque essai, le problème étant que le nombre de permutation augmenterait de façon factorielle : $o(n!)$, ce qui est encore pire qu’une croissance exponentielle.

Plutôt que de mettre en place cette solution qui serait extrêmement inefficace en termes de puissance de calcul, on peut alors chercher des moyens “d’approximer” une solution suffisamment et calculable à des échelles de temps humaines. C’est alors que les algorithmes d’optimisation interviennent.

Application du TSP à la chimie :

Le problème du voyageur de commerce peut s’appliquer aisément dans la chimie, que ce soit en chimie organique, en chimie analytique comme présenté ci dessous:

En effet, le TSP est utilisé en chimie organique afin de déterminer des **conformations moléculaires** qui seraient les “villes” et l’énergie nécessaire pour changer de conformation serait le “coût”. Ainsi c’est un outil intéressant afin de déterminer le chemin le moins coûteux pour obtenir toutes les conformations possibles. De plus, il est utilisé dans la **synthèse de molécules** afin d’estimer le coût et de choisir la voie la plus rentable ou bien le chemin le moins coûteux lors d’une synthèse multi-étape.

Concernant le domaine de la chimie analytique, il est utilisé pour la **spectrométrie de masse,** afin de calculer la probabilité de fragmentation.

Optimisation par Recuit simulé :

Introduction :

Le problème de Métropolis a été introduit dans les années 50 afin de simuler l'évolution du processus de recuit physique. Ainsi, de cette idée s'est imposée d'elle-même la curiosité d'utiliser Métropolis afin de résoudre des problèmes d'optimisation combinatoire dans les années 80. [1,2]

Le recuit simulé est la première métaheuristique a avoir vu le jour grâce à Kirkpatrick S. et ses collègues, spécialistes de physiques statistiques. Leur domaine d'étude concernait les configurations de basse énergie dans des matériaux magnétiques désordonnés. Ils ont été confrontés au problème de détermination numérique de ces configurations présentant une grande variété de profondeurs inégales.

Ainsi, travaillant dans les matériaux, ils se sont inspirés du processus de recuit en métallurgie afin d'appliquer ça sur leur problème et ont tenté de perfectionner cet algorithme afin qu'il soit répétable et applicable à tout domaine d'applications.

Principe :

Le concept principal derrière le recuit simulé s'apparente comme dit précédemment au processus de recuit en métallurgie. En effet, nous introduisons dans cet algorithme la notion de température fictive à laquelle on va appliquer une décroissance progressive. Cet algorithme est d'autant plus intéressant par la fait qu'il est probabiliste. [3,4]

1. En effet, la première étape est de chauffer à une température élevée une combinaison initiale qui représente notre solution initiale.

2. Cette combinaison va subir des modifications sous l'effet de la température à l'instant T grâce à une différence d'énergie ΔE. Le but est d'autoriser des mouvements/permutations au sein de la combinaison qui pourraient nous donner une solution temporairement moins optimale.

3. Ensuite, grâce à la règle de Métropolis, on peut tout simplement accepter si la différence d'énergie est négative ou acceptée sous la condition d'une probabilité si elle est négative. Ainsi, en explorant l'espace des solutions et en acceptant temporairement des solutions non idéales, on espère limiter les minima locaux.

4. Tant que le système n'est pas thermodynamiquement stable et figé, on abaisse la température et donc on effectue un recuit qui va donc rendre l'espace des solutions de plus en plus sélectif jusqu'à se rapprocher de la solution optimale.

D'un point de vue physique et chimique, les molécules suivent un mouvement brownien dû à l'agitation thermique. Comme dans la chimie en général, plus la température est élevée et plus les molécules sont en mouvements, ce qui reprend la même idée que l'utilisation que l'on fait de la température dans cet algorithme et plus elle est basse comme dans les polymères et plus celui-ci est vitreux donc les molécules sont immobiles.

Ainsi, les paramètres d'entrée sont :

- la température initiale
- un facteur de refroidissement
- la configuration initiale

Et les critères sont :

- le nombre de configurations visitées dans un palier de température
- la loi de décroissance de température
- le critère d'arrêt du programme

La température de départ est fonction du nombre d'éléments dans la combinaison. En effet, plus on a d'éléments et plus la température initiale doit être importante afin qu'il y ait assez d'itérations afin d'arriver à la solution optimale.

Concernant, le facteur de refroidissement, nous avons décidé de le définir le plus proche possible de 1 afin que le refroidissement soit le plus lent possible mais cela dépend de ce que l'on souhaite en terme d'efficacité et rapidité de l'algorithme.

Les critères d'arrêts sont atteints en général lorsque la température atteint une valeur presque nulle ou bien que plus aucun mouvement améliorant la fonction metropolis n'ait été accepté.

Schéma de fonctionnement :

L'algorithme du recuit simulé peut être simplement expliqué à l'aide d'un organigramme qui clarifie les différentes conditions que doit questionner celui-ci à chaque étape. En effet, étant donné qu'il s'agit d'une succession de la même opération en suivant un

gradient descendant de température, il faut bien comprendre quand l'algorithme est censé s'arrêter, ayant trouvé une solution idéale satisfaisante: [2]

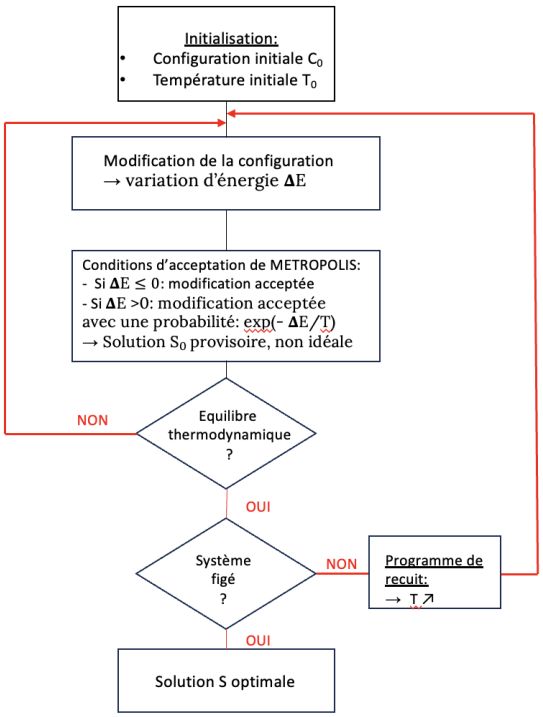


Figure 1 : Schéma de fonctionnement du recuit simulé

Ainsi, les deux conditions les plus importantes à vérifier sont l'équilibre thermodynamique afin que le système soit stable et ensuite qu'il soit figé, pour savoir s' il reste encore des possibilités menant à une solution plus idéale.

Évidemment, c'est un schéma général pour lequel l'algorithme est fonctionnel. Cependant, pour qu'il soit efficace, il est important de choisir les bons paramètres initiaux et d'effectuer des tests car c'est par la méthode empirique et l'amélioration continue que l'on obtient un résultat qui par notre sens critique nous paraît satisfaisant.

Formalisme mathématique :

Cet algorithme probabiliste repose principalement sur la mesure de Gibbs énoncé comme suivant qui permet de définir le noyau de probabilités de transitions suivant: [5]

$$\mu_\beta(x) = \frac{1}{Z_\beta} e^{-\beta L(x)}, \quad \text{avec } Z_\beta = \sum_{x \in E} e^{-\beta L(x)}$$

Figure 2 : Mesure de Gibbs

On remarque que plus L(x) est petit et plus le noyau de probabilité de transition est grand. Ainsi, cette simulation nous permet d'obtenir une probabilité élevée d'obtenir le plus petit chemin.

Nous pouvons également remarquer que cette formule est très similaire à la fonction de partition en mécanique quantique qui permet de donner la probabilité qu'à une particule d'être dans un état quantique donné. Ainsi, dans notre cas, on obtient la probabilité que la température soit refusée par l'algorithme de métropolis et donc nous indique indirectement la probabilité qu'à une configuration d'obtenir la longueur donnée.

Le facteur β est égal à l'inverse de la température, ce qui explicite bien que cette probabilité est dépendante de la température selon la relation suivante:

$$\forall \sigma \in E, \quad \nu_\sigma^T = \frac{1}{Z(T)} \exp\left(-\frac{1}{T} f(\sigma)\right)$$

Figure 3 : Relation entre la température et la mesure de Gibbs

La fonction $f(\sigma)$ correspond à la différence d'énergie générée par la modification de la configuration par la règle de Métropolis et est dépendante de la longueur du chemin.

Concernant l'évolution de la probabilité en fonction de probabilité, on peut se trouver dans 3 cas différents:

1. La température est élevée → $-\frac{1}{T} f(\sigma)$ tend vers 0 → $e^{(-\frac{1}{T} f(\boldsymbol{\sigma}))}$ tend vers 1 → probabilité maximale → la plupart des mouvements sont acceptés→ système non figé et non équilibré → solution non idéale
2. La température est élevée → $-\frac{1}{T} f(\sigma)$ tend vers 0 → $e^{(-\frac{1}{T} f(\boldsymbol{\sigma}))}$ tend vers 1 → probabilité maximale → la plupart des mouvements sont acceptés→ système non figé et non équilibré → solution non idéale
3. La température basse → $-\frac{1}{T} f(\sigma)$ tend vers -∞ → $e^{(-\frac{1}{T} f(\boldsymbol{\sigma}))}$ tend vers 0 → probabilité minimale → la plupart des mouvements sont refusés→ système figé et équilibré → solution idéale

Un algorithme adapté à de nombreux concept scientifiques :

Dans notre problème du voyageur du commerce, il représente un algorithme intéressant car il est capable de s'adapter à n'importe quel volume de données, à condition que l'on adapte le paramètre de température en conséquence.

Comme dit en introduction, de nos jours, cet algorithme est appliqué à d'autres problématiques scientifiques plus concrètes.

A titre d'exemple, le traitement d'image de tomographie et d'images satellites prend grandement appui sur le recuit simulé qui est prometteur dans ce domaine. En effet, on souhaite restaurer une image à partir de fragments d'images sans contours qui représentent la configuration initiale.
A travers la règle de Metropolis, on peut modifier les contours et les variables pixels au fur et à mesure des itérations successives. Ainsi, le processus est très similaire au recuit simulé car il fonctionne selon des variations d'énergie dues à la décroissance de la température. Cette décroissance s'arrête lorsque l'image restaurée est satisfaisante.

Les avantages et inconvénients :

Cet algorithme présente de nombreux avantages mais également de nombreux désavantages qui sont des critères à prendre en compte lors du choix d'un algorithme en particulier, optimal pour l'application désirée.

AVANTAGES:

1. Multiplicité des domaines d'applications: applicable à une grande variété de problèmes d'optimisation.
2. Permet d'éviter les minima locaux : La probabilité d'accepter des solutions moins bonnes au début permet d'éviter de se retrouver piégé dans des minima locaux.
3. Flexibilité : Plusieurs paramètres peuvent être ajustés pour s'adapter à des problèmes spécifiques.
4. Simplicité : Relativement simple à comprendre et à mettre en œuvre par rapport à d'autres techniques d'optimisation globale.
5. Des résultats qualitatifs Avec les bons paramètres, peut produire des solutions de haute qualité.

INCONVÉNIENTS:

1. Sensibilité aux paramètres : Les performances dépendent fortement du choix des paramètres, comme la température initiale et le programme de refroidissement.
2. Pas de garantie d'optimalité : L'algorithme ne garantit pas de trouver la solution optimale, seulement une solution de bonne qualité.
3. Complexité temporelle : Pour certains problèmes, l'algorithme peut nécessiter beaucoup de temps pour converger.
4. Manque d'une méthode standard : Il n'y a pas de méthode unique pour définir la température initiale, le programme de refroidissement, etc. Ceci peut rendre l'application de l'algorithme à de nouveaux problèmes un peu délicate.
5. Comparaison difficile : En raison de sa nature stochastique, les exécutions multiples peuvent produire des résultats différents, ce qui peut rendre difficile la comparaison avec d'autres méthodes.

Optimisation par algorithme génétique :

Introduction :

Une approche très populaire afin de s'atteler à ce genre de problèmes d'optimisation est l'algorithme génétique, qui fonctionne de la même manière que l'évolution des espèces réelles sur terre. Les algorithmes génétiques sont assez anciens, puisque leur première mention date de 1962 par John Holland [1], qui explique que des systèmes peuvent générer eux-mêmes les procédures qui permettent de s'adapter à leur environnement. Il introduit alors la principale problématique qui gouverne la conception d'algorithmes génétiques: ce n'est plus un problème pour le développeur de créer un algorithme en particulier, mais plutôt de créer un environnement propice à l'évolution du système. En pratique, la mise en place d'un algorithme génétique demande au développeur de bien calibrer de nombreux paramètres du programme qui vont permettre une bonne recherche de l'espace pour trouver la meilleure solution.

Principe :

L'algorithme génétique commence par la génération initiale d'un nombre N d'individus totalement aléatoires, qui forme notre génération de départ. Par la suite, l'algorithme va répéter le même processus d'évolution et de sélection plusieurs fois [2,3]:

1 - Calcul de la fitness, le score de chaque individu
2 - Elimination des chemins les moins optimaux
3 - Recroisement des survivants afin de se ramener à une génération de N individus
4 - Mutation des nouveaux individus

Lors de la première étape, on va additionner les distances entre les différentes villes ensemble en prenant en compte le modèle de la terre sphérique à l'aide de leur coordonnées GPS. Calculer la fitness va permettre de sélectionner les chemins les plus efficaces de manière majoritaire tout en permettant quand même à de la diversité d'exister en laissant quelques individus moins bons passer à la génération suivante, cela permet également de se rapprocher plus de la sélection naturelle réelle.

Lors de l'étape d'élimination, on sélectionne aléatoirement un certain nombre de survivants selon une probabilité calculée à partir de la fitness, les individus avec la fitness la plus élevée ont une plus grande chance de s'en sortir, bien que ça ne soit pas garanti.

Lors du recroisement, on sélectionne des parents 2 à 2 et on crée à partir d'eux 2 nouveaux enfants selon la méthode dite du "Order-Based Crossover", aussi connue sous l'identifiant OX1. Lors de cette étape, on va sélectionner une région aléatoire des gènes de chaque parent (au même endroit dans leur séquence génétique, et de même longueur). Chacune de ces séquences est passée à un enfant différent, et le génome de chaque enfant est complété par les gènes de l'autre parent [4].

La mutation est réalisée avec une probabilité assez élevée afin d'explorer plus de solutions possibles et d'explorer l'espace. Il existe de très nombreuses manières de causer des mutations, mais l'une des meilleures est la méthode dite de Reverse Sequence Mutation (RSM), et c'est celle que l'on a utilisée ici [5].

Schéma fonctionnel :

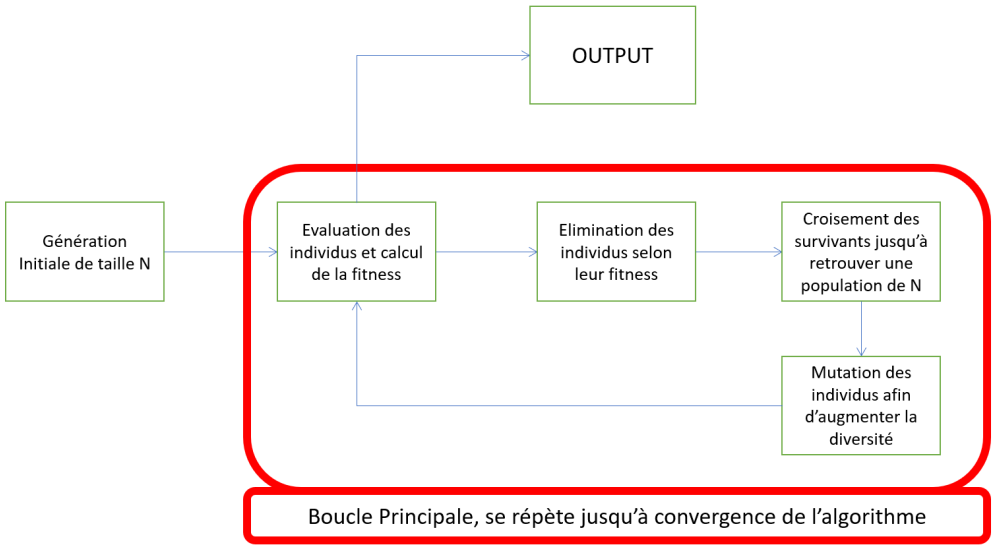


Figure 4 : Schéma fonctionnel de l'algorithme génétique

Formalisme mathématique :

Afin de pouvoir différencier les chemins les uns des autres, il faut calculer leur score, leur fitness. Pour cela, on peut tout simplement utiliser cette formule, avec L(Ri) la longueur du chemin Ri [3]:

$$f_r(R_i) = \frac{1}{L(R_i)}$$

Figure 5 : calcul de la fitness

On peut aussi tout simplement utiliser la longueur des chemins et ne sélectionner que les plus courts, mais avoir une fitness nous permet de réaliser plusieurs opérations dessus afin de pouvoir sélectionner plus précisément les survivants.
Par exemple, on peut utiliser la fitness afin de réaliser un scaling. Ce scaling, caractérisé par le facteur k, nous permet de contrôler précisément le fonctionnement de l'algorithme de sélection avec par exemple:

$$f_s = (f_r)^{k(n)}$$

Figure 6 : Calcul de la fitness scalée fs

avec n la génération courante et k(n) :

$$k = \left(\tan\left[\left(\frac{n}{N+1}\right)\frac{\pi}{2}\right]\right)^p$$

Figure 7 : Calcul de k

Avec dans cette formule p un paramètre à choisir. La littérature recommande une valeur d'environ 0.1, mais il est beaucoup plus judicieux d'utiliser une valeur élevée ici afin de pouvoir permettre à l'algorithme de différencier plus facilement de faibles écarts de fitness et de converger rapidement.
Finalement, la probabilité de sélection de l'individu Ri est donnée par:

$$p(R_i) = \frac{f_s(R_i)}{\sum_{j=1}^{N} f_s(R_j)}$$

Figure 8 : Calcul de la probabilité de sélection de l'individu Ri

Pour ce qui est de l'algorithme de mutation on a utilisé la méthode RSM, qui choisit aléatoirement une séquence dans la le chemin et la renverse. Cette méthode est l'un de

celles qui fournit les meilleurs résultats en terme de mutation, et également une de celles qui converge le plus rapidement [5]:

Figure 9 : Fonctionnement de la mutation RSM

Pour ce qui est de la méthode de croisement, on utilise la méthode dite du “order-based crossover” repéré par l’identifiant OX1 afin de croiser les parents 2 à 2. La méthode utilisée pour réaliser ce croisement est la suivante: [4]

1 - Choix aléatoire de 2 parents P0 et P1, caractérisés par leur séquence
2 - Création de 2 enfants Pc et Pd vides
3 - Chaque enfant reçoit une partie des gènes de son parent associé
4 - La séquence génétique de l’enfant est complétée avec les gènes manquants dans l’ordre ou ces gènes apparaissent dans la séquence génétique de l’autre parent

Avantages et inconvénients :

Les avantages:
- Le fonctionnement de l'algorithme est facilement contrôlable, la vitesse de convergence peut par exemple avec une simple modification du paramètre k. Elle n’a pas été réalisée ici, mais cela rend simple la création par exemple d’une grid search aisée.

Inconvénients:
- Si l’algorithme converge trop vite, alors ce dernier risque d'explorer l’espace et de converger trop vite vers un minimum local très rapidement. Si l’on réduit sa convergence, alors il risque de tourner très lentement.

- Peut rapidement devenir gourmand en calcul si la convergence est lente.

Optimisation par algorithme de colonie de fourmis:

Principe de l'algorithme :

L'algorithme d'optimisation par colonies de fourmis a été mis au point en 1992 par Marco Dorigo et s'inspire du comportement des fourmis qui sont capables de trouver le chemin le plus court vers une source de nourriture sans pouvoir se repérer visuellement. Les fourmis sont capables de communiquer entre elles en laissant derrière elles des phéromones, une hormone que les autres fourmis sont capables de détecter. Utiliser des phéromones pour communiquer est un moyen de communication très répandu dans la nature et est utilisé par des animaux, des insectes et même des plantes. Les fourmis prise une à une ne sont pas intelligentes, elles répondent seulement à un ensemble limité de règles simples : suivre en priorité un parcours sur lequel une fourmi est déjà passée et laissé sur son chemin de phéromones si une source de nourriture est détectée. En revanche une colonie de plusieurs fourmis exhibe en suivant ces règles une intelligence collective assez remarquable.

Le principe de l'algorithme d'optimisation par colonie de fourmis est de simuler ce comportement naturel en créant des fourmis artificielles qui déposent elles-mêmes des phéromones détectables par les autres fourmis. De la même façon qu'une colonie de fourmis utilisera sa population pour trouver une source de nourriture, ici les fourmis cherchent la solution pour minimiser la fonction coût du problème à résoudre, dans notre cas, le problème du voyageur de commerce, il s'agit de minimiser la distance du parcours.

Mise en place de l'algorithme :

Dans tout le programme on utilise le package numpy pour accélérer grandement les calculs, notamment en remplaçant les listes par des arrays partout où cela est pertinent (la seule opération pour laquelle un array numpy n'est pas plus efficace qu'une liste est la concaténation).

On importe d'abord la liste des villes à utiliser (les 100 villes les plus peuplées du monde, qui est triée dans l'ordre décroissant). A chaque ville est associé un indice pour pouvoir la repérer, par exemple Shanghaï qui est la ville la plus peuplée donc la première sur la liste à l'indice 0 et ainsi de suite . Un parcours réalisé par une fourmi est donc une liste (ou plutôt un array numpy) de taille 100 et qui comprend toutes les villes dans l'ordre où elles sont parcourues. On calcule également en avance une matrice des distances de taille 100*100 qui répertorie toutes les distances entre les villes (et est donc symétrique). La distance entre une ville i et une autre ville j correspond alors à l'élément de la i-ème ligne et j-ème colonne de la matrice. En utilisant cette matrice il sera possible de calculer la distance totale d'un parcours.

Une autre matrice à initialiser, de la même taille et d'une très grande importance, est la matrice des phéromones. Cette matrice associe à chaque transition d'une ville *i* à une ville *j* une quantité de phéromones. Les phéromones sont déposées par les fourmis et influencent le choix de leur parcours. A chaque itération du programme une partie des phéromones s'évaporent.

Cette matrice est également symétrique mais ici les éléments (i, j) et (j, i) ne seront pas forcément les mêmes puisque le sens du trajet a une importance. Ici τ_{ij} représente le taux de phéromone sur le trajet allant de la ville i à la ville j.

$$\mathrm{Ph} = \begin{pmatrix} 0 & \tau_{1,2} & \cdots & \tau_{1,n-1} & \tau_{1,n} \\ \tau_{2,1} & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \tau_{n-1,1} & & & \ddots & \tau_{n-1,n} \\ \tau_{n,1} & \cdots & \cdots & \tau_{n,n-1} & 0 \end{pmatrix}$$

Figure 10 : Matrice des Phéromones

Pour initialiser le programme, chaque fourmi emprunte un parcours complètement aléatoire, hormis la première ville qui sera toujours Shanghaï. Ces parcours sont classés et les meilleurs (les 30 meilleurs dans le programme) voient les transitions qu'ils ont empruntés récompensés par un ajout de phéromone (la valeur ajoutée est constante même si en théorie elle devrait être inversement proportionnelle à la distance).

Ensuite pour chaque itération, les fourmis vont une à une faire leur parcours. A chaque étape, elles doivent choisir vers quelle ville s'orienter et donc quelle transition prendre. Cela est déterminé par la distance qui la sépare de chaque ville et la quantité de phéromones sur chaque transition. La probabilité de faire la transition d'une ville *i* à une ville *j* est donnée par la formule ci-dessous :

$$\begin{cases} P_{ij} = \dfrac{(\tau_{i,j})^{\alpha} \cdot (d_{i,j}^{-1})^{\beta}}{\sum\limits_{j} (\tau_{i,j})^{\alpha} \cdot (d_{i,j}^{-1})^{\beta}} & \text{Si j} \in \text{Villes non parcourues} \\ 0 & \text{Sinon} \end{cases}$$

Figure 11 : Probabilité de passer d'une ville à une autre

Ici d^{-1} correspond à l'inverse de la distance entre deux villes. La somme des probabilités est bien égale à 1 à chaque fois que la fourmis doit faire un choix. Une fonction random.choice avec chaque transition ville pondérée par la probabilité associée calculée comme figure ci-dessus. Les transitions avec le plus de phéromones sont favorisées. Les paramètres α et β permettent de régler l'algorithme, en pratique ils ont été réglés à 1.

Lorsqu'il ne reste plus de ville, la fourmi a fini son parcours, qui est gardé en mémoire. Toutes les autres fourmis font de même. A la fin de cette itération les parcours sont comparés et sur cette base la matrice phéromone est alors mise à jour, de la même façon que dans l'initialisation. En revanche cette fois-ci la matrice des phéromones subit également un traitement d'évaporation pour atténuer les mauvais parcours. Ici ρ est un paramètre que l'on fixe et n le numéro de l'itération.

$$(\tau_{i,j})_{n+1} = (1-\rho) \cdot (\tau_{i,j})_n$$

Figure 12: Formule d'évaporation de la matrice des Phéromone

Ces étapes sont répétées autant de fois qu'il y a d'itérations. Au fur et à mesure que les meilleures transitions sont récompensées, les fourmis deviennent de plus en plus susceptibles de les emprunter. L'algorithme finit par converger vers le meilleur chemin qu'il a trouvé, ce qu'on peut observer en visualisant la matrice des Phéromones :

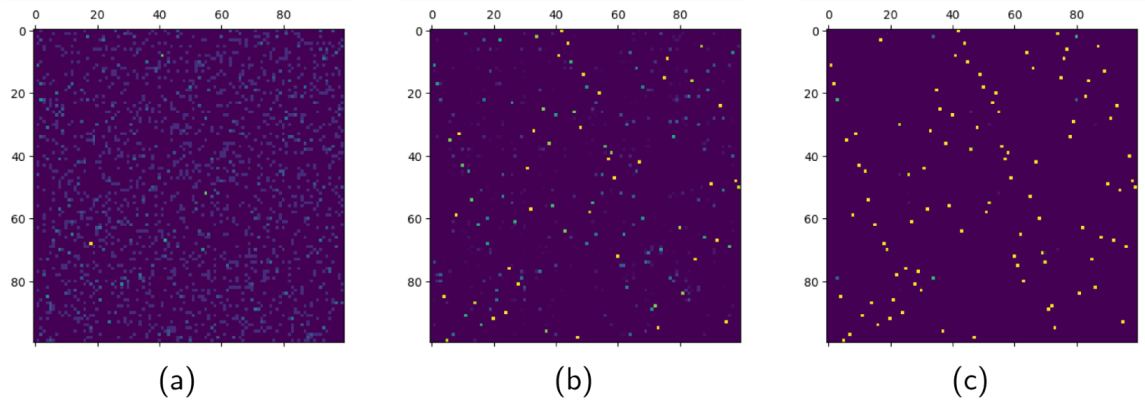


Figure 13 : Evolution de la matrice Phéromone au cours de l'algorithme : (a)-Début, (b)-Milieu, (c)-Fin.

A la fin de l'algorithme le meilleur (le dernier en fait) parcours est affiché ainsi que la distance qui lui est associée. On peut résumer le programme avec ce schéma de fonctionnement qui répertorie les grandes étapes :

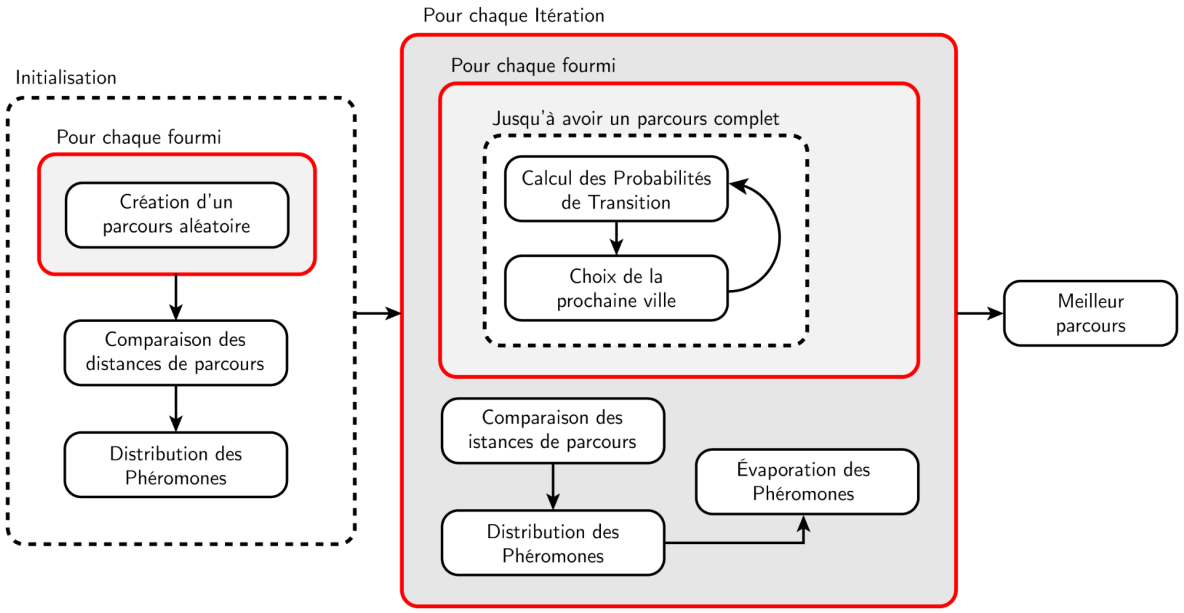


Figure 14 : Schéma de Fonctionnement de l'algorithme de Colonie de Fourmis

Avantages et inconvénients de l'algorithme :

L'algorithme de colonie de fourmi présente quelques avantages : Son fonctionnement de calcul fait qu'il est hautement parallélisable, c'est à dire qu'il peut tirer profit des architectures de cartes graphiques par exemple et peut donc être poussé plus loin que d'autres algorithmes. Son fonctionnement, basé sur un cycle vertueux où les fourmis vont rapidement trouver de bonnes solutions, signifie qu'il converge assez rapidement, ce qui est visible sur les graphiques. Il a également été montré que cet algorithme est plutôt bon pour résoudre le problème du voyageur de commerce [5] et peut s'adapter à une situation changeante. L'algorithme peut en revanche converger trop tôt et souffrir d'un manque d'exploration des solutions en plus d'être peu adapté à des problèmes non-discrets.

Comparaison des algorithmes et des résultats :

Grâce à cet exercice, nous avons pu étudier 3 algorithmes métaheuristique pour lesquels nous avons obtenu des résultats différents mais qui mènent bien à un solution pour laquelle le système est thermodynamiquement équilibré et figé:

Algorithme génétique :

- Meilleur parcours : 127680 km
- 300 itérations
- Temps de fonctionnement: 83 secondes

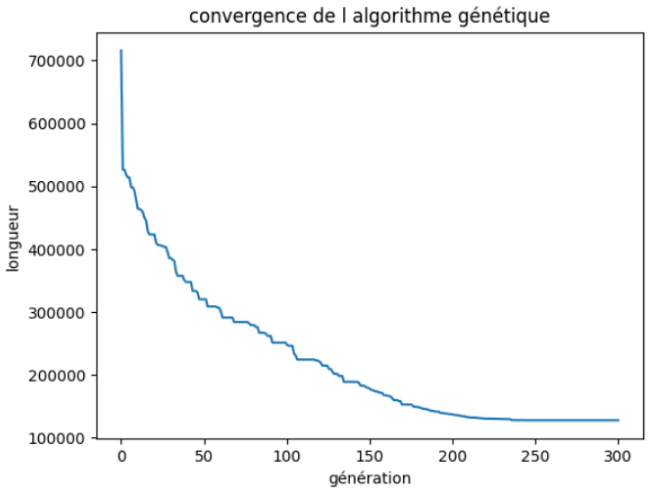


Figure 15 : Distance du meilleur parcours en fonction du nombre de générations

Colonie de Fourmis :

- Meilleur parcours : 90943 km
- 100 itérations
- Temps de fonctionnement: 108 secondes

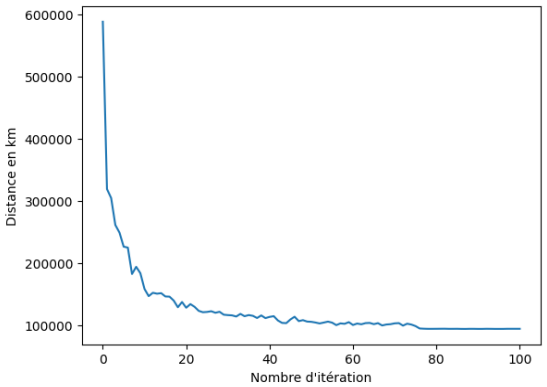


Figure 16 : Distance du meilleur parcours en fonction du nombre d'itérations

Recuit simulé :

- Meilleur parcours :198045 km
- 6300 itérations
- Temps de fonctionnement: 9 secondes

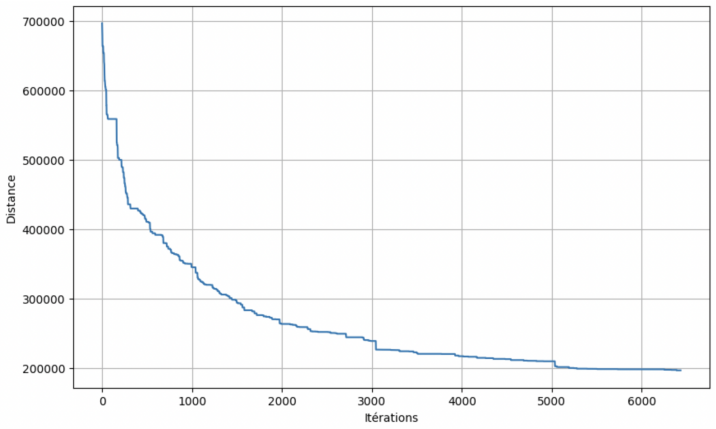


Figure 17 : Distance du meilleur parcours en fonction du nombre d'itérations

Interprétations:

Comme dit précédemment, nous avons réussi à obtenir de ces trois méthodes un résultat qui n'est pas aberrant.

En termes de distance, nous remarquons que la méthode de colonie de fourmis nous permet d'obtenir la plus petite distance contrairement au recuit simulé et à l'algorithme génétique qui donnent des distances bien supérieures mais qui sont toutes les deux dans le même ordre de grandeur.

Concernant le nombre d'itérations, la colonie de Fourmis implique le moins d'itérations, suivie de près par l'algorithme génétique en convergeant relativement rapidement contrairement au recuit simulé impliquant le plus d'itérations et convergeant lentement.

Enfin, en termes de performances, le recuit simulé est le plus rapide et la colonie de fourmis et l'algorithme génétique se valent en termes de temps. Il faut quand même considérer que ce sont des temps très courts pour des algorithmes complexes comme ceci.

Finalement, prenant en compte tous ces facteurs, la colonie de Fourmis semble être le meilleur algorithme pour ce problème étant rapide, précis et peu coûteux en termes d'exécution. Il semble plus fiable car il passe plus de temps par itérations. De plus, la distance en km semble cohérente si on compare avec des logiciels de cartographie permettant de calculer la distance.
Le recuit simulé est la parfaite démonstration du fait que rapidité ne garantit pas la qualité car malgré le fait qu'il soit le plus rapide, la distance semble trop importante, d'autant plus au nombre colossal d'itérations. Ce décalage peut être dû à un problème d'ordre dans les permutations.
L'algorithme génétique peut présenter le même problème que le recuit simulé mais est tout de même plus adapté au problème du TSP que le recuit simulé car moins coûteux et plus précis tout de même.

Bibliographie :

Algorithme génétique :

[1] John H. Holland. (1962,). Outline for a Logical Theory of Adaptive Systems. J. ACM 9, 3

[2] Fu, C., Zhang, L., Wang, X., & Qiao, L. (2018, May). Solving TSP problem with improved genetic algorithm. In *AIP Conference Proceedings* (Vol. 1967, No. 1). AIP Publishing.

[3] Algorithmes génétiques, JM Alliot, N Durand, visité le 21/10/2022, http://pom.tls.cena.fr/GA/FAG/ag.html

[4] Deep, K., & Mebrahtu, H. (2011). New variations of order crossover for travelling salesman problem. *International Journal of Combinatorial Optimization Problems and Informatics*, *2*(1), 2-13.

[5] Bye, R. T., Gribbestad, M., Chandra, R., & Osen, O. L. (2021). A comparison of ga crossover and mutation methods for the traveling salesman problem. In *Innovations in Computational Intelligence and Computer Vision: Proceedings of ICICV 2020* (pp. 529-542). Springer Singapore.

Algorithme du recuit simulé :

[1] Celeux G. , Diebolt J. (1989), *Une version de type recuit simulé de l'algorithme* EM. (RR-1123, programme 5), INRIA. , France

[2] Siarry P. (2003), *Métaheuristiques*, édition Eyrolles, France

[3] Hafez N. (1999), *Conditions d'équilibre et gestion d'unités de transport en libre service avec demandes aléatoires*, Université de Metz, Faculté des sciences, UFR de mathématiques , informatique et mécanique, France

[4] Constantini C., Poissat J. (2017), *Le problème du voyageur de commerce*, mémoire d'initiation à la recherche, , PSL research university, Dauphine Université Paris, France

[5] Agrégation externe de mathématiques (2008), *Recuit simulé et le voyageur de commerce*, épreuve de modélisation, université de Rennes 1, France

Algorithme de Colonie de Fourmis :

[1] Jinhui Yang; Xiaohu Shi; Maurizio Marchese; Yanchun Liang (2008). An ant colony optimization method for generalized TSP problem. , 18(11), 1417–1422.

[2] Oliveira, Sabrina & Hussin, Mohamed Saifullah & Stützle, Thomas & Roli, Andrea & Dorigo, Marco. (2011). A detailed analysis of the population-based ant colony optimization algorithm for the TSP and the QAP. Genetic and Evolutionary Computation Conference, GECCO'11 - Companion Publication.

[3] Yong Wang;Zunpu Han; (2021). Ant colony optimization for traveling salesman problem based on parameters optimization . Applied Soft Computing

[4] Dorigo M, Gambardella LM. Ant colonies for the travelling salesman problem. Biosystems. 1997;43(2):73-81.

[5] Selvi, V., & Umarani, R. (2010). Comparative analysis of ant colony and particle swarm optimization techniques. *International Journal of Computer Applications*, 5(4), 1-6.