Projet Intelligence Artificielle

Partie I : Définition du problème

Le but du projet est la recherche d'un circuit hamiltonien de plus petit poids dans un graphe complet. Pour cela, nous allons utiliser deux types d'algorithmes, la recherche informée et la recherche locale.

A- Recherche informée

Il s'agit d'un parcours dans un espace d'état guidé par une information relative à l'estimation du cout. A chaque état, correspond un ensemble de villes visités et la ville courante. L'état initial est un ensemble vide avec une ville de départ fixée. L'état final (goal test) est l'état où toutes les villes ont été visité et où la ville courante est la ville de départ. Une action est le fait de passer d'un état initial à un autre, c'est-à-dire de se déplacer dans une ville non visitée par cet état initial.

Remarque : On peut se déplacer dans la ville de départ que lorsqu'il reste un seul sommet à visiter.

L'ordre de visite se fait selon un cout égal à la somme du cout du chemin partiel avec une heuristique. Comme précisé dans l'énoncé, l'heuristique se calcule en trouvant le poids d'un arbre couvrant minimal qui lui dépend des villes déjà visités. L'heuristique est admissible car le poids de l'arbre couvrant d'un graphe réduit par les nœuds déjà visités est nécessairement plus grand ou égal au poids du plus court chemin vers le nœud de départ. En effet, ce chemin est un arbre avec une disposition particulière. L'arbre couvrant minimal est plus général que le chemin minimal (il est inclus dans les arbres) celui-ci peut donc être soit plus grand si l'arbre couvrant minimal n'est pas le chemin minimal ou exactement égal si cet arbre est le chemin minimal.

Exemple. Ensemble des villes E = {1, 2, 3, 4, …, N}, Ville de départ = x0.

L'état initial est un ensemble vide avec comme ville courante x0.
Etat initial= [{}, x0]

L'état final est un ensemble qui contient toutes les villes du graphe et avec comme ville courant x0.
Etat final = [E, x0]

Actions : Soit un état courant qui a pour ensemble de villes visitée Ev, Si la taille de Ev est inférieure à N-1, les actions possibles sont les suivantes :
= {->xi | avec xi ∈ E\Ev et xi ≠ x0}

Sinon il reste une ville à visiter, l'action possible est d'aller dans la ville de départ.
= {->x0}

Une transition est le passage d'un état à un autre à partir d'une action.
Résultat([Ev, xi] , ->xn) = [Ev U xn, xn]

Principe : Au départ, on initialise l'état courant par l'état initial. Tant que l'état courant n'est pas égal à l'état final, on construit la frontière des états explorés qui est l'union des frontières de chacun d'eux. A partir de cette frontière, on calcule le minimum, c'est-à-dire l'état dont la somme du cout du chemin et de la valeur de l'heuristique est la plus petite. Puis l'état courant devient ce minimum et on continue jusqu'à qu'on se trouve dans un état final.

B- Recherche locale

Pour les deux algorithmes de recherche locale implémentés, nous définissons le problème de la même façon. A chaque état correspond un circuit hamiltonien. L'état final n'est pas clairement défini car on cherche un circuit de plus petit cout, on procède par itération jusqu'à que le circuit trouvé soit minimal, où jusqu'à que le cout ne s'améliore plus.

Une action possible est d'inverser l'ordre de parcours entre deux sommets du circuit. Cela est implémenté par l'algorithme 2-opt indiqué par l'énoncé. Le voisinage d'un état correspond donc aux états produits en inversant le circuit entre toutes paires possibles de position du circuit de l'état courant.

Exemple. Ensemble des villes E = {1, 2, 3, 4, …, N}, Ville de départ = x0.

L'état initial est un circuit hamiltonien de taille N.
Etat initial= (x1,x2,…,xi,xn) de coût ci.
remarque : on considère que le circuit est composé des arêtes de tout sommet consécutif de cette séquence auquel on ajoute l'arête (xn,x1)

L'état final est aussi un circuit hamiltonien, qui représente une « bonne » solution.
Etat final = (y1,y2,…,yi,yn) de coût cf tel que cf ≤ ci.

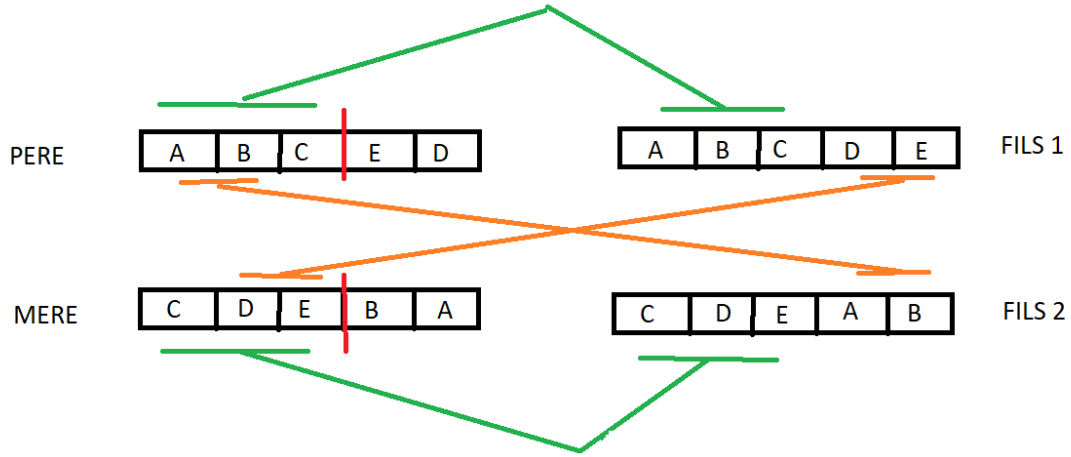
Actions : Une action est effectuée par un état. Elle est caractérisée par un couple (i,j) avec i,j ∈ E
Elle consiste à inverser le circuit entre i et j.

Une transition est le passage d'un état à un autre à partir d'une action.
Résultat((x1, xi, xi+1, … , xj-1, xj, xn) , (xi, xj)) = (x1,xj, xj-1, …, xi+1, xi, xn)

Algorithme local Beam : On commence avec un nombre k d'états aléatoires (circuits hamiltoniens aléatoires), puis on génère tous les successeurs avec l'algorithme 2-opt. Parmi ces successeurs, on choisit les k meilleurs, et on recommence le cycle.

Algorithme génétique : On commence aussi avec un nombre d'état k pair d'état aléatoire. Ces états représentent la population initiale. A partir de cette population, on forme des paires d'état qui sont au nombre de k/2. La probabilité des individus pour former les paires est inversement proportionnelle au coût. Plus un individu a un cout faible, plus il aura de chance d'être choisi. Puis pour chaque paire, on forme deux enfants qui sont des arrangements entre le père et la mère comme le montre le schéma suivant :

Un entier k aléatoire est généré pour déterminer où faire le découpage.
Ainsi si k = 3, les 3 premiers éléments de la séquence du fils 1 seront ceux du père, et le reste sera complété par l'ordre de la mère. Et inversement pour le fils 2, il prendra les 3 premiers éléments de la séquence de la mère et les éléments restants copieront l'ordre dans lequel ils sont situés chez le père.



Un bon circuit se caractérise par une bonne suite de sommets, l'idée de ce couplage est de pouvoir possiblement garder des bonnes parties de la séquence du père et de la mère qui ont de bons couts. Et donc avoir la possibilité de générer des fils « meilleurs » ou « pas trop pire » des parents.

Lorsque la population enfant est créé. Ils peuvent avec une probabilité faible subir une légère mutation qui n'est rien d'autre que la permutation entre deux sommets du circuit. Cela permet entre autres de garantir la diversification de la population pour améliorer les performances de l'algorithme.

Enfin pour former la génération suivante, nous remplaçons les enfants avec les circuits ayant les plus gros coûts par un pourcentage fixé des parents avec les plus petits coûts.
Puis on recommence l'algorithme de génération en génération.

Partie II : Implémentation

Le langage de programmation choisi est le Java. Nous utilisons des bibliothèques externes pour les structures de données de base (liste, ensemble) :
- *ArrayList*
- *LinkedList*
- *HashSet + Iterator*

Pour les algorithmes A* et Local Beam, nous nous basons sur les algorithmes du cours que nous adaptons à notre problème.
Pour l'algorithme génétique, il est fortement inspiré de celui fait en TP avec des modifications pour l'adapter à notre problème.

L'implémentation des graphes est entièrement faite manuellement, on les représente par des matrices d'adjacence comme ce qui se fait usuellement.

Partie III : Mesure de performance

Les algorithmes de recherche locale ont des paramètres.
- Local Beam : k = le nombre d'états traqués, n = le nombre d'itérations.
- Algorithme génétique : k = la taille de la population, m = le taux de mutation, e = le taux d'élitisme (% de la bonne population qu'on garde à la prochaine génération), n = le nombre d'itérations.

Pour la comparaison des algorithmes, nous fixons certains paramètres et nous faisons varier ce qu'on veut mesurer. Il y a plusieurs critères pour évaluer nos différents algorithmes : L'optimalité de la solution, et le temps d'exécution.

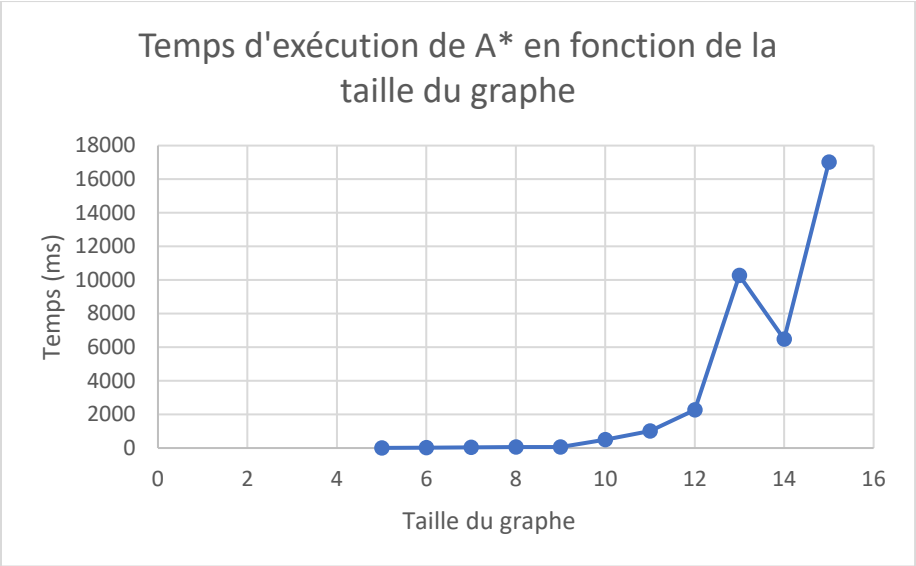
Nous utilisons des matrices aléatoires. Pour cela, on génère d'abord une liste de villes de position aléatoire dans une portée donnée, puis on crée la matrice en fonction de la distance des villes. Pour compenser l'aspect aléatoire, chaque mesure pour des paramètres fixés est une moyenne de 5 exécutions.

Les données des mesures se trouvent dans le fichier Excel, en annexe.

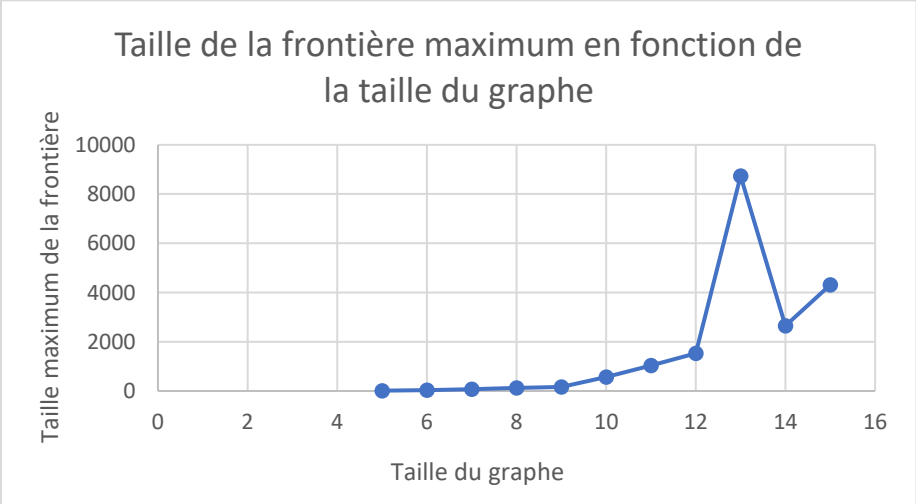
1. Algorithme A*

Optimalité : Pour n'importe quel graphe, l'écart à l'optimum est nul. L'algorithme donne toujours une solution optimale.

Cependant, on remarque que le temps d'exécution augmente exponentiellement en fonction de la taille du graphe :



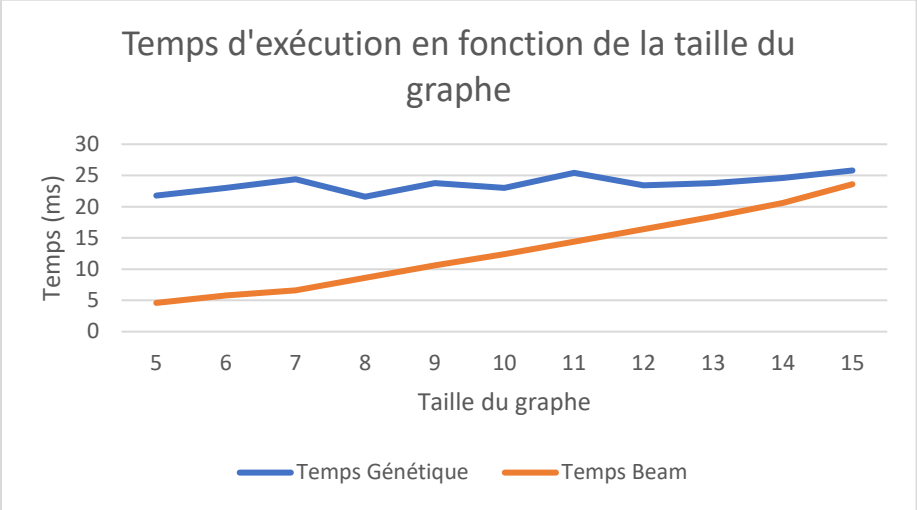
Cela est corrélé à la taille de la frontière qu'on explore :



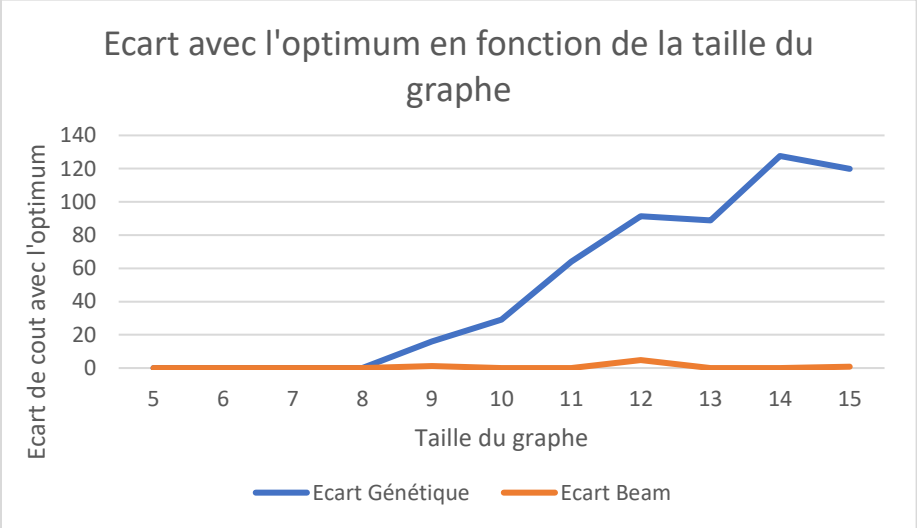
Le pic qu'on observe est dû au fait qu'il est possible dans les pires cas de parcourir tous les voisins. Mais on observe une tendance globale.

2. Algorithme Local Beam et Algorithme génétique

Pour Local Beam, on fixe comme paramètres : le nombre d'itérations n = 10 et le nombre d'états traqués k= 5. Pour l'algorithme génétique, on fixe le nombre d'itérations n = 10 , le taux de mutation m = 0.15, Le taux d'élitisme e = 0.20 et la taille de la population k = 50. Puis on mesure le temps d'exécution qui est considérablement amélioré par rapport à A*.



La contrepartie est l'optimalité. Pour les deux algorithmes, on mesure l'écart avec la solution optimale trouvée avec A* en fonction de la taille du graphe :



Partie IV : Analyse

Algorithme A* : C'est l'algorithme le plus lent, il ne parcourt pas tous les états possibles grâce à l'heuristique mais c'est celui qui passe par le plus d'états. Il donne cependant toujours la solution optimale. Aux alentours des graphes de taille 15, le temps dans le pire des cas pour trouver la solution peut être très grand et augmente exponentiellement, cependant ce n'est pas toujours le cas. Cela s'explique par le nombre d'états qu'on explore et donc par la taille de la frontière.

Algorithme Génétique : Augmenter la taille de la population permet de s'approcher de l'optimum. Cependant le temps d'exécution augmente aussi parallèlement car on fait plus de calcul pour recalculer les générations. La taille du graphe n'influe pas sur le temps d'exécution, car l'algorithme construit directement un nombre fixé (taille de la population) de chemins hamiltoniens possibles. Le nombre d'itération influe quant à lui aussi sur le temps de manière linéaire.

Dans notre cas, l'algorithme a du mal à trouver une solution optimale mais on peut l'approcher considérablement en le paramétrant bien. L'écart à l'optimum augmente avec la taille du graphe, il est donc nécessaire d'adapter nos paramètres à cette taille (par exemple une population plus grande si le graphe est grand). Il est probablement possible d'obtenir des meilleurs résultats en trouvant un meilleur moyen de coupler les parents pour pouvoir garder leur bon cotés dans les générations suivantes.

Algorithme Local Beam :
Le temps d'exécution augmente avec la taille du graphe . Le nombre d'état traqué n'a pas besoin d'être très grand pour obtenir de bons résultats. En le fixant à 5, l'écart à l'optimum est déjà quasiment nul pour les tailles de graphe testés. Pour trouver les voisins, on utilise l'algorithme opt qui permet d'arriver rapidement à la solution. Pour notre problème, Local Beam est l'algorithme qui offre le meilleur compromis entre optimalité et temps d'exécution.

Projet réalisé par : HUANG Roger, MEZIANE Abdesamad, FLAUBERT Marc.