

Étude comparative sur les heuristiques de résolution du voyageur de commerce

SOUS-TITRE DU RAPPORT

Evan Damas et Élia Gautier | SAÉ 2.01 | Mai 2022

# Heuristiques gloutonnes

## Plus proche voisin

Cet algorithme a pour principe de toujours ajouter à la tournée le sommet non visité le plus proche du dernier ajouté à la tournée. Pour l’initialisation on choisit arbitrairement l’usine comme point de départ, mais la tournée étant cyclique on peut théoriquement démarrer à n’importe quel sommet.

Cet algorithme est très simple algorithmiquement ce qui le rend très rapide sur la plupart des graphes tout en maintenant des résultats tout à fait satisfaisants. Il dévoile tout son potentiel lorsqu’il s’agit de traiter des graphes bipartis, ce qui s’explique par leur nature qui fait que l’algorithme alterne entre les deux parties du graphe assez facilement.

## christofides

Nous appellerons le graphe de départ G.

Pour commencer nous avons besoin de faire un arbre couvrant de notre graphe de poids minimum, pour ce faire j’ai décidé d’utiliser l’algorithme de Kruskal (consiste à prendre les arêtes par ordre croissant selon leur poids tant qu’elles ne crées pas de cycle, et on s’arrête lorsqu’on a qu’une composante connexe). On appellera cet arbre couvrant de poids minimum T. Kruskal()

Ensuite on récupère les sommets de degrés impair de T dans un ensemble I. ListeLieuDegreImpair()

On fait un graphe induit de G par les sommets de I. Pour cela, on prend les sommets de I et les arêtes de G, et chaque arête à laquelle il manque un sommet de départ ou un sommet d’arrivée est supprimée. On appellera ce nouveau graphe induit V. SupprimeRouteEnTrop()

Par la suite, on fait un couplage minimal du graphe V. Couplage()

On fusionne l’arbre couvrant T avec le couplage obtenu Union(), et on fait un tour hamiltonien de ce nouveau graphe, et on a enfin de notre résultat (en supprimant les doublons). Final() insertion au plus proche

Le principe de cette heuristique est de partir des deux sommets les plus éloignés l’un de l’autre puis de toujours ajouter le sommet le plus proche de la tournée à la position qui rallongerait le moins la tournée, et ce jusqu’à avoir intégré tous les sommets désirés à la tournée.

Dans la plupart des cas cet algorithme est comparable en termes de temps d’exécution à celui du plus proche voisin, mais avec des résultats qui sont la plupart du temps en deçà de ceux de ce dernier. En outre sur certains graphes on parvient à un temps d’exécution ou un résultat bien plus efficient, cependant nous ne sommes pas parvenus à en identifier les causes.

## insertion au plus loin

Cette heuristique fonctionne selon le même principe que la précédente, à la différence près qu’au lieu d’ajouter systématiquement le sommet le plus proche de la tournée on ajoute celui le plus éloigné de cette dernière.

On s’attend naturellement à ce que dans la plupart des cas cet algorithme donne des résultats moins efficaces que l’insertion au plus proches, et les faits donnent raison à notre intuition première sur ce point, mais surprenamment on constate que ces deux algorithmes obtiennent régulièrement des résultats similaires voire parfois en faveur de l’insertion au plus loin.

# Heuristiques locales

## Voisinage d’une tournée

Ici on part d’une tournée initiale et on cherche parmi ses voisins si l’un d’entre eux est plus optimisé qu’elle. Une fois qu’on en a trouvé un plus performant on remplace la tournée initiale et on réitère jusqu’à ne plus trouver d’améliorations.

On définit ici une voisine d’une tournée comme étant une tournée différente pour laquelle il suffirait d’inverser deux éléments pour obtenir la tournée originale.

# Heuristiques génétiques

Les algorithmes génétiques se basent sur les principes biologiques de l’évolution, ils intègrent pour la plupart d’entre eux une part de hasard. Ici nous en avons créé deux similaires à la différence de la notion de « sélection » qui varie de l’un à l’autre.

On définit ici un individu comme étant une tournée de sommets et une population comme étant un groupe d’individus.

Dans un premier temps l’algorithme s’initialise en générant une population d’individus composés aléatoirement. A partir de ce moment il va faire évoluer sa population jusqu’à ce que le meilleur individu de celle-ci ne s’améliore pas lors de l’évolution de la population n vers la population n + 1 pendant un nombre de générations défini dans le constructeur.

L’évolution comporte trois étapes : la sélection, la reproduction et la mutation. Lorsqu’on effectue une sélection on cherche à sélectionner les individus les plus performants de la population précédente sans détruire la diversité de la population ce qui empêcherait son évolution dans le futur. Une fois la sélection terminée on passe à la mutation en faisant se reproduire entre eux les couples d’individus sélectionnés.

La reproduction entre deux individus consiste au fait de « mélanger » leurs deux listes de lieux. Pour cela on sélectionne un indice aléatoire dans le premier parent et on prend tous ses sommets jusqu’à l’indice choisi. Une fois cela fait on retire du second parent les sommets déjà ajoutés à l’enfant et on insère ceux restants à la suite de ceux déjà présents chez l’enfant.

Cette reproduction faite on applique une mutation, définie ici comme l’inversion de deux sommets sélectionnés aléatoirement au sein d’un individu, sur certains individus choisis au hasard.

## sélection par roulette

La sélection par roulette consiste à donner à chaque individu une probabilité d’être sélectionné proportionnelle à la fitness de l’individu. On appelle en biologie « fitness » le taux d’adaptation d’un individu.

## selection par rang

La sélection par rang consiste à donner à chaque individu une probabilité de sélection proportionnelle à son rang dans le classement des individus composant la population.