

**Etude comparative des heuristiques**

S2.02 - Exploration algorithmique d'un problème

Léo Bourdin / Antoine Lindimer / Romain Barabant

IUT DIJON AUXERRE  Département informatique

# Critères de comparaison

## Complexité algorithmique

La complexité algorithmique représente le nombre de calculs basiques que fera l’algorithme selon la taille du problème. Pour un même algorithme, la complexité peut parfois être légèrement différente selon la qualité du code.

**Pour ce critère** : le meilleur algorithme sera celui avec la complexité algorithmique la plus petite.

## Temps d’exécution

Le temps d’exécution représente la durée que prendra l’algorithme pour calculer l’ordre de la tournée finale. La Stopwatch mesurera et affichera cette durée en milliseconde pour chaque algorithme.

**Pour ce critère** : nous examinerons les temps d’exécution des différents cas du jeu d’essais afin d’en conclure quel algorithme est le plus efficace dans chaque situation.

*Remarque : pour que ce critère soit sensé, on exécutera chaque algorithme sur l’ensemble du jeu d’essai afin qu’il soit soumis à différentes situations.*

## Distance

La distance représente la distance totale de la tournée finale trouvée par l’algorithme.

**Pour ce critère** : nous examinerons les temps d’exécution des différents cas du jeu d’essais afin d’en conclure quel algorithme est le plus efficace dans chaque situation.

*Remarque : pour que ce critère soit sensé, on exécutera chaque algorithme sur l’ensemble du jeu d’essai afin qu’il soit soumis à différentes situations.*

# Jeu d’essai de graphes

Comme susdit, le **jeu d’essai** aura pour but de **faire rencontrer aux heuristiques des situations diverses et variées** afin de récolter des données plus précises, donc utiles pour la comparaison. Nous allons donc **explorer différents cas concrets** de la vie utilisant la notion de graphes.

Pour créer un jeu d’essai pertinent, nous avons commencé par recréer les graphes que nous avions étudié en séance de TP. Puis nous avons décidé de créer nos propres graphes afin d’obtenir des résultats exploitables.

Avec des graphes simples (où l’on peut obtenir facilement un résultat en faisant tourner les algorithmes à la main), les résultats ne sont pas très intéressants. Le temps d’exécution est inférieur à 10 ms dans tous les cas, on observe tout de même de petites différences dans les résultats des tournées. Ces résultats étaient à prévoir puisque nous les avions observés durant les séances de découvertes.

En utilisant le graphe fourni avec le sujet (GrapheSimple2.gph), nous avons pu tester nos algorithmes et observer des résultats différents en fonction de l’algorithme.

Pour obtenir des résultats intéressant nous avons commencé par créer des graphes ou tous les sommets sont reliés entre eux, ce sont des cliques. Nous avons essayé de créer une clique de degré 5 « à la main », cela s’est révélé très long et inutile car il semble impossible de créer des cliques de degré supérieur à 10 « à la main ». Nous avons donc créé un algorithme qui permet de générer automatiquement dans la console les lignes qui permettent de créer un graphe avec un fichier en .gph dans le projet.

Cet algorithme imprime dans la console :

* L’usine qui est numéroté 0
* La liste des magasins
* La liste des routes entre tous les sommets

Pour les sommets, les coordonnées sont générées en se basant sur le cercle trigonométrique. On place dans un premier temps l’usine à un abscisse définie et avec une ordonnée de 0, puis on place les sommets sur le cercle à intervalle régulier. En ce qui concerne les routes, avec deux boucles imbriquées, on peut générer chaque route entre tous les sommets. Pour ce qui est de la pondération, elle est aléatoire et comprise entre 1 et le nombres de sommets.

Grâce à cet algorithme, nous avons réalisés une clique de taille 10 (ci-dessous à gauche) et une clique de taille 50 (ci-dessous à droite).

Une image contenant melon, fruit, melon d’Espagne, blanc

Description générée automatiquementUne image contenant dôme

Description générée automatiquement

Grâce à ces deux graphes, nous avons pu tester nos algorithmes et observer des différences, premièrement, en termes de temps d’exécution. Puisque tous les sommets sont reliés entre eux, on retrouve beaucoup moins de contraintes pour passer d’un sommet à un autre. Ce qui fait que le temps d’exécution est très court pour tous les algorithmes implémentés, aux alentours de 50 ms. Deuxièmement

\*différent résultat

* Graph bi partie : génération et exploitation

# Mesure qualitative des algorithmes

## Graphes trois chemins

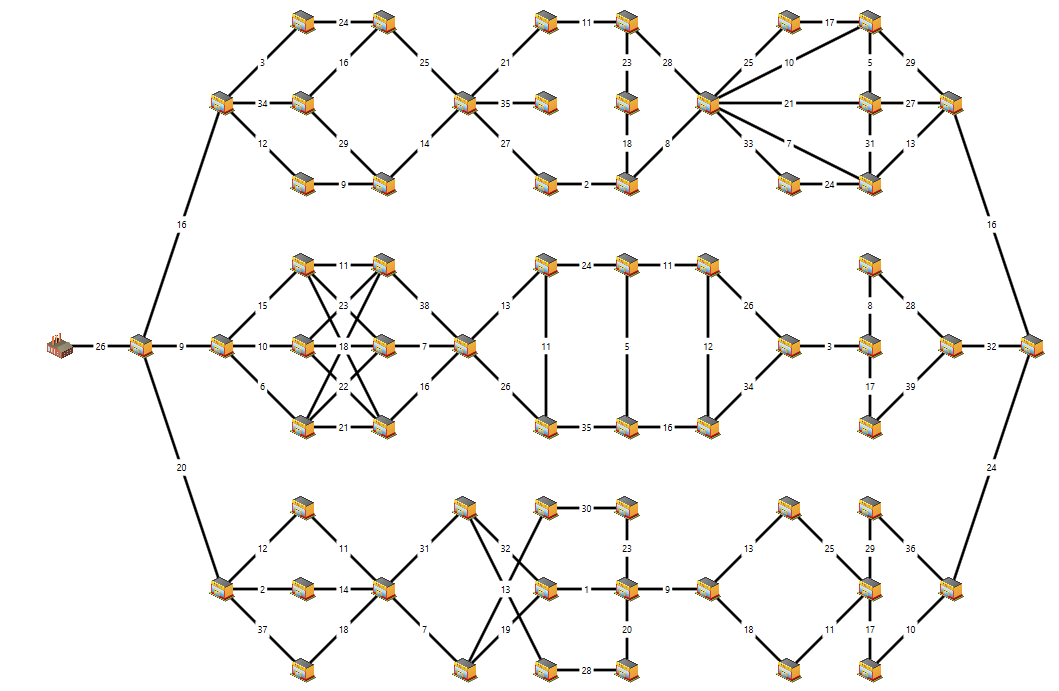


Figure 2 - Graphe avec chemins à longueur aléatoires

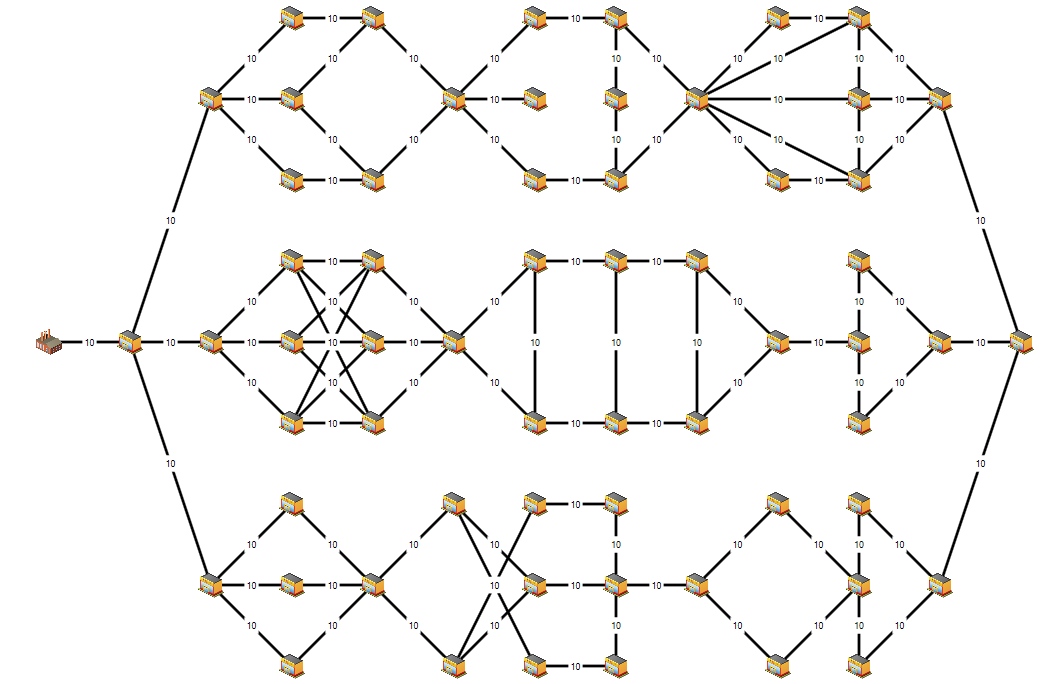


Figure 1 - Graphe avec chemins à longueurs constantes

Pour ces graphes, nous avons voulu observer le comportement de chaque heuristique en lui laissant le choix en trois grands chemins (l’usine étant située tout à gauche).

Nous avons créé deux versions à partir de cette idée : la seule différence est la valeur des distances entre les sommets.

La Figure 1 représente le graphe avec les valeurs constantes (10 partout) et la Figure 2 représente le graphe avec les valeurs aléatoires (entre 0 et 40).

### Tableau de données

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Longueurs constantes | | Longueurs aléatoires | |
|  | Temps d’exécution | Distance | Temps d’exécution | Distance |
| Plus proche voisin | **75** | 900 | **75** | 1520 |
| Insertion proche | 100 | **810** | 100 | 1557 |
| Insertion loin | 100 | 860 | 100 | **1279** |
| Recherche locale | 80 | **810** | 100 | 1613 |
| Aléatoire | 80 | 3900-4500 | 80 | 5500-6500 |

### Comparaison et analyse

Avec des longueurs de chemins aléatoires, on obtient des résultats complètement différents qu’avec des longueurs constantes.

Ainsi, plusieurs données ressortent :

**Graphe à longueurs constantes**

* L’ordre de grandeur des distances des différentes heuristiques est le même (entre 800 et 900, excepté pour l’algo aléatoire)
* Le Plus Proche Voisin à un temps d’exécution plus faible que tous les autres

**Graphe à longueurs aléatoires**

* L’insertion loin trouve une distance bien inférieure à tous les autres
* Le Plus Proche Voisin à un temps d’exécution plus faible que tous les autres ET trouve une distance inférieure à l’insertion proche

Pour ces graphes, on peut donc clairement affirmer que selon les longueurs des chemins, l’algorithme qui s’en sort le mieux n’est jamais le même.

L’insertion proche est plus adaptée au graphe constant et l’insertion loin au graphe aléatoire. Cependant, nous remarquons que si l’on veut prioriser la rapidité d’exécution, Le Plus Proche Voisin est le meilleur compromis.