INF8775 – Analyse et conception d'algorithmes

Hiver 2021

Laboratoire 2

Travail présenté à :

Mohammed Najib Haouas

1897222 – Alexandre Morinvil

1609536 – Mohamed Laziz Taoual

Date de remise :

Lundi 5 avril 2021

# 

# Table des matières

[Table des matières 2](#_Toc68555244)

[Instruction 3](#_Toc68555245)

[Présentation des résultats 4](#_Toc68555246)

[Analyse et discussion 9](#_Toc68555247)

[Faites une analyse asymptotique du temps de calcul pour chaque algorithme. 9](#_Toc68555248)

[Servez-vous de vos temps d'exécution pour confirmer et/ou préciser l'analyse asymptotique théorique de vos algorithmes avec la méthode hybride de votre choix. 9](#_Toc68555249)

[Discutez des trois algorithmes en fonction de la qualité respective des solutions obtenues, de la consommation de ressources (temps de calcul, espace mémoire) et de la difficulté d'implantation. 11](#_Toc68555250)

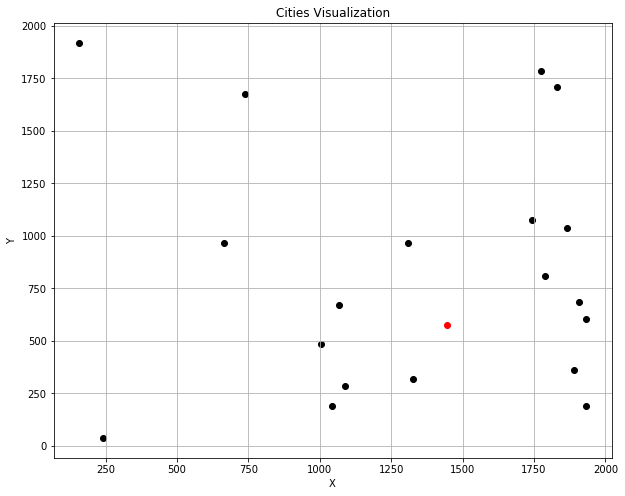
[Indiquez sous quelles conditions vous utiliseriez chaque algorithme. 14](#_Toc68555251)

[On vous fournit 5 exemplaires difficiles à résoudre jusqu’à optimalité (fichiers avec préfixe hard). Tentez de les résoudre à l’aide de vos algorithmes glouton et approximatif et discuter des écarts obtenus. 15](#_Toc68555252)

# Instruction

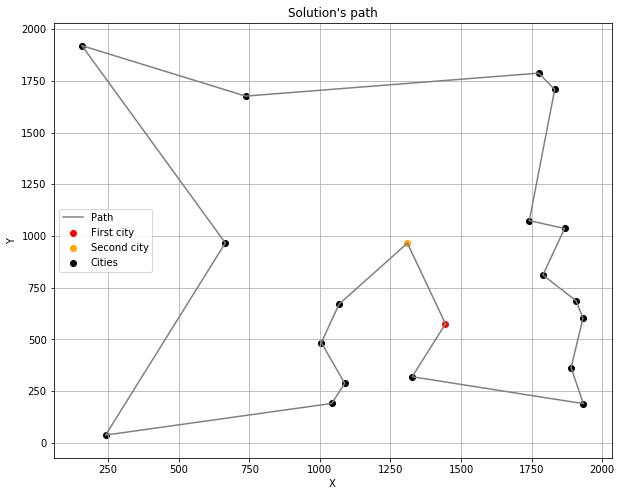
Ce TP vise à se familiariser avec l’analyse empirique et hybride d’algorithmes.

Ainsi, l’objectif de ce TP se d’implémenter trois algorithmes permettant de régler le problème du voyageur. Ce problème se résout, à partir d’un ensemble de points disposés aléatoirement dans un espace en 2 dimension, tel qu’affiché dans l’exemple suivant :



**Figure 1 :** Exemple de visualisation d’un ensemble de données devant être traité

À partir de ces données, le chemin le plus court partant d’un point et terminant au même point doit être déterminé :



**Figure 2 :** Exemple de visualisation de la solution optimale du problème de la figure 1

Un algorithme glouton, un algorithme utilisant le patron de programmation dynamique ainsi qu’un algorithme de résolution approximatif seront utilisés pour résoudre ce problème et une analyse des résultats obtenus suivra.

# Présentation des résultats

Le tableau suivant affiche le temps d’exécution des exemplaires pour chaque taille d’exemplaire ainsi que la distance totale de la solution obtenue pour chaque algorithme. Afin de pouvoir comparer l’algorithme par programmation dynamique avec les autres algorithmes, des exemplaires ayant une taille allant de 6 à 20 ont été utilisés (des tailles supérieures n’ont pas été utilisés car le temps de calcul de l’algorithme de programmation dynamique devient trop grand pour des tailles supérieures).

**Tableau 1 :** Temps de calcul et distance totale résultant de la résolution du problème du voyageur pour différentes taille d’exemplaire pour les trois algorithmes

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Taille de l’exemplaire | Temps Prog. dyn.  [s] | Distance  Prog. dyn. | Temps Glouton  [s] | Distance Glouton | Temps Approx.  [s] | Distance Approx. |
| 6 | 0.004986 | 5701 | 0\* | 5727 | 0\* | 7390 |
| 7 | 0.010971 | 2956 | 0.000997 | 3867 | 0.000997 | 2946 |
| 8 | 0.028923 | 4905 | 0\* | 6109 | 0.001994 | 9233 |
| 9 | 0.554522 | 4748 | 0.000998 | 4983 | 0.000997 | 8414 |
| 10 | 0.148601 | 6982 | 0\* | 6996 | 0.000997 | 8463 |
| 11 | 0.294213 | 6315 | 0\* | 6801 | 0.001995 | 9450 |
| 12 | 0.580445 | 7122 | 0\* | 8193 | 0.001994 | 10934 |
| 13 | 1.523928 | 6777 | 0.000997 | 8870 | 0.001995 | 15253 |
| 14 | 3.369991 | 6652 | 0.000993 | 7447 | 0.00399 | 13832 |
| 15 | 8.166425 | 5889 | 0.000997 | 7069 | 0.004987 | 11634 |
| 16 | 22.29713 | 7673 | 0.000995 | 8849 | 0.00595 | 13245 |
| 17 | 51.12833 | 6691 | 0.001001 | 7760 | 0.009007 | 10786 |
| 18 | 118.4132 | 7022 | 0.000998 | 9633 | 0.007978 | 15252 |
| 19 | 271.2454 | 7725 | 0.001998 | 9476 | 0.011964 | 16419 |
| 20 | 553.6534 | 6205 | 0.001995 | 9110 | 0.011968 | 13504 |

\*Les valeurs « 0 » sont des valeurs trop petites qui ont été ignorées par l’incertitude machine.

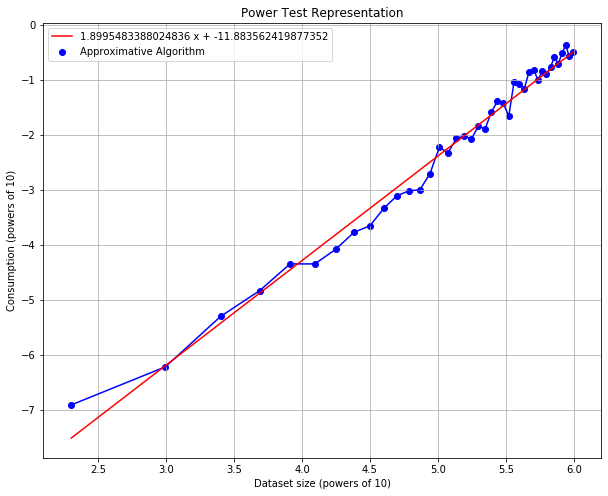
Le tableau suivant affiche le temps d’exécution des exemplaires pour chaque taille d’exemplaire ainsi que la distance totale de la solution obtenue pour les algorithmes glouton .

**Tableau 2 :** Temps de calcul et distance totale résultant de la résolution du problème du voyageur pour différentes taille d’exemplaire pour les algorithmes gloutons et approximatifs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Taille de l’exemplaire | Temps Glouton  [s] | Distance Glouton | Temps Approximatif  [s] | Distance approximatif |
| 10 | 0.001 | 4873 | 0.003987 | 7609 |
| 20 | 0.001999 | 8614 | 0.016949 | 24808 |
| 30 | 0.004992 | 9934 | 0.058839 | 25068 |
| 40 | 0.00798 | 15025 | 0.127661 | 43965 |
| 50 | 0.012962 | 14650 | 0.174567 | 41408 |
| 60 | 0.012966 | 15801 | 0.28025 | 53027 |
| 70 | 0.016918 | 16866 | 0.441823 | 75404 |
| 80 | 0.022972 | 17338 | 0.75694 | 76445 |
| 90 | 0.025926 | 17670 | 0.863719 | 85667 |
| 100 | 0.035873 | 19289 | 1.242679 | 79511 |
| 110 | 0.044881 | 21040 | 1.813196 | 91943 |
| 120 | 0.048864 | 22878 | 2.14289 | 118275 |
| 130 | 0.049873 | 20268 | 2.585119 | 118246 |
| 140 | 0.066784 | 21672 | 3.628051 | 129830 |
| 150 | 0.107747 | 21649 | 4.301617 | 153635 |
| 160 | 0.097734 | 23680 | 5.461737 | 139831 |
| 170 | 0.126698 | 24316 | 6.773643 | 206376 |
| 180 | 0.131649 | 25965 | 8.153194 | 146236 |
| 190 | 0.125665 | 25216 | 9.236877 | 163997 |
| 200 | 0.158612 | 25201 | 11.00587 | 212514 |
| 210 | 0.150598 | 27372 | 13.60352 | 269724 |
| 220 | 0.204456 | 26938 | 15.72571 | 212272 |
| 230 | 0.249323 | 30065 | 16.63719 | 267097 |
| 240 | 0.241319 | 28432 | 18.42211 | 213547 |
| 250 | 0.189491 | 30666 | 21.41514 | 280726 |
| 260 | 0.356045 | 30468 | 26.4161 | 299369 |
| 270 | 0.343118 | 31143 | 31.85104 | 312899 |
| 280 | 0.310171 | 31672 | 33.48491 | 279912 |
| 290 | 0.42187 | 31165 | 34.86172 | 298999 |
| 300 | 0.441913 | 31418 | 37.56479 | 243211 |
| 310 | 0.367985 | 33705 | 41.95971 | 330536 |
| 320 | 0.427969 | 33844 | 44.99357 | 333376 |
| 330 | 0.410901 | 32959 | 49.33257 | 416671 |
| 340 | 0.463867 | 34049 | 54.4801 | 276790 |
| 350 | 0.557509 | 34942 | 57.65085 | 362970 |
| 360 | 0.491682 | 34325 | 62.49633 | 309521 |
| 370 | 0.597445 | 35500 | 72.39982 | 354939 |
| 380 | 0.68716 | 35616 | 76.02821 | 317107 |
| 390 | 0.568483 | 35341 | 85.86241 | 343875 |
| 400 | 0.607366 | 38132 | 94.78853 | 309604 |

### Graphique de l’analyse asymptotique de l’algorithme glouton

Pour l’algorithme glouton, l’on soupçonnait qu’il avait une complexité asymptotique polynomiale. Par conséquent, un test du de puissance a été utilisé. Le graphique résultant de ce test de puissance est affiché dans la figure suivante. Il est à noter que les axes des abscisses et des ordonnées affichent des puissances de 10, il s’agit donc d’un graphique logarithmique.

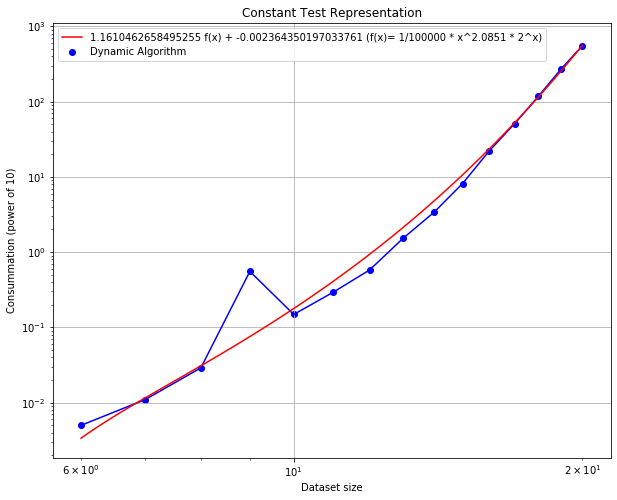


**Figure 3 :** Test de puissance appliqué pour l’algorithme glouton

L’équation résultante du test de puissance pour l’algorithme glouton est la suivante :

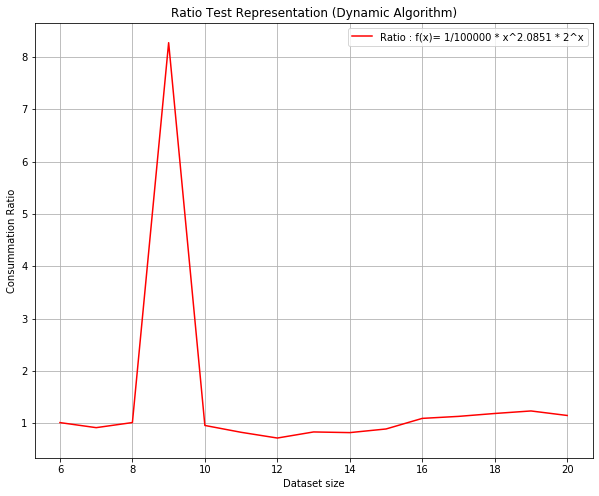
### Graphique de l’analyse asymptotique de l’algorithme par programmation dynamique

Pour l’algorithme par programmation dynamique, l’on soupçonnait qu’il avait une complexité exponentielle de complexité . Par conséquent, un test du ratio et un test de de constantes ont été utilisés. Les graphiques obtenus sont affichés ci-bas. La fonction utilisée pour ces deux tests est la fonction suivante :



**Figure 4 :** Test de constant appliqué pour l’algorithme par programmation dynamique

L’équation résultante du test de constantes pour l’algorithme dynamique est la suivante :

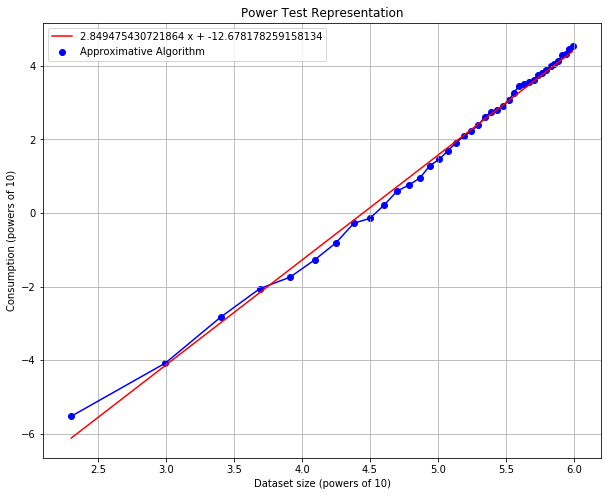


**Figure 5 :** Test du rapport appliqué pour l’algorithme par programmation dynamique

On observe qualitativement que cette fonction temps à converger vers la valeur 1.

### Graphique de l’analyse asymptotique de l’algorithme approximatif

Pour l’algorithme glouton, l’on soupçonnait qu’il avait une complexité asymptotique polynomiale. Par conséquent, un test du de puissance a été utilisé. Le graphique résultant de ce test de puissance est affiché dans la figure suivante. Il est à noter que les axes des abscisses et des ordonnées affichent des puissances de 10, il s’agit donc d’un graphique logarithmique.



**Figure 6 :** Test de puissance appliqué pour l’algorithme approximatif

L’équation résultante du test de puissance pour l’algorithme approximatif est la suivante :

# Analyse et discussion

## Faites une analyse asymptotique du temps de calcul pour chaque algorithme.

L’analyse asymptotique déterminée théoriquement pour l’implémentation faite de chacun des algorithmes est fournie dans un ficher PDF fournis avec ce rapport. Le nom du fichier PDF en question est le suivant :

analyse\_complexite\_1897222\_1609536.pdf

## Servez-vous de vos temps d'exécution pour confirmer et/ou préciser l'analyse asymptotique théorique de vos algorithmes avec la méthode hybride de votre choix.

### Algorithme glouton

Pour l’algorithme glouton, l’on soupçonnait qu’il avait une complexité asymptotique polynomiale. Par conséquent, un test du de puissance a été utilisé. Tel qu’affiché dans la section de la présentation des résultats, l’équation obtenue suite à l’application. L’équation résultante du test de puissance pour l’algorithme glouton est la suivante :

On constate d’une part de manière qualitative que la régression linéaire obtenu via le test de puissance permet de couvrir assez fidèlement le taux de croissance de l’algorithme. Ceci est une confirmation que la complexité algorithme de cet algorithme est polynomiale.

Ainsi, on par analogie avec la signification de la régression linéaire obtenue dans le test de puissance, l’on déterminerait expérimentalement que la complexité asymptotique de cet algorithme serait de l’ordre de 1,9, soit approximativement :

Ceci est cohérent avec la complexité théorique estimée à .

### Algorithme par programmation dynamique

Pour ce qui est du test du rapport réalisé, on observe qualitativement que le graphique du rapport du temps de consommation divisé par la fonction suivante :

tendrait à converger vers une valeur de 1. Ainsi, ceci serait une indication que la complexité asymptotique de cet algorithme serait de l’ordre de :

Ce qui est cohérent avec la complexité asymptotique théorique estimée à :

De plus, pour confirmer cette observation, un test de constante a été appliqué avec la même fonction f(x). Ainsi, il peut être observé qualitativement que la régression linéaire :

obtenue couvre relativement bien le graphique de la consommation de temps. Ceci confirme la complexité expérimentale de déterminée expérimentale.

### Algorithme approximatif

Pour l’algorithme approximatif, l’on soupçonnait qu’il avait une complexité asymptotique polynomiale. Par conséquent, un test du de puissance a été utilisé. Tel qu’affiché dans la section de la présentation des résultats, l’équation obtenue suite à l’application. L’équation résultante du test de puissance pour l’algorithme glouton est la suivante :

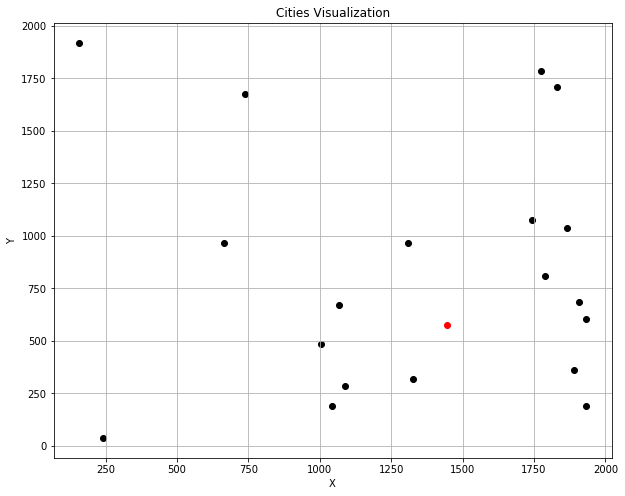
On constate d’une part de manière qualitative que la régression linéaire obtenu via le test de puissance permet de couvrir assez fidèlement le taux de croissance de l’algorithme. Ceci est une confirmation que la complexité algorithme de cet algorithme est polynomiale.

Ainsi, on par analogie avec la signification de la régression linéaire obtenue dans le test de puissance, l’on déterminerait expérimentalement que la complexité asymptotique de cet algorithme serait de l’ordre de 1,9, soit approximativement :

Ceci est cohérent avec la complexité théorique estimée à .

## Discutez des trois algorithmes en fonction de la qualité respective des solutions obtenues, de la consommation de ressources (temps de calcul, espace mémoire) et de la difficulté d'implantation.

Pour un ensemble de donné quelconque contenant 20 points, tel que celui affiché ci-dessous :



**Figure 7 :** Ensemble de donnés utilisé pour soutenir la discussion sur les trois algorithmes

Le point rouge étant le point initial.

Pour les trois algorithmes donnerait les solutions ainsi que les temps de calculs suivants :

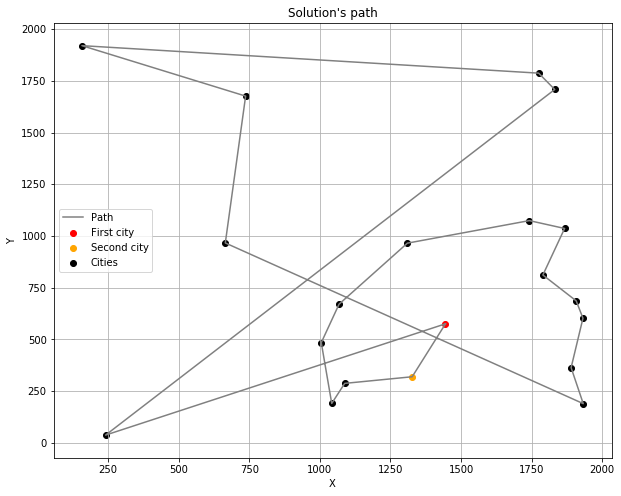
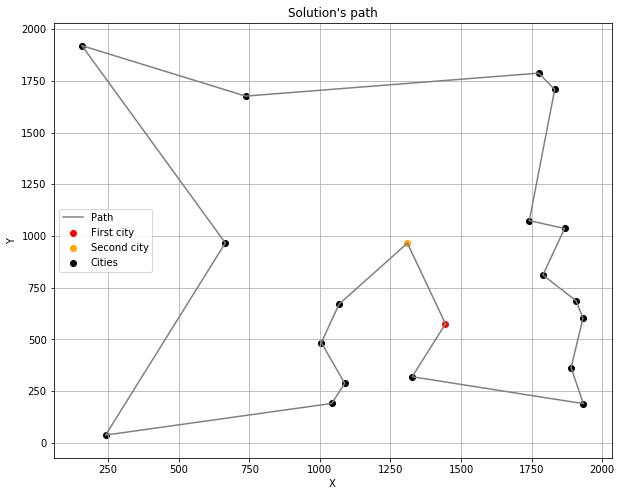


Figure 8 : Solution obtenue pour l’algorithme glouton

Temps utilisé : 0.000998 secondes

Distance totale : 11176

Solution obtenue : [0, 18, 5, 12, 10, 7, 11, 1, 2, 9, 4, 16, 8, 14, 19, 6, 13, 3, 17, 15, 0]

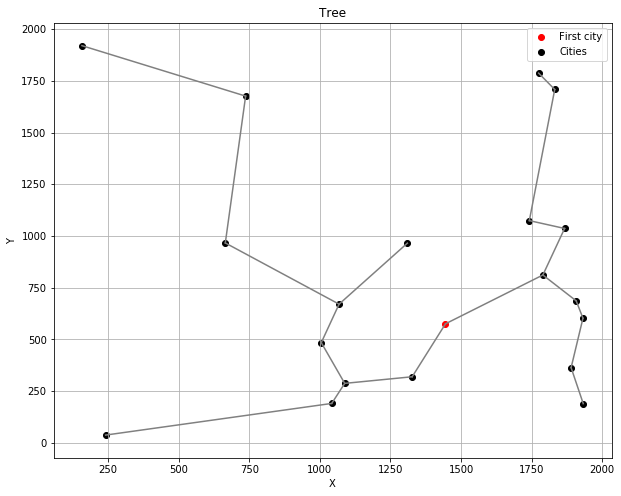


**Figure 9 :** Solution obtenue pour l’algorithme de programmation dynamique

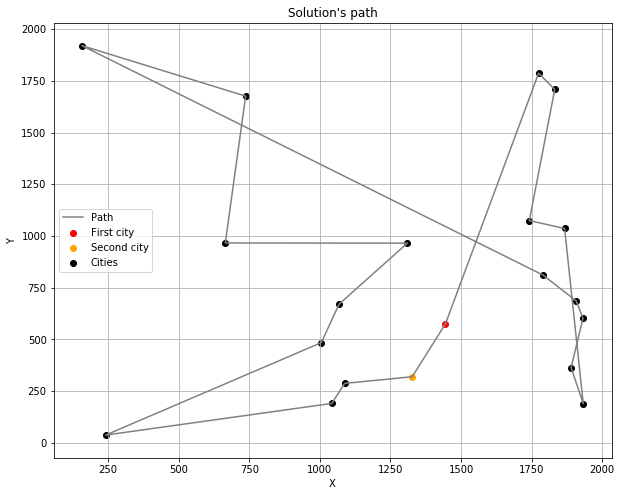
Temps utilisé : 482.601066 secondes

Distance totale : 8591

Solution obtenue : [0, 11, 7, 10, 5, 12, 15, 19, 13, 6, 3, 17, 1, 2, 9, 4, 16, 8, 14, 18, 0]



**Figure 10 :** Arbre sous-tendant minimal utilisé pour la calcul de la solution approximative



**Figure 11 :** Solution obtenue pour l’algorithme approximatif

Temps utilisé : 0.010970 secondes

Distance totale : 16818

Solution obtenue : [0, 18, 5, 12, 15, 10, 7, 11, 19, 6, 13, 9, 4, 16, 8, 14, 2, 1, 17, 3, 0]

**Temps de calcul**

Ainsi, de manière générale, l’on observe que le temps de calcul obtenu pour ces trois algorithmes est d’en l’ordre croissant :

* Algorithme glouton
* Algorithme approximatif
* Algorithme par programmation dynamique

Ceci est dû au fait que l’algorithme par programmation dynamique a une complexité asymptotique exponentielle, ce qui a pour conséquence que le temps nécessaire pour déterminer le chemin optimal augmente de manière monstrueusement rapide pour chaque point ajouté.

De plus, l’on observe que l’algorithme approximatif nécessite également un temps significativement supérieur à l’algorithme glouton. Ceci est dû à sa complexité asymptotique qui est d’un ordre supérieur à celle de l’algorithme glouton. Ceci dit, le goulot d’étranglement qui augmente la complexité asymptotique de cet algorithme d’un ordre 3 plutôt qu’un ordre 2 est l’algorithme de génération d’un arbre sous-tendant minimal. L’implémentation utilisé dans le code de ce laboratoire est d’ordre . Ainsi, si un une implémentation d’ordre avait été utilisé, cet algorithme aurait eu des temps de calculs similaire à ceux de l’algorithme glouton.

**Utilisation de mémoire**

Pour ce qui est de l’utilisation de la mémoire, elle est du plus petit au plus grand dans l’ordre suivant :

* Algorithme glouton
* Algorithme approximatif
* Algorithme par programmation dynamique

L’algorithme par programmation dynamique a une utilisation de la mémoire exponentielle (de l’ordre ) en raison du/des tableaux de distances que cet algorithme génère et utilise.

L’algorithme approximatif utilise la mémoire pour mémoriser le graphe sous-tendant minimal généré et utilisé pour déterminer le parcours optimal approximatif.

Enfin, l’algorithme glouton ne fait pas d’utilisation de la mémoire. Cet algorithme ne fait que parcourir les données non visitées de l’ensemble de donné initial. Ainsi, autre que les données initiales desquels l’on peut retirer les points visités, il n’y a pas d’utilisation additionnelle de la mémoire requise.

**Optimalité de la réponse**

Ceci dit, du plus petit au plus grand, les distances totales déterminée sont données par les algorithmes dans l’ordre suivant :

* Algorithme par programmation dynamique
* Algorithme glouton
* Algorithme approximatif

L’algorithme par programmation dynamique est garanti de trouver la solution optimale par le principe d’optimalité.

L’algorithme glouton est un algorithme naïf qui peut donner la solution optimale dans le meilleur cas, mais souvent peut ne pas donner la meilleure solution parce que cet algorithme ne prévoit pas l’impact d’une décision sur le reste du parcours.

Enfin, l’algorithme approximatif a une très forte probabilité de ne pas donner la solution optimale (il est très improbable de trouver la solution optimale avec cet algorithme). Cet algorithme a plutôt pour objectif de trouver une approximation de la réponse optimale

## Indiquez sous quelles conditions vous utiliseriez chaque algorithme.

**Algorithme glouton**

Cet algorithme peut être utilisé lorsque :

* Un très grand nombre de points est présents (C’est un algorithme relativement rapide)
* Une très petite mémoire est à disposition
* Une solution non optimale peut être acceptée

**Algorithme par programmation dynamique**

Cet algorithme peut être utilisé lorsque :

* Un relativement petit nombre de points est présent (ou des capacités de calculs très puissantes et rapides sont à disposition si un grand nombre de points doit être évalué)
* Une grande mémoire est à disposition.
* La véritable solution optimale est désirée

**Algorithme approximatif**

* Un relativement grand nombre de points est présents (C’est un algorithme relativement rapide)
* Une solution très approximative est acceptable

Ceci dit, la puissance de cet algorithme pourrait réellement être mis à profit seulement si une meilleure implémentation de la génération de l’arbre sous-tendant minimal serait utilisé, faisant passer la complexité de cet algorithme de à . Autrement, cet algorithme est en tout point inférieur à l’algorithme glouton.

## On vous fournit 5 exemplaires difficiles à résoudre jusqu’à optimalité[[1]](#footnote-2) (fichiers avec préfixe hard). Tentez de les résoudre à l’aide de vos algorithmes glouton et approximatif et discuter des écarts obtenus.

Le tableau suivant affiche les résultats obtenus pour les algorithmes gloutons et approximatifs utilisés sur les exemplaires difficiles.

**Tableau 3 :** Résultats optimaux de la résolution des 5 exemplaires difficile

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fichier | Nombre de villes | Solution optimale | Solution alg.  glouton | Solution alg.  approximatif |
| hard\_N52 | 52 | 551609 | 632300 | 3222705 |
| hard\_N91 | 91 | 1228726 | 1338026 | 12533077 |
| hard\_N130 | 130 | 1928734 | 2135153 | 26884848 |
| hard\_N169 | 169 | 2600546 | 2857273 | 46670663 |
| hard\_N199 | 199 | 3139778 | 3352771 | 66289464 |

Ce que l’on observe que pour tous les échantillons, la solution optimale est plus petite que la solution de l’algorithme glouton qui lui-même est inférieur à l’algorithme approximatif.

Ceci correspond au résultat attendu. L’algorithme glouton offrira toujours une réponse plus optimale que l’algorithme approximatif puisque l’algorithme glouton pour chaque position cherchera à obtenir la position suivante qui minimisera immédiatement sa distance, tandis que l’algorithme approximatif quant à lui cherche à minimiser les distances lors de la génération de son arbre sous-tendant, ce qui ne garantit pas la minimalité du parcours lorsque cet arbre est parcouru en ordre préfix (sans répétition de nœud).

De plus, l’algorithme glouton lui-même ne donne jamais une solution optimale parce que cet algorithme cherche à minimiser la distance immédiate de sa prochaine destination, sans prendre en considération l’impacte de ce choix sur le reste de son trajet.

Ainsi, l’optimalité des résultats du tableau 3 correspond aux résultats attendus pour chacun des algorithmes.

1. [↑](#footnote-ref-2)