École Polytechnique de Montréal

Département de génie informatique et génie logiciel

INF4705

Analyse et conception d’algorithmes

Travail pratique 3 : Rapport

Soumis par

Francis de Lasalle, #1706231

Zihui Zhong, #1687994

13 avril 2016

Table des matières

[Introduction 2](#_Toc446042585)

[Description du jeu de données 3](#_Toc446042586)

[Résultats expérimentaux 3](#_Toc446042587)

[Analyse et discussion 7](#_Toc446042588)

[Analyse asymptotique des algorithmes 7](#_Toc446042589)

[Algorithme vorace probabiliste 7](#_Toc446042590)

[Algorithme de programmation dynamique 7](#_Toc446042591)

[Heuristique d’amélioration locale 7](#_Toc446042592)

[Analyse empirique des algorithmes 7](#_Toc446042593)

[Algorithme vorace probabiliste 7](#_Toc446042594)

[Algorithme de programmation dynamique 7](#_Toc446042595)

[Heuristique d’amélioration locale 8](#_Toc446042596)

[Qualité des algorithmes 8](#_Toc446042597)

[Conclusion 9](#_Toc446042598)

[Références 9](#_Toc446042599)

# Introduction

Ce troisième travail pratique est une compétition visant à produire le meilleur algorithme possible pour résoudre un certain problème donné. La compétition se déroule entre les étudiants du cours INF4705.

Le problème consiste à trouver le meilleur ordonnancement possible pour minimiser le nombre d’écoliers dans une file ayant leur vue obstruée, tout en respectant certaines contraintes. Plus précisément, le problème se présente par une liste contenant la hauteur d’écoliers et une liste contenant des paires de numéros d’écoliers pouvant être jumelés. Les écoliers regardent tous devant eux et l’objectif est de minimiser le nombre d’écoliers ayant leur vue obstruée par un écolier plus grand devant lui (directement ou plus loin). De plus, un écolier peut être précédé ou suivit d’un autre écolier seulement si les deux figurent dans l’une des paires de numéros d’écoliers pouvant être jumelés.

Le choix d’algorithme à implémenter est à la discrétion des étudiants. L’algorithme sera exécuté pendant 3 minutes et la meilleure solution obtenue au terme de ces 3 minutes sera retenue. Il convient donc de judicieusement utiliser le temps nous étant imparti.

# Description du jeu de données

Le jeu de données se présente sous la forme de fichiers texte dont la première ligne indique le nombre d’écoliers et la seconde ligne le nombre de paires d’écolier pouvant être jumelés. Le fichier se poursuit avec la liste de la hauteur de chaque écolier (ligne par ligne), puis la liste des paires d’écoliers pouvant être jumelés (ligne par ligne).

Plusieurs exemplaires sont fournis, ayant différent nombres d’écoliers et de paires (liens).

# Présentation de l’algorithme

BLAH

L’algorithme peut-être brièvement représenté sous-forme de pseudocode ainsi :

|  |
| --- |
| backtrack (List<Node> currentIn, Node currentNode)  Si currentIn comprend tout les nodes  retourne true  Si valide(currentIn)  pour tout les nodes n avec un lien avec currentNode  si n pas dans currentIn  ajoute n dans currentIn  si aucun thread est en attende de travail  si(backTrack(currentIn, n))  retourne true  sinon  envoy backTrack(currentIn, n) au thread  enleve n de currenIn  retourne false  valide(List<Node> currentIn)  vérifie que le graph est encore connexe si on enlève ces noeuds avec un DFS. (O(n)) |

# Justification de l’originalité de l’algorithme

Le problème peut être abstrait à celui d’un parcours de graphe. En effet, chaque nœud du graphe est représenté par un écolier et chaque lien par une paire permise d’écoliers. Par exemple, pour 5 écoliers où l’écolier 1 peut être jumelé avec l’écolier 3 et 4, l’écolier 2 avec l’écolier 3 et 5 et l’écolier 3 avec l’écolier 4, le graphe ressemblerait à :

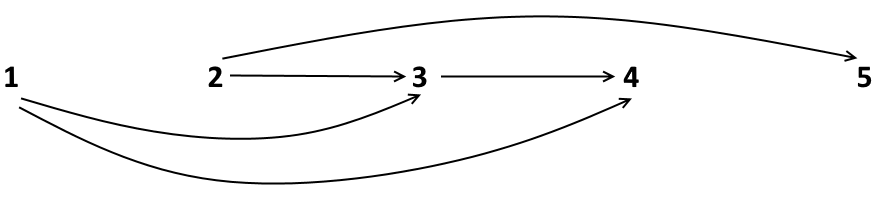


Figure 1. Abstraction du problème sous forme de graphe, notez que le graphe en réalité n’est *pas* orienté

L’objectif peut se reformuler à trouver une chaîne hamiltonienne qui minimise le nombre d’écoliers ayant leur vue obstruée. En effet, une chaîne hamiltonienne est une chaîne parcourant chaque élément passant par chaque élément du graphe une et une seule fois; la chaîne hamiltonienne abstrait donc le concept de file d’écoliers.

Trouver une telle chaîne se fait en faisant une fouille en profondeur du graphe. Un écolier est choisi et, parmi les écoliers pouvant être jumelés, un autre est choisi et ainsi de suite (cette sélection se fait de façon récursive). Les écoliers sont d’abord trillés en ordre croissant de taille, afin que les premiers écoliers considérés dans la file étant construite soient les plus petits. Cette astuce très simple permet de significativement améliorer la solution initiale. En effet, il est évident que la solution sera plus optimale en choisissant d’abord les plus petits écoliers.

À chaque ajout d’un écolier à la file d’écoliers, celui-ci est enlevé des écoliers restant et l’algorithme teste si le graphe restant (avec ces nœuds en moins) est toujours connexe. Il s’agit de l’astuce pour effectuer le retour arrière (« backtracking »). En effet, si le graphe restant après avoir enlevé les écoliers ajoutés à la file en construction n’est plus connexe, c’est qu’il n’existe aucune chaîne passant par chaque nœud et donc, il est impossible que les écoliers restants dans le graphe puissent être jumelés ensemble. Si la vérification e connexité échoue, l’algorithme retourne en arrière, c’est-à-dire qu’il considère le nœud suivant au dernier nœud ajouté plutôt que celui ayant fait échoué le test de connexité.

Comme cette recherche peut-être longue et comme il convient de maximiser l’utilisation de ressources matérielles disponibles pour optimiser la solution, cet algorithme est parallélisé sur plusieurs fils d’exécution différents, afin de répartir la charge de travail.

Une fois une première chaîne trouvée, celle-ci est gardée comme point de départ d’une batterie d’algorithmes d’optimisation locale (heuristiques d’optimisation locale). Le premier algorithme utilisé est une version « allégée » d’un algorithme « complet » de permutation de nœuds. La méthode nOptOptimization() permet d’échanger de place deux séquences de nœuds consécutifs au sein de la chaîne. Le nombre de nœuds échangés de place va de 1 à la moitié de la taille de la chaîne. L’exemple suivant illustre l’un des 3 échanges qui seraient considérés dans une chaîne de 5 éléments si un échange de 2 nœuds consécutifs est fait :

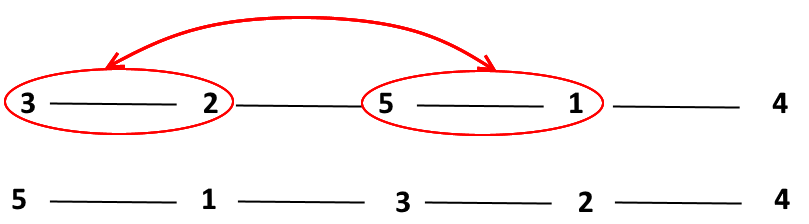


Figure 2. Exemple de l’algorithme d’optimisation locale par échange de nœuds

Cette heuristique s’est avérée, selon nos tests, être fort performance et très peu coûteuse en temps de calcul. La complexité asymptotique de cet algorithme est en , ce qui correspond à la complexité d’un algorithme 2-opt « complet », mais permet de tester plusieurs transformations très variées.

Cet algorithme s’exécute tant qu’une amélioration est observée. Plus précisément, des groupes de 1 à la taille maximale de nœuds sont échangés et l’algorithme recommence à 1 tant qu’une amélioration a été faite. La taille maximale des blocs de nœuds pouvant être échangé est fixée arbitrairement à 20, c’est-à-dire que pour de petites chaînes (moins de 40), l’algorithme testerait des échanges allant jusqu’à la moitié de la taille de la chaîne, mais, en pratique, aucune amélioration n’a été observée lors d’échanges de groupes de plus de 20 nœuds (des améliorations ont été observés jusqu’à environ 16). Il est donc inutile d’essayer d’échanger des groupes de 50 nœuds dans une chaîne de 100 nœuds.

Lorsque cet algorithme est terminé, un second algorithme d’optimisation locale est lancé...