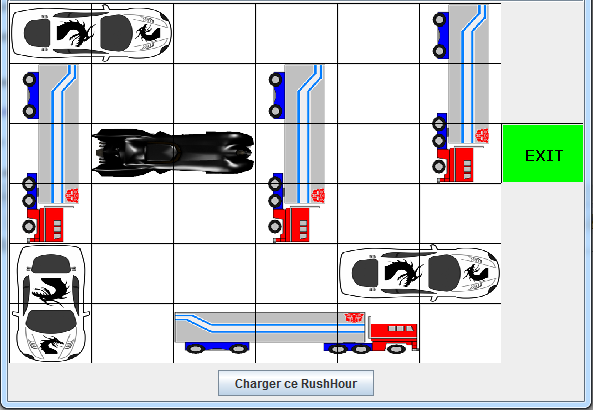
07/12/2016



# Sous la direction de Hung Nguyen

Loïc LAFONTAINE

Maxime LAVASTE

Projet de MOGPL

Résolution d’un casse-tête Rush Hour

Table des matières

[Introduction 1](#_Toc468306000)

[I. Résolution par programmation linéaire en variables binaires 1](#_Toc468306001)

[1. Déterminer les fonctions objectives 1](#_Toc468306002)

[2. Déterminer les contraintes additionnelles 2](#_Toc468306003)

[II. Résolution par l’algorithme de Dijkstra 4](#_Toc468306004)

[1. Résolution de RHC 4](#_Toc468306005)

[2. Résolution de RHM 4](#_Toc468306006)

# Introduction

Dans le cadre de l’enseignement de l’UE MOGPL, nous avons créé un outil de résolution automatisé d’un problème du puzzle de Rush Hour.

Le jeu simule une congestion automobile dans un parking à l’heure de pointe (appelée « *rush hour* » en anglais, d’où son nom) ; le but du jeu est d’extraire le véhicule rouge d’une grille dans laquelle plusieurs autres véhicules bloquent la sortie. Il faut pour cela les déplacer, mais ils sont suffisamment entrecroisés et il y a suffisamment de contraintes de déplacement pour que la solution ne soit pas triviale.[[1]](#footnote-1)

# Résolution par programmation linéaire en variables binaires

correspond à la position du marqueur du véhicule , au tour .

correspond aux positions occupées par un véhicule au marqueur au tour .

correspond à un mouvement d’un véhicule , au tour du marqueur vers le marqueur .

le nombre de mouvements possible pour que la voiture rouge atteigne l’arrivée.

Le but de cet exercice est de programmer un programme de résolution linéaire d’un problème de RushHour. Nous allons devoir générer de nombreuses contraintes pour résoudre ce problème.

## Déterminer les fonctions objectives

Plusieurs chemins sont possibles pour atteindre la victoire, d’un coût plus ou moins important. Nous devons créer deux formules qui minimisent le coût de déplacement pour RHM ainsi que pour RHC.

On cherche, dans RHM à minimiser le nombre de mouvements. Or, un mouvement correspond à une valeur de . Nous devons donc minimiser cette fonction objective.

Avec , la fonction valeur absolue.

## Déterminer les contraintes additionnelles

Nous avons déjà quatre contraintes données par le sujet.

≤ 1 -

a. La voiture rouge est positionnée devant la sortie au terme du dernier mouvement.

Avec i, l’identifiant de la voiture rouge. En informatique, la 17ème case d’un tableau est au 16ème élément.

b. Au plus un véhicule est déplacé par tour.

La somme des pour chaque k correspond au nombre de mouvement effectué pour un tour. On ajoute donc comme contrainte, pour chaque k, la somme de doit être comprise entre .

c. La position du marqueur d’un véhicule i est bien mise à jour si le véhicule i se déplace.

On regarde pour chaque véhicule la position de son marqueur au mouvement précédent.

De plus, dans notre implémentation, nous avons ajoutés des contraintes pour pour initialiser le problème pour et .

# II. Résolution par l’algorithme de Dijkstra

Nous avons utilisé l’algorithme de Breadth-first search pour générer tous les configurations de graphes possibles à partir du graphe de départ.

## 1. Résolution de Rush Hour Cases

Trouver la solution

Le poids d’une arrête entre deux sommets représentent le nombre de cases d’un mouvement entre deux configurations différentes. En appliquant l’algorithme de Dijkstra, nous obtenons le plus court chemin pour arriver à une configuration de victoire et donc un chemin avec le moins de cases de déplacements.

À chaque itération de l’algorithme, on marque le sommet (non marqué) qu’on peut atteindre avec la plus petite distance.

Ainsi, la condition d’arrêt pour Dijkstra est le marquage d’un sommet correspondant à une configuration-but, c’est-à-dire quand la voiture rouge est devant la sortie.

Afficher la solution

En stockant les prédécesseurs de chaque sommet lors de l’exécution de l’algorithme de Dijkstra, on est capable de recréer la séquence de déplacements. En effet, un sommet équivaut à une configuration. Il suffit alors de partir de la solution et de remonter les prédécesseurs jusqu’à retrouver la configuration de départ. On a au final la séquence de déplacements de la solution vers la configuration initiale, il suffit alors de l’inverser pour avoir cette séquence dans le bon ordre.

NB : Il aurait été plus difficile d’utiliser un tableau de successeurs vu qu’un sommet du sous-graphe à la fin de l’algorithme peut avoir plusieurs successeurs alors qu’il n’a qu’un seul prédécesseur.

## 2. Résolution de Rush Hour Mouvements

S’il existe une arrête entre deux sommets du graphe des configurations, il existe alors un déplacement (d’un véhicule) permettant de passer de l’un à l’autre. Dans un problème RHM, on cherche à minimiser le nombre de ces déplacements. Lorsque l’on déplace un véhicule de 1,2,3, etc. cases, on n’effectue au final qu’un seul déplacement, le poids de chaque arête est donc fixé à 1. Nous pouvons ensuite appliquer Dijkstra pour obtenir un chemin de poids minimal (et donc effectuant le moins de mouvement) de la configuration de départ vers un sommet représentant une configuration-but.

11.

nombre de config réalisable a.k.a

soit le nb total de config (config.size()) ?

soit compter le nb de config pour N mv donné

2e possibilité :

On envoie un paramètre supplémentaire à la fonction de Dijkstra pour le nombre de mouvements. On utilise deux compteurs internes : un pour compter le nombre de configurations réalisables et un autre pour savoir si on a atteint le nombre de mouvements passé en paramètre.

Initialisation du compteur mv à 1 ?

Quand est-ce qu’on incrémente le compteur mv ?

Pour chaque sommet marqué, on mettait à jour la distance pour atteindre ses successeurs, il suffit d’incrémenter de 1 le compteur autant de fois qu’il y a de successeurs. Cette incrémentation ne s’effectue pas si on atteint le nombre de mouvements demandé.

1. https://fr.wikipedia.org/wiki/Rush\_hour\_(casse-t%C3%AAte) [↑](#footnote-ref-1)