Table des matières

Table des figures

Préambule

La décision est le propre de l'homme, puisqu'elle implique le choix conscient entre plusieurs solutions possibles. Si la décision est aux antipodes de l'instinct et du réflexe, elle est cependant maintes fois dictée, dans une proportion variable, par l'intuition.

Au fil du temps, l'homme a changé d'attitude en face des situations d'organisations ou économiques : au lieu de se contenter de l'intuition ou d'une déduction qualitative, il réclame souvent une expression numérique des faits.

La Recherche Opérationelle est une science récente datant au plus de la deuxième guerre mondiale, d'où elle doit son nom, grâce aux opération militaires associées aux problèmes mathématiques par biais de la modélisation. En réalité, elle est bien plus ancienne, car dès le dix-septième siécle PASCAL et FERMAT proposèrent la notion d'espérence mathématique pour résoudre les problèmes de décisions dans l'incertain.

Mais ce n'est qu'en 1938 que le physicien anglais BEACKETT a pu réunir la première équipe de la Recherche Opérationelle, et a eu le mérite de proposer une solution qui a pu permettre de traiter rapidement et avec succès la question difficile de l'implantation optimale des radars de surveillance qui allait jouer un rôle déterminant dans la bataille d'Angleterre, ainsi que l'optimisation des ressources militaires limitées.

L'évolution économique et sociale du $20^{ème}$ siécle observée mondialement pointe vers l'augmentation des niveaux de services, cette évolution se manifeste à tous les niveaux, souvent accompagnées d'exigences accrues pour que les services soient rapides, fiables, et peu coûteux.

La recherche opérationnelle offre des méthodologies qui permettent d'accroître l'efficacité et la qualité, en terme tant économiques que de services,

des entreprises et des systèmes de transport. D'un autre côté, la complexité des opérations et des processus de planification et de gestion des systèmes de transport offrent de grandes opportunités de modélisation et algorithmiques qui font avancer la discipline.

Introduction

L'histoire de la théorie des graphes débute peut $\hat{e}tre$ avec les travaux d'Euler au 18^{eme} siècle et trouve son origine dans l'étude de certains problèmes, tels que celui des ponts de Königsberg , la marche du cavalier sur l'échiquier ou le problème de coloriage de cartes. La théorie des graphes s'est alors développée dans divers domaines tels que la chimie, la biologie et les sciences sociales. Depuis le début du $20^{éme}$ siécle, elle constitue une branche part entiére des mathématiques, grâce aux travaux de König, Menger, Cayley puis de Berge et $d'Erdos$.

De manière générale, un graphe permet de représenter la structure, les connexions d'un ensemble complexe en exprimant les relations entre ses élèments : réseau de communication, réseaux routiers, interaction de diverses espéces animales, circuits électriques,. . .

Les graphes constituent donc une façon très ingénieuse de modélisation d'un grand nombre de problèmes variés en se ramenant $\grave{a}$ l'étude de sommets et d'arcs. Les derniers travaux en théorie des graphes sont souvent effectués par des informaticiens, du fait de l'importance qu'y $revêt$ l'aspect algorithmique.

De nombreux problèmes de la vie réelle ont été résolus grâce à La théorie des graphes. Parmis eux les problèmes de calcul d'itinéraire dans le domaine du transport. Néanmoins, pour certaines applications, aucun des algorithmes présents dans la littérature n'est approprié. Une de ces applications est le calcul d'un itinéraire pour rejoindre un véhicule se déplaçant dans un réseau routier. Ce besoin a été exprimé par les entreprises de transport public urbain qui doivent gérer des interventions sur les bus de leur flotte. En effet, parfois, le chauffeur d'un bus est confronté à un incident pendant son trajet qui peut être de nature diverse : incident technique, des passagers qui provoquent un désordre. Dans ce cas, une équipe d'intervention compétente doit être envoyée pour régler le problème. Si l'incident est

grave, le bus est immobilisé,ce qui fait que l'équipe d'intervention n'a aucun mal à calculer son itinéraire pour le rejoindre dans les plus brefs délais. Par contre, si l'incident est mineur et ne nécessite pas l'arrêt du bus, ce dernier doit continuer son itinéraire, sachant que , les compagnies de transport public possèdent des chartes de qualité à respecter en termes d'horaires. De plus, dans le cas où la compagnie de transport est une entreprise privée, les autorités locales infligent des pénalités financières importantes en cas de non respect des tableaux de marche affichés dans les stations. Par suite, une première problématique consiste à calculer l'itinéraire de l'équipe d'intervention pour rejoindre le bus, qui est en mouvement, le plus rapidement possible. D'autre part, généralement plusieurs équipes d'intervention disponibles et plusieurs incidents peuvent se déclarer en même temps. Par conséquent, une deuxième problématique consiste à réaliser une affectation optimale des équipes.

Ce mémoire est devisé en quatre chapitres, qui se présentent comme suit :

Dans le premier chapitre, on a donné les éléments de la théorie des graphes en présentant quelques définitions et concepts de base sur les graphes.

Dans le deuxième chapitre, on s'est intéressé à l'algorithme de Djikstra pour résoudre le problème de plus court chemin et l'algorithme hangrois pour résoudre le problème d'affectation dans le but d'utiliser ces deux algorithmes pour la résolution de la problématique.

Le troisième chapitre est consacré à la modélisation de notre problème à l'aide des graphes et dans ce chapitre un algorithme de résolution sera proposé. On terminera par la vérification de l'efficacité de la résolution en terme de temps.

Le quatrième chapitre sera consacré à l'implémentation des résultats développés dans ce mémoire en utilisant le logiciel MATLAB et nous exécutant notre programme sur un exemple pratique dans les réseaux routiers.

Nous terminrons notre travail par une conclusion générale.

Chapitre 1

Définitions de base et notation

Les graphes permettent de modéliser toute situation dans laquelle il y a des interactions entre les objets. Les techniques utilisées en théorie de graphes permettent de répondre à beaucoup de problémes algorithmiques posés. Ce premier chapitre fournit au lecteur les bases en théorie des graphes, nécessaires à la bonne compréhension des notions abordées dans la suite de ce mémoire. Ce chapitre, reprend ainsi ce qu'on peut trouver dans la majeure partie des ouvrages d'introduction sur ce sujet.

1.1 Concepts et définition de base

1.1.1 Les graphes

Un graphe G est un couple d'ensembles finis $(X(G), E(G))$ *où* $E(G)$ est constitué de paires de $X(G)$. Les éléments de $X(G)$ sont appelés sommets de G et ceux de $E(G)$ *arêtes* de G . Si aucune confusion n'est possible on note X au lieu de $X(G)$ et E au lieu de $E(G)$.

1.1.1.1 Graphe orienté

Un graphe orienté G est un couple (X, E) avec X un ensemble dont les éléments sont appelés noeuds et U un ensemble dont les éléments sont des couples ordonnés de sommets appelés arcs.

Exemple 1.1.1
Soient $G = (X, U)$, $i, j \in X$.
i et j sont appelés noeuds du graphe G. Si G est un graphe orienté, $a = (i, j) \in E$ *est appelé arc de G, i représente l'origine de l'arc, on note :* $I(a) = i$. *et j représente sa destination, on note :* $T(a) = j$. *Si G est un graphe non orienté,* $(i, j) \in E$ *est appelée arête de G et nous ne pouvons pas parler d'origine ou de destination dans ce cas.*
En effet, dans le cas d'un graphe orienté, $(i, j) \neq (j, i)$. *Dans le cas contraire,*

$(i, j) = (j, i)$. *Les figures 1.1 et 1.2 présentent respectivement un exemple d'un graphe orienté et un exemple d'un graphe non orienté.*

FIGURE 1.1 – Graphe orienté

FIGURE 1.2 – Graphe non orienté

Remarque 1.1.2
Une arête peut être transformée en deux arcs de sens différents.

1.1.2 Le voisinage :

1.1.2.1 Le voisinage d'un sommet :

Définition 1.1.3
Si $e = (x, y) est une arête de G on dit que y et x sont voisins dans G et qu'il forment les extrémités de e.
On définit le voisinage d'un sommet x dans un graphe G comme l'ensemble de ses voisins.

1.1.2.2 Les successeurs et les prédécesseurs :

Soit $G = (X, U)$ un graphe orienté avec $x,\ y$ deux sommets de X.
– L'ensemble des prédécesseurs d'un sommet x est défini par :

$$\Gamma^-(x) = \{y \in X / \exists\ u \in U \text{ où } I(u) = y \text{ et } T(u) = x\} \tag{1.1}$$

– L'ensemble des successeurs d'un sommet x est définit par :

$$\Gamma^+(x) = \{y \in X / \exists\ u \in U \text{ où } T(u) = y \text{ et } I(u) = x\} \tag{1.2}$$

– Dans un graphe orienté, l'ensemble des voisins du sommet x est égal à la réunion de l'ensemble de ses prédécesseurs et ses successeurs :

$$\Gamma(x) = \Gamma^+(x) \cup \Gamma^-(x) \tag{1.3}$$

1.1.2.3 Les arêtes incidentes

Définition 1.1.4
Deux arêtes sont dites incidentes si elles ont une extrémité en commun.

1.1.2.4 Le degré d'un sommet

Soit $G = (X, U)$ un graphe orienté on a :

1. Le demi-degré extérieur d'un sommet x est égal au nombre d'arcs ayant le sommet x comme extrémité initiale, on dit aussi le nombre d'arcs incidents extérieurs au sommet x. On le note :

$$d_G^+(x) = |\{u \in U / I(u) = x\}| \tag{1.4}$$

Le demi degré intérieur d'un sommet x est égal au nombre d'arcs ayant le sommet x comme extrémité terminale, on dit aussi le nombre d'arcs incidents intérieurs au sommet x. On le note :

$$d_G^-(x) = |\{u \in U / T(u) = x\}| \tag{1.5}$$

1. Le degré d'un sommet x est le nombre d'arcs ayant x comme extrémité initiale ou terminale, on dit aussi le nombre d'arcs adjacents à x. On le note :

$$d_G(x) = d_G^+(x) + d_G^-(x) \tag{1.6}$$

1.1.3 Les Graphes pondérés

Définition 1.1.5
Un graphe G est dit pondéré si une fonction de poids lui est associée. Cette fonction, notée d, peut prendre différentes formes.

Définition 1.1.6
Un réseau est un graphe $G = (X, U)$ muni d'une application $d : U \to \mathbb{R}$ qui à chaque arc fait correspondre sa longueur $d(u)$, on note un tel réseau par :
$R = (X, U, d)$. En pratique, $d(u)$ peut matérialiser un coût, une distance, une durée etc.

Cependant, dans la suite d sera supposée positive ($\forall (i, j) \in U, d(i, j) \geq 0$). Cette hypothèse a été considérée car elle permet de modéliser un coût positif afin de refléter la réalité des problèmes traités.

1.2 Quelques types de graphes :

1.2.1 Graphe biparti :

Un graphe est biparti si l'ensemble de ses sommets peut être partitionné en deux classes X_1 et X_2 de sorte que deux sommets de la même classe ne soient jamais adjacents, il se note $G = (X_1, X_2, E)$.

1.2.1.1 Graphe biparti complet

Un graphe $G = (X_1, X_2, E)$ est biparti complet, si tout sommet de X_1 est adjacent à tout sommet de X_2

1.2.2 Le multigraphe

Définition 1.2.1
Les arêtes multiples sont des arêtes qui ont les mêmes extrémités.

Définition 1.2.2
Un multigraphe ou graphe multiple est un graphe qui contient des arêtes multiples.

1.2.3 Un pseudographe

Définition 1.2.3
Une arête (arc) dont les extrémités sont confondues est une boucle.

Définition 1.2.4 *Un pseudographe est un multigraphe qui contient des boucles*

1.2.4 Un graphe simple :

Un graphe simple est un graphe sans boucle ni arcs (arêtes) multiples.

1.3 Le couplage dans les graphes :

1.3.1 Le couplage

Un couplage dans un graphe est un ensemble d'arêtes M tel que :

1. M ne contient pas de boucles .
2. deux arêtes quelconques de M n'ont pas d'extrêmité commune.

1.3.2 Le couplage maximum

Un couplage maximum est un couplage dont le nombre d'arêtes est maximal.

1.3.3 Le couplage parfait

Un couplage parfait est un couplage qui est incident à tous les nœuds.

FIGURE 1.3 – Le couplage parfait

1.4 Cheminement dans un graphe

1.4.1 La chaine

Soit $G = (X, E)$ un graphe. Une chaine joignant deux sommets x_t et x_k dans un graphe G est une suite de sommets reliés par des arêtes tels que, deux sommets successifs ont une arête commune.
On la note : (x_t, x_1,x_2,...,x_k) et on dit que x_t et x_k sont les extrémités de la chaine.

1.4.2 Le chemin

Un chemin entre deux sommets i et j du graphe est une suite d'arcs liant i à j.
Ce chemin est noté $Ch(i, j)$. L'ensemble de ces chemins est noté $ECh(i; j)$.
Formellement, $Ch(i; j) = (u_0; u_1; \ldots; u_n)$, avec :

- $u_k \in X, \forall k \in [0, n]$
- $u_0 = i,\ et\ u_n = j$
- $\forall u_k, u_{k+1} \in Ch(i, j); (u_k, u_{k+1}) \in U$

Définition 1.4.1

- *x est ascendant de y s'il existe un chemin de x à y.*

- *x est descendant de y s'il existe un chemin de y à x.*

1.4.3 Le cycle

Un cycle est une chaine simple dont les deux extrémités coïncident.

1.4.4 Le circuit

Un circuit est un chemin dont les deux extrémités sont confondues.

1.5 La représentation de graphe sur machine

1.5.1 La matrice d'adjacence

Soit un graphe $G = (X, U)$ contenant n sommets et m arcs ($|X| = n$ et $|U| = m$)
La matrice d'adjacence du graphe $G = (X, U)$ est une matrice $n * n$, ses éléments prennent deux valeurs 1 ou 0. Chaque ligne et chaque colonne correspondent à un sommet du graphe.
Ainsi chaque élément de la matrice indique la relation qui existe entre deux sommets :

- 1 signifie que les deux sommets sont reliés par un arc orienté.

- 0 signifie que les deux sommets ne sont pas reliés par un arc. orienté.

1.5.2 La liste d'adjacence

Une liste d'adjacence est une structure de données utilisée pour représenter un graphe dans laquelle on associe à chaque sommet sa liste de voisins. Ainsi, on ne stocke que les informations " utiles " concernant l'adjacence dans le graphe
Cette représentation est particulièrement adaptée aux graphes creux (c'est-à-dire peu denses), contrairement à la matrice d'adjacence adaptée aux graphes denses.

1.5.3 Matrice d'adjacence pondérée

On definit un élément a_{ij} de la matrice d'ajacence pondérée d'un graphe simple par :

$$a_{ij} = \begin{cases} d(i,j) \quad si & \text{(i,j)} \in \text{U} \\ 0 \quad si & \text{i=j} \\ \infty & \text{sinon} \end{cases} \tag{1.7}$$

1.6 La connexité dans un graphe

1.6.1 La connexité

On définit la connexité dans un graphe, par la relation entre deux sommets de la manière suivante :

Deux sommets x et y ont une relation de connexité

$$\Longleftrightarrow$$

Il existe une chaine entre x et y ou bien $x = y$.

1.6.2 Les composantes connexes :

On appelle composante connexe un ensemble de sommets, qui ont deux à deux la relation de connexité, de plus tout sommet en dehors de la composante n'a pas de relation de connexité avec les sommets de cette composante.

1.6.3 Le graphe connexe

Un graphe $G = (X, E)$ est dit graphe connexe si tous ses sommets ont deux à deux la relation de connexité. Autrement dit, G contient une seule composante connexe.

Un graphe est connexe .

$$\Longleftrightarrow$$

Il possède une seule composante connexe.

FIGURE 1.4 – Le graphe connexe

1.7 Arbres et arborescences

Définition 1.7.1
Un arbre est un graphe connexe et sans cycle.

Définition 1.7.2
Un sommet s d'un graphe G est une racine (resp.une (anti-racine) s'il existe un chemin joignant s à chaque sommet du graphe G (reps. Joignant chaque sommet de G a s) à l'exception du sommet lui-même.

Remarque 1.7.3
Une arborescence est un arbre mais la réciproque est fausse.

Définition 1.7.4
Un graphe $G = (X, U)$, avec $n = |X| \geq 2$ sommets. G est une arborescence de racine s si :

- G est un arbre
- s est une racine de G

$MinZ = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} c_{ij}$

$$\begin{cases} \sum_{i=1}^{n} x_{ij} = 1 & \text{i=1,...,n} \\ \sum_{j=1}^{n} x_{ij} = 1 & \text{j=1,...,n} ; \\ x_{ij} \in \{i, j\}, & \forall \text{ i, j= 1,...,n.} \end{cases}$$

Chapitre 2

Problèmes d'optimisation

Dans ce chapitre nous donnons un bref aperçu de la théorie de la complexité des algorithmes en insistant sur les notions fondamentales de classes de problèmes P (polynomial). Et puis nous présentons, le problème classique du plus court chemin ainsi que le problème d'affectation en donnant un algorithme de résolution pour chacun de ces derniers problèmes.

2.1 Théorie de la complexité

La théorie de la complexité est une branche des mathématiques et de l'informatique ayant pour cadre l'étude de la difficulté des problèmes algorithmiques, et qui vise à classer ces problèmes en fonction de cette difficulté. Ici, les mots "complexité" et "difficulté" ne se rapportent pas à la mise au point d'un algorithme de résolution, ou aux concepts avancés auxquels il peut faire appel (une structure), mais plutôt à la quantité de ressource à utiliser pour résoudre le probléme.

En ce qui nous concerne, les ressources consistent en le temps que met un algorithme à résoudre le probléme (c'est sa complexité temporelle) et l'espace mémoire qu'il utilise au cours de son exécution (sa complexité spatiale). Le principal atout de la théorie de la complexité est que ces grandeurs sont exprimées indépendamment de tout dispositif physique concret, plutôt, elle est basée sur un modéle de calcul abstrait, généralement une machine de Turing. Ce qui permet de comparer facilement l'efficacité de deux algorithmes en s'affranchissant de considérations telles que la vitesse du processeur. On est quasiment sûrs aujourd'hui que certains problèmes nécessite, pour leur résolution, des algorithmes dont le temps de calcul est bien supérieure à celui d'autres problèmes. Et c'est en ce sens que l'on dira qu'ils sont plus difficiles

2.1.1 Concepts de base

Définition 2.1.1
Un problème est une question générale possédant des paramétres dont la valeur n'est pas connue.

Une instance d'un problème est obtenue en affectant une valeur à chacun de ses paramètres. La taille d'une instance désigne généralement la quantité de cases mémoires nécessaires pour décrire les paramètres.

2.1.2 Problèmes de décision et d'optimisation

Définition 2.1.2
Un problème de décision est un problème auquel la réponse est oui ou non.

Définition 2.1.3
Un problème d'optimisation consiste à déterminer la meilleure solution parmi toutes les solutions réalisables.

2.1.3 Complexité en temps et en espace

Définition 2.1.4
La complexité temporelle d'un algorithme correspond au nombre d'instructions élémentaires (opérations arithmétiques, comparaison, affectation. . .) effectuées au cours de son exécution.

Définition 2.1.5
La complexité spatiale d'un algorithme correspond au nombre de cases mémoires occupées par les données manipulées par l'algorithme au cours de son exécution.

En général, le temps d'exécution dépend de la taille de l'instance du problème considéré ; en genérale, plus une instance est grande, plus le problème demandera de temps pour être résolu. Par exemple, si l'on considère un algorithme de tri d'un tableau d'entiers, le nombre d'instructions et l'espace occupé ne seront pas les mêmes si l'on travaille sur un tableau de 10 entiers ou sur un tableau de 10 000 entiers. On exprime donc le temps (ou toute autre mesure de complexité) en fonction de la taille d'une instance générale du problème considéré, souvent notée n. En algorithmique des graphes, en fonction du problème traité on choisit généralement le nombre n de sommets, ou le nombre m d'arêtes du graphe. Mais même sur des instances de même taille, une complexité peut varier d'une instance à une

autre : par exemple, rechercher une valeur dans un tableau demande plus de temps dans un tableau dont les éléments sont désordonnés que dans le même tableau trié. Aussi définit on généralement une complexité en considérant la pire instance possible parmi toutes les instances de taille n, c'est-à-dire celle demandant le plus de ressources.
La notation grand "O" dite aussi symbole de Landau, décrit le comportement asymptotique d'une fonction, exprimé à l'aide d'une autre fonction généralement plus simple.

Définition 2.1.6
(Notation grand O)
$F(n) = O(g(n))(F(n)$ *est en grand* $O(g(n)))$ *quand* $N \to \infty$ *si et seulement si* $\exists M > 0, n_0 \in \mathbb{R}$ *telque* $\forall n \geq \quad n_0 |F(n)| \leq M|g(n)|$

Intuitivement, ceci signifie qu'à partir de n_0 *et à un facteur constant prés,* F *ne croît pas plus rapidement que* g*.*

Un problème de décision X est dans la classe P si, pour chacune de ses instances, dont la taille est notée n , il existe un réel positif k tel qu'il peut être résolu par un algorithme de complexité temporelle $O(n^k)$ c'est-à-dire qu'il peut être décidé en temps polynomial.
Les problèmes de la classe P sont dits faciles. Ce sont ceux que l'on sait résoudre efficacement.

2.2 Problème classique du plus court chemin

2.2.1 la longueur d'un chemin

Dans un réseau $R = (X, U, d)$. La fonction de coût d'un chemin dans G, notée C, est définie comme suit

$$C(u_0, u_1, \ldots, u_n) = \begin{cases} \sum_{k=0}^{n-1} d(u_k, u_{k+1}) & \text{Si} \quad \text{n}\neq 0 \text{ et } \forall \text{ K, } (u_k, u_{k+1}) \in U \\ 0 & \text{si n=0} \\ \infty & \text{Sinon} \end{cases}$$

Étant donnés deux sommets x et y d'un réseau $R = (X, U, d)$, trois cas peuvent se présenter :

1. Il n'existe pas de chemin de x à y dans $\mathbb{R}$.

2. Il existe un chemin de x à y dans R mais pas de chemin de longueur minimum.
3. Dans l'ensemble des chemins joignant x à y dans $\mathbb{R}$, il en existe un de longueur minimum.

2.2.2 La notion du plus court chemin

Soit $R(X, U, d)$ un réseau. Le plus court chemin d'un sommet i à un sommet j s'il existe est noté $PCC(i, j)$:
PCC(i,j) = Ch(i,j) avec $C(Ch(i, j)) = min_{y \in ECh(i,j)}\{C(y)\}$.

Dans le cas où il existe plusieurs chemins avec un coût minimal, un parmi eux est choisi aléatoirement.

2.2.3 Position du probléme

Soit le problème suivant :
A chaque sommet x, associer un chemin de longueur minimum joignant un la racine s donné à x dans le réseau $R = (X, U, d)$

Définition 2.2.1
Un circuit de longueur négative est appelé circuit absorbant

Théorème 2.2.2
Une condition nécessaire et suffisante pour que le probleme posé dans la section 2.2.3 ait une solution est que :

1. *s soit racine*
2. *R ne contienne pas de circuit absorbant*

2.3 Algorithme de Résolution

2.3.1 Algorithme de Dijkstra

On applique cette algorithme pour déterminer une arborescence des plus courtes distances sur un réseau $R = (X, U, d)$, où les longueurs des arcs sont positives ou nulles ($d(e) \geq 0 \; \forall \; e \in U$).

Soit un reseau $R = (X; U, d)$ et $S_{source} \in X$ une racine de R. L'algorithme de Dijkstra est présenté ci-dessous. Cet algorithme utilise deux ensembles particuliers : l'ensemble des noeuds candidats et l'ensemble des

noeuds fermés. L'ensemble des noeuds candidats, noté Q, est l'ensemble des noeuds dont le plus court chemin n'a pas encore été déterminé. Par contre, l'ensemble des noeuds fermés, noté F, est l'ensemble des noeuds dont le plus court chemin a été calculé de manière définitive.

- //Initialiser les plus court chemins
- $PCC(S_{source}, S_{source})$= (S_{source}), CPCC(source,source)=0
- $\forall i \in N \backslash i \neq S_{source}$, $PCC(S_{source}, i) = \phi$, $CPCC(source, i) = \infty$
- //Initialiser l'ensemble Q
- $Q = \{S_{source}\}$
- //Initialiser l'emsemble F
- $F = \phi$
- Tant que Q $\neq \phi$ faire
- Choisir i de Q tel que $C(PCC(S_{source}, i))$ soit minimal
- $Q = Q \setminus \{i\}$ et $F = F \cup \{i\}$
- Développer les successeur j de i
- Pour chaque j faire
- Si $C(PCC(S_{source}, i)) + d(i, j) < C(PCC(S_{source}, j))$ Alors
- $PCC(S_{source}, j) = PCC(S_{source}, i) \cup \{j\}$
- $C(PCC(source, j) = C(PCC(source, i) + d(i, j)$
- $Q = Q \cup \{j\}$
- Fin Si
- Fin Pour
- Fin Tant que

2.3.2 Principe de l'algorithme

L'idée de l'algorithme de Dijkstra et de calculer de proche en proche, l'arborescence des plus courtes distances, issue du sommet s à un sommet donné x ; Une particularité de cet algorithme est que les distances s'introduient dans l'ordre croissant.

L'algorithme maintient une distance d_i pour chaque noeud i avec un statut permanent ou temporaire
A chaque itération, on choisit le noeud i dont la distance temporaire est minimale, on en fait une distance permanente et on ajuste les distances temporaires des noeuds qu'on peut atteindre du noeud i.

2.3.3 Complexité de l'algorithme

Au total, l'algorithme effectue au plus $|x|^2+|U|$ *operation* $\leq 2|x|^2$ *operation* [13] L'utilisation d'un tas de Fibonacci [11] donne un meilleur temps d'exécution amorti : $0(|U| + |X|.log|X|)$.

Remarque 2.3.1

2.4 Problème d'affectation et l'algorithme hongrois :

2.4.1 Problème d'affectation des tâches

Le problème d'affectation [13] est un problème classique de la recherche opérationnelle. Informellement ce problème consiste à attribuer au mieux des tâches à des machines. Plus formellement, l'objectif est de déterminer un couplage parfait de poids minimum (ou un couplage maximum) dans un graphe biparti valué. Le problème d'affectation peut être résolu en temps polynomial ($0(n^3)$) [14] [15] par l'algorithme hongrois, il appartient par conséquent à la classe de complexité P.

2.4.2 Interprétation du probléme :

2.4.2.1 Calcul numérique :

Soit une matrice $A = (c_{ij})$, i, $j = 1, \ldots n$. Il s'agit de trouver une plus petite somme d'éléments de A n'appartenant ni à la *même* ligne, ni à la *même* colonne, pris deux à deux.

2.4.3 Problème de couplage :

Soit un graphe biparti complet $G = (X, Y, E)$ et une application *coût* $d : U \to \mathbb{R}$ qui associe à chaque *arête* du graphe, son *coût*. Si $\mid X \mid = \mid Y \mid$ = n, G est donc le graphe biparti complet sur 2n sommets.
Le problème est de trouver un couplage de cardinalité n et de valeur minimale.

2.4.4 Réolution par l'algorithme hongrois :

Soit un probléme d'ffectation représenté par sa matrice des *coûts* $A = (c_{ij})$, $i, j = 1, \ldots, n$. La résolution de ce problème comporte deux étapes :

– **Etape 1** : Création de zéro : On crée un zéro, au moins, sur chaque ligne et chaque colonne de la matrice, en soustrayant le plus petit élément de la ligne (respectivement, la colonne) de toute la ligne (resp. La colonne).

– **Etape 2** : Application de l'algorithme hongrois.

On conviendra qu'un zéro sélectionné est un zéro indépendant, et que tout zéro sur la *même* ligne ou colonne qu'un zéro sélectionné dit un zéro barré.

2.4.5 Principe de résolution :

– Si, à l'issue de l'étape 1, il existe un nombre de zéros indépendants égal à l'ordre de la matrice, alors la solution est trouveé.
– Sinon, on passe à l'étape 2.

2.4.6 L'algorithme hongrois :

C'ést un algorithme de marquage. Il marque, alternativement, des lignes et des colonnes de la matrice.

1. Marquer toute ligne n'ayant pas de zéro sélectionné
2. Marquer toute colonne ayant un zéro barré sur une ligne marquée
3. Marquer toute ligne ayant un zéro sélectionné sur une colonne marquée
4. Répéter (2) et (3) jusqu'à ce que le marquage ne soit plus possible
5. Barrer toute ligne non marquée et toute colonne marquée.
   Il exixte alors, dans cette matrice, trois sortes d'éléments :
   – Des éléments non barrés
   – Des éléments barrés une fois
   – Des éléments barrés deux fois
6. On forme une nouvelle matrice, issue de la première, de la manière suivante :
   – Soit a_0 le plus petit élément non barré. On retranche a_0 de tous les éléments non barrés.
   – Les éléments barrés une fois restent inchangés.
   – a_0 est rajouté aux éléments barrés deux fois.
7. Recommencer la procédure de résolution, à deux étapes, sur la nouvelle matrice.

2.5 Quelques cas particuliers :

2.5.1 Cas de matrice non carrée :

Si le nombre de ligne (rep. colonnes) de la matrice d'affectation est supérieur à celui des colonnes (resp. ligne), rendre la matrice carrée, en rajoutant les colonnes (resp. ligne) manquantes de zéros. Puis, appliquer le procédé de résolution à cette matrice

2.5.2 Présence de $< \infty >$ dans la matrice :

Pour signifier une interdiction d'affecter un élément i sur un élément j de la matrice, il ya lieu de mettre $a_{ij} = \infty$.

Chapitre 3

Interception d'un mobile dans un graphe

Dans ce chapitre on proposera des algorithmes pour résoudre la problématique posée dans l'introduction et pour cela on doit modéliser le problème par les graphes. a la fin on fera une analyse de la complexité de ces algorithmes afin de verifier leur efficacité.

3.1 Problème d'interception un-à-un

3.1.1 Le modèle

Le réseau routier [16] est représenté par un graphe statique déterministe $G = (X; U)$ avec X l'ensemble des neouds et U l'ensemble des arcs. Dans la pratique, un noeud modélise un arrêt de bus ou un croisement et un arc représente une route liant directement deux noeuds. Soit $d(u, v)$ le poids de l'arc $(u, v) \in U$. Dans la pratique, le poids d'un arc représente le temps nécessaire pour le parcourir. Cette métrique a été choisie car elle est fréquemment utilisée dans le domaine des transports et elle est simple à mesurer.

Les bus et les voitures n'utilisent pas nécessairement les mêmes voies de circulation et ne roulent pas forcément à la même vitesse.

En effet, dans la majorité des réseaux routiers urbains, des voies dédiées aux bus sont aménagées afin qu'ils évitent les embouteillages. En conséquence, nous considérons deux graphes G_o et G_p telsque $G_o = (X_o; U_o)$ est le graphe représentant les routes sur lesquelles les bus se déplacent et $G_p = (X_p; U_p)$ celui où les voitures se déplacent. Les fonctions de poids associées à chaque graphe est respectivement d_o et d_p.

Dans notre modèle, G_o et G_p doivent avoir des noeuds en commun mais

ne partagent aucun arc. Même si les bus et les voitures utilisent la même route (où il n'existe pas de voie bus), cette route est représentée par deux arcs distincts, l'un dans G_O et l'autre dans G_p. Donc, dans notre modèle, $X_O \cap X_p \neq \emptyset$; et $U_o \cap U_p = \emptyset$.

Dans la suite, le terme Mobile Objectif (MO) désigne le bus et le terme Mobile Poursuivant (MP) désigne l'équipe d'intervention. $Init_{MO} \in X_O$ et $Init_{MP} \in X_P$ sont des noeuds spéciaux représentant respectivement la position initiale de MO dans G_O et MP dans Gp. La figure 3.1 présente un exemple. G_O possède neuf noeuds $X_O = \{1; 2; 4; 5; 6; 7; 8; 9; 10\}$ et G_p possède neuf noeuds $Xp = \{1; 2; 3; 4; 5; 6; 7; 8; 9\}$.

L'ensemble des noeuds partagés est $X_O \cap X_P = \{1; 2; 4; 5; 6; 7; 8; 9\ \}$. Dans la figure 3.1, les arcs appartenant à U_o sont représentés par des lignes discontinues et les arcs appartenant à U_P sont représentés par des lignes continues. La position initiale de MO est le noeud 1 et la position initiale de MP est le noeud 7.

FIGURE 3.1 – Graphes G_O et G_p

Dans la suite, un chemin entre $Init_{MO}$ et $w \in X_O$ est noté $Ch_{MO}(w)$. L'ensemble de ces chemins est noté $ECh_{MO}(w)$. De même, un chemin entre $Init_{Mp}$ et $r \in X_p$ est noté $Ch_{MP}(r)$. l'ensemble de ces chemins est noté $ECh_{MP}(r)$.

Hypothèse 1 :
MO ne peut suivre qu'un seul chemin appelé itinéraire du mobile objectif et noté $It(MO)$, ce chemin relie la position initiale de MO ($Init_{MO}$) à sa position finale ($Dest_{MO}$). $It(MO)$ est supposé connu et fixe. De plus MO ne peut passer par le même noeud plus d'une fois.

L'hypothèse précédente est adaptée à notre contexte de travail car un bus possède un itinéraire connu et fixe. De plus, en général, il ne passe pas par le même croisement ou par le même arrêt plus d'une fois. Cependant, il est toujours possible de modéliser cette dernière situation. En effet, il sufit de dupliquer le noeud par lequel le bus passe plusieurs fois en dupliquant tous les arcs le reliant aux autres noeuds dans G_o et G_p. comme dans la figure 3.2 qui montre un exemple où le noeud 1 a été dupliqué. Supposons qu'au départ, $It(MO) = (1; 4; 5; 2; 1; 10)$, en dupliquant le noeud 1, le nouvel itinéraire de MO devient le suivant $It(MO) = (1; 4; 5; 2; 1; 10)$.

FIGURE 3.2 – la duplication d'un noeud

Définition 3.1.1
Soit $It(MO) = (u_0, \ldots, u_q, \ldots, u_l)$ *avec* $u_0 = InitMO$ *et* $u_l = Dest_{MO}$. *l'itinéraire partiel au noeud* u_q *est déffenit comme étant le chemin* $I_t(MO; u_q) = (u_0, \ldots, u_q)$.
Dans l'exemple de la figure 3.3, $Init_{MO} = 1$, $Dest_{MO} = 9$

FIGURE 3.3 – Ecxemple de graphes G_0 et G_p avec un itineraire de MO

$It(MO)$=*(1 ;10 ;2 ;6 ;9) et* $It(MO; 2) = (1; 10; 2)$.
Dans la suite, la fonction de coût associée à G_o *(respectivement à* G_p*) est notée* C_o *respectivement* C_p*). De plus, le plus court chemin de* $Init_{MP}$ *à* $r \in Np$ *est noté* $PCC_{MP}(r)$.

Définition 3.1.2
Un noeud w *est appelé noeud d'interception si :*

1. $w \in Xo \cap Xp$ *(*w *est un noeud partagé entre* G_o *et* G_p*) ;*
2. $w \in It(MO)$ *(*w *appartient à l'itinéraire de MO) ;*
3. $C_o(It(MO; w)) \geq C_p(PCC_{MP}(w))$*(MP atteint* w *avant MO).*

But : Trouver un noeud d'interception optimale tel que $C_o(It(MO; OPT))$ soit minimal. Un tel noeud est appelé noeud d'interception optimal.

3.1.2 L'algorithme de référence

3.1.3 Principe de l'algorithme

Une première idée [16] pour trouver le noeud d'interception optimal consiste à calculer les plus courts chemins de la position initiale de MP à tous les autres noeuds du graphe. Il devient alors facile de trouver l'ensemble des noeuds d'interception et de choisir le meilleur. Cet algorithme est appelé algorithme de référence . Pour cet algorithme, l'ensemble $Intercep$ est déféni comme étant l'ensemble des noeuds d'interception. Pour calculer le noeud d'interception optimal, il suffit de trouver l'ensemble $Intercep$ et les $PCCh_{MP}(u)$ avec $u \in Intercep$. Ensuite, le noeud d'interception optimal OPT est le noeud de l'ensemble $Intercep$ tel que $C_o(It(MO; u))$ est minimal. Dans cette optique, l'algorithme de Dijkstra est utilisé. En effet, l'algorithme de Dijkstra permet de calculer le chemin

de coût minimal d'un noeud source vers tous les autres noeuds du graphe.

3.1.4 Algorithme d'interception de référence

1. //Initiliser l'ensemble Intercep
2. Intercep=$\emptyset$
3. Calculer les plus courts chemins de $Init_{MP}$ vers tous les noeuds de G_p
4. Pour $\forall$u $\in$ It(MO)$\cap$ G_p fair
5. Si C_p $(PCC_{MP}(u)) \leq C_0(It(MO,u)))$ alors
6. Intercep=Intercep $\cup$ { u}
7. Fin si
8. Fin pour
9. Si Intercep= $\emptyset$ Alors ECHEC
10. Sinon Déterminer le noeud OPT $\in$ Intercep tel que $C_o(It(MO,OPT)$ est minimal

3.1.5 Complexité de l'algorithme

La complexité de l'algorithme egale a la complexité de l'algorithme de Dijkstra plus la complexité de la boucle a la lingne 4 qui s'exécute $|X_O|$ fois
Dans le pire des cas ce qui donne : $|X_O| + |U_p| + |X_P|.log|X_p|$

3.1.6 Inconvénient

Cette démarche implique d'explorer entièrement le graphe représentant le réseau routier. Étant donné que ces graphes peuvent être de taille importante, cette solution est rejetée. D'où une autre approche est proposée, qui consiste à adapter un algorithme de Dijkstra au cas de l'interception afin de minimiser le nombre de noeuds traités.

3.1.7 L'algorithme de dijkstra adapté a l'interception :

L'idée principale consiste à changer le noeud de destination pendant le déroulement. Au début, le premier noeud de $It(MO)$ G_p en partant de $InitMO$ est choisi comme étant le noeud destination Obj. Pendant l'exécution, si $C_p(PCC_{MP}(Obj)) \leq C_o(It(MO;Obj))$ alors Obj est le noeud d'interception optimal. Sinon le noeud suivant de $It(MO) \cap G_p$ est choisi

comme étant le nouveau noeud destination. L'algorithme est décrit ci-dessous. Dans l'algorithme, l'ensemble F représente l'ensemble des noeuds avec un coût final et l'ensemble Q représente l'ensemble des noeuds candidats. Soit $It(MO) = (u_0, \ldots, u_q, \ldots, u_m)$, la fonction 'suivant$(It(MO); u_q)$est défénie comme suit :

3.1.8 Algorithme d'interception un-à-un

1. //calculer le cout des noeuds de $It(MO)$
2. Soit $It(MO) = (u_0, u_1, \ldots, u_m)$
3. $\forall u_k \in It(MO)$ $calculer$ C_0 $(It(MO, u_k))$
4. //Initialiser les plus court chemins de G_p
5. $PCC_{MP}(Init_{MP}) = (Init_{MP})$
6. $\forall r \in X_p$ $/r \neq Init_{MP}, PCC_{MP}(r) = \phi$
7. $//(PCC_{MP}(r)) = \infty$
8. // Initialiser l'emsemble Q
9. $Q = \{Init_{MP}\}$
10. //Initialiser l'ensemble F
11. $F = \phi$
12. //Initialiser le noeud de destination
13. $Obj = Init_{MO}$
14. Si $Obj \notin It(MO) \cap G_p$ $Alors$
15. $\quad Obj = suivant(It(MO), Obj)$
16. $\quad Si$ $aucun$ $noeuds$ $partager$ $alors$ $ECHEC$
17. $\quad //ECHEC1$
18. $\quad\quad Si$ $Obj = \phi Alors$ $retourner$ $ECHEC1$
19. $\quad\quad$ Fin Si
20. Fin Si
21. $s_i = InitMP$
22. $Tantque$ $Q \neq \phi$ $faire$
23. $\quad Tantque$ $C_p(PccM_p(s_i) \leq C_0(it(MO, Obj)$ et $Q \neq \phi$
24. $\quad Q = Q \setminus \{s_i\}$ et $F = F \cup \{s_i\}$
25. $\quad$ Si $s_i = Obj$ Alors succés

26. Sinon
27. Développer les successeurs s_j de s_i
28. *Pour chaque* s_j *faire*
29. *Si* $C_p(PCC_{MP}(s_i)) + d_p(s_i, s_j) < C_p(PCC_{MP}(s_j))$
30. *Alors* $PCC_{MP}(s_j) = PCC_{MP}(s_i) \cup \{s_j)\}$
31. $C_p(PCC_{MP}(s_i) = C_p(PCC_{MP}(s_i)) + d(s_i, s_j)$
32. $Q = Q \cup \{s_j\}$
33. Fin Si
34. Fin pour
35. Choisir s_i de Q avec $Cp(PCC_{MP})(s_i)$ minimale
36. Fin tant que
37. Si $C_p(PCC_{MP}(s_i) > C_0((it(MO, Obj))$
38. Suivant 2
39. Obj= suivant(It(MO),Obj)
40. *Si* $Obj = \phi$ *alors* $ECHEC2$
41. Sinon
42. *Si* $Obj \in F$
43. Alors Retourner Succés
44. Fin tanque

3.1.9 Optimalité de l' algorithme :

Dans cette section l'optimalité de la solution fournie par notre algorithme est verifieé selon les lemmes suivants :

Lemme 3.1.3

Si dans une itération de l'algorithme :

$C_p(PCC_{MP}(Si)) > C_o(It(MO; Obj))$ *alors* $C_p(PCC_{MP}(Obj)) > C_o(It(MO; Obj))$.

Preuve 1 :

Ce lemme est une conséquence directe de l'algorithme de Dijkstra. En effet, dans n'importe quelle itération de l'algorithme, le noeud en cours de traitement possède un coût inférieur au coût des noeuds qui seront

traités dans les itérations suivantes. Comme le noeud *Obj* n'a pas été encore sélectionné, alors $C_p(PCC_{MP}(\text{Obj})) \geq C_p(PCC_{MP}(S_i)) > C_o(It(MO;\text{Obj}))$.

Lemme 3.1.4

Si l'algorithme traite un noeud Obj alors $\nexists w \in It(MO) \cap G_p \setminus \{Obj\}$ *tel que :*

$C_o(It(MO;w)) < C_o(It(MO;Obj))et C_p(PCC_{MP}(w)) \leq C_o(It(MO;w))$.

En d'autres termes, il n'existe pas un noeud d'interception w qui soit situé avant Obj dans $It(MO)$.

Preuve 2 :

Si la fonction $suivant(It(MO);\text{Obj})$ est appelée alors *Obj* courant n'est pas un noeud d'interception. En effet, si la fonction '$suivant(It(MO);\text{Obj})$' est appelée, alors
Soit $obj \notin it(MO) \cap G_p$ donc *obj* n'est pas un noeud d'interception

ou $C_p(PCC_{MP}(Si)) > C_o(It(MO;Obj))(SUIVANT\ \ 1\ \ ou\ \ 2)$.

D'où, d'après le lemme 1, *Obj* n'est pas un noeud d'interception. Comme la fonction $'suivant'$ est appelée pour changer le noeud *Obj*, alors lorsque l'algorithme traite un noeud de $It(MO)$, tous les noeuds traités précédemment ne sont pas des noeuds d'interception.

Lemme 3.1.5
Si l'algorithme retourne SUCCES alors le noeud Obj est un noeud d'interception. Autrement dit $C_p(PCC_{MP}(Obj)) \leq C_o(It(MO;Obj))$.

Preuve 3 :

L'algorithme admet deux issues ou il retourne le résultat succès, la première dans la ligne 25 et pour y arriver il faut vérifier la condition d'accès à la boucle de la ligne 23 c'est-à-dire :

$$(C_p(PCC_{MP}(Obj)) \leq C_o(It(MO;Obj))) \tag{3.1}$$

la deuxième issue se situe à la ligne 43 . L'instruction de cette ligne n'est exécuter sauf si $obj \in F$ sachant que pour ajouter un sommet à l'ensemble F il doit vérifier la condition de la boucle a la ligne 23.

Proposition 1 :
Si l'algorithme retourne$SUCCÈS$ alors Obj est le noeud d'interception optimal. En d'autres termes, il n'existe pas un noeud d'interception

$$w \in It(MO) \cap G_p \backslash \{Obj\} telque C_o(It(MO; w)) < C_o(It(MO; Obj)). \quad (3.2)$$

Preuve 4 :
D'après le lemme 3.1.5, si l'algorithme retourne SUCCÈS alors Obj est un noeud d'interception. De plus, d'après le lemme 3.1.4, il n'existe pas un noeud d'interception avec un coût inférieur à $C_o(It(MO;\text{Obj}))$. Par conséquent, Obj est le noeud d'interception optimal.
Proposition 2 :
Si l'algorithme retourne ÉCHEC alors il n'existe aucun noeud d'interception.

Preuve 5 :
Si l'algorithme retourne ÉCHEC alors soit il n'existe aucun noeud partagé entre G_o etG_p (ÉCHEC 1), soit il n'existe pas de noeud partagé w tel que $C_o(It(MO; w)) \geq C_p(PCC_{MP}(w))$(ÉCHEC 1 ou ÉCHEC 2 avec lemme 2). En conclusion, il n'existe aucun noeud d'interception.

3.1.10 Complexité de l'algorithme :

L'algorithme développé dans cette section résulte de l'ajout de certaines instructions à l'algorithme de Dijkstra original, ses instructions son localisé dans la ligne 1 à 3, de la ligne 12 à 20 et de la ligne 37 à 44. La majorité des instructions ajoutées sont des instructions conditionnelles 'Si', qui n'ont aucun effet sur la complexité. Cependant, il existe deux boucles, la première se situe à la ligne 3. Et la deuxième se situe à la ligne 23. La boucle à la ligne 3 calcule le temps d'arrivée pour tous les noeuds de $It(Mo)$, d'où au pire des cas, cette boucle réalise $|X_o|$ opérations. Quant à la boucle localisée à la ligne 23 permet d'interrompre l'algorithme de Djikstra temporairement l'orsque c'est nécessaire afin de changer le neoud de destination objectif grâce à la procédure "suivant" et reprend les calcule des plus courts chemins à partir du point ou on'est arrêté donc cette boucle n'a aucune influence sur la complexité de la boucle principale sauf que le teste d'entré à cette boucle sera effectué au plus $|X_p|$ fois.
En conséquence la complexité de la totalité de l'algorithme et :

$$O(|X_o| + |U_p| + |X_p| + |X_p|.ln(X_p)) = O(|X_o| + |U_p| + |X_p|.ln(X_p)) \quad (3.3)$$

3.2 Problème d'interception plusieurs-à-plusieurs

3.2.1 Le modèle

Le même modèle que celui dans la section 3.1.1 est utilisé excepté que maintenant il existe un ensemble de K mobiles poursuivants $\{MP_1, MP_2, \ldots, MP_K\}$ dans G_p et un ensemble de L mobiles objectifs $\{MO_1, MO_2 \ldots, MO_L\}$ dans G_0 [16]. Un exemple est présenté dans la figure 3.4. Les notations utilisées pour l'interception un-à-un sont étendues à l'interception plusieurs-à-plusieurs comme suit.

- $OPT(MP_i; MO_j)$ représente le noeud d'interception optimale de MOj par MPi ;
- $It(MO_j)$ représente l'itinéraire deMO_j ;
- $It(MO_j; w)$ avec $w \in N_t$ représente l'itinéraire partiel de MOj jusqu'au noeud w ;
- $InitM_{oj}$ représente le noeud initial de MO_j ;
- $DestMO_j$ représente le noeud final de MO_j ;
- $InitMP_i$ représente le noeud initial de MP_i.

Hypothèse 4 :
Chaque MP ne peut poursuivre qu'un seul MO et chaque MO ne peut être poursuivi que par un seul MP.

FIGURE 3.4 – Exemple avec plusieurs mobiles objectifs et plusieurs mobiles poursuivants

Définition 3.2.1
La fonction de coût $PairC$ est une fonction attribuant un coût à la paire $(MP_i; MO_j)$ et représente le coût de la poursuite de MO_j par MP_i.

Définition 3.2.2
Soit aff_m est une fonction de l'ensemble des MP vers l'ensemble des MO, telque elle affecte chaque MP à la poursuite d'un seul MO.
Si le nombre des MO est supérieur au nombre des MP, aff_m affecte le maximum de MP et laisse les MO restants sans affectation.

Définition 3.2.3
Soit $CG(aff_m)$ une fonction de coût global, résultant de l'affectation de l'ensemble des MP par aff_m. Elle est définie comme suit :

But : Trouver la fonction d'affectation optimale $affopt$ qui optimise CG.

Pour atteindre notre but, la définition de $PairC$ est un point important à traiter. En effet, cette fonction est l'élément fondamental pour l'évaluation d'une fonction d'affectation. Elle permet aussi de définir le critère d'optimisation dans le modèle. Par exemple, si le but est de maximiser uniquement le nombre d'interception, $PairC$ peut être définie comme suit :

$$PairC(MP_i, MO_j) = \begin{cases} 1 & \text{Si interception ;} \\ o & \text{Sinon.} \end{cases}$$

Avec une telle définition, CG doit être maximisée. Cependant, Si le but est de minimiser le temps d'interception moyen, PairC peut être définie comme suit :

$$PairC(MP_i, MO_j) = \begin{cases} c_o(It(MO_j, OPT(MP_i, MO_j))) & \text{Si interception ;} \\ \infty & \text{Sinon.} \end{cases}$$

Dans ce cas, CG doit *être* minimisée.

3.2.2 Algorithme d'affectation :

Une solution évidente peut consister à essayer toutes les paires possibles de MP et MO, sachant que la complexité d'une telle approche est exponentielle.[14] [15] En conséquence, cette méthode est rejetée.

1. **Cas** *où* **K=L** Dans ce cas, notre solution se présente en deux etapes. Dans la première étape, nous calculons $PairC(MP_i, MO_j) \forall i \in [1, k]$ et $\forall j \in [1, L]$ en utilisant l'algorithme de section 3.2, ensuite, les résultats sont représentés par une matrice carrée appelée matrice des coûts. Des exemples de cette matrice sont présentés dans la figure 3.5 et la figure 3.6.
   La figure 3.5 montre un exemple où le but est de maximiser uniquement le nombre d'interceptions réussies, et la figure 3.6 montre un exemple où le but est de minimiser le temps moyen d'interception.
   Ayant cette matrice, notre problème devient un Problème d'Affectation.
   La deuxième étape consiste à résoudre le probléme d'afectation des taches. en utilisant l'algorithme hongrois qui permet de résoudre le problème d'affectation en un temps polynomial.

FIGURE 3.5 – Premier exemple d'une matrice de coût où K = L

FIGURE 3.6 – Deuxième exemple d'une matrice de coût où K = L

2. **Cas** *où* **K** $\neq$ **L**

   Dans ce cas, une étape supplémentaire est nécessaire pour pouvoir utiliser l'algorithme du probléme d'afectation des taches. Cette étape consiste à ajouter des MP ou des MO virtuels pour obtenir le même nombre de MP et de MO et revenir ainsi au premier cas. Les mobiles ajoutés ne doivent en aucun cas influencer le résultat final de l'algorithme, c'est pourquoi, les MO virtuels sont des mobiles qui ne peuvent pas être interceptés et les MP virtuels sont des mobiles qui ne peuvent intercepter aucun MO.
   Les figures 3.7 et 3.8 présentent des exemples avec des MO virtuels et des MP virtuels.

3.2.3 Algorithme d'interception plusieur a plusieur

1. Pour i=1 à K
2. pour j=1 à L
3. Apliquer algorithme d'interception 1 à 1 pour (MP_i, MO_J)
4. Si resultat =succeé alors
5. a_{ij}= C_0 $(it(M_{oj}, Opt(MP_i, MO_j)$
6. Sinon
7. $a_{ij} = \infty$
8. Fin Pour
9. Fin Pour
10. Apliquer l'algorithme hongroie a la matrice A

3.2.4 Complexité de l'algorithme

Cas *où* **K=L** Dans la première étape, l'algorithme d'interception est exécuté pour chaque paire $(MP_i; MO_j)$. La complexité de cette étape est

$$0(K.L.(|X_0| + |U_p| + |X_p|.ln(X_p))) = 0(K^2.(|X_0| + |U_p| + |X_p|.ln(|X_p|))) \tag{3.4}$$

Pour la deuxième étape, le probléme dafectation des taches peut être résolu avec l'algorithme de Hangrois. La complexité de cet algorithme est

$$\circ\,(K^3) \tag{3.5}$$

Donc la complexité globale est

$$0(K^2.(|X_0| + |U_p| + |X_p|.ln(|X_p|)) + K^3) \tag{3.6}$$

FIGURE 3.7 – Exemple d'une matrice de coût avec un MO virtuel

FIGURE 3.8 – Exemple d'une matrice de coût avec un MP virtuel

Cas *où* $\mathbf{K} \neq \mathbf{L}$ Pour ce cas, les seuls opérations supplémentaires sont l'ajout des MO ou MP virtuels. Ces opérations s'exécutent en $\circ(L)$ ou $\circ(K)$. Par conséquent, la complexité en fonction du nombre des noeuds reste

$$0(|U_p| + |X_p|.ln(|X_p|)) \tag{3.7}$$

et la complexité en fonction du nombre de MP et MO reste $0(K^3)$.

3.2.5 Conclusion

L'algorithme de référence et l'algorithme d'interception un-à-un ont la même complexité dans le pire des cas. Pour cela une analyse empirique des deux algorithmes est nécessaire pour determiner lequel des deux est meilleur.

Chapitre 4

Implementation

Dans ce chapitre, nous implémentons l'algorithme de référence et l'algorithme d'interception un-a-un puis nous allons faire une comparaison entre le temps d'execution de chacun sur les mêmes intances du probleme d'interception.

4.0.6 Univers de travail :

Matlab (Matrix Laboratory)
Matlab dispose d'un langage de programmation basé essentiellement sur le calcul matriciel, avec des fonctionnalités mathématiques et graphiques étendues, ce qui correspond parfaitement à notre cas, puisque l'algorithme que nous présentant manipule essentiellement des graphes et des matrices.

4.0.7 Exemple d'application :

Il s'agit de visualiser les résultats obtenus avec l'exécution des deux algorithmes sur l'exemple dans la figure suivante :.

FIGURE 4.1 – Itenéraire du mobile

Les paramaitres d'entré sont :
it_{MO}=[1,10,2,6,9] : L'itenéraire du mobile objectife.
$init_{MP}$: le point initiale du mobile poursuivant.

d_p : La matrice d'adjacence pondérée du graphe G_p.

$$d_p = \begin{pmatrix} 0 & 9 & inf & 5 & inf & inf & inf & inf & inf & inf \\ 9 & 0 & 5 & inf & 7 & inf & inf & inf & inf & inf \\ inf & 5 & 0 & inf & inf & 1 & inf & inf & inf & inf \\ 5 & inf & inf & 0 & 3 & inf & 4 & inf & inf & inf \\ inf & 7 & inf & 3 & 0 & 5 & 0 & 3 & inf & inf \\ inf & inf & 1 & inf & 5 & 0 & inf & inf & inf & inf \\ inf & inf & inf & 4 & inf & inf & 0 & 5 & inf & inf \\ inf & inf & inf & inf & 3 & inf & 5 & 0 & 2 & inf \\ inf & inf & inf & inf & inf & 5 & inf & 2 & 0 & inf \\ inf & inf & inf & inf & inf & inf & inf & inf & inf & 0 \end{pmatrix}$$

d_o : La matrice d'adjacence penderée du graphe G_o

$$d_o = \begin{pmatrix} 0 & inf & inf & 5 & inf & inf & inf & inf & inf & 2 \\ inf & 0 & inf & inf & 7 & 3 & inf & inf & inf & 2 \\ inf & inf & 0 & inf & inf & inf & inf & inf & inf & inf \\ 5 & inf & inf & 0 & 3 & inf & 2 & inf & inf & inf \\ inf & 7 & inf & 3 & 0 & inf & inf & 3 & inf & inf \\ inf & 3 & inf & inf & inf & 0 & inf & inf & 5 & inf \\ inf & inf & inf & 2 & inf & inf & 0 & 2 & inf & inf \\ inf & inf & inf & inf & 3 & inf & 2 & 0 & 2 & inf \\ inf & inf & inf & inf & inf & 5 & inf & 2 & 0 & inf \\ 2 & 2 & inf & inf & inf & inf & inf & inf & inf & 0 \end{pmatrix}$$

A la sortie si l'interception est possible on récupère le point d'interception optimale du véhicule objectif par le véhicule poursuivant ainsi que le plus court chemin du Initiale MP vers le point d'interception optimale. Si l'interception est impossible, un message qu'il l'indique sera afficher a l'interface.

4.0.8 Procedure de teste

Pour les tests plusieurs graphe sont générés aléatoirement. mais pour que la differance entre les deux algorithme soit remarquable il faudrai travailler sur des graphes ayant un nombre de sommet asser grand.

4.0.9 La méthode waxmane

Waxmane [17] a proposé une méthode pour générer aléatoirement des graphes cette méthode nécessite deux étapes. La premiere consiste a distri-

buer N noeuds dans un espace rectangulaire. Chaque noeud est placé dans une position avec des coordonnées entieres. La deuxiéme étape consiste à crées des arcs entre les différents noeuds. Soint deux noeuds p et q, d la distance euclidienne maximale entre deux noeuds du graphe.
Un arc est crée entre p et q suivant la loi de probabilité.

$$P = \beta cxp(\frac{-d}{L\alpha})$$

α et β sont des paramétres entre 0 et 1 pérmettant d'inffluer sur la probabilité de création des arcs. Plus la valeur de β augmente, plus la densité des arcs augmente. Plus la densité des arcs courts augmente par rapport à celle des arcs longs.

Pour les tests un graphe est généré de la maniere suivante. La structure du graphe est générée grâce à la méthode Waxman avec α=0,5 et β=0,5. Puis, un poids aléatoire est assigné à chacun de ses arcs. Pour chaque graphe, plusieurs tests sont réalisés en générant alétoirement, à chaque fois $Init_{MP}$ et $It(MO)$. Tous les tests sont réalisés sur un PC équipé d'un proceseur Intel Core Due de fréquence 1,83 Ghz. Le temps d'éxucution moyen pour chaque graphe est donné dans le tableau. L'emsemble des résultat est représenté dans le tableau.

Nombre de noeuds	Algorithme d'interception	Algorithme de référence
50	0,4	0,3
100	0,6	1,0
150	0,9	2,0
200	1,1	3,1
250	1,5	3,9
300	2,3	5,7
350	2,2	6,7
400	2,5	9,6
450	3,0	11,9
500	4,1	14,5

FIGURE 4.2 – evaluation de temps d'execution moyenne de l'algorithme d'interception et de l'algorithme de référence en fonction du nombre de noeud

4.0.10 Inteprétation des résultats

D'après la figure , nous remarquons d'abord que notre algorithme d'interception est plus rapide que l'algorithme de réfference et que la différence

dans le temps d'éxucution devient plus inportante à mesure que la taille du graphe devient plus importante. Par conséquent, notre algorithme apporte une réelle améllioration en termes de temps d'exécution. De plus, l'algorithme d'interception est très rapide est peut être utilisé dans des applications à fortes contraintes de temps d'excution. Effectivement, il ne nécessite que quelques secondes pour s'exécuter même pour des graphes de taille importante.

Conclusion

Dans ce memoire le problème de l'interception d'un mobile dans un graphe statique déterministe avec des poids positif a été formalisé pour deux cas de figures : un-à-un, plusieurs-à-plusieurs. Pour chaque cas, un algorithme de résolution a été proposé et l'optimalité de la solution retournée a été prouvée. De plus, l'efficaté des algorithmes en termes de temps de calcul a été vérifiée.
Les algorithmes développés peuvent parfaitement être utilisés dans un contexte industriel vu leur simplicité et leur efficacité en termes de calul. Non seulement cette algorithme permet de localiser le point d'interception optimal mais il donne aussi le plus court chemin vers ce dernier. Par conséquent, ils peuvent apporter une réponse concréte au problème de la gestion des interventions sur les bus en mouvement.

Bibliographie

[1] C. BERGE. *Graphes et hypergraphes*. FRANCE, 1970.

[2] P.H.D. Mohamed Mejdi HIZEM. *PhD Title*. PhD thesis, UniversitÃ© l'École Centrale de Lille, 2008.

[3] H W. KUHN. The hungarian method for the assignment and transportation problems. Naval Research Logistics Quarterly 2. 1955.

[4] J. MUNKRES. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics 5,*, pages 32–38, 1957.