République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي والبحث العلمي
Université Chadli Bendjedid
جامعة الشاذلي بن جديد
Faculté des Sciences et de la Technologie
كلية العلوم و التكنولوجيا
Département d'Informatique
قسم الاعلام الالي

MEMOIRE

Présentée par

Nasri Zineb

Pour l'obtention de diplôme de

MASTER

Filière : Informatique

Spécialité : Systèmes Informatiques Intelligents

Thème

RECHERCHE DU PLUS COURT CHEMIN DANS LES RESEAUX TEMPORELMULTIMODAUX

Soutenue le : 07/2021 Devant le Jury composé de :

Qualité	Nom et Prénom	Grade	Université
Président	Mme. FARROUM A	MCB	Chadli Bendjedid El-Tarf
Rapporteur	**Mme. Makhlouf Amina**	**MCB**	**Chadli Bendjedid El-Tarf**
Examinateur	Mr. CHEMAME C	MAA	Chadli Bendjedid El-Tarf

Année Universitaire : 2020/2021

Remerciements

Avec des sentiments profonds "Nous remercions d'abord ***« Dieu »****qui nous a aidés à faire ce travail, respect et amour*

A mes parents que j'aime tant pour leur présence et

Soutenez-les pour leur dévouement mais surtout pour leur bien

Crois-moi.

A ma chère soeur Imen.

A mon frère bien-aimé Mohamed que j'admire beaucoup, et à mon jeune frère Chiheb Eddin.

Pour tous les membres de ma famille, oncles, tantes et cousins (Buthaina, Abir, Marwa, Khawla, Linda, Nadjat)

Pour les encourager.

Pour la famille Nasser.

A tous mes amis (Kholoud, Sarah, Hana et Ikram, Kholoud, Khalida)à eux

Encouragements particuliers, soutien et dévouement

A tous ceux que j'aime et ceux qui m'ont aidé.

A tous ceux qui me sont chers. Il faut mentionner son aide et sa patience dans l'élaboration de ce travail, Dr Makhlouf Amina Merci beaucoup pour son soutien.

NASRI ZINEB

Dédicace

Je dédie ce travail à :

Spécialement à mon père Ma mère qui m'a très encouragé

Toutes mes sœurs

Tous mes frères

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

PCC	Plus Court Chemin
A*	Astar
Matlab	MATrixLABoratory
PC	Personnel Computer
RAM	Randomaccess memory
BFS	Breadth First Search
BOF	Bacterial Foraging Optimisation

Introduction Générale

De nos jours, les réseaux de transport au sein d'une ville sont de plus en plus complexes.

1. Contexte du projet et problématique

Un voyageur commutateur se voit confronté à déterminer un trajet optimal combinant plusieurs modes de transport publics et/ou privés de disponibilité différentes. C'est l'une des plus intéressantes applications du problème du plus court chemin en raison de l'intérêt croissant pour la gestion dynamique des systèmes de transport complexes. Ainsi, nombreuses applications algorithmiques calculant le chemin optimal d'une source à une destination dans ce type de réseau, ont été proposées pour accompagner les voyageurs dans leur planification d'itinéraire. Nous étudions dans cet article, le problème du plus court chemin dans un réseau multimodal dont les données de disponibilité des moyens de transport changent dans le temps à cause des congestions, des retards et des tableaux d'horaire pour le transport public. A cet effet, nous discuterons les techniques de modélisation du réseau multimodal pour ensuite étudier le modèle dynamique. Propose une approche optimale pour résoudre le problème de chemin le plus court dans le réseau multimodal.

2. Objectifs

Il est courant, lorsque l'on cherche à se rendre d'un point à un autre dans un réseau (routier, par exemple), de chercher le plus court chemin en anglais "shortestpath", c'est-à-dire celui dont la distance est la plus petite. Si le nombre de trajets possibles entre le point de départ et le point d'arrivée est faible, il suffira de calculer les longueurs de chacun des trajets en additionnant la longueur des liens qui le composent et de comparer directement les longueurs obtenues. Mais une telle solution exhaustive devient rapidement impraticable si le nombre de trajets possibles est grand. Heureusement, il existe des algorithmes qui évitent d'avoir à calculer tous les trajets possibles. Pour cela, ils mettent en œuvre diverses stratégies.

3. Contenu du mémoire

Dans notre travail, nous allons étudier le problème de recherche du plus court chemin, notre mémoire comprendra une introduction générale, trois chapitres, et une conclusion générale.

Des Notions générales sur l'intelligence artificielle, des rappels sur la théorie des graphes, ainsi une présentation des différents algorithmes de recherche du plus court chemin sont introduites dans le premier chapitre.

Le deuxième chapitre présentera la conception de notre système, quelques résultats, et discussion des résultats.

En troisième chapitre, les étapes d'implémentation des méthodes utilisées sont résumées en cadre d'une application sous Matlab. Nous terminons avec une conclusion générale clôturera la comparaison entre les méthodes implémentées.

1. Introduction

Dans ce chapitre, nous fournirons une définition de l'intelligence artificielle et de Le chemin le plus court dans les réseaux multimodaux temporel, quel est le problème auquel les Voyageurs sont confrontés et quelles sont les méthodes possibles pour résoudre le problème.

2.L'INTELLIGENCE ARTIFICIELLE

L'intelligence artificielle (IA) renvoie à "une application capable de traiter des tâches qui sont, pour l'heure, réalisées de manière plus satisfaisante par des êtres humains dans la mesure où elles impliquent des processus mentaux de haut niveau comme l'apprentissage perceptuel, l'organisation de la mémoire et l'esprit critique". C'est ainsi que le scientifique américain Marvin Lee Minsky, considéré comme le père de l'IA, définit ce concept. C'est en 1956 à l'occasion d'une rencontre de scientifiques à Dartmouth (au sud de Boston) organisée en vue d'envisager la création de machines pensantes qu'il parvient à convaincre son auditoire d'accepter le terme [1].

Figure.1.1: Image traverser l'intelligence artificielle

- ✓ consiste à mettre en œuvre un certain nombre de techniques visant à permettre aux machines d'imiter une forme d'intelligence réelle
- ✓ Le but de l'IA: simulation le raisonnement humain.

3. LE PCC

Dans un graphe value, le poids c(p) d'un chemin p est la somme des poids des arêtes (arcs) le long du chemin. Dans ce qui suit nous appellerons le poids d'un chemin sa longueur.
Le PCC entre deux sommets s et t est alors défini comme le chemin de plus Faible poids reliant s et t. **[3]**

Remarque : Pour différencier la longueur d'un chemin telle que nous l'avons définie pour un graphe non value, nous préciserons qu'il s'agit de la longueur en nombre d'arêtes.

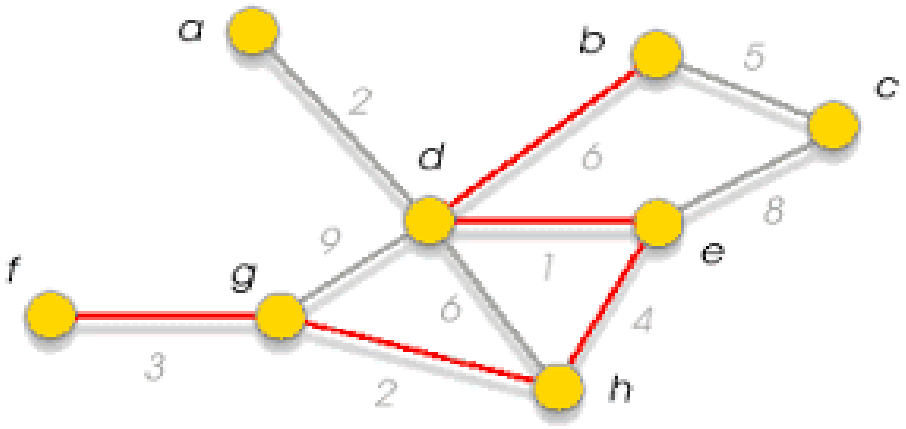


Figure.1.2 : exemple d'un graphe

Cette confusion des deux définitions de la longueur se justifie par le fait que la longueur en nombre d'arêtes (arcs) correspond au poids d'une chaine (chemin) avec une évaluation de 1 pour toutes les arêtes (arcs).

Dans le graphe value ci-contre, p = (f, g, h, e, d, b) a une longueur de 16. Il est le plus court chemin reliant les sommets f et b.

4. LE PROBLEME DE PCC

En théorie des graphes, le problème de PCC est le problème algorithmique qui consiste à trouver un chemin d'un sommet à un autre de façon que la somme des poids des arcs de ce chemin soit minimale.Les problèmes de cheminement dans les graphes sont parmi les plus anciens de la théorie des graphes.

Ce type de problèmes se rencontre soit directement, soit comme sous problème dansde nombreuses applications.

Exemples :

- les problèmes de tournées, de gestion de stock, d'investissements
- les problèmes d'optimisation de réseaux.

5. LES ALGORITHME DU RECHERCHE DE PCC

Il existe de nombreux algorithmes qui aident à trouver le PCC dans plusieurs réseaux temporels et parmi ces algorithmes, nous en présentons certains :

5.1. Algorithme de DIJKSTRA :

L'algorithme de **Dijkstra** est sans doute l'un des algorithmes de plus court chemin le plus connu. Ce dernier se base sur le parcours en largeur afin de trouver le plus court chemin dans un graphe pondéré **positivement** (ce point est très important, mais nous y reviendrons plus en détails après). On sait qu'un parcours en largeur permet de trouver le plus court chemin sur un graphe non pondéré (où on associe chaque arc avec une longueur arbitraire de 1 unité) et qu'il utilise une file pour stocker les éléments afin de les parcourir profondeur par profondeur. L'algorithme de**Dijkstra** s'appuie sur la même idée, sauf qu'il ne va pas utiliser une simple file (car tous les arcs n'ont pas la même pondération) mais une file à priorité. Cette dernière permet de toujours avoir le nœud avec la distance minimale par rapport au départ en tête de file, et garantie alors qu'on a trouvé le plus court chemin lorsqu'on atteint le nœud de sortie. La file à priorité a aussi l'avantage d'être une structure de données très efficace et rapide, ce qui rend notre algorithme d'autant plus intéressant.[4]

Exemple :Soit le graphe suivant [15]:

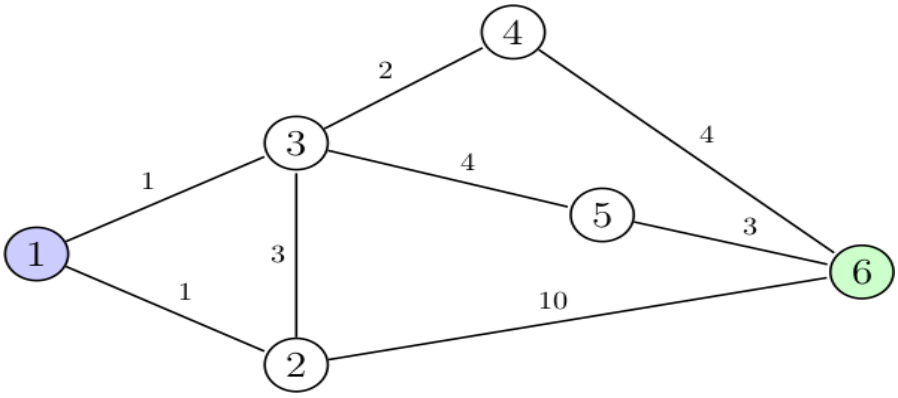


Figure.1.3 : Exemple de graphe pondéré positivement

On souhaite trouver le plus court chemin pour aller du nœud de départ (en bleu) au nœud d'arrivée (en vert).

L'algorithme de Dijkstra commence par le nœud de départ (1), et examine ses voisins :

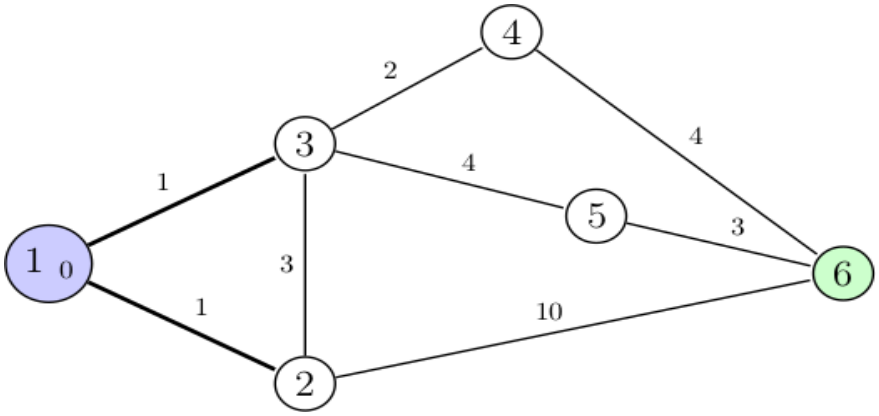


Figure.1.4 : Tour 1 de l'algorithmede Dijkstra

Le nœud en bleu représente désormais le nœud en tête de notre file à priorité, et j'ai aussi indiqué la distance parcourue depuis le nœud initial en indice. Le nœud 1 possède deux arcs qui représentent donc deux possibilités pour le moment. L'algorithme prend en compte la distance depuis le nœud de départ ainsi que le poids des arcs pour choisir le prochain nœud à visiter. Les différentes combinaisons actuellement sont donc : 0 + 1 (nœud 1 vers nœud 2) et 0 + 1 (1 vers 3). Dans notre cas les deux nœuds sont équivalents en termes d'efficacité, mais notre implémentation doit bien choisir lequel parcourir et nous imaginerons que l'on visite le nœud 2 (le nœud 3 menant directement au plus court chemin, il est plus intéressant de voir comment notre algorithme va rectifier son tir). Notre file à priorité contient donc les nœuds 2 et 3 (le 1 a été retiré), mais le 2 apparait avant c'est donc celui-ci qu'on va visiter au prochain tour :

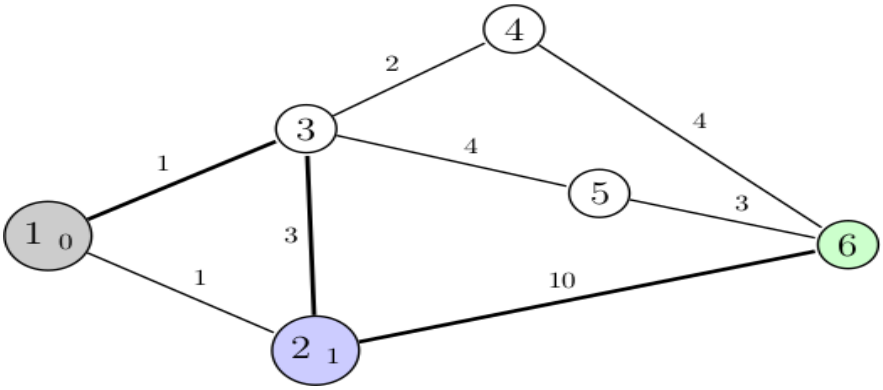


Figure.1.5 : Tour 2de l'algorithme de Dijkstra

Notre algorithme nous mène au nœud 2, avec comme distance totale 1. Le point très important ici, est que lors du dernier tour, on avait deux choix possibles d'arcs à emprunter, en choisissant l'un des deux on n'oublie pas de conserver l'autre au cas où justement ce n'est pas le chemin le plus optimal qu'on vient de prendre. Et la file à priorité est le point essentiel car cette structure se chargera de faire remonter automatiquement le nœud 3 en tête si ce dernier devient finalement un choix plus intéressant. Lorsqu'on ajoute les voisins du nœud 2 à notre file, on doit choisir entre : 0 + 1 (nœud 1 vers 3), 1 + 3 (2 vers 3), ou 1 + 10 (2 vers 6). On voit que le minimum est atteint pour 0 + 1, soit le nœud 1 vers le 3. Notre file à priorité ne possède donc plus le nœud 2, et a en tête le nœud 3 qu'on parcourt lors du prochain tour :

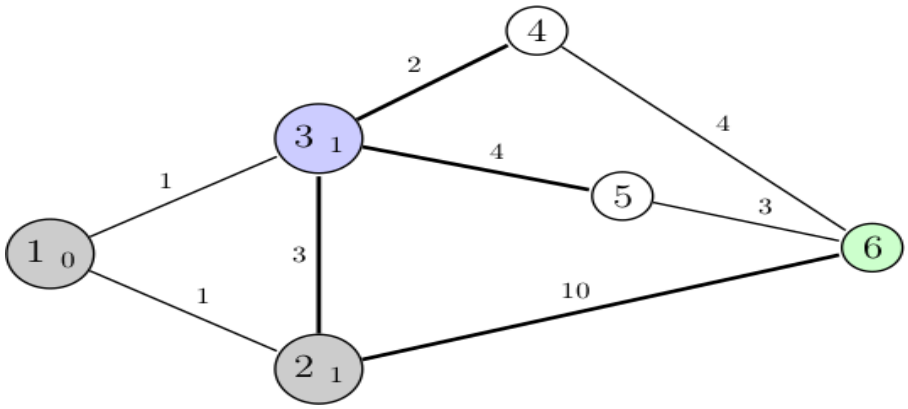


Figure.1.6 : Tour 3de l'algorithme de Dijkstra

Nous sommes au nœud 3 avec une distance totale depuis le nœud de départ de 1, comme d'habitude on va chercher à comparer nos anciennes possibilités à celles qui se sont rajoutées (les voisins de 3). L'algorithme doit donc choisir entre : 1 + 2 (nœud 3 vers 4), 1 + 4 (3 vers 5), 1 + 3 (2 vers 3), et 1 + 10 (2 vers 6). Le choix avec une distance minimale est donc le premier : allant du nœud 3 vers le nœud 4. On continue l'algorithme :

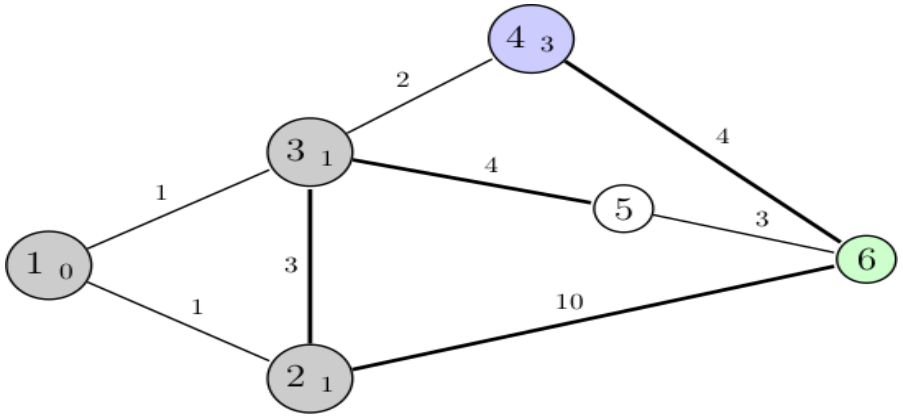


Figure.1.7 : Tour 4de l'algorithme de Dijkstra

On se retrouve avec un nouvel arc disponible, et on va comparer de nouveau les différents chemins possibles : 3 + 4 (nœud 4 vers 6), 1 + 4 (3 vers 5), 1 + 3 (2 vers 3), et 1 + 10 (2 vers 6). L'algorithme choisit donc le chemin 2 vers 3 :

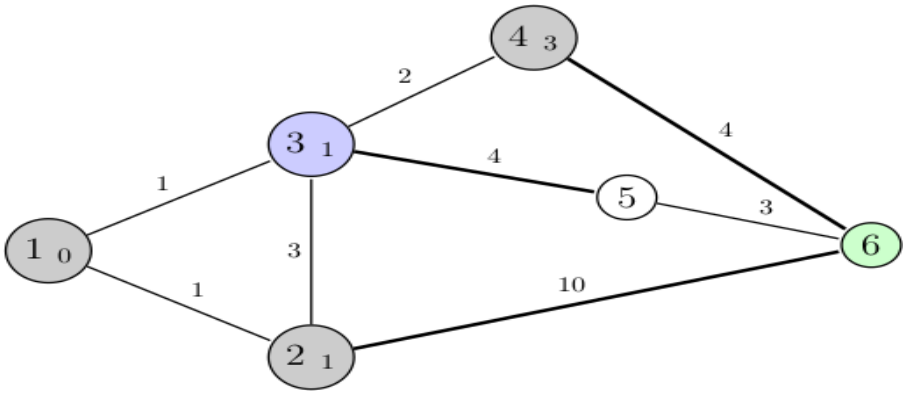


Figure.1.8 : Tour 5de l'algorithme de Dijkstra

Cependant lorsqu'on arrive sur le nœud 3, on s'aperçoit qu'on a déjà visité ce nœud (il est donc tout à fait inutile de recommencer cette opération), et l'algorithme va immédiatement choisir un autre chemin entre : 3 + 4 (nœud 4 vers 6), 1 + 4 (3 vers 5), et 1 + 10 (2 vers 6). Le chemin choisi est donc celui reliant le nœud 3 au 5 :

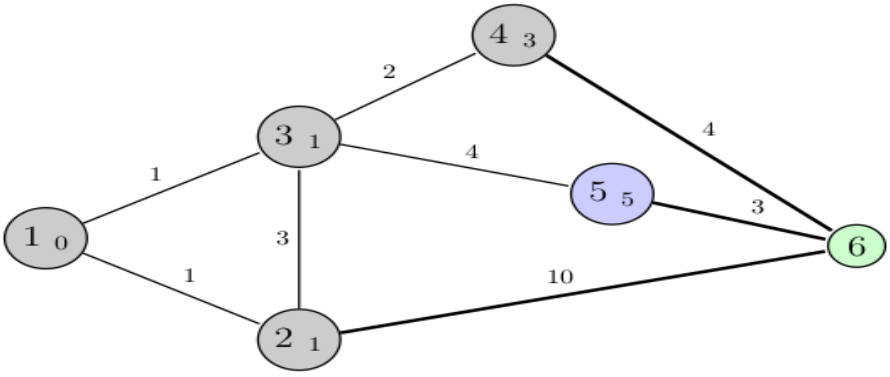


Figure.1.9 : Tour 6de l'algorithme de Dijkstra

Comme auparavant, on prend en compte le nouvel arc possible et on choisit entre : 3 + 4 (nœud 4 vers 6), 5 + 3 (5 vers 6) et 1 + 10 (2 vers 6). Le premier choix étant celui avec le plus petit résultat, c'est ce qu'on décide de réaliser :

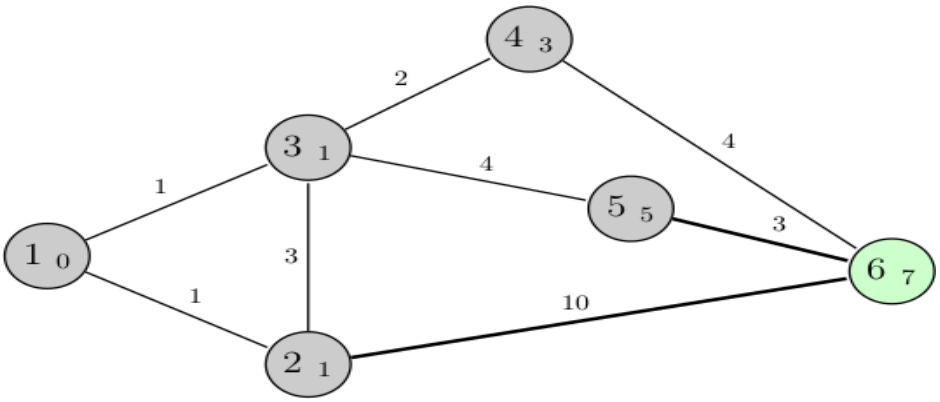


Figure.1.10 : Tour 7de l'algorithme de Dijkstra

On arrive au nœud vert (celui d'arrivée), notre algorithme a donc terminé et on a trouvé le plus court chemin pour s'y rendre (les autres possibilités restantes étant forcément des chemins avec une plus longue distance).

Cet exemple peut prêter à confusion car finalement on a visité tous les nœuds du graphe ainsi que la plupart des arcs, mais j'ai choisi ceci pour justement montrer au mieux le fonctionnement de l'algorithme et surtout comment ce dernier réalise son choix pour trouver le chemin optimal en termes de distance. Sur d'énormes graphes, l'algorithme de Dijkstra est très utile car il ne visitera que ce dont il a réellement besoin puisque si on réfléchit à sa manière de fonctionner, on ne peut pas savoir à l'avance si le chemin qu'il emprunte actuellement restera optimal jusqu'au bout (d'où la nécessité de conserver les différents choix rencontrés lors du parcours), et inversement il se peut que les autres chemins soient en réalité pires et qu'il faut donc bien garder celui actuel [17].

- **Les avantages :**

  Avec cet algorithme, nous pouvons nous arrêter dès que le point d'arrivé devient lepoint à traiter puisque le chemin que nous avons construit jusque-là est un chemin minimal.

Le système de marque empêche l'algorithme de revenir en arrière. Il améliore ainsi sa performance.

- **Les inconvénients :**

Avec cette version, nous devons réitérer l'algorithme pour chaque véhicule présent sur la carte, ce qui le rend gourmand en temps de calcul. Nous pensons que si le nombre de point est important et la carte à traiter est relativement petite, nous pourrons faire Fonctionner l'algorithme sur toute la carte et obtenir ainsi l'ensemble des chemins optimaux entre deux points quelconques. Ensuite, il nous reste plus qu'à reconstituer le cheminoptimal pour chaque point.

Cet algorithme est de type glouton. C'est pourquoi dans un certain nombre de cas où les cases à traiter ont un même niveau de trafic l'algorithme n'est pas très performant car il les parcourt toutes. C'est pareil si nous prenons un autre critère de tri comme la distance entre la case à traiter et la case d'arrivée. Nous verrons plus loin comment on peut guider cet algorithme dans la recherche afin d'éviter de parcourir toutes les casesinutilement.

Puisque chaque nœud traité est marqué, nous ne pouvons pas revenir en arrière.

De ce fait, l'algorithme de Dijkstra est inadapté quand nous avons des arcs de poids négatif. Dans ce genre de situation, l'algorithme doit pouvoir revenir en arrière pourmodifier les étiquettes.

5.2. Algorithme de BELLMEN-FORD :

L'**algorithme de Bellman-Ford**, aussi appelé **algorithme de Bellman–Ford**, est un algorithme qui calcule des plus courts chemins depuis un sommet source donné dans un graphe orienté pondéré. Il porte le nom de ses inventeurs **Richard Bellman** et Lester Randolph Ford junior (publications en 1956 et 1958), et d'Edward Forrest Moore qui le redécouvrit en 1959.

Contrairement à l'**algorithme de Dijkstra**, l'algorithme de Bellman-Ford autorise la présence de certains arcs de poids négatif et permet de détecter l'existence d'un circuit absorbant, c'est-à-dire de poids total strictement négatif, accessible depuis le sommet source.

Cet algorithme est basé sur une approche de programmation dynamique, similaire dans sa structure de base à **l'algorithme Dijkstra**, mais au lieu de choisir le nœud de poids le plus bas qui n'a pas été étendu, il étend tous les bords et le fait | S-1 | Heure, où | s | C'est le nombre de sommets du tracé, la répétition permet d'obtenir la plus petite distance possible le long du tracé, en l'absence de cercles négatifs le chemin le plus court passe sur chaque nœud une fois au plus.[5]

5.3. Algorithme de JOHNSON :

L'algorithme de Johnson calcule les PCC pour tout couple de sommets du graphe. Il a une bonne complexité pour les graphes creux (peu d'arêtes en comparaison du nombre de sommets).

Il utilise les algorithmes de **Dijkstra** et de **Bellman-Ford,** et renvoie soit la matrice des poids des plus courts chemins, soit l'assertion que le graphe possède un circuit de poids négatif.

La technique utilisée consiste à se ramener au cas avec uniquement des poids positifs, puis de faire tourner l'algorithme de **Dijkstra** en partant de chaque sommet. Si les poids ne sont pas tous positifs, on les redéfinit pour pouvoir se ramener au premier cas.[13]

L'algorithme de Johnson est composé des étapes suivantes :

1. Ajouter un nouveau nœud *q* au graphe, et connecter ce nœud par des arcs de poids nul à tous les autres nœuds.
2. Utiliser **l'algorithme de Bellman-Ford** en partant du nouveau nœud *q* pour déterminer, pour chaque sommet *v*, le poids minimum *h*(*v*) d'un chemin de *q* à *v*. Si on détecte un cycle négatif, l'algorithme s'arrête sur un échec.
3. Ré pondérer les arcs du graphe initial en utilisant les valeurs calculées par l'algorithme de **Bellman-Ford** : un arc de *u* à *v*, dont l'ancien poids ou longueur est le nombre *w(u ,v)*, a pour nouvelle longueur le nombre non négatif *w(u ,v)* + *h(u)* − *h(v)*.
4. Enlever le sommet *q* et utiliser l'algorithme de **Dijkstra** pour déterminer les chemins les plus courts.

Exemple de l'algorithme de Johnson :

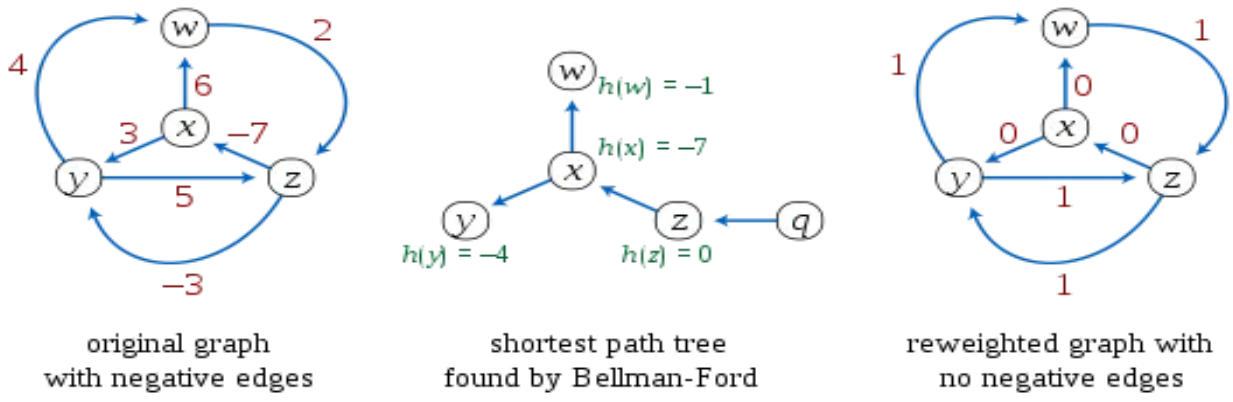


Figure .1.11 : Exemple de l'algorithme de johnson. [13]

5.4. La recherche du meilleur premier (BFS BREADTH FIRST SEARCH) :

L'algorithme travaille d'une façon semblable, sauf qu'il doit faire quelques évaluations (appelées heuristique). Au lieu de la sélection du sommet le plus proche au point de départ, il choisit le sommet le plus proche au but.[4]

- **Inconvénient :**
  - BFS ne garantit pas de trouver le PCC.
- **Avantage :**

Cet algorithme converge beaucoup plus rapidement que celui proposé par **Dijkstra.** Ceci est dû au fait qu'il emploie la fonction heuristique pour guider sa voie vers le but très rapidement.

Par exemple, si le but est au sud de la position de départ, l'algorithme BFS aura tendance à se concentrer sur des chemins orientés vers le sud. Dans le diagramme suivant, le jaune représente ces nœuds avec une haute valeur heuristique (coût élevé pour atteindre le but). Le noir représente, par contre, des nœuds avec une valeur heuristique basse (coût bas pour arriver au but). Il montre que l'algorithme BFS peut trouver des chemins très rapidement comparativement à l'algorithme de **Dijkstra.**

5.5. Algorithme FLOYD-WERSHALL:

L'algorithme de Floyd-Warshall permet de trouver le **PCC entre toute paire de sommets**. Il fait partie de la famille des algorithmes à correction d'étiquettes, mais comme il permet de calculer le plus court chemin pour tout couple (x, y) de sommets du graphe, les étiquettes ne sont plus un tableau (une étiquette par sommet), mais une matrice M de taille $N \times N$ où l'entrée $M_{i,j}$ correspond au plus court chemin entre les sommets i et j. Cet algorithme est **valable quelles que soient les évaluations des arcs**, y compris si cela implique des circuits négatifs (l'algorithme permet de prouver l'existence ou l'inexistence de tels circuits). L'algorithme est constitué de N itérations principales ; pour chaque itération k, on calcule les **plus courts chemins entre toute paire de sommets avec des sommets intermédiaires appartenant uniquement à l'ensemble** $\{1, 2, \ldots, k\}$. A l'initialisation, on calcule le plus court chemin entre toute paire de sommets n'ayant pas de sommets intermédiaires, donc il suffit de prendre la longueur des arcs qui existent et mettre un poids infini si l'arc n'existe pas. Par la suite, si on note $M^k_{i,j}$ la valeur du plus court chemin de i à j dont les seuls sommets intermédiaires sont dans l'ensemble $\{1, 2, \ldots, k\}$, alors on a l'égalité suivante [6]:

$$M_{i,j}^{k} = \min\left(M_{i,j}^{k-1}, M_{i,k}^{k-1} + M_{k,j}^{k-1}\right) \quad (1)$$

Exemple :

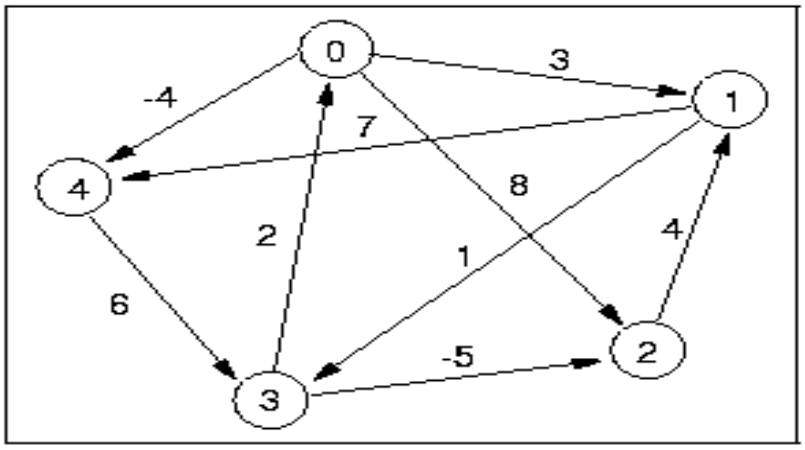


Figure.1.12 : un graphe valu par les couts des arcs

Voici sur l'exemple de **la figure 1.12**, les matrices d^ksuccessives, calculées d'après cette relation (on a utilisé ∞ pour indiquer la non-existence d'un arc) :

$$d^0 = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad d^1 = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d^2 = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad d^3 = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d^4 = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad d^5 = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

5.6. Algorithme A*:

L'algorithme de recherche A* (qui se prononce **A étoile**, ou ***A star*** à l'anglaise) est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final tous deux donnés. De par sa simplicité il est souvent exhibé comme un exemple typique d'algorithme de planification, un domaine de l'intelligence artificielle. **L'algorithme A*** a été créé pour que la première solution trouvée soit l'une des meilleures, c'est pourquoi il est célèbre dans des applications comme les jeux vidéo privilégiant le temps de calcul à l'exactitude des résultats.

L'algorithme A* est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final. Il utilise une évaluation heuristique sur chaque nœud pour estimer le meilleur chemin y passant, et visite ensuite les nœuds par ordre de cette évaluation heuristique. C'est un algorithme simple, ne nécessitant pas de prétraitement, et ne consommant que peu de mémoire. [7]

Fonction d'évaluation : [8]

$$f(n) = g(n) + h(n). \qquad (2)$$

g(n) est le cout de l'état initial à l'état n.

h(n) est le cout estime pour atteindre l'état final.

f(n) est le cout total estime pour aller de l'état initial à l'état final en passant par n

- **A*** utilise une heuristique admissible.
- **$h(n) \leq h^*(n)$** où **$h^*(n)$** est le cout réel pour aller de **n** jusqu'à l'état final. Une heuristique admissible ne surestime jamais le cout réel pour atteindre le but. Elle est optimiste
- Si h(n) = 0 pour tout n, alors **A*** est équivalent à l'algorithme de **Dijkstra** de calcul du plus court chemin.
- Théorème : **A* est optimale.**

Exemple de recherche PCC avec A* :

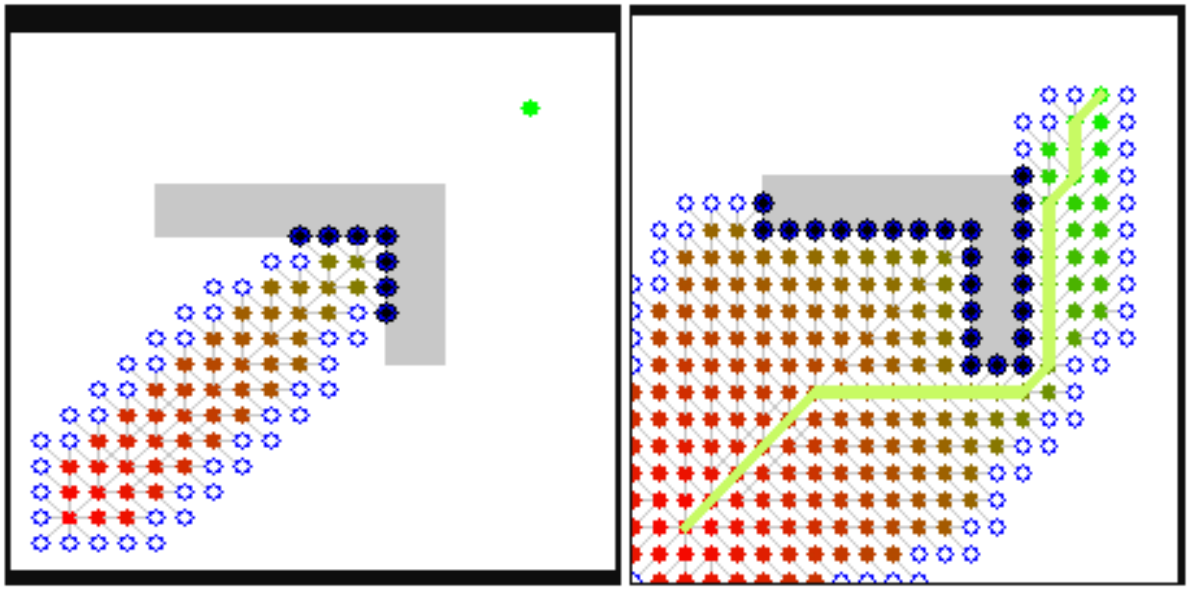


Figure.1.13 : Exemple de recherche le PCC avec **A*.**[10]

5.7. Algorithm Bacterial Foraging Optimisation(BFO):

L'algorithme d'optimisation de la recherche de nourriture bactérienne est inspiré du comportement de recherche de nourriture de groupe de bactéries telles que E. coli et M.xanthus. Plus précisément, l'algorithme d'optimisation de la recherche de nourriture bactérienne est inspiré du comportement de chimiotaxie des bactéries qui percevront les gradients chimiques dans l'environnement (tels que les nutriments) et se rapprocheront ou s'éloigneront de signaux spécifiques.[18]

Les bactéries perçoivent la direction de la nourriture en fonction des gradients de produits chimiques dans leur environnement. De même, les bactéries sécrètent des produits chimiques attirant et repoussant dans l'environnement et peuvent se percevoir de la même manière. À l'aide de mécanismes de locomotion (tels que les flagelles), les bactéries peuvent se déplacer dans leur environnement, parfois de manière chaotique (tourbillon et rotation) et d'autres fois de manière dirigée, ce que l'on peut appeler la nage. Les cellules bactériennes sont traitées comme des agents dans un environnement, utilisant leur perception de la nourriture et d'autres cellules comme motivation pour se déplacer, et le basculement stochastique et la nage comme un mouvement pour se déplacer. Selon les interactions cellule-cellule, les cellules peuvent envahir une source de nourriture et/ou peuvent se repousser ou s'ignorer de manière agressive.

La stratégie de traitement de l'information de l'algorithme est de permettre aux cellules d'essaimer stochastiquement et collectivement vers les optima. Ceci est réalisé grâce à une série de trois processus sur une population de cellules simulées :

1)Chimiotaxie où le coût des cellules est déclassé par la proximité d'autres cellules et les cellules se déplacent le long de la surface de coût manipulée une à la fois (la majorité du travail de l'algorithme).

2) Reproduction où seules les cellules qui ont bien fonctionné au cours de leur durée de vie peuvent contribuer à la prochaine génération.

3) Élimination-dispersion où les cellules sont rejetées et de nouveaux échantillons aléatoires sont insérés avec une faible probabilité. [16]

6.Conclusion

Dans ce chapitre, nous avons fourni une définition du chemin le plus court et des problèmes que nous avons rencontrés pour la recherche. Et nous avons présenté quelques méthodes qui nous aident à trouver des solutions appropriées à ce problème.

La partie qui suit, est consacrée à l'analyse et à la conception de notre système, qui consiste à implémenter un système qui sert à rechercher du plus court chemin dans les réseaux multimodaux temporel.

Chapitre 2 :ANALYSE ET CONCEPTION

1. Introduction

La réalisation d'un système suit généralement des méthodes qui conduisent à modéliser et construire des produits de manière fiable. C'est dans cet objectif que nous abordons dans ce chapitre l'architecture de notre système Liés à la recherche le plus court chemin dans les réseaux multimodaux temporel et les différentes étapes pour modéliser notre système en utilisant deux méthodes de recherche**dijkstra** et **A* modifié** qui sont plus efficaces dans la vitesse de recherche du chemin le plus court.

2. CAHIER DE CHARGE DE SYSTEME

2.1. CONTEXTE ET DEFINITION DU PROBLEME :

•Le problème du PCC consiste à déterminer lechemin de coût minimum reliant un nœud a à un nœud b.

- On peut le voir comme un problème de transbordement.
- Cependant, il est plus efficace d'utiliser des algorithmes spécialisés. **[9]**

Figure.2.1 : exemple d'une image map

***Problème :**

Soit :

•Un réseau G = (N, A).

• Un coût aij est associé à chaque arc (i, j) ∈ A:

• distance,

• temps de trajet,

• etc.

Soit un nœud appelé *origine*. Par convention, ce sera le nœud1.

Nous cherchons le chemin de coût minimum reliant le nœud 1 à n'importe quel autre nœud du réseau.

2.2. OBJECTIF DE SYSTEME :

L'objectif de notre système est utiliser des méthodes **(Dijekstra et AStar modifié)** pour faciliter la recherche de plus court chemin dans les réseaux multimodaux rapidement dans tempe court avec diminuant des points visiter.

3. ARCHITECTURE DE SYSTEME

Ce diagramme montre comment utiliser le système :

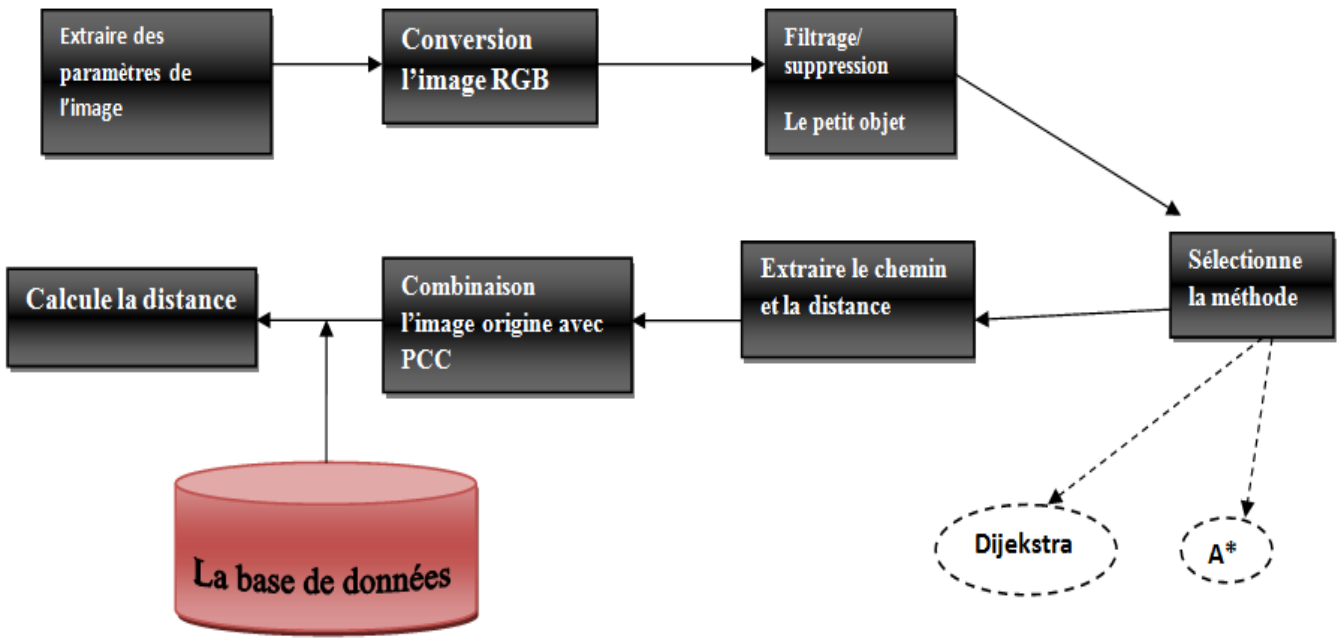


Figure.2.2 : Architecture de système.

4. DESCRIPTIONS DE SYSTEME

4.1. DESCRIPTION DE LA BASE DE DONNEE :

Nom	Description
La base de données :	Il contient des images de sites de Google Maps. À partir de chaque image, nous déterminons deux points (départ, destination), et pour chaque point ayant des coordonnées réelles dans la carte, nous les extrayons et prenons l'irréel correspondant du dessin et le mettons dans la base de données. Pour que nous puissions l'explorer là ou : **-Image name :** le nom de l'image qui est exporté. -'**lat of start' et 'long of start' :** sont les coordonnées réelles de point de départ. **- 'X start' et 'Y start' :** sont des coordonnées non réelles de point de départ prise en utilisant le logiciel « paint ». -'**destination lat' et 'destination long' :** sont des coordonnées réelles de point de destination. -'**X destination' et 'Y destination' :** sont des coordonnées non réelles de point de destination prises en utilisant le logiciel « paint ». La base de données de notre système va être utilisée automatiquement.

Tableau 2.1. : Description textuelle de la base de données

Exemple de la base de donnée :

I8 ▾ | Jx | 535

base de donnee.xlsx

	A	B	C	D	E	F	G	H	I
1	image name_lat of start_long of start_X start_Y start_destination lat_distination long_X destination_Y destination								
2	image 1,PNG	36,7655	8,32328	457	23	36,76205	8,3211	66	333
3	image2,PNG	36,76722	8,30738	56	348	36,76912	8,31006	307	128
4	image3,PNG	36,89563	8,44317	199	224	36,89764	8,44729	497	50
5	image4,PNG	36,7817	8,1935	182	290	36,78899	8,20419	432	77
6	image5,PNG	36,76061	8,32554	272	91	36,7572	8,32745	356	281
7	image6,PNG	36,79772	7,73317	189	34	36,7901	7,73244	155	480
8	image7,PNG	36,80945	7,74025	320	18	36,80414	7,74462	525	535
9	image8,PNG	36,85388	7,74307	267	109	36,84928	7,75015	595	370
10	image9,PNG	36,86445	7,77563	93	91	36,85917	7,77672	146	400

Figure.2.3 : Notre base de données

Exemple de **l'image4,PNG** sur map :

FIGURE.2.4 :IMAGE PRISE DE GOOGLE MAPS.

FIGURE.2.5: IMAGE SUR LOGICIEL PAINT

4.2. DESCRIPTION DES PARAMETRES PRINCIPALE DE NOTRE SYSTEME :

Nom	Description
Select point	***beggin** : nous marquons le point de départ en vert, à partir de là, le processus d'exploration des routes possibles pour atteindre l'objectif. ***destination** : nous définissons la destination en bleu, qui est l'objectif que nous voulons atteindre.
Show steps:	Grace à lui, toutes les étapes d'exploration sont affichées afin de trouver le chemin le plus court.
Use speed	la vitesse diminue dès qu'elle est exposée à des obstacles (les dodanes , les circulation , les travaux , etc…), qui l'empêchent d'atteindre l'endroit souhaité dans les plus brefs délais, et elle s'est exprimée dans notre système dans un dégradé de couleurs clair, donc plus la couleur est claire plus la vitesse est élevée et vice versa . **La vitesse** est calculée par la relation suivant : $\mathbf{V} = \frac{d}{t}$(3) **V** : la vitesse **d :** la distance **t :** le temps Dans notre système, lorsque la vitesse est utilisée pour calculer la distance du chemin le plus court, il peut y avoir, par exemple, deux chemins : dont l'un est long et n'a pas d'obstacles, et l'autre est court mais a des obstacles, nous nous concentrons ici sur temps et il est calculé par la relation suivante : $\mathbf{T} = \frac{d}{v}$(4)
Withe factor	c'est un factor qui donne de l'importance à la distance droite, car nous lui donnons une valeur supérieure à 1, donc chaque fois que nous augmentons sa valeur de 0.1, cela donne un bon résultat.
Resize image	Cela fonctionne pour réduire l'image afin que nous puissions travailler dessus et faciliter la recherche et l'exploration pour rechercher le chemin le plus court, et nous lui donnons une valeur de 0.5.
Real position	***Start latitude et Start longitude** : ce sont les coordonnés real du point de

	départ, extraites automatiquement de la base de données existant. *__destination latitude et destination longitude :__ce sont des coordonnés real du point de destination, il se trouve automatiquement de la base de données. __*calculatemapscal__ : l'échelle d'une carte est le rapport d'une distance sur le carte à la distance correspondant le sol. Il se trouve de la base de données automatiquement.

Tableau 2.2 : Description textuelle des paramètres.

4.3. DESCRIPTION DES METHODE DE RECHERCHE UTILISEE :

4.3.1. ALGORITHME Dijekstra :

Les travaux de l'algorithme de **Dijkstra**consistent en un parcours des nœuds dans un graphe et ce en commençant avec le point de départ de l'objet. L'algorithme de **Dijkstra**calcule le chemin le plus court entre deux nœuds d'un réseau. L'idée de base est de maintenir pour chaque ensemble de nœuds P le chemin le plus court ayant été trouvé. Chaque nœud extérieur à P doit être atteint à partir d'un nœud déjà dans P. Il s'étend ainsi à l'extérieur du point de départ avant qu'il n'accède le but.[4]

Le principe de l'algorithme :

- L'acteur part du point de départ puis calcule le poids de ses points adjacents ;
- Il examine le point dont le poids est le plus faible ;
- Il calcule, ensuite, le poids de ses points adjacents. Leurs poids sont éventuellement mis à jour (s'il est plus faible);
- Ensuite, il passe au point dont le poids est le plus faible parmi l'ensemble des points.
- Etc ,….

L'implémentation d'un tel algorithme s'effectue en utilisant une structure de file. Quand un point est placé dans la file, il l'est en fonction de son poids calculé. Le nouveau point à examiner est alors celui dont le poids et le plus faible, qui se trouve en tête de la file. Cette fois-ci, le poids de la cellule est pris en compte, mais, comme l'algorithme précédent, les directions sont équivalentes.

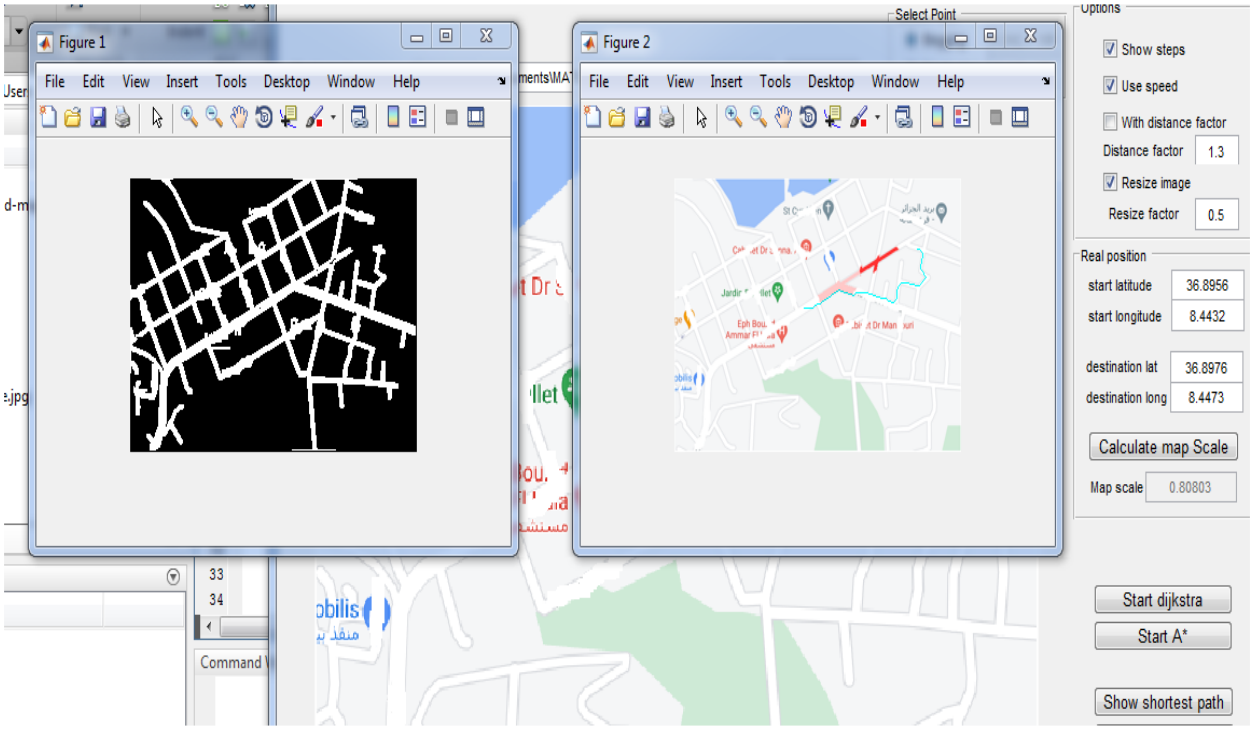


Figure.2.6: recherche **PCC** par DIJEKSTRA

4.3.2.LA METHODEAstar original (a*) :

L'algorithme de recherche A* (A star) est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final tous deux donnés. De par sa simplicité il est souvent présenté comme exemple typique d'algorithme utilisé en **intelligence artificielle**. L'algorithme A* a été créé pour que la première solution trouvée soit l'une des meilleures, c'est pourquoi il est célèbre dans des applications comme les jeux vidéo privilégiant la vitesse de calcul sur l'exactitude des résultats. Cet algorithme a été proposé pour la première fois par Peter E. Hart, Nils Nilsson et Bertram Raphael en 1968. Il s'agit d'une extension de l'algorithme de Dijkstra de 1959.[12]

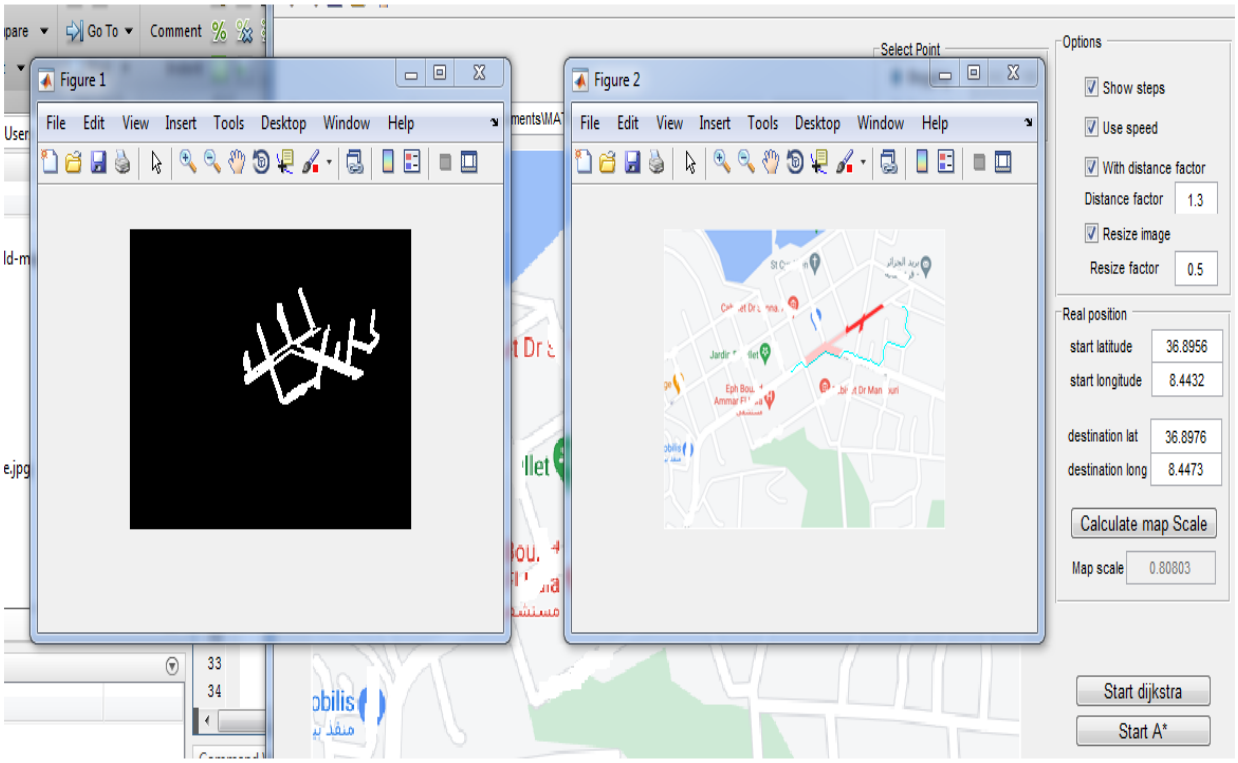


Figure.2.7 : recherche **PCC** par A*

A chaque itération, A* va tenter de prendre le chemin le plus court pour aller d'une position**Src** à une position **Dst**.

Pour déterminer si un nœud est susceptible de faire partie du chemin solution, il faut pouvoirquantifier sa qualité. Le plus souvent, on utilise la distance en "vol d'oiseau" entre la positiondu nœud et Dst, mais d'autres fonctions peuvent être utilisées.

On travaille à partir d'une position de départ, d'une arrivée, d'un terrain, et de deux listes de nœuds (ouverte et fermée), vides à l'origine.

1. A partir du départ, on regarde quels sont les nœuds voisins (correspondant aux libertés de mouvement – Par exemple, on peut aller à droite, à gauche, sauter ou ramper, reculer ou avancer – 6 possibilités = 6 nœuds). On place le nœud de départ dans la liste fermée.
2. Pour chaque nœud voisin :
   - Si c'est un obstacle, on passe (on ne traite pas ce nœud, il ne sera ajouté nulle part)
   - S'il est déjà dans la liste fermée, on passe (on a déjà étudié ce trajet)
   - S'il est déjà dans la liste ouvert, On vérifie sa qualité. Si la qualité actuelle est meilleure, c'est que le chemin est plus court en passant par le trajet actuel, donc on met à jour sa qualité dans la liste ouverte, et on met à jour son lien parent (avec nœud courant qui correspond à un meilleur chemin)
   - Sinon, on ajoute ce nœud à la liste ouverte avec comme parent le nœud courant

3. On parcourt la liste ouverte à la recherche du nœud ayant la meilleure qualité. Si la liste ouverte est vide, il n'y a pas de solution, fin de l'algorithme.
4. On prend le meilleur, et on le retire de la liste ouverte pour le mettre dans la liste fermée
5. Nœud courant = nœud qu'on vient d'insérer
6. Si nœud courant = Dst alors c'est bon, on a trouvé un bon chemin, sinon on réitère en 2.

La liste ouverte contient toutes les pistes de trajet qui sont encore possibles. La liste ferméecontient toutes les solutions étudiées, bonnes et mauvaises. Le "bon chemin" trouvé se retrouve en remontant le long des parents de nœuds.

Le secret lié au succès de **l'algorithme A*,** consiste dans le fait qu'il combine des morceaux d'information, les même que celles utilisées par l'algorithme de Dijkstra, ainsi que l'information exploitée par **l'algorithme BFS.**

Dans la terminologie standard employée dans un **algorithme A*:**

g (n): représente le coût du chemin du point de départ à n'importe quel sommet n.
h (n): représente le coût heuristique estimé du sommet n au but.
L'algorithme A* équilibre les deux tout en se déplaçant du point de départ au but. Achaque pas de la boucle principale, il examine le sommet n qui possède le coût le plus bas.

$$\mathbf{f(n) = g(n) + h(n)}(5)$$

4.3.3. LA METHODEa* modifié :

4.3.3.1 : AJOUTER UN FACTEUR :

La distance la plus proche entre deux points est la ligne droite, Et d'après l'être humain conscient, s'intéresse normalement à ce chemin, même sur des routes sinueuses (indirectes), car le chemin le plus droit ou le plus proche du montant est souvent le plus court. Et comme la méthode A* utilise cette distance, nous avons donné de l'importance à la distance rectiligne en définissant **un facteur** qui augmente l'importance de cette distance par rapport à la distance parcourue précédemment, et elle a obtenu de bons résultats en diminuant le nombre de points à visiter avant d'atteindre l'objectif.

	Nombres de points visiter						
	Avec factor	Image 1	Image 2	Image 3	Image 4	Image 5	Image 6
A* original	1	2097	1644	1690	585	882	309
A* modifier	1.1	1748	1501	1448	514	735	244
	1.2	1569	1393	1242	388	732	220
	1.3	1378	1278	1017	332	618	204
	1.4	1181	1064	852	273	608	188
dijekstra		5285	6429	6112	645	3166	2580

Tableau 2.3 : étude comparative qui montre l'influence de paramètre 'factor' sur lesméthodes utilisées.

*On observe que le paramètre 'factor' affecte clairement sur **a*** et aidez-nous à obtenir les meilleurs résultats, on observe également qu'il peut diminuer le nombre de points visités pendant la recherche.

4.3.3.2.La distance Euclidienne :

Si les unités peuvent se déplacer suivant un angle (au lieu des directions du réseau), on donc probablement employer une distance de ligne droite :

$$h(n) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (4)$$

Cependant, si c'est le cas, on peut avoir donc des difficultés pour l'utilisation directe de **l'algorithme A*** parce que la fonction du coût **g** ne correspondra plus à la fonction heuristique **h** [4].

De plus, pour cette méthode, la distance Euclidienne est plus courte que la distance de Manhattan ou la distance diagonale. On obtient toujours le chemin les plus courts, cependant **l'algorithme A*** prendra plus de temps à l'obtenir.

5. COMPARAISON DU METHODES APPROUVEE DANS LE TEMPS

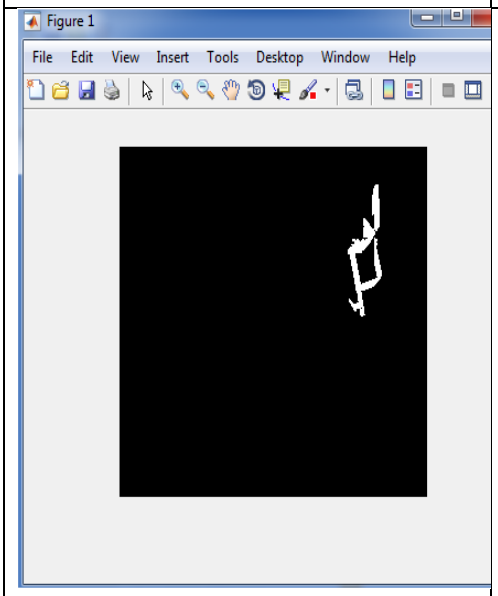
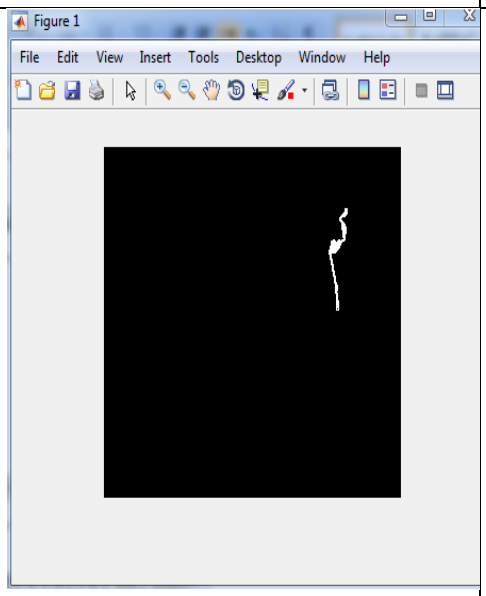
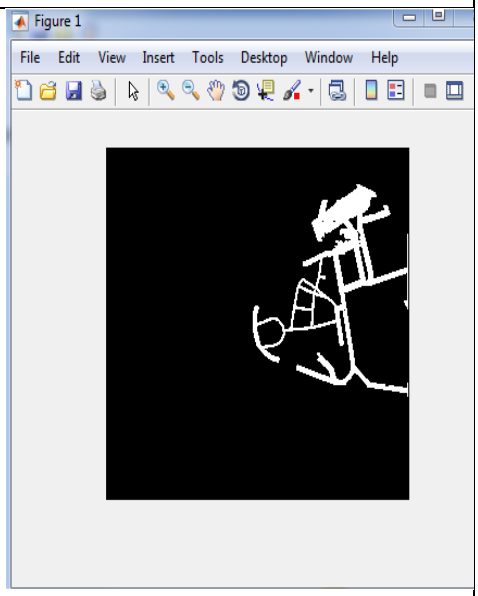
Les méthodes de recherche		
A*originale	A* modifiéé	Dijekstra
		
11.45 s	06 :07 s	38.47 s

Tableau 2.4 : comparaison de recherche entre les méthodes utilisées en termes de temps

Observation :

A travers ce tableau comparatif, nous observe que l'algorithme qui est le plus rapide pour trouver le chemin le plus court est a*modifié.

6. RESULTATS ET DECISION

- L'algorithme **a* modifié** donne une bonne solution en prendre en considérant le temps nécessaire pour arriver à la destination désirée.
- On pourrait comparer ses performances avec celles de l'algorithme de **dijekstra.**
- **Dijekstra** donne une meilleure solution. mais **a* modifié** est plus rapide.
- On peut bien conclure que l'algorithme**a*modifié**est plus fiable que **dijekstra**afin de faire la recherche du plus court chemin dans les réseaux multimodaux temporel.

7. CONCLUSION

Dans ce chapitre, nous avons présenté le principe du « pathfinding », et les divers algorithmes mis à notre disposition pour le gérer, pour la découverte des plus courts chemins et des distances dans des graphes plats qui exploitent la topologie particulière de l'entrée graphique. Nous avons vu aussi, les différentes étapes, puis les points forts et les points faibles de chacun des algorithmes. L'algorithme A* prend les avantages de l'algorithme BFS et de l'algorithme Dijkstra. Il est ainsi semblable à l'algorithme de Dijkstra dans lequel il peut être employé pour trouver le plus court chemin, et également semblable à l'algorithme BFS dans lequel il peut employer un heuristique pour se guider. Nous avons, par ailleurs, présenté l'heuristique liée à l'algorithme A*, et enfin les différentes variantes de l'algorithme A*.

Chapitre 3 : Implémentation et réalisation

1. Introduction

Après avoir réalisé une conception qui répondait bien aux besoins de l'application, nous entamons la partie réalisation de système que nous avons développée pour les plus court chemins, en exposons les différents outils et langages de développement utilisés lors de la réalisation et l'implémentation de la base de données ainsi qu'un aperçu sur les interfaces de notre système.

2. ENVIRONNEMENT DE TRAVAIL

La réalisation de ce projet a nécessité l'utilisation d'un nombre d'outils et de technologies cités ci-dessous.

2.1. ENVIRONNEMENT MATERIEL :

Afin de réaliser notre système, nous avons utilisé :

- PC portable avec les caractéristiques suivantes :
  - ✓ Processeur : Intel i3 CPU M370 @ 2.40 GH
  - ✓ RAM : 8GB DDR3
  - ✓ Disque dur : 500Go
  - ✓ Système d'exploitation : Windows 7

2.2. ENVIRONNEMENT LOGICIEL :

Le long de la phase de développement, nous avons utilisé l'environnement logiciel suivant :

2.2.1. MATLAB :

MATLAB est abréviation de MATrixLABoratory, est un logiciel commercial de calculinteractif. Il permet de réaliser des simulations numériques basées sur des algorithmesd'analyse numérique. Il peut donc être utilisé pour la résolution approchée d'équationsdifférentielles, d'équations aux dérivées partielles ou de systèmes linéaires, etc...

MATLAB peut être aussi considéré comme un langage de programmation adapté pour lesproblèmes scientifiques. **[14]**

Mode de fonctionnement :

MATLAB est un interpréteur, les instructions sont interprétées et exécutées ligne par ligne.
Il existe deux modes de fonctionnement :

- **Mode interactif :** MATLAB exécute les instructions au fur et à mesure qu'elles sont données par l'usager (fenêtre de commande).

- **Mode exécutif :** MATLAB exécute ligne par ligne un « fichier .M » (programme en langage MATLAB)

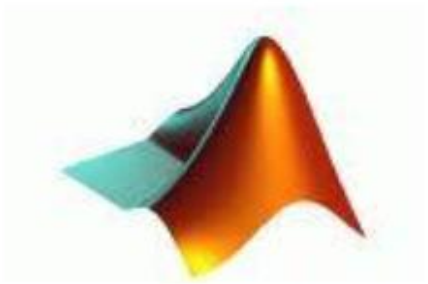


Figure.3.1: Logo de MATLAB

3. INTERFACE DE SYSTEME

Les interfaces graphiques de système sont très importantes, car elles permettent de faciliter le dialogue entre l'homme et la machine ainsi que d'améliorer les performances de système. Dans la conception des interfaces de notre système nous avons respecté un ensemble des choix ergonomiques comme la lisibilité, la compréhensibilité, etc. Dans ce qui suit une présentation des captures écrans des plus importantes interfaces de système.

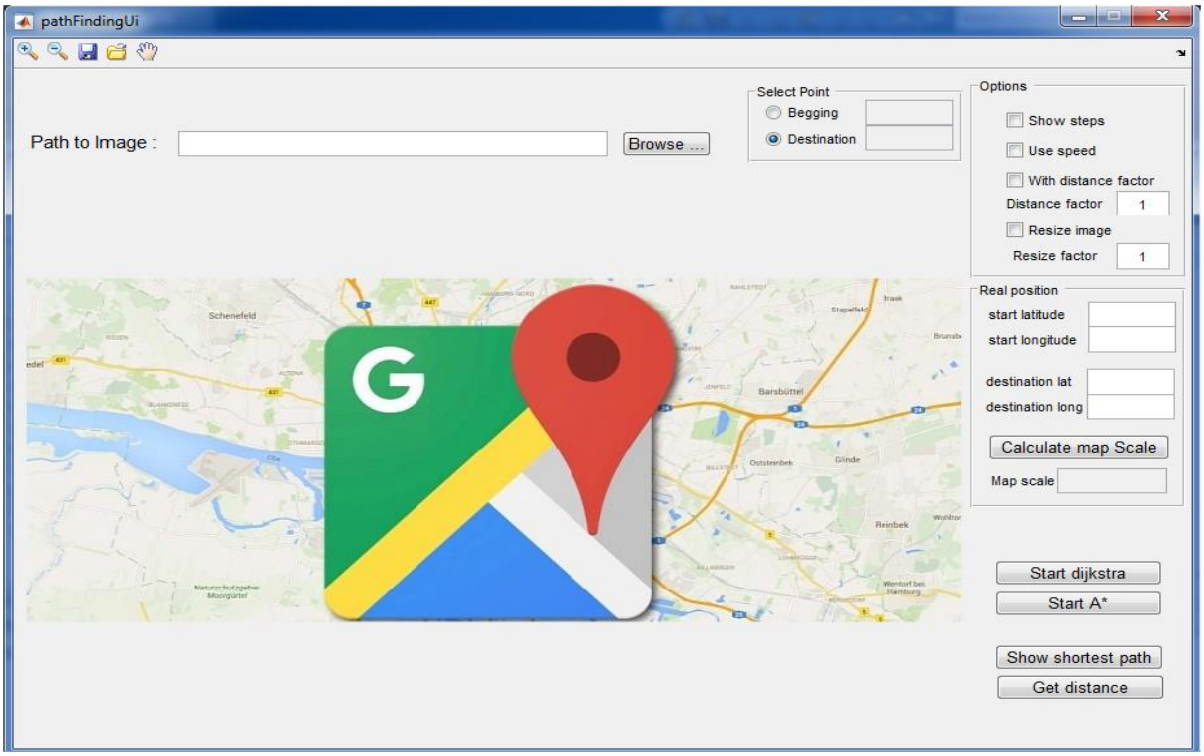


Figure.3.2 : interface de système

3.1. IMPORTER L'IMAGE :

- **BROWSER** : En utilise le Botton **Browser** pour choisirez une image map

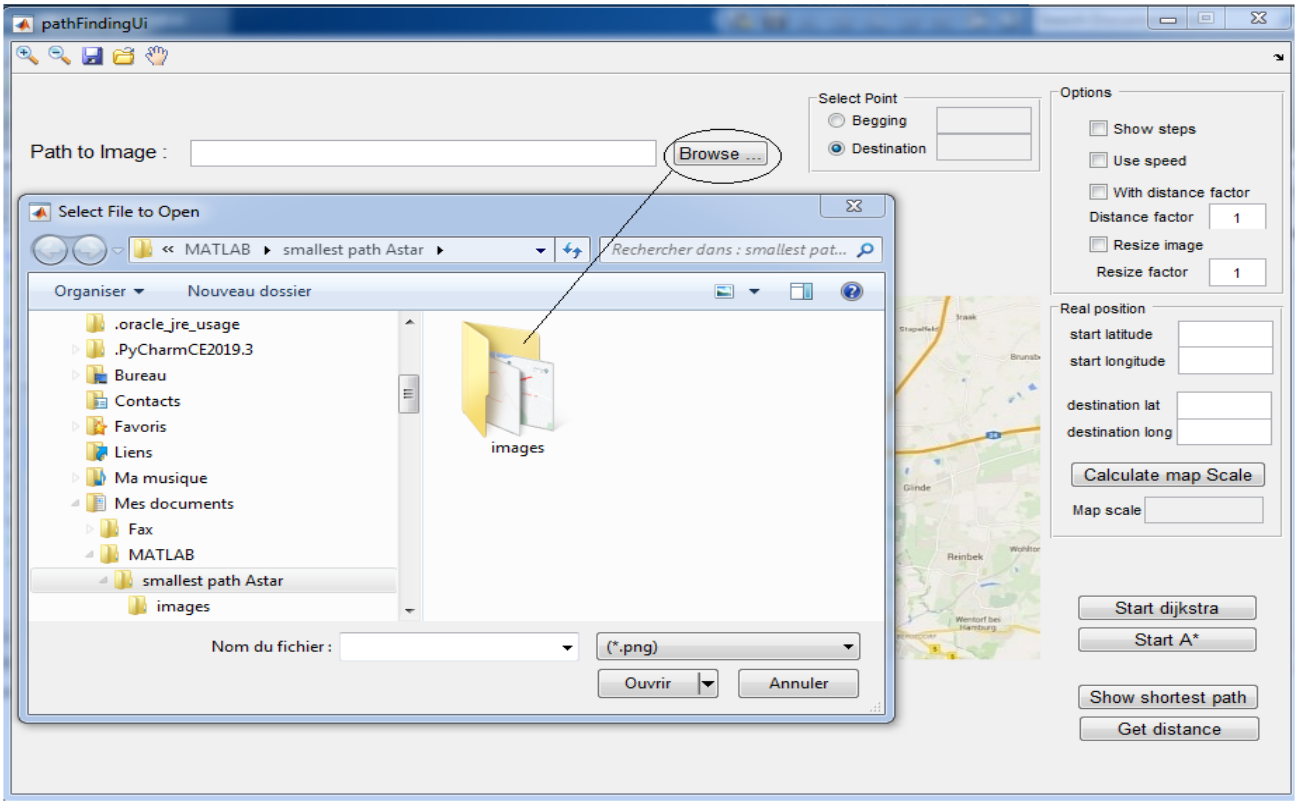


Figure.3.3 :(Browser) pour choisirez une image map

- Sélectiond’une image map

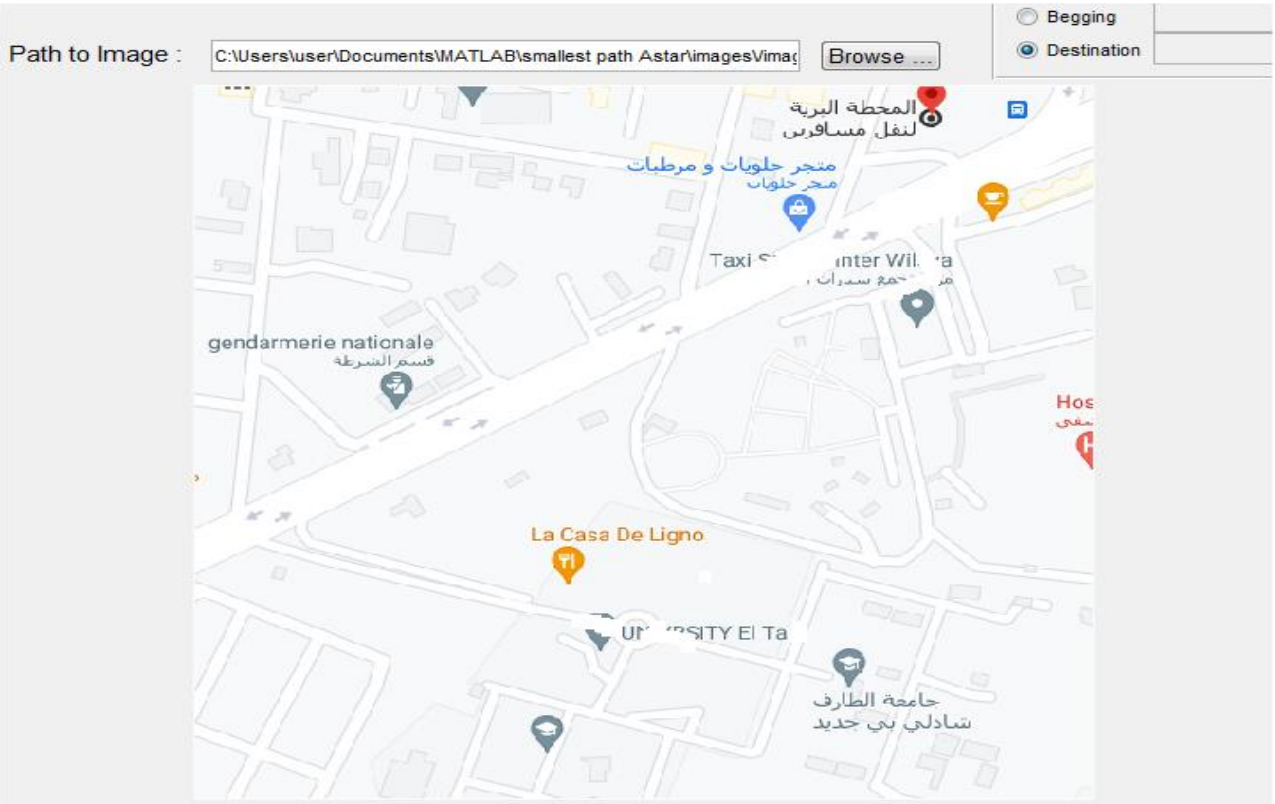


Figure.3.4: sélection de l’image map

3.2. CONFIGURATION :

- **SELECTIONNEE UNE POINTS (BEGINE& DESTINATION) :**

Sélection des points « point de départ en vertele point de destination en bleu

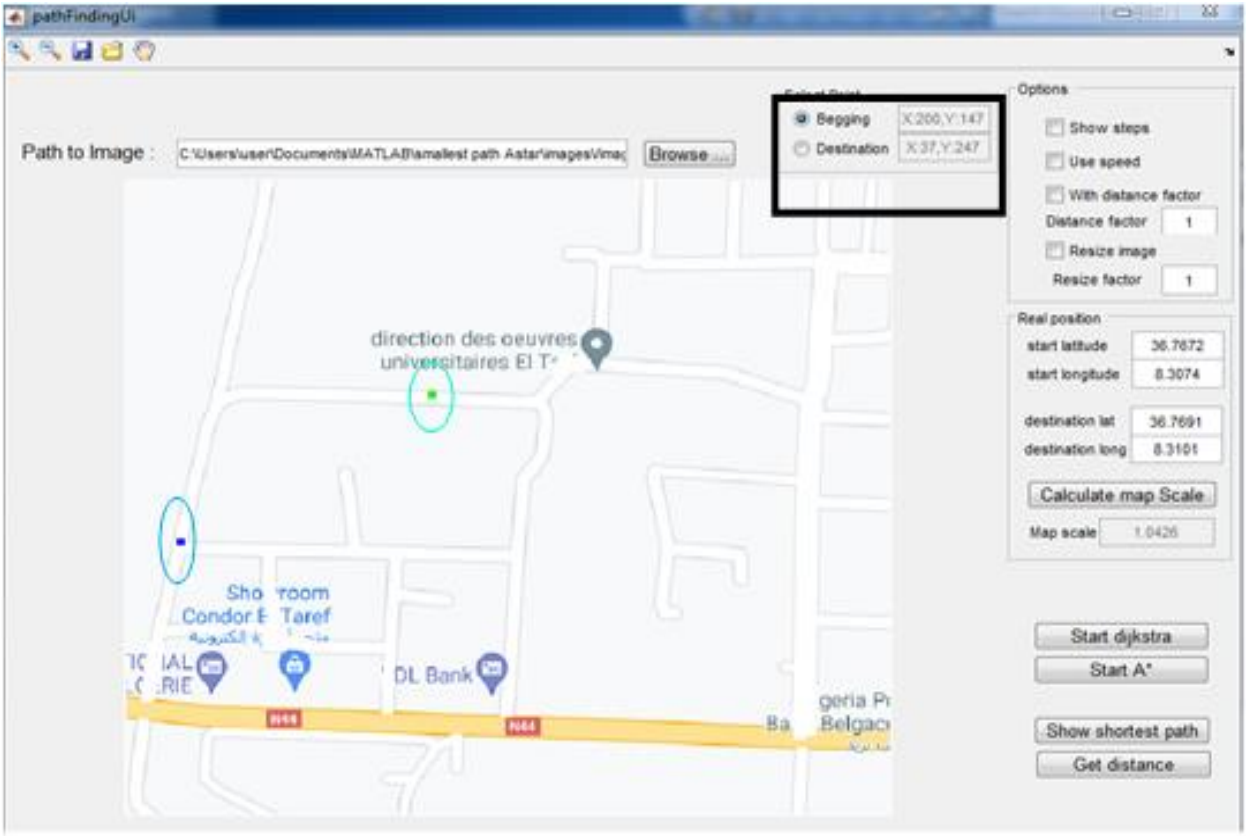


Figure.3.5 : Sélectionnée les points (beggin& destination)

- **MONTREZ LES ÉTAPES (SHOW STEPS) :**

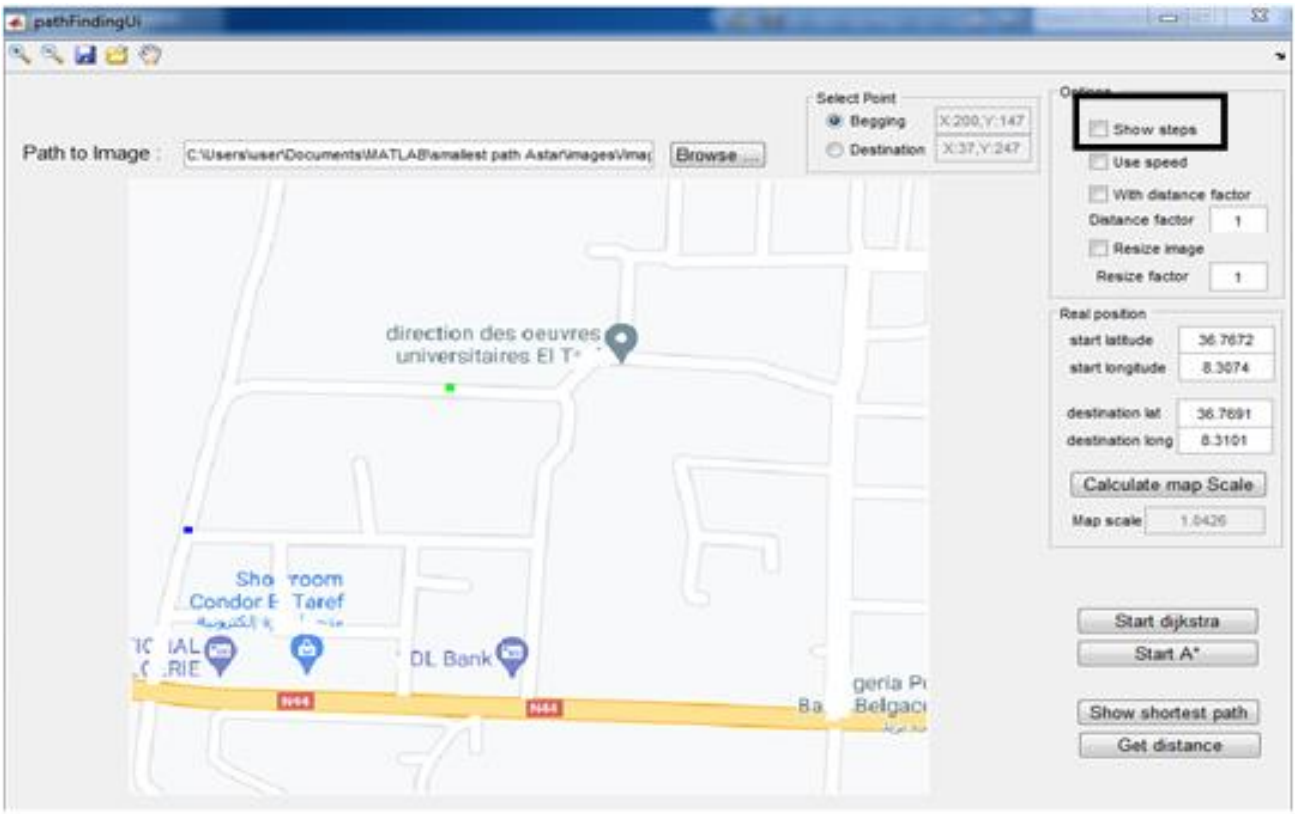


Figure.3.6: Sélectionnez SHOW STEPS (pour montrer les étapes)

- **SELECTIONNEE USE SPEED :**

L'utilisation de la vitesse : la vitesse est liée à la gradation des couleurs, plus la couleur est claire, plus la vitesse est élevée et vice versa.

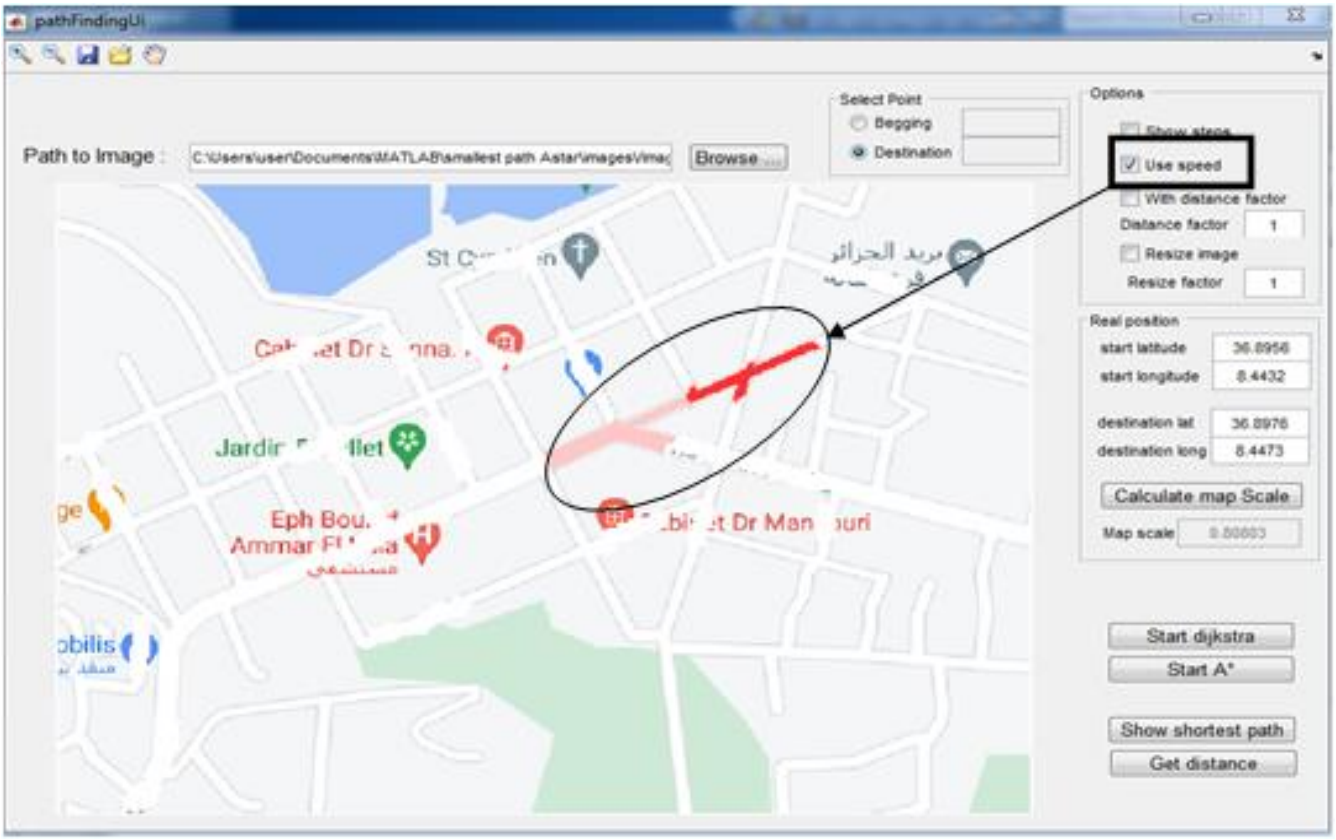


Figure.3.7 : Sélectionnée (use speed)

- **SELECTIONNEE (WITHE DISTANCE FACTORE):**

Utiliser le facteur pour réduire les points que vous souhaitez visiter avant d'atteindre l'objectif.

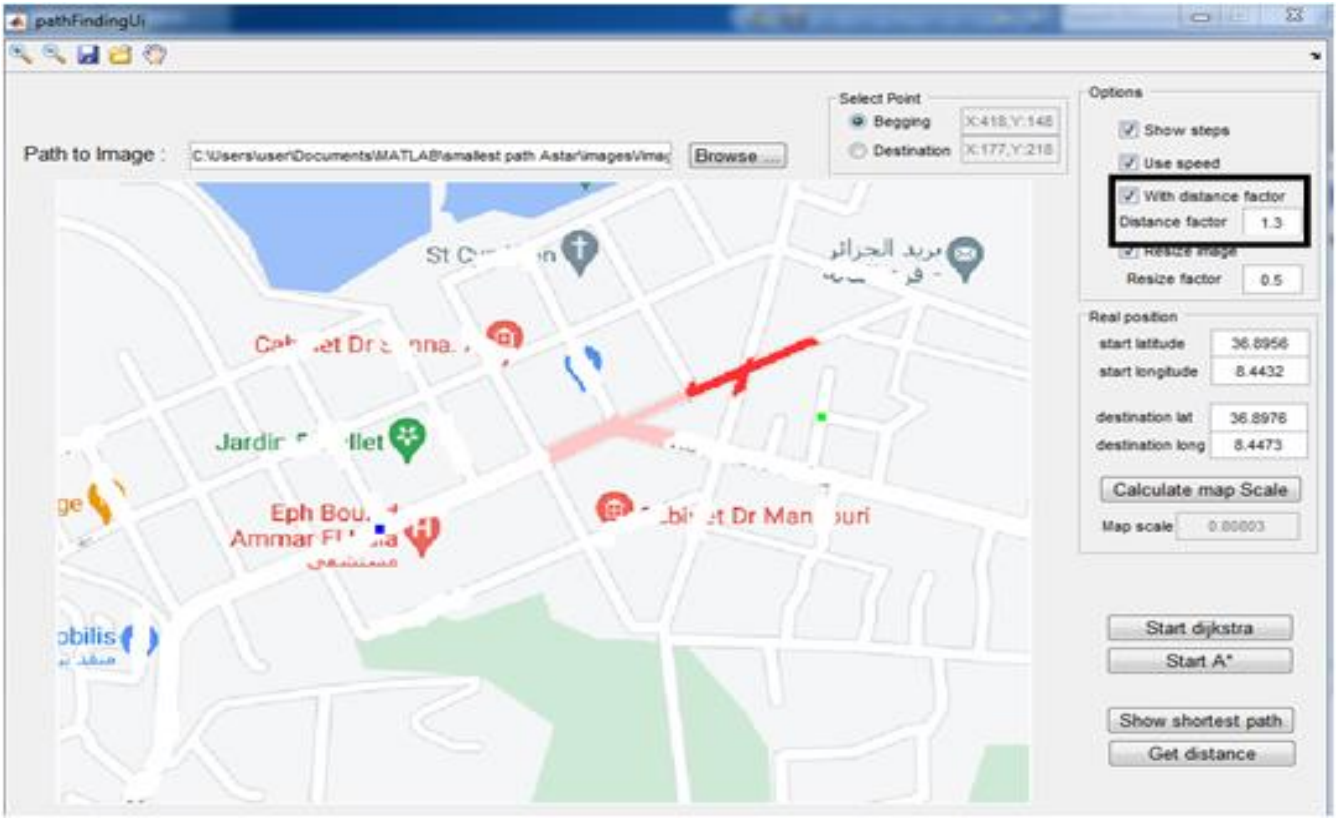


Figure.3.8 : Sélectionnée (withe distance factor)

- **SELECTIONNEE (RESIZE IMAGE):**

Minimiser la taille de l'image pour accélérer la recherche.

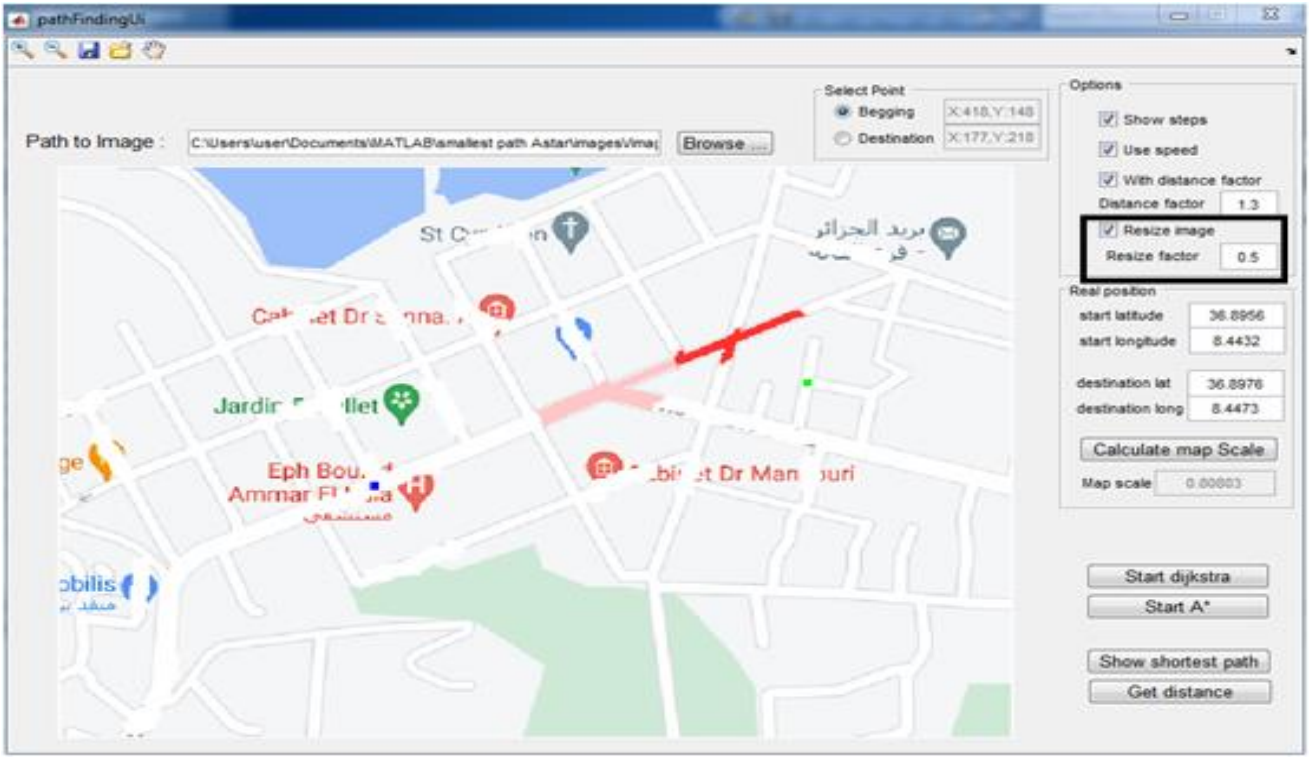


Figure .3.9 : Sélectionnée (Resize image)

- **REAL POSITION:**

Montrez la position réelle :

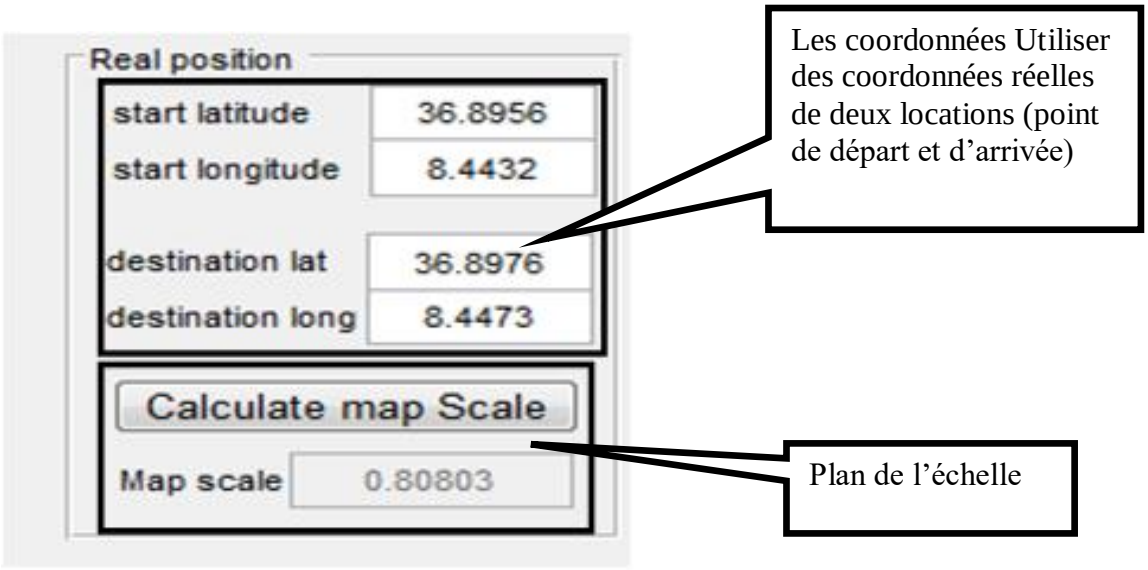


Figure.3.10 : les options real

3.3. LES METHODES DE RECHERCHE :

3.3.1 : Dijekstra :

Méthode pour recherche duPCC.

- On utilisant la vitesse :

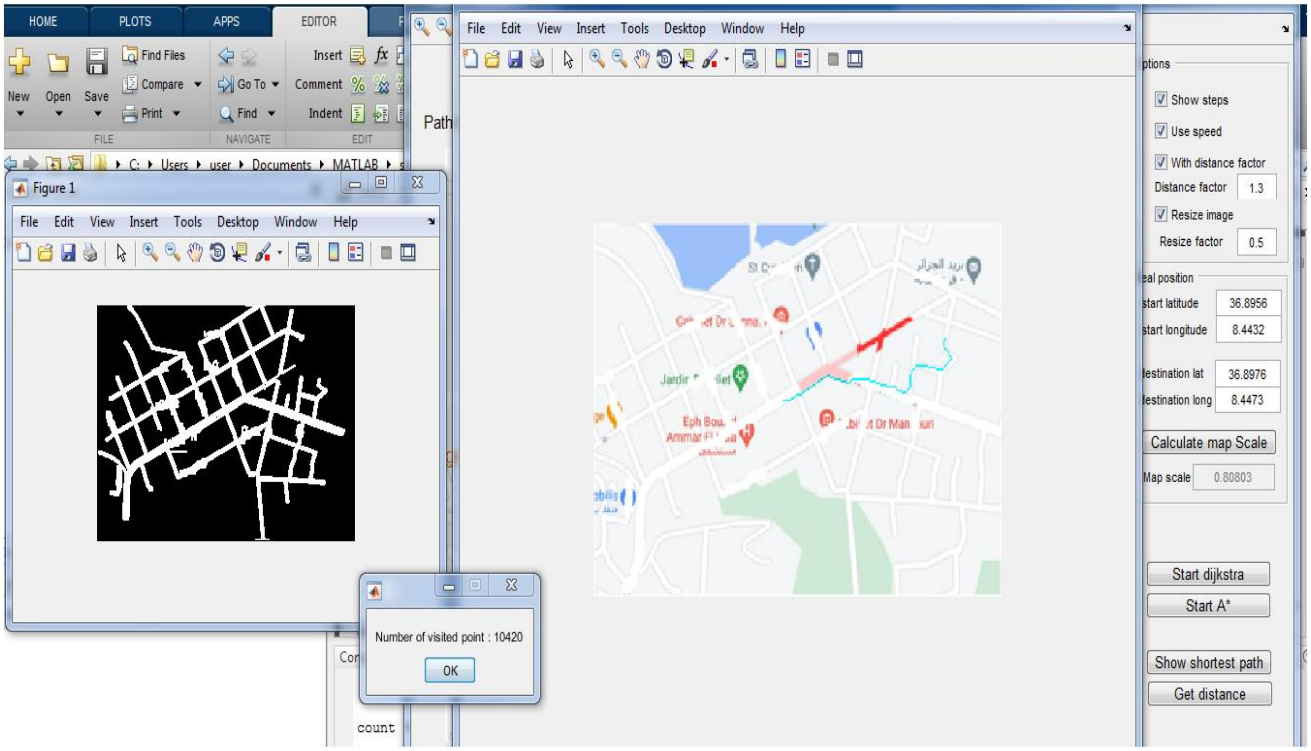


Figure.3.11 :le PCC en utilisant la vitesse

- la recherche avec DIJEKSTRA sans vitesse :

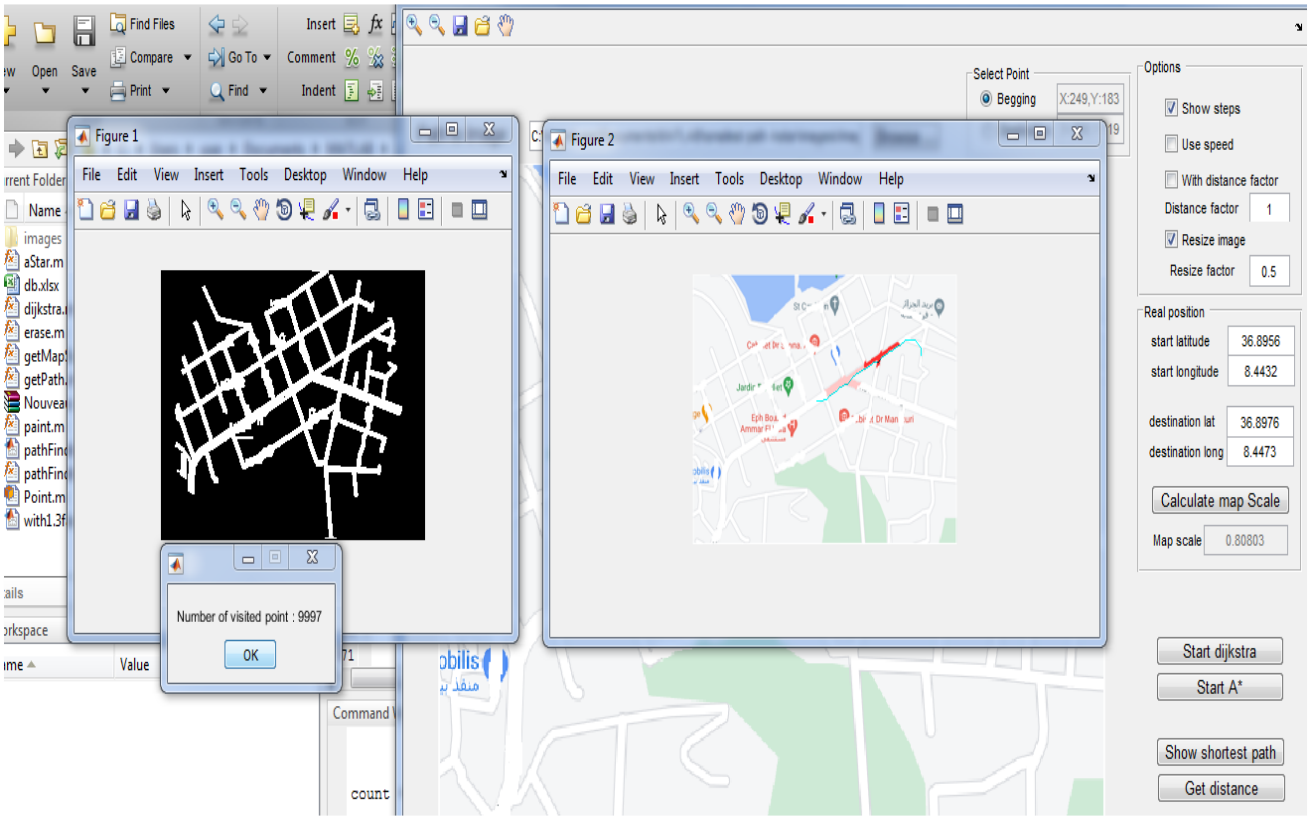


Figure .3.12 : le PCC sans utiliser vitesse

3.3.2: A* modifié:

Méthode de recherche pour trouver le plus court chemin.

- En utilisant la vitesse et le facteur :

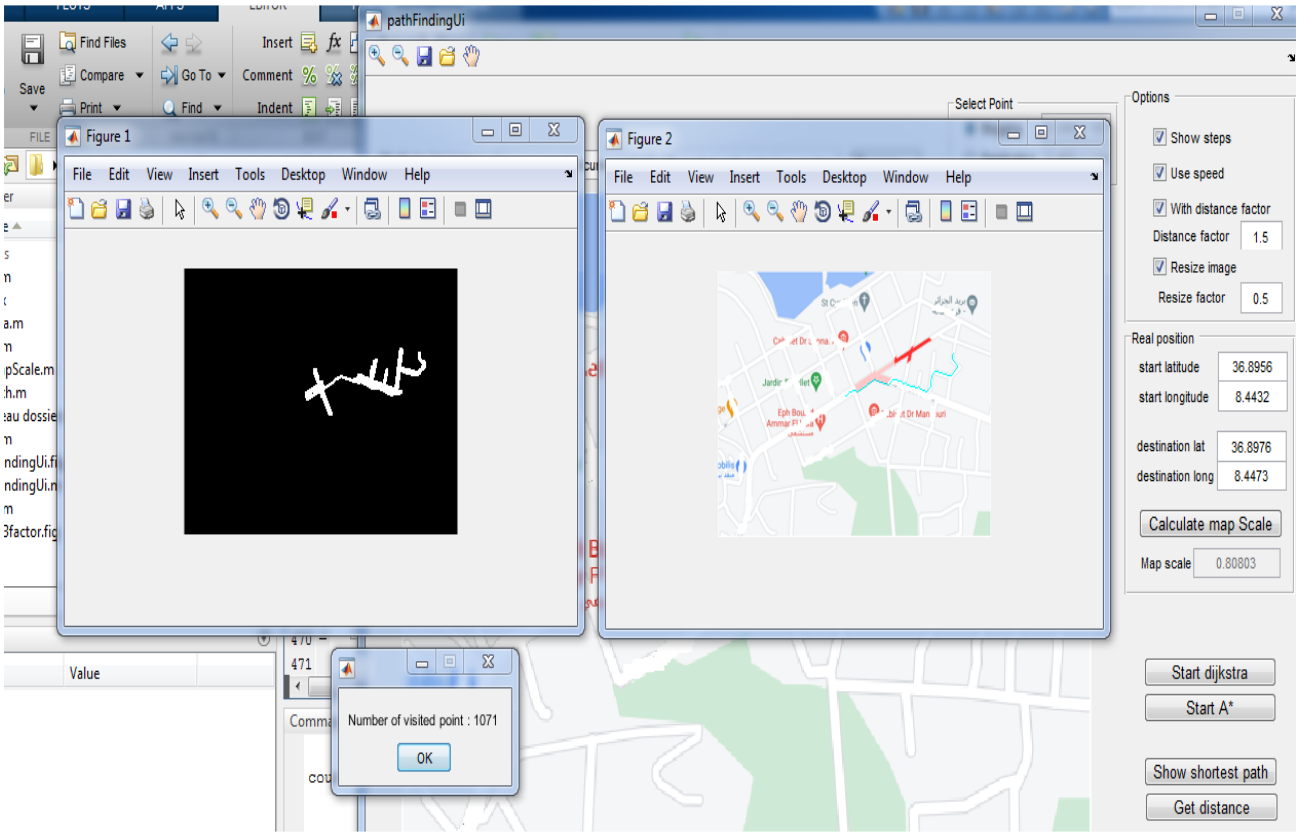


Figure.3.13 :lePCC avec vitesse et le facteur.

3.3.3. A*original :

- La PCC par A* avec l'utilisation de la vitesse et sans facteur :

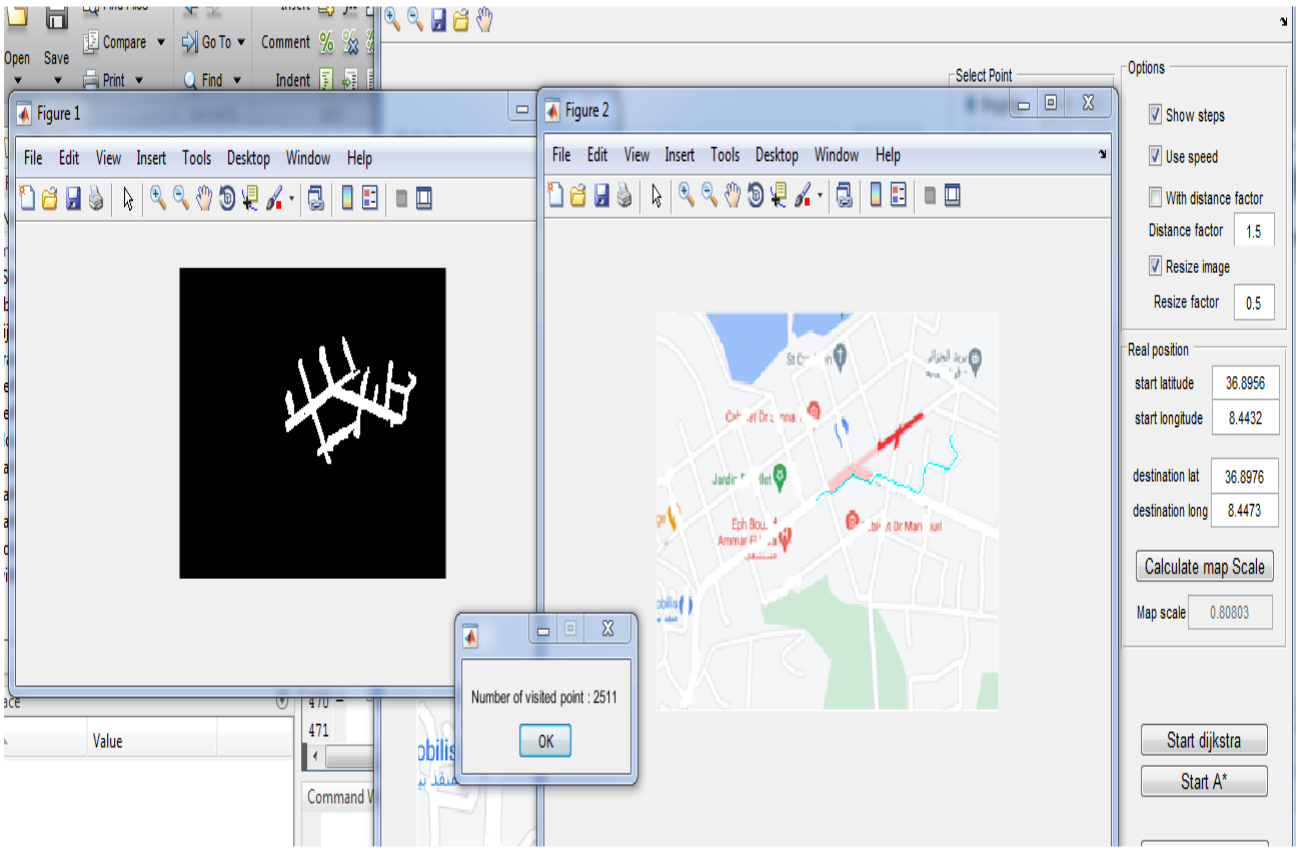


Figure.3.14 : le PCC avec vitesse et sans facteur

3.4. LES POINTS VISITEES :

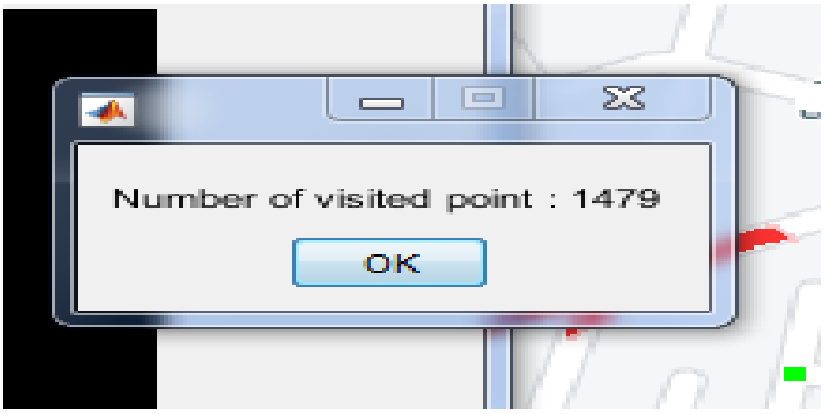


Figure.3.15: Nombre de points visités

3.5. CALCULE DE LA DISTANCE:

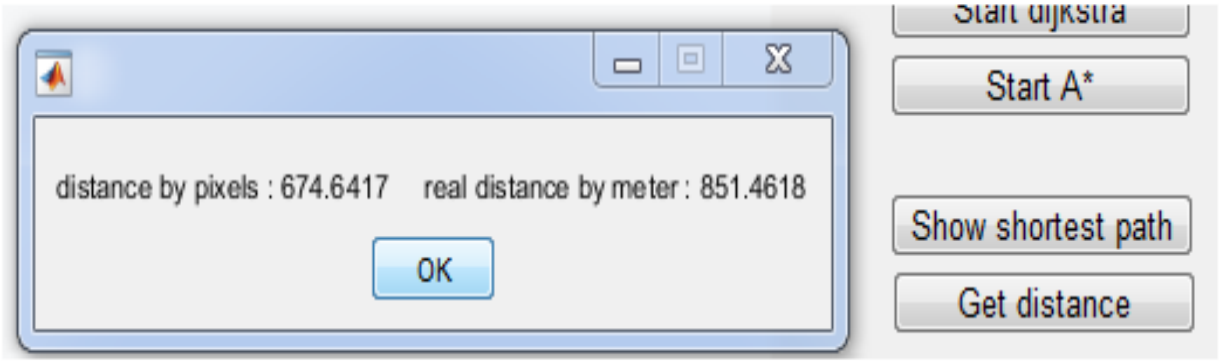


Figure.3.16: Interface de calcule la distance.

4. Conclusion

Dans ce dernier chapitre, nous avons essayé de présenter les notions essentielles relatives à l'implémentation de notre système.

Pour faciliter la compréhension de notre système, nous avons présenté également les outils, la plateforme et le langage utilisé pour le développement de ce système, comme nous avons donné des exemples d'interfaces d'accès à notre système et nous avons aussi montré des extraits du code sources.

Conclusion et Perspectives

Ce projet représente le fruit de cinq années d'études, où on a appliqué les notions théoriques et pratiques acquises au cours de notre formation.

Ce travail nous a donné une idée de carrière dans l'informatique, mais aussi une expérience de gestion d'un projet informatique dans les aspects conception et développement en utilisant différentes méthodes et outils, tels que les langages de programmation, la gestion de bases de données et la capacité à développer les systèmes. Les difficultés rencontrées lors de la mise en œuvre du projet ont été surmontées grâce à une bonne méthodologie de recherche.

Le résultat de ce projet est un système qui permet à l'exploration de trouver le chemin le plus court dans le temps le plus rapide.
Notre objectif était donc de concevoir, modéliser et développer un système répondant aux exigences ci-dessus. Pour y parvenir, les travaux se sont déroulés principalement en deux étapes :

- **Etape analyse et conception :** l'architecture de systèmeet dans laquelle méthodes qui utiliser.
- **Etape implémentation et réalisation :** Cette étape nous a permis d'acquérir et d'approfondir nos connaissances du langage' Matlab'.

Les principales fonctions du système ont été mises en œuvre avec succès, telles que : déterminer le point de départ et la destination, afficher toutes les étapes de l'exploration de l'itinéraire, rechercher l'itinéraire le plus court et le plus rapide, choisir l'algorithme de recherche, soit algorithme**dijekstra**oualgorithme **A***qui utilise le facteur et calcule la distance réelle.Après les résultats obtenus dans notre système en appliquant les deux algorithmes (**Dijekstra et A*),** nous choisissons A* pour rechercher le chemin le plus court par ce que A*est l'algorithme informatique le plus efficace garantissant de trouver le chemin le plus court.

En raison des contraintes de temps, nous avons dû concentrer nos efforts sur de nombreuses activités autres que la partie conception et modélisation ainsi que la partie mise en œuvre.

Références

[1]	intelligence artificielle/futuratechhttps://www.futura-sciences.com/tech/definitions/informatique-intelligence-artificielle-555/
[2]	le plus court cheminhttps://sites.google.com/site/cuatmessaoudi/home/cours-3
[3]	problème de plus court cheminhttps://studylibfr.com/doc/855461/chapitre-3.-le-probleme-du-plus-court-chemin.
[4]	algorithme dijekstrahttp://thesis.univ-biskra.dz/3447/3/2eme%20chapitre.pdf
[5]	https://fr.wikipedia.org/wiki/Algorithme_de_Bellman-Ford.
[6]	algorithmefloyd-warshallhttps://www.pairform.fr/doc/1/32/180/web/co/Floyd.html
[7]	Algorithme A* https://www.techno-science.net/definition/6469.html
[8]	http://helios.mi.parisdescartes.fr/~bouzy/Doc/IAL3/03_IA_heuristique_BB.pdf
[9]	http://cours-examens.org/images/An_2013_2/Etude_superieure/Ingeniorat_informatique/RO_EFPL/09-_pcc.
[10]	A* http://createursdemondes.fr/creation-de-jeux-video/pathfinding-algorithmes-en-a/
[11]	Mémoire de fin de cycle. AKOU HAMZA ET HARKOUK SAID .Conception et réalisation d'une application web de gestion d'hôtel sous java cas : « Hôtel ROYAL » promotion 2014/2015. Université ABDERRAHMENE MIRA-BEDJAIA. http://www.univ-bejaia.dz/xmlui/handle/123456789/740
[12]	A* https://www.createursdemondes.fr/creation-de-jeux-video/pathfinding-algorithmes-en-a/
[13]	algorithme de JHONSON https://complex-systems-ai.com/recherche-de-chemin-theorie-des-graphes/algorithme-de-johnson/
[14]	MATLAB http://elearning.univ-relizane.dz/moodle/pluginfile.php/36080/course/overviewfiles/TP1%20info3.pdf?forcedownload=1
[15]	https://haltode.fr/algo/structure/graphe/plus_court_chemin/dijkstra.html
[16]	(BFO) https://complex-systems-ai.com/swarm-algorithms/bacterial-foraging-optimization-algorithm/
[17]	exemplefloydwarshallhttp://cermics.enpc.fr/polys/oap/node77.html#f-___graph
[18]	**(BFO)**https://complex-systems-ai.com/algorithmes-dessaims/algorithme-doptimisation-de-lalimentation-bacterienne/

Résumé

Dans cette étude, nous avons présenté les différentes étapes de conception et de réalisation d'un système permettant la recherche du plus court chemin par des algorithmes (dijekstra et a*modifié)

Afin d'y accéder le plus rapidement possible et dans les plus brefs délais, nous avons collecté un certain nombre d'images de certains endroits à partir de Google Map et créé une base de données et nous les avons converties en une image à l'aide de logiciel Paint afin que nous puissions les traiter pour rechercher le chemin le plus court entre deux points, nous avons mené un certain nombre d'expériences dans différentes images afin de déterminer la structure idéale pour notre problème qui nous permettra d'obtenir les meilleurs résultats possibles. En effet, les résultats obtenus sont prometteurs pour résoudre le problème du plus court chemin.

Mots clés : plus court chemin, algorithme, facteur, vitesse, longitude, latitude, destination, dijekstra, a*modifié.

Abstract

In this study, we presented the different stages of design and realization of a system allowing the search for the shortest path by algorithms (dijekstra and a* modified)

In order to reach them as quickly as possible and in the shortest possible time, we have collected a number of images of some places from a Google map, created a database and converted them to an image in Paint software, for what we can deal with there. In order to find the shortest path between two points, we performed a number of experiments in different images to determine the ideal structure for our problem that will allow us to achieve the best possible results. Indeed, the results obtained are promising for solving the shortest path problem.

Keywords: shortest path, algorithm, factor, speed, longitude, latitude, destination, dijekstra, a *modified.

ملخّص

في هذه الدراسة، قدمنا المراحل المختلفة لتصميم وتحقيق نظام يسمح بالبحث عن أقصر مسار بواسطة الخوارزميات (Adijekstraو* معدل)
من أجل الوصول إليهم في أسرع و أقصر وقت ممكن ، قمنا بجمع عدد من الصور لبعض الأماكن من Google Map وإنشاء قاعدة بيانات وتحويلها إلى صورة عن طريق برنامج Paint من أجل ما يمكننا معالجته من أجل البحث عن أقصر طريق بين نقطتين ، أجرينا عددًا من التجارب في صور مختلفة لتحديد البنية المثالية لمشكلتنا التي ستتيح لنا تحقيق أفضل النتائج الممكنة. في الواقع، النتائج التي تم الحصول عليها واعدة لحل مشكلة أقصر طريق.

الكلماتالمفتاحية:أقصر مسار ، خوارزمية ، عامل ، سرعة ، خط طول ، خط عرض ، وجهة ، ديجسترا ، A* معدل.