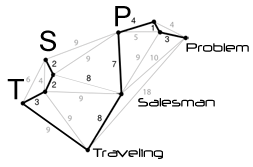
Yann DEBAIN 2 Juin 2017

Antoine MILCENT

**Rapport de Projet Informatique : Problème du Voyageur de Commerce**



**Sommaire :**

1 – Introduction………………………………………...……………2

2 – Spécifications

2.1 - Données : description des structures de données.…3

2.2 - Fonctions : prototypes et rôle des fonctions essentielles…………………………………………………….……4

2.3 – Tests……………………………………….…………….7

2.4 - Répartition du travail et planning prévu…………..…..8

3 – Implantation

3.1 - État du logiciel…………………………..………………9

3.2 - Tests effectués…………………………….…..………10

3.3 - Exemple d'exécution……………………...…………..12

4 – Suivi

4.1 - Problèmes rencontrés………………...………………13

4.2 - Planning effectif………………………………………..14

4.3 - Qu'avons-nous appris et que faudrait-il de plus ?....15

4.4 - Suggestions d'améliorations du projet………………15

5 – Conclusion……………………………………………………..16

**1 – Introduction**

L’objectif du projet est de faire le tour du monde le moins cher possible. C’est le problème du « voyageur de commerce ». Malgré la simplicité de son énoncé, il s'agit d'un problème d'optimisation pour lequel on ne connait pas d'algorithme permettant de trouver une solution exacte, rapidement dans tous les cas. Plus précisément, on ne connait pas d'algorithme en [temps polynomial](https://fr.wikipedia.org/wiki/P_(complexit%C3%A9)).

Pour résoudre ce problème, on va utiliser un algorithme de colonie de fourmis pour trouver le plus court chemin. Cet algorithme est inspiré des déplacements réels des fourmis qui vont se déplacer sur le chemin optimal par dépôt de phéromones.

Le résultat du projet sera donc un résultat heuristique donnant le chemin le plus court en s’appuyant sur une méthode de déplacement qui a fait ses preuves dans la réalité en faisant se déplacer les fourmis sur le graphe un grand nombre de fois.

**2 – Spécifications**

**2.1 - Données : description des structures de données**

La carte du problème est représentée par un graphe, ce graphe est implémenté par un tableau de sommets.

Un arc est un quadruplet :

* Numéro de la ville de départ.
* Numéro de la ville d'arrivée.
* Distance entre les deux .
* Quantité de phéromones.

Chacun des sommets contient les informations suivantes :

* Le numéro du sommet (numéro de la ville).
* Le nom du sommet (nom de la ville).
* Les coordonnées du sommet.
* La liste de ses voisins.

La liste des voisins est la liste d'adjacence associée à ce sommet, c'est une liste chaînée répertoriant les pointeurs sur les arcs qui partent de ce sommet.

Les fourmis virtuelles sont des entités qui parcourent l’intégralité du graphe en mémorisant leur parcours (appelé solution de la fourmi).

A chaque fois que la fourmi part d'une ville, elle décide dans quelle ville elle se va se rendre. Cette décision est aléatoire mais influencée par la distance entre les villes considérées et la quantité de phéromone déposée sur le chemin vers la ville.

Chaque fourmi est représentée par :

* Sa solution, c'est-à-dire l'ensemble des pointeurs d'arcs entre les villes déjà visitées qui sera une “file”.
* Le numéro de la ville de laquelle elle est partie.
* Le numéro de la ville dans laquelle elle se trouve actuellement.

|  |
| --- |
| **structure.h** |
| typedef struct arc\* POINTEUR\_ARC;  struct cellule {  POINTEUR\_ARC val;  struct cellule \* suiv;} ;  typedef struct cellule \* Liste;  /\*  Definition de la structure d'un arc  \*/  struct arc {  unsigned int depart;  unsigned int arrive;  double cout;  double pheromone;  };  /\*  Definition de la structure d'un sommet  \*/  struct sommet {  char nom[64];  unsigned int numero;  double x;  double y;  Liste voisin;  };  typedef Liste File;  /\*  Definition de la structure de fourmi  \*/  struct fourmis {  unsigned int ville\_depart;  unsigned int ville\_courante;  File solution;  };  typedef struct arc ARC;  typedef struct arc\* POINTEUR\_ARC;  typedef struct sommet SOMMET;  typedef struct sommet\* GRAPHE;  typedef struct fourmis FOURMIS; |

**2.2 - Fonctions : prototypes et rôle des fonctions essentielles**

L’ensemble des prototypes se trouvent dans les « .h » correspondant.

|  |
| --- |
| **Prototypes des fonctions « liste.h »** |
| /\*  Créer une liste vide.  \*/  Liste creer\_liste(void);  /\*  Teste si la liste est vide.  \*/  int est\_vide(Liste L);  /\*  Ajoute un pointeur d'arc à la liste.  On ajoute le pointeur de d'arc d'un voisin du sommet du graphe choisi pour le compléter et avoir la liste complète des voisins de chaque sommet  \*/  Liste ajout\_tete(POINTEUR\_ARC e, Liste L);  /\*  Visualise les différentes informations contenues dans la liste.  Ici permet d'afficher les voisins de chaque sommet du graphe.  \*/  void visualiser\_liste(Liste L); |

|  |
| --- |
| **Prototypes des fonctions « file.h »** |
| /\*  Créer une file vide.  \*/  File creer\_file(void);  /\*  Teste si la file est vide.  \*/  int file\_vide(File f);  /\*  Visualise les différentes informations contenues dans la file.  Ici permet d'afficher le parcours de la fourmi.  \*/  void visualiser\_file(File f);  /\*  Ajoute un pointeur d'arc à la file par la queue.  On ajoute le pointeur de d'arc de la prochaine ville choisie pour completer la solution de la fourmi.  \*/  File enfiler(POINTEUR\_ARC e, File f); |

|  |
| --- |
| **Prototypes des fonctions « fichiers.h »** |
| /\*  Charge le fichier graphe dans le répertoire source pour pouvoir extraire les différentes données (comme le nombre de sommet, le cout d'un arc et les voisins) et ainsi creer le graphe.  \*/  GRAPHE chargeFichierGraphe(char\* nomDuFichier,unsigned int \*pTailleGraphe); |

|  |
| --- |
| **Prototypes des fonctions « graphe.h »** |
| /\*  Retourne un graphe vide de taille "taille".  \*/  GRAPHE creer\_graphe(unsigned int taille);  /\*  Retourne un graphe complété.  On ajoute un arc au graphe pour le complété et avoir le graphe complet demandé par la gestion de fichier  \*/  GRAPHE ajout\_arc(GRAPHE g, ARC a);  /\*  Permet de visualiser le graphe.  On peut suivre l'évolution du graphe au fil du temps.  \*/  void visualiser\_graphe(GRAPHE g, unsigned int\* taille); |

|  |
| --- |
| **Prototypes des fonctions « fourmis.h »** |
| /\*  Créer le tableau de fourmis qui parcourra le graphe.  \*/  FOURMIS\* creer\_fourmis(GRAPHE g, unsigned int taille\_graphe);  /\*  Retourne le pointeur sur arc de la prochaine ville pas encore visité par la fourmi.  Le calcul de cette prochaine ville se fait par probabilité.  \*/  POINTEUR\_ARC choix\_prochaine\_ville(GRAPHE g, FOURMIS\* pf, unsigned int taille);  /\*  Met à jour la quantité de phéromones sur un arc.  Cette fonction va permettre aux fourmis de se déplacer selon le nombre de phéromones et ainsi donner une préférence sur le chemin.  \*/  double mise\_a\_jour\_des\_pheromones(GRAPHE g, FOURMIS f);  /\*  Retourne la file qui contieint le parcours de la fourmi f.  \*/  File parcours(GRAPHE g, FOURMIS\* pf, unsigned int taille); |

**2.3 – Tests**

Les tests seront faits en deux parties. Tout d'abord celui qui écrit le programme fera des tests simples pour vérifier si le programme est bien écrit, compile et s’exécute en donnant le bon résultat. Puis celui qui n’a pas écrit le programme cherchera à “casser” le code pour voir si le programme prend en compte toutes les possibilités.

On testera donc en premier lieu le chargement de fichiers, la gestion des graphes puis après la gestion des fourmis et enfin l’ensemble des fichiers pour répondre à la problématique du projet.

**2.4 - Répartition du travail et planning prévu**

Yann DEBAIN :

* Définition des structures.
* Gestion des graphes.
* Gestion des fourmis.
* Tests.

Antoine MILCENT :

* Définition des structures.
* Gestion des fichiers.
* Makefile & README.
* Tests.

**Diagramme de Gant prévisionnel**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tâches** | **Séance 1** | **Séance 2** | **Séance 3** | **Séance 4** |
| Définition des structures |  |  |  |  |
| Gestion des fichiers |  |  |  |  |
| Gestions des graphes |  |  |  |  |
| Gestion des fourmis |  |  |  |  |
| Makefile & README |  |  |  |  |
| Tests de la gestion des fichiers |  |  |  |  |
| Tests de la construction de graphes |  |  |  |  |
| Test de la gestion des fourmis |  |  |  |  |
| Test global du projet |  |  |  |  |

Les fonctions pour les listes et les files ont déjà été écrites lors des séances précédentes et ont été adaptées au sujet. Ce qui nous a permis de gagner un peu de temps.

**3 – Implantation**

**3.1 - État du logiciel**

Le programme livré avec le rapport compile, s’exécute et retourne le meilleur chemin et son coût sur un nombre de cycles précédemment défini.

Le programme prend donc en compte toutes les villes parcourues par la fourmi même si elle est déjà passée par ces dernières dans un cycle (il arrive à la fourmi de de faire des aller-retours dans son parcours), ce qui n’est pas voulu d’après le cahier des charges.

Le cahier des charges est globalement respecté sauf le point cité au-dessus :

* On ouvre et lit un fichier représentant un graphe.
* La construction du graphe est respectée.
* La solution de chaque fourmi regroupe bien l’ensemble des villes parcourues par celle-ci.
* Les phéromones sont déposées et mises à jour à chaque cycle.
* On retourne le meilleur chemin et son coût sur l’ensemble des cycles.

**3.2 - Tests effectués**

Une fois le programme de gestion de fichiers écrit (le code qui avait la priorité) nous avons pu tester les différentes parties du code séparément, puis ensemble.

Les différents tests effectués ont permis de faire un point sur l’avancement du projet et régler un grand nombre de problèmes dans la compréhension du sujet et dans la conception des différentes fonctions.

|  |
| --- |
| **Test des fonctions de gestion des fichiers et de construction de graphe (graphe11.txt)** |
|  |

|  |
| --- |
| **Test des fonctions de gestion du parcours des fourmis** |
|  |

**3.3 - Exemple d'exécution**

|  |
| --- |
| **Exécution du projet avec les paramètres de déplacement de la fourmi** |
| #define NOMBRE\_DE\_FOURMIS 2 //optimisation en puissance de 2  #define COEFFICIENT\_EVAPORATION\_PHEROMONE 0.5  #define IMPORTANCE\_PHEROMONE 1  #define IMPORTANCE\_VISIBILITEE 2  #define VALEUR\_INITIALE\_PHEROMONE 1E-5  #define QUANTITE\_PHEROMONE\_A\_DEPOSEE 1  #define MAX\_CYCLE 1 |
|  |

Pour avoir un meilleur résultat, il faut avoir un grand nombre de cycle et un grand nombre de fourmis par rapport au nombre de sommet du graphe. Les résultats peuvent varier selon les différents paramètres de déplacements des fourmis.

**4 – Suivi**

**4.1 - Problèmes rencontrés**

Nous avons rencontré un certain nombre de problèmes au cours des différents tests effectués.

Tout d’abord lors de la lecture des fichiers pour trouver la bonne ligne à lire (prévoir le bon nombre de ligne à sauter) et avoir accès à la bonne information sur le graphe.

La construction du graphe a posé un certain nombre de problèmes à cause de codes mal écrits et surtout inutiles. Combiner gestion des fichiers et construction du graphe a permis de mettre en lumière les codes inutiles et non fonctionnels. Ceci permet au code d’être plus rapide, moins gourmand en place (les anciennes fonctions créaient des copies) et d’exécuter exactement ce qu’on veut.

La gestion des fourmis est le code qui nous a pris le plus de temps et qui a posé le plus de problèmes. Au niveau de la compréhension du sujet, la solution sous forme de file de pointeur d’arc a été un choix peu compréhensible (pour nous un tableau aurait été plus facile à traiter). Le choix de la prochaine ville a donc posé des problèmes d’adressages et de compréhension (le code était gourmand en temps lors de la première écriture).

Une fois que les problèmes de la prochaine ville ont été réglés, le parcours de la fourmi a juste posé des problèmes de sauvegarde de données (il fallait penser à stocker le choix de la prochaine de ville dans une variable).

Le dernier problème rencontré est celui qui n’a pas encore été résolu, pour répondre complètement aux exigences du sujet, il faut que la fourmi parcoure le graphe sans faire d’aller-retour dans un cycle. Pour éviter ces aller-retours qui faussent le résultat final, l’algorithme doit tourner sur des petits cycles.

**4.2 - Planning effectif**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tâches** | **Séance 1** | **Séance 2** | **Séance 3** | **Séance 4** |
| Définition des structures |  |  |  |  |
| Gestion des fichiers |  |  |  |  |
| Gestions des graphes |  |  |  |  |
| Gestion des fourmis |  |  |  |  |
| Makefile & README |  |  |  |  |
| Tests de la gestion des fichiers |  |  |  |  |
| Tests de la construction de graphes |  |  |  |  |
| Test de la gestion des fourmis |  |  |  |  |
| Test global du projet |  |  |  |  |

La répartition des tâches a été respectée. Chaque personne s’est occupée de ce qui lui est attribué et a apportée son aide à l’autre en cas de besoin pour résoudre des problèmes qui empêchaient l’avancement du projet.

Le planning a été globalement respecté, mais lors de l’apparition de tous les problèmes cités plus haut lors des tests pour la gestion de fourmis, nous avons pris du retard car nous ne comprenions pas d’où venait le problème (accès à de la mémoire non autorisé). Pour arriver au code final, du travail personnel a été nécessaire (dans la compréhension du sujet, dans l’écriture des fonctions pour pouvoir solliciter au maximum les professeurs en cas de problèmes et dans les tests finaux pour épurer le code).

L’aide apportée par les professeurs a permis de nous débloquer et nous permettre d’avancer dans l’écriture du code et d’effectuer encore plus de tests pour arriver au programme final.

**4.3 - Qu'avons-nous appris et que faudrait-il de plus ?**

Ce projet nous a permis d’apprendre à implémenter des graphes et à recopier un mouvement réel sous forme d’algorithme pour résoudre un problème apparemment sans solution théorique optimale.

Le projet, se faisant en binôme, nous a permis d’apprendre à travailler en groupe sur un même projet informatique et nous a apporté une certaine rigueur dans l’écriture du code (chacun écrit sa partie du code sans que l’autre intervienne sauf pour le test, le style d’écriture doit être le même pour que chacun se comprenne).

**4.4 - Suggestion d'améliorations du projet**

D’un point de vue de notre algorithme :

Résoudre le problème des aller-retours des fourmis dans leur parcours.

Réduire au maximum les copies inutiles d’informations (comme passer en paramètre un sommet au lieu de son indice).

Faire un affichage graphique pour avoir une meilleure visualisation du graphe en exploitant les coordonnées de chaque sommet (qui sont données dans la structure).

Trouver les paramètres d’optimisation pour réduire la complexité en temps et en espace du programme.

D’un point de vu organisation :

Approfondir le cours sur les graphes avant d’entamer le projet pour démarrer plus facilement.

**5 – Conclusion**

En somme, l’implémentation d’un algorithme comme celui des fourmis est un exemple concret d’utilisation des files et permet d’approcher la résolution de problèmes avec des méthodes heuristiques.

Notre programme permet finalement de trouver une solution et répond globalement au cahier des charges.

Aussi, il serait tout à fait utile d’ajouter à ce programme de base les fonctionnalités proposées dans ce rapport pour le rendre plus ergonomique.