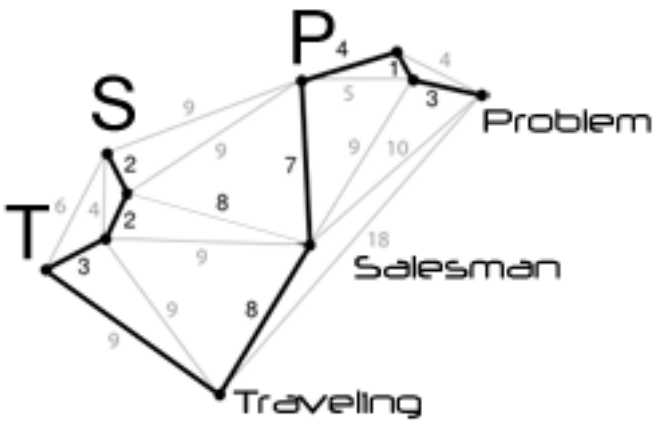
Yann DEBAIN
Antoine MILCENT

2 Juin 2017

Rapport de Projet Informatique : Problème du Voyageur de Commerce



Sommaire :

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.

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

```c
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)

```
minatecap42-212:Graphe_LPC yanndebain$ make
gcc -c -g  main.c
gcc -c -g  fichiers.c
gcc  -o main main.o fichiers.o graphe.o liste.o fourmis.o file.o  -lm
minatecap42-212:Graphe_LPC yanndebain$ ./main
6 sommets du graph chargés avec succes
15 arcs chargés avec succes
chargement ok
Voisins du sommet 0
depart  = 0  arrivée = 1  cout = 0.823365 pheromone = 0.000010
depart  = 0  arrivée = 2  cout = 0.539884 pheromone = 0.000010
depart  = 0  arrivée = 3  cout = 0.549344 pheromone = 0.000010
depart  = 0  arrivée = 4  cout = 1.051899 pheromone = 0.000010
depart  = 0  arrivée = 5  cout = 0.545452 pheromone = 0.000010
Voisins du sommet 1
depart  = 1  arrivée = 2  cout = 0.327274 pheromone = 0.000010
depart  = 1  arrivée = 3  cout = 0.741992 pheromone = 0.000010
depart  = 1  arrivée = 4  cout = 0.482120 pheromone = 0.000010
depart  = 1  arrivée = 5  cout = 0.377032 pheromone = 0.000010
depart  = 1  arrivée = 0  cout = 0.823365 pheromone = 0.000010
Voisins du sommet 2
depart  = 2  arrivée = 3  cout = 0.668909 pheromone = 0.000010
depart  = 2  arrivée = 4  cout = 0.730579 pheromone = 0.000010
depart  = 2  arrivée = 5  cout = 0.335492 pheromone = 0.000010
depart  = 2  arrivée = 1  cout = 0.327274 pheromone = 0.000010
depart  = 2  arrivée = 0  cout = 0.539884 pheromone = 0.000010
Voisins du sommet 3
depart  = 3  arrivée = 4  cout = 0.682049 pheromone = 0.000010
depart  = 3  arrivée = 5  cout = 0.372327 pheromone = 0.000010
depart  = 3  arrivée = 2  cout = 0.668909 pheromone = 0.000010
depart  = 3  arrivée = 1  cout = 0.741992 pheromone = 0.000010
depart  = 3  arrivée = 0  cout = 0.549344 pheromone = 0.000010
Voisins du sommet 4
depart  = 4  arrivée = 5  cout = 0.509852 pheromone = 0.000010
depart  = 4  arrivée = 3  cout = 0.682049 pheromone = 0.000010
depart  = 4  arrivée = 2  cout = 0.730579 pheromone = 0.000010
depart  = 4  arrivée = 1  cout = 0.482120 pheromone = 0.000010
depart  = 4  arrivée = 0  cout = 1.051899 pheromone = 0.000010
Voisins du sommet 5
depart  = 5  arrivée = 4  cout = 0.509852 pheromone = 0.000010
depart  = 5  arrivée = 3  cout = 0.372327 pheromone = 0.000010
depart  = 5  arrivée = 2  cout = 0.335492 pheromone = 0.000010
depart  = 5  arrivée = 1  cout = 0.377032 pheromone = 0.000010
depart  = 5  arrivée = 0  cout = 0.545452 pheromone = 0.000010
minatecap42-212:Graphe_LPC yanndebain$
```

Test des fonctions de gestion du parcours des fourmis

```
minatecap42-212:Graphe_LPC yanndebain$ ./main
6 sommets du graph chargés avec succes
15 arcs chargés avec succes

File de la fourmis 1
depart = 1 arrivée = 0 cout = 0.823365 pheromone = 1.214533
depart = 0 arrivée = 3 cout = 0.549344 pheromone = 0.728492
depart = 3 arrivée = 1 cout = 0.741992 pheromone = 0.472885
depart = 1 arrivée = 3 cout = 0.741992 pheromone = 0.350060
depart = 3 arrivée = 4 cout = 0.682049 pheromone = 0.282591

poids de la solution de la fourmis n°1  = 3.538742

File de la fourmis 2
depart = 1 arrivée = 5 cout = 0.377032 pheromone = 2.652300
depart = 5 arrivée = 1 cout = 0.377032 pheromone = 1.326152
depart = 1 arrivée = 0 cout = 0.823365 pheromone = 0.633948
depart = 0 arrivée = 3 cout = 0.549344 pheromone = 0.611492
depart = 3 arrivée = 1 cout = 0.741992 pheromone = 0.348587

poids de la solution de la fourmis n°2  = 2.868765
minatecap42-212:Graphe_LPC yanndebain$
```

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
```

```
MacBook-Pro-de-Yann:Graphe_LPC yanndebain$ ./main
6 sommets du graph chargés avec succes
15 arcs chargés avec succes
chemin le plus cours
depart  = 0  arrivée = 3  cout = 0.549344 pheromone = 0.611492
depart  = 1  arrivée = 5  cout = 0.377032 pheromone = 2.652300
depart  = 3  arrivée = 1  cout = 0.741992 pheromone = 0.348587
depart  = 5  arrivée = 1  cout = 0.377032 pheromone = 1.326152

cout du meilleur chemin = 2.045400

MacBook-Pro-de-Yann:Graphe_LPC yanndebain$
```

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

<u>D'un point de vue de notre algorithme :</u>

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.

<u>D'un point de vu organisation :</u>

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.