Anteur Neil: nanteur@etudiant.univ-mlv.fr
Hamdi Karim: khamdi@etudiant.univ-mlv.fr
Programmation 4 - TP3

Compte Rendu Projet:

Problème du Voyageur du Commerce
PVC

Sommaire :

I) Problématique :

Un voyageur commercial souhaite parcourir toutes les villes d'une carte sans visiter deux fois la même ville, et que son voyage soit le plus court possible, c'est-à-dire avec la distance la plus courte.

Il n'existe pas de solution en moins de O(n!), avec n le nombre de villes, nous allons faire une approximation d'une meilleure solution.

II) Analyse fonctionnelle générale :

Les données traitées, structures, dans ce programme sont :

- Villes :

```
typedef struct{
        char * nom;
        int x, y;
}Ville;
```

- Cartes :

```
typedef struct{
        Ville * villes;
        int nb_ville;
}Carte;
```

- Visites :

```
typedef struct{
        int* parcours;
        int nb_ville;
        int distance;
```

```
}Visite;
```

- Population :

```
typedef struct{
    Visite* individus;
    Carte* carte;
    int nb_individus;
}Population;
```

Toutes les actions de notre programme exposé ici peuvent être lancées grâce à des options (cf. READ_ME.txt).

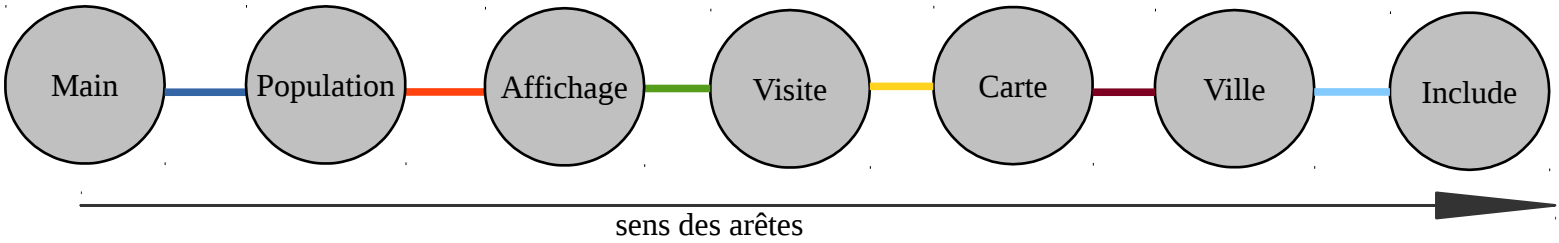
Le programme permet de traiter une résolution du PVC avec une partie du programme élaborée en programmation génétique, croisement et mutation, avec une population de visites.

Le programme calcule la distance de visite et peut calculer une visite dite gloutonne, c'est-à-dire on avance de la ville de départ vers la ville d'arrivée en choisissant à chaque fois la ville suivante en prenant celle qui a la plus petite distance entre la ville où l'on est et les autres villes non visitées.

Il permet également de charger ou de sauvegarder une visite avec la carte. Notre programme sauvegardera la dernière meilleure visite obtenue dans un fichier passé en paramètre.

Notre programme supporte également la création de carte par clique à la souris où l'utilisateur clique la dans la fenêtre, qui est la carte, pour mettre les villes où il le souhaite.

Nous avons décidé de découper notre programme, tout d'abord par structure et également avec certaines fonctionnalités qui ne dépendent d'aucune autre structure, il y a donc plusieurs modules représentés ici :



Nous avons décidé pour nos modules d'avoir un module Include qui n'a pas de ".h" et qui contient toutes les inclusions de bibliothèques ainsi que toutes les structures et les macros "define".

Nous avons décidé d'utiliser un modulo pour rester dans la taille de fenêtre et de mettre des bordures. C'était l'implantation qui nous paraissait le plus naturel, en revanche, on a quand même fait une mise à jour des coordonnées, car si on récupère un fichier extérieur, on calculera et affichera tout grâce a une fonction de mise à l'échelle, tout en conservant les distances données dans le fichier.

Dans notre affichage, nous avons décidé de montrer à l'utilisateur dans le bandeau de notre fenêtre la comparaison de distance avec glouton. Et de montrer par le symbole "<" ou ">" quelle distance est la plus petite.

Nous avons décidé de ne pas toucher au premier tiers des visites qui sont les meilleures actuelles, mais on les copie à la fin du tableau de visites pour écraser les pires visites puis on les mute et les croise pour trouver des meilleures visites.

III) Analyse détaillée :

Nous allons désormais analysé en détail les modules :

i) Main

Il est le corps du programme le menu y est géré, il permet également de faire le traitement des options du programme, pour cela on utilise la fonction getopt() avec un switch pour parcourir l'ensemble des options possibles. Des tests sur les options y sont effectués afin de contrôler ce que donne l'utilisateur.

Il prend en entrée des options et renvoie un "exit" pour sortir en cas d'échec "EXIT_FAILURE" ou en cas de réussite "EXIT_SUCESS".

ii) Ville

Il y a une structure ville, alors pour respecter les consignes de la programmation, il y a un module ville. Ce module est le noyau des données que l'on traite. Tout tourne autour des villes, une carte est un ensemble de villes, un visite est un ordre de passage de ville, une population est un ensemble de plusieurs visites donc différents ordres de passage de ville. Il y a les fonctions de bases sur les structures :

- copie profonde, allocation, destruction, est_egale et initialiser.

Ce module prend en entrée des adresses de villes et renvoie soit un code de retour soit rien car les variables sont envoyés par adresse et donc les modifications sont persistantes.

iii) Carte

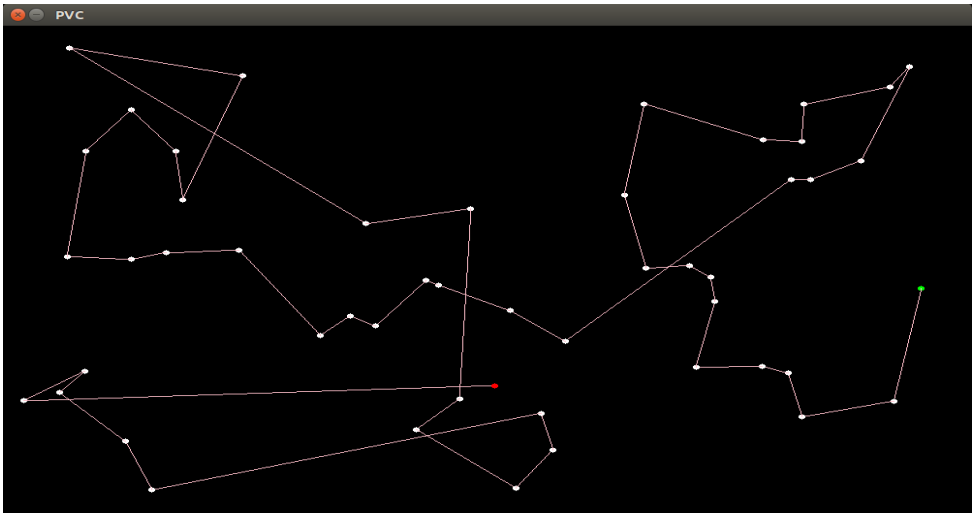
Il y a une structure carte alors pour respecter les consignes de programmation, il y a un module carte. Ce module gère l'ensemble des villes. Il y a aussi la gestion des différents types d’implémentation pour une carte, soit par clique de la souris soit aléatoirement. Le choix est géré dans par l'utilisateur dans les options. Il y a les fonctions de bases sur les structures :

- copie profonde, allocation, destruction, est_egale et initialiser.

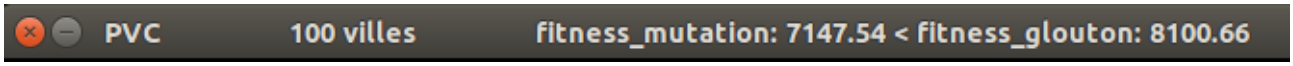
Le module prendra en entrée des adresses de carte et les modifiera à souhait et ne renvoie rien, car les modifications seront persistantes.

iv) Visite

Il y a une structure visite alors pour respecter les consignes de programmation, il y a un module visite. Ce module est le début de la résolution du problème du PVC car la notion de distance ou de meilleur chemin est maintenant intégré. La visite de toutes les villes est représenté graphiquement par des traits entre les villes. Il y a différents algorithmes qui entrent en jeu à ce niveau-là, comme l'algorithme Glouton :



Or nous allons vite nous rendre compte que même si l'algorithme Glouton est une solution pour le PVC, il ne donne pas forcément la meilleure, on peut encore mieux faire, car les dernières villes sont souvent éloignées. Pour ça, on commence la programmation génétique avec les fonctions croisement et mutation.



Il y a les fonctions de bases sur les structures :

- copie profonde, allocation, destruction, est_egale et initialiser.

Le module prendra en entrée des adresses de visites et les modifiera à souhait et ne renvoie rien, car les modifications seront persistantes.

v) Population

Il y a une structure population, alors pour respecter les consignes de programmation, il y a un module population. Ce module est la "résolution" ou du moins l'approximation de la meilleure solution pour le PVC. Avec evolution() on utilise les mutations et les croisements sur les meilleures visites pour les rendre meilleures à chaque passage du tableau de visite on trie à l'aide de qsort() pour avoir toujours les meilleures dans le début du tableau. Au bout d'un certain temps, on se retrouve avec une très bonne solution pour le PVC, on pourra arrêter à tout moment en appuyant sur la touche "Entrée".

Il y a les fonctions de bases sur les structures :

- copie profonde, allocation, destruction, est_egale et initialiser.

Le module prendra en entrée des adresses de populations et de carte parfois et les modifiera à souhait et ne renvoie rien, car les modifications seront persistantes.

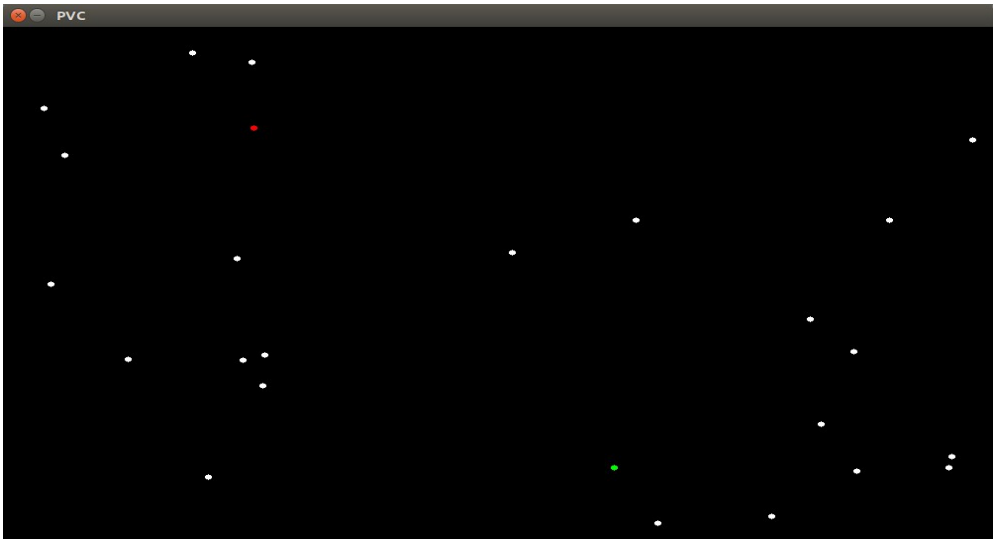
vi) Affichage

Ce module gère toutes les fonctions d'affichage, dans le terminal ou graphiquement avec la bibliothèque <MLV>. Il y a donc toutes les représentations des données :

- Les villes sont représentées par des points :
  Vert pour la ville de départ ;
  Rouge pour la ville d'arrivée ;
  Blanc pour les autres.

Dans le terminal, les villes sont représentées par un nom et par des coordonnées, pour le choix du nom nous avons décidé de prendre la représentation "XLY", où X est sa coordonnée en x, Y sa coordonnée en y et L une lettre majuscule prise au hasard.

- La carte est une fenêtre noire avec toutes les villes.

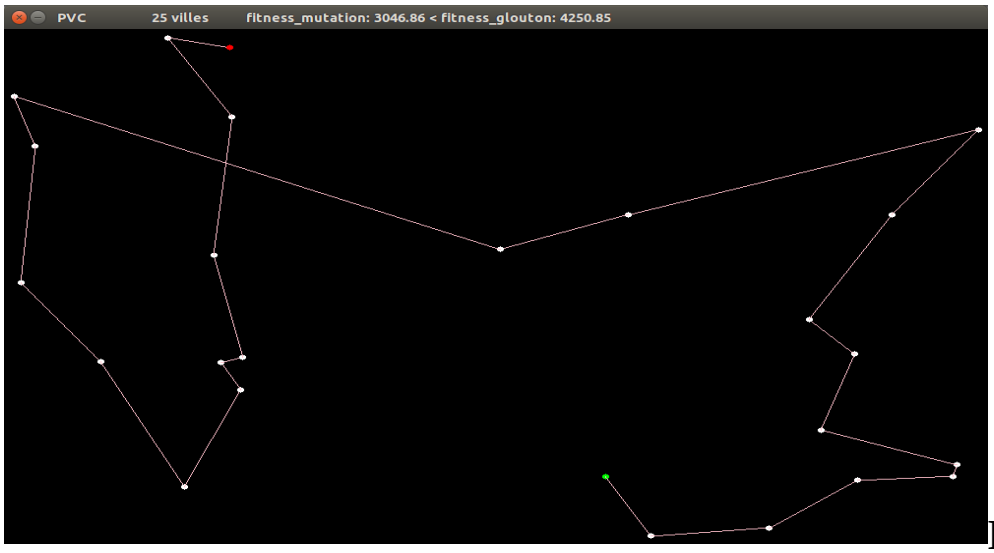


Dans le terminal, les cartes sont représentées par une liste de ville avec leur numérotation.

Nous avons choisi de prendre la ville 0 comme la ville de départ et la ville 1 la ville d'arrivée.

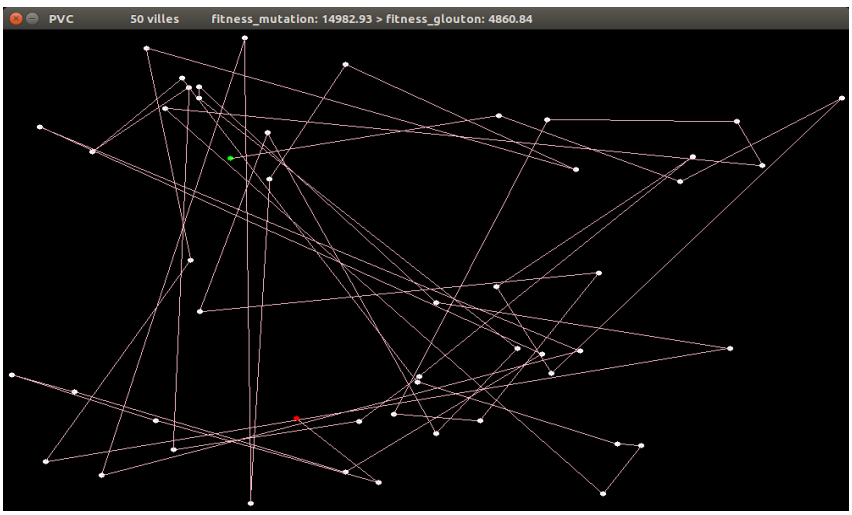
```
617A516 617 516
251D41 251 41
126D389 126 389
958G503 958 503
639X226 639 226
826K465 826 465
514F264 514 264
242J390 242 390
207I527 207 527
776A573 776 573
```

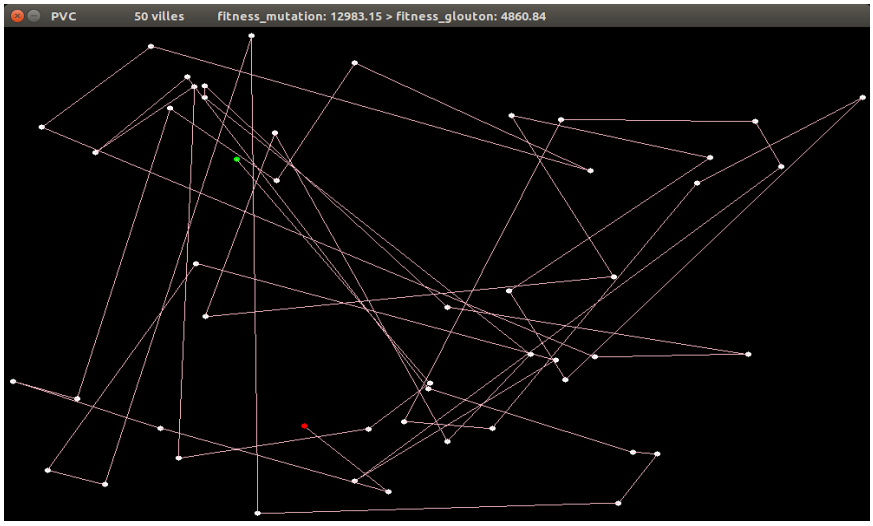
- La Visite est représentée graphiquement avec les traits entres les villes.



La -

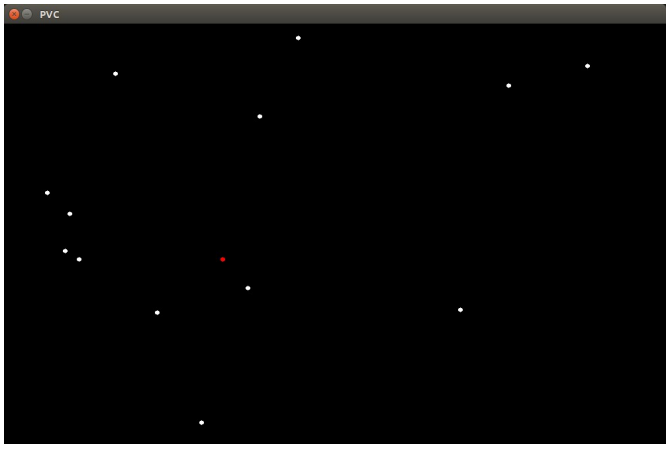
- Population n'affiche qu'une visite, toujours la meilleure parmi son tableau de visite.

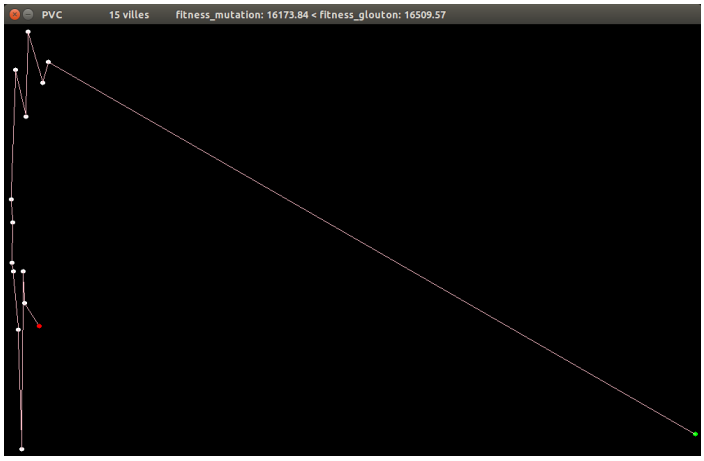




Il y a également une fonction de mise à l'échelle de l'affichage de la carte, on cherche parmi toutes les villes de la carte les coordonnées minimum et maximum pour les villes et on les convertit pour que toutes les villes tiennent dans notre fenêtre graphique.

la ville de départ a pour coordonnée (15000,550) ce qui est supérieur a notre fenêtre, pour preuve on ne vois pas de point vert dans la carte, or pour la visite on met a l'échelle et nous conservons les distances.





vii) Include

stocke tout ce qui est "inclure dans les fichiers"

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <time.h>
#include <unistd.h>
#include <math.h>
#include <MLV/MLV_all.h>

#define LARGEUR_FENETRE 1000
#define HAUTEUR_FENETRE 600
#define MAX 728
#define TAILLE_POINT 3
#define BORDURE 10
#define TAILLE_POPULATION 1000
```

IV) Conclusion

Nous avons réussi à nous rapprocher d'une meilleure solution pour le PVC en respectant la problématique. Nous avons réussi à approcher une meilleure solution avec une complexité en $O [n^2 \lg(n)]$.

On aurait pu améliorer ce programme :

- en essayant ce programme sur un cas réel, avec une vraie carte et des villes existantes, il faudra cependant prendre d'autres problèmes en compte (comme les terrains non accessibles par le voyageur, etc.).

V) Problèmes rencontrés

- Les options, qui ont été corrigées grâce au dernier TD de Programmation 4 sur la fonction getopt().
- Nous avons eu des problèmes avec la mise à l'échelle des coordonnées, mais nous l'avons résolu en tp.

VI) Annexe

- Nous avons utilisé les ressources des TP et des années précédentes.
- Et l'aide du chargé de TD Michel Landschoot.