

Projet Génétique

Diaz Raphael - Pascal Florian - Camin Enzo

YNOV Bordeaux - 24 décembre 2020

Réalisation d'un générateur d'un trajet optimal appliquant de la génétique.



Sommaire

I- Ligne de réflexion.....	3
A- Objectifs	3
B- Notre projet	3
C- Logiciels.....	3
II- La base de donnée et MYSQL.....	4
III- Le programme C++	11
A- Réflexion autour du programme	11
B- Structure de notre programme	11
C- Notre programme en détail	11
1- Temporaire1.hpp.....	11
2- Les « include ».....	12
3- La connexion à notre BDD	12
4- Création d'une liste	13
5- L'affichage	14
6- main.cpp.....	15
7- Résultat de l'exécution	16
IV- Remarques et Conclusion	17
A- Remarques	17
B- Conclusion.....	17

I- Ligne de réflexion

A- Objectifs

Le projet a pour but de concevoir un programme utilisant de la génétique pour aboutir à une optimisation d'une liste de données. Le sujet est libre, sans contrainte, ni cahier des charges.

B- Notre projet

Notre projet a pour but la conception d'un programme permettant la génération d'un trajet entre deux villes selon des critères d'optimisation bien précis : la distance, la vitesse, la consommation. Pour cela, nous avons convenu de l'utilisation d'une base de donnée. L'ajout de la génétique doit intervenir après la génération de cinq listes aléatoires de trajet entre deux points. Bien évidemment, les trajets doivent suivre une succession de routes réelles et qui sont liées les unes avec les autres. Le trajet doit être optimal, chaque paramètres doivent être optimaux.

C- Logiciels

Le projet devait être réalisé sur QT creator en « c++ ». La base de donnée est conçue et réalisée via SQL. Cependant, après de multiples essais et manipulations, nous avons déduit que notre projet est infaisable sur QT Creator. Effectivement, la compatibilité de SQL et Qt Creator est quasi nulle. Nous nous sommes rabattus sur un autre compilateur qui est Code Blocks en version 20.03. Ce logiciel permet de programmer en c++ et d'intégrer SQL à notre code.

II- La base de donnée et MySQL

Nous avons décidé de réaliser notre propre base de donnée, celle-ci n'est donc pas disponible sur internet.

Pour créer notre BDD manuellement, nous sommes partis dans l'idée de récupérer toutes les routes les plus importantes de France. Pour ce faire nous avons décidé de récupérer ces routes avec la carte de géoportail ([https://www.geoportail.gouv.fr/carte?c=-0.1328513839260712,45.28136543834333&z=9&i0=ORTHOIMAGERY.ORTHOPHOTOS:WMTS\(1\)&i1=TRANSPORTNETWORKS.ROADS:WMTS\(1\)&i2=GEOGRAPHICALGRIDSYSTEMS.MAPS.SCAN-EXPRESS.STANDARD::GEOPORTAIL:OGC:WMTS\(1\)&permalink=yes](https://www.geoportail.gouv.fr/carte?c=-0.1328513839260712,45.28136543834333&z=9&i0=ORTHOIMAGERY.ORTHOPHOTOS:WMTS(1)&i1=TRANSPORTNETWORKS.ROADS:WMTS(1)&i2=GEOGRAPHICALGRIDSYSTEMS.MAPS.SCAN-EXPRESS.STANDARD::GEOPORTAIL:OGC:WMTS(1)&permalink=yes))

Exemple : pour la route allant de Bordeaux à Périgueux nous utilisons la N89 pour relier Bordeaux à Libourne, puis l'A89 pour relier Libourne à Périgueux :



Dans notre base de donnée, voici ce que donne cette route :

```
(124, 'Bordeaux', 'N89', 'Libourne', 39, 70, 28, 139, 72, 93),  
(333, 'Libourne', 'A89', 'Périgueux', 99, 100, 81, 122, 82, 101),
```

Voici la composition de notre base de donnée :

```
CREATE TABLE `routefr` (
  `id` int(11) NOT NULL,
  `ville_A` varchar(23) NOT NULL,
  `Route` varchar(10) NOT NULL,
  `ville_B` varchar(23) NOT NULL,
  `distance` int(11) DEFAULT NULL,
  `vitesse` int(5) DEFAULT NULL,
  `distance_vol_oiseau` int(11) DEFAULT NULL,
  `distance_optimal` int(11) DEFAULT NULL,
  `conso_optimal` int(11) DEFAULT NULL,
  `fitness` int(11) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

- **Ville_A** correspond au départ de notre route
- **Route** correspond à la route utilisée
- **Ville_B** correspond à la ville d'arrivée de notre route
- **Distance** correspond à la distance entre la Ville_A et la Ville_B
- **Vitesse** correspond à la vitesse réglementaire en lien avec le code de la route
- **Distance_vol_oiseau** correspond à la distance entre 2 villes en ligne droite
- **Distance_optimal**
- **Conso_Optimal**
- **Fitness** correspond à la moyenne des 3 critères : Distance_Optimal ; Conso_Optimal ; Vitesse.

Voici l'explication d'une ligne de notre Base de donnée :

```
(124, 'Bordeaux', 'N89', 'Libourne', 39, 70, 28, 139, 72, 93),
```

- **124** correspond à l'id de notre route.
- '**Bordeaux**' correspond à notre ville de départ.
- '**N89**' correspond à la route utilisée.
- '**Libourne**' correspond à notre ville d'arrivée.
- **39** correspond au nombre de kilomètres entre ces 2 villes en utilisant la N89.
- **70** correspond à un produit en croix qui calcul le pourcentage pour une vitesse optimale. Nous avons décidé de partir sur le fait que 130 km/h correspond à 100% et que sur une route nationale, nous roulons à 90 km/h. Si nous réalisons ce produit en croix nous obtenons un pourcentage de 70%.
- **28** correspond à la distance en vol d'oiseau : une ligne droite entre les 2 villes qui correspond à une route optimale car elle ne rencontre aucun problème.
- **139** correspond à un produit en croix qui calcul le pourcentage pour une distance optimale. Nous avons décidé de partir sur le fait que la distance en vol d'oiseau correspond à une distance optimale. Ce qui représente un pourcentage de 100%. La distance que nous trouvons ici est 39 (géopointail). Après un produit en croix : $(\text{Distance} * 100) / \text{Distance_vol_oiseau} = 139\%$ pour notre exemple.
- **72** correspond à un pourcentage lui aussi. Nous avons utilisé ce site : <https://calculis.net/consommation#resultat>, pour nous permettre de calculer la consommation optimale en pourcentage. Pour notre exemple, voici ce que nous trouvons avec ce site : pour le nombre de kilomètre nous regardons simplement dans notre base de donnée.

Nombre de km (ou miles) parcourus :

39

Nombre de Litres (ou gallons) :

50

Prix d'un Litre (gallon) de carburant :

1.35

Litre et km gallon et mile

Calculer la consommation

chaque km parcouru vous avez consommé 1.28 L

Nombre de km (ou miles) parcourus :

28

Nombre de Litres (ou gallons) :

50

Prix d'un Litre (gallon) de carburant :

1.35

Litre et km gallon et mile

Calculer la consommation

chaque km parcouru vous avez consommé 1.79 L

Ensuite, nous avons décidé que la consommation durant un trajet à vol d'oiseau correspondait à 100% , pour connaître la consommation optimale, il fallait réaliser un produit en croix : (Conso_distance_normal * 100) / Conso_distance_vol_oiseau. Pour notre exemple $(1.28 \times 100) / 1.79 = 72\%$

- **93** correspond simplement à notre fitness. Notre fitness se calcul simplement en faisant la moyenne de la vitesse, de la distance_optimale et de la consommation optimale. Pour notre exemple $(70+139+72) / 3 = 93\%$.

Quelques explications concernant la vitesse :

- 130 Km/h = **100%**
- 90 Km/h = **70%**
- 50 Km/h = **40%**

Ensuite pour :

- Une Autoroute + une Nationale = **85%**
- Une Nationale + une Départementale = **55%**
- Une Autoroute + une Départementale = **70%**

Afin de pouvoir faire fonctionner notre base de donnée, nous sommes passés par EasyPHP Devserver qui propose un ensemble d'applications permettant de faire fonctionner des scripts PHP en local.

The screenshot shows the main interface of the EasyPHP Devserver. At the top, there's a navigation bar with links for applications, tools, support, documentation, website, and warehouse. A bell icon is also present. Below the navigation bar, there are two main sections: 'HTTP SERVER' and 'DATABASE SERVER'. The 'HTTP SERVER' section shows Apache 2.4.25 x86 - PHP 5.6.30 x86 running on Port 80. The 'DATABASE SERVER' section shows MySQL 5.7.17 x86 running on Port 3306. Both sections have 'stop', 'refresh', 'settings', and 'restart' buttons. Below these sections, there's a 'WORKING DIRECTORIES' section where users can organize their working environment by listing working directories. It includes a 'Portable Directory' entry pointing to C:\Program Files (x86)\EasyPHP-Devserver-17\eds-www. There's also a '+ add directory' button. The 'MODULES' section shows MySQL Administration : PhpMyAdmin 4.7.0, with an 'open' button and a '+ add modules' button.

EasyPHP est un environnement de développement composé de deux serveurs : un serveur Web Apache et un serveur de base de données MySQL. De plus, il possède un module d'administration SQL nommé PhpMyAdmin qui permet ainsi de faire fonctionner toutes sortes de scripts PHP et notamment de vérifier le bon fonctionnement des scripts.

Pour créer notre base de donnée, nous avons réalisé le cheminement suivant :

- 1 – Lancer EasyPHP
- 2 – Ouvrir les serveurs (HTTP SERVER et DATABASE SERVER)
- 3 – Ouvrir un invité de commande
- 4 – Renseigner le chemin afin d'utiliser notre fichier .sql :

A screenshot of a file explorer window showing the path: « Programmes (x86) > EasyPHP-Devserver-17 > eds-binaries > dbserver > mysql5717x86x201023103138 > bin ». This indicates the location of the MySQL binary files used for the connection.

```
cd C:\Program Files (x86)\EasyPHP-Devserver-17\eds-binaries\dbserver\mysql5717x86x201023103138\bin
```

5 – Connexion à MYSQL :

```
mysql -hlocalhost -uroot -p (password)
```

6 – Entrer le mot de passe (s'il y en a un de configurer)

7 – Création de la base de donnée :

```
mysql> create database gps;
Query OK, 1 row affected (0.00 sec)
```

Notre base de donnée est donc créée mais vide, il ne manque plus qu'à importer notre fichier « .sql » que nous avons préalablement placé dans le dossier « bin » de notre dossier « EasyPHP Devserver » afin d'implémenter nos tables ainsi que les valeurs à l'intérieur. Pour importer notre fichier la manipulation à suivre est :

```
Microsoft Windows [version 10.0.19041.630]
(c) 2020 Microsoft Corporation. Tous droits réservés.

C:\Users\enzoc>cd C:\Program Files (x86)\EasyPHP-Devserver-17\eds-binaries\dbserver\mysql5717x86x201023103138\bin

C:\Program Files (x86)\EasyPHP-Devserver-17\eds-binaries\dbserver\mysql5717x86x201023103138\bin>mysql -hlocalhost -uroot -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 28
Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use gps;
Database changed
mysql> \. gps_poo.sql
Query OK, 0 rows affected (0.00 sec)
```

Afin de vérifier si la manipulation s'est bien passée en restant sur le cmd, nous pouvons afficher les tables de notre base de donnée avec un « show tables » :

```
mysql> show tables;
+-----+
| Tables_in_gps |
+-----+
| commune      |
| routefr     |
+-----+
2 rows in set (0.00 sec)
```

Ensuite, nous pouvons retourner sur EasyPHP, ouvrir PhpMyAdmin afin de vérifier si nos tables comportent bien nos données :

Nous pouvons aussi faire apparaître une de nos tables via le cmd en tapant la requête sql :

```
mysql> use gps
Database changed
mysql> select * FROM routefr;
+---+---+---+---+---+---+---+---+---+---+
| id | ville_A | Route | ville_B | distance | vitesse | distance_vol_oiseau | distance_optimal | conso_optimal | fitness |
+---+---+---+---+---+---+---+---+---+---+
| 1 | Nantes | A11 | Angers | 88 | 100 | 81 | 108 | 92 | 100 |
| 2 | Nantes | A83 | Niort | 143 | 100 | 130 | 110 | 92 | 100 |
| 3 | Niort | A10 | Poitiers | 80 | 100 | 68 | 117 | 85 | 100 |
| 4 | Niort | A10 | Saintes | 77 | 100 | 66 | 116 | 86 | 100 |
| 5 | Poitiers | A10 | Tours | 105 | 100 | 94 | 116 | 90 | 102 |
| 6 | Tours | A28 | Le Mans | 100 | 100 | 78 | 128 | 78 | 102 |
| 7 | Tours | A10 | Orleans | 118 | 100 | 188 | 189 | 91 | 100 |
| 8 | Orleans | A71 | Bourges | 125 | 100 | 99 | 126 | 78 | 100 |
| 9 | Bourges | A85 | Tours | 165 | 100 | 134 | 123 | 81 | 101 |
| 10 | Bourges | A71 | Clermont-Ferrand | 191 | 100 | 154 | 124 | 81 | 101 |
| 11 | Orleans | A19 | Courtenay | 115 | 100 | 87 | 132 | 75 | 100 |
| 12 | Courtenay | A6 | Beaune | 197 | 100 | 150 | 131 | 76 | 102 |
| 13 | Beaune | A31 | Dijon | 47 | 100 | 37 | 127 | 78 | 101 |
| 14 | Courtenay | A19-A5 | Troyes | 99 | 100 | 80 | 123 | 81 | 101 |
| 15 | Troyes | A31-A5 | Dijon | 185 | 100 | 130 | 142 | 71 | 104 |
| 16 | Dijon | A31 | Nancy | 217 | 100 | 175 | 124 | 79 | 101 |
| 17 | Beaune | A6 | Macon | 87 | 100 | 80 | 108 | 90 | 99 |
| 18 | Dijon | A39-A36 | Besancon | 98 | 100 | 75 | 130 | 81 | 103 |
| 19 | Besancon | A36 | Montbeliard | 79 | 100 | 66 | 119 | 83 | 100 |
| 20 | Montbeliard | A36 | Mulhouse | 60 | 100 | 49 | 122 | 81 | 101 |
| 21 | Mulhouse | A35 | Strasbourg | 114 | 100 | 98 | 116 | 86 | 100 |
| 22 | Courtenay | A19-A77 | Cosne | 116 | 100 | 106 | 109 | 91 | 100 |
| 23 | Macon | A40-A39 | Lons-le-Saunier | 102 | 100 | 69 | 147 | 68 | 105 |
| 24 | La Roche-sur-Yon | A87 | Cholet | 70 | 100 | 60 | 116 | 86 | 100 |
| 25 | La Roche-sur-Yon | A87-A83 | Niort | 90 | 100 | 83 | 108 | 93 | 100 |
| 26 | Rochefort | A837-A10 | Niort | 62 | 100 | 57 | 108 | 92 | 100 |
| 27 | Nantes | N249 | Cholet | 64 | 70 | 54 | 118 | 84 | 90 |
| 28 | Bressuire | N149 | Parthenay | 32 | 70 | 29 | 110 | 91 | 90 |
| 29 | Parthenay | N149 | Migne-Auxances | 45 | 70 | 43 | 104 | 96 | 90 |
| 30 | Migne-Auxances | N147 | Poitiers | 11 | 70 | 6 | 183 | 55 | 102 |
+---+---+---+---+---+---+---+---+---+---+
```

Notre base de donnée est maintenant prête à être utilisée.

III- Le programme C++

A- Réflexion autour de la création du programme

Afin de bien concevoir notre programme à partir de notre base de donnée, nous avons exploré plusieurs possibilités et essayé différentes approches. Pour commencer, nous voulons créer un trajet cohérents entre deux points, quelque soit sa longueur et sa cohérence. Pour cela nous devons donc réaliser plusieurs requêtes SQL et stocker nos résultats dans une liste ou un tableau. Bien évidemment, l'utilisation d'un tableau dynamique nous semble le plus adapté à notre projet. Après avoir généré un tableau dynamique contenant notre trajet, il faut généraliser notre programme pour qu'il puisse générer cinq trajets dans cinq tableaux différents afin d'avoir la possibilité d'utiliser la génétique.

B- Structure de notre programme

Notre programme est découpé en trois fichiers : un « main.cpp », un « Temporaire1.cpp » et « Temporaire1.hpp ». Dans notre fichier « Temporaire1.hpp » nous avons donc sept fonctions : cinq pour la création de liste, une pour la connexion et une pour l'affichage contenu dans une classe « temporaire1 ». Le « int main » est utilisé pour appeler nos fonctions et notre classe.

C- Notre programme en détail

1- Temporaire1.hpp

```
class Temporaire1
{
private:
    MYSQL_ROW row;
    MYSQL_ROW row2;
    MYSQL_ROW row3;
    MYSQL_ROW row4;
    MYSQL_ROW row5;

    MYSQL_RES* res;
    MYSQL_RES* res2;
    MYSQL_RES* res3;
    MYSQL_RES* res4;
    MYSQL_RES* res5;

    MYSQL_RES* res10;
    MYSQL_RES* res11;
    MYSQL_RES* res12;
    MYSQL_RES* res13;
    MYSQL_RES* res14;
    MYSQL_RES* res15;

public:
    string m, a, b, c, d, e, f;
    string qstate0, qstate1, qstate2, qstate3, qstate4, qstate5, qstate6, qstate7, qstate8;
    string str10, str11, str12, str13, str14;
    void liste1();
    void liste2();
    void liste3();
    void liste4();
    void liste5();
    void affichage();
    void connection();
};
```

Voici le détail de notre classe « Temporaire1 ». Les variables en « private » sont propres à MYSQL, cela permet la création de requêtes et la récupération des résultats. Il faut noter que l'ensemble de ces variables doivent être utilisées dans des requêtes différentes. Les fonctions et variables en « public » sont utilisées également dans les requêtes.

2- Les « include »

```
#ifndef TEMPORAIRE1_H
#define TEMPORAIRE1_H
#include <mysql.h>
#include <vector>
#include <iostream>
#include <windows.h>
using namespace std;
```

Ici nous avons donc :

- <mysql> qui permet d'utiliser toutes les commandes de MYSQL et ainsi faire nos requêtes.
- <vector> permet la création de notre tableau dynamique.
- <iostream> permet le bon fonctionnement du programme c++.
- « using namespace std » permet de simplifier l'écriture de notre programme.

3- La connexion à notre BDD.

```
void Temporaire1::connection() {
    conn = mysql_init(0);
    conn = mysql_real_connect(conn, "localhost", "root", "", "gps", 0, NULL, 0);

    if (conn)
    {
        cout<<"OK"<<endl;
    }
    else
    {
        cout<<"NO"<<endl;
    }
}
```

Ici nous utilisons une requête SQL afin de nous connecter à notre BDD. Nous vérifions ensuite notre connexion grâce à un If/else.

4- Crédation d'une liste.

```
11 | void Temporaire1::listel() {
12 |     Temporaire1();
13 |
14 |     m="Paris";
15 |     d="Bordeaux";
16 |     while(d!=m)
17 |     {
18 |         (
19 |
20 |
21 |         qstate = "SELECT * FROM routefr WHERE ville_A ='" + d + "' order by rand() limit 1";
22 |         mysql_query(conn,qstate.c_str());
23 |         res = mysql_store_result(conn);
24 |         row=mysql_fetch_row(res);
25 |         row=mysql_fetch_row(res);
26 |
27 |         a=row[0];
28 |         b=row[1];
29 |         c=row[2];
30 |         d=row[3];
31 |         e=row[9];
32 |
33 |
34 |
35 |         gstate1 = "INSERT INTO temporaire1 (id, ville_A, Route, ville_B,fitness) VALUES ('" + a + "','" + b + "','" + c + "','" + d + "','" + e + "')";
36 |         mysql_query(conn,gstate1.c_str());
37 |
38 |
39 |         f = a + " " + b + " " + c + " " + d + " " + e;
40 |         tableau1.push_back(f);
41 |
42 |
43 |
44 |         str10 = mysql_query(conn,"DELETE FROM temporaire1");
45 |         res10 = mysql_use_result(conn);
46 |
47 |
48 |
49 |
50 |
51 |     }
52 |
53 | }
```

Activer Windows
Accédez aux paramètres pour activer

Dans un premier temps, nous initialisons notre ville grâce à la variable « m ». Bien évidemment, « m » est un type string. Le point de départ est la variable « d », celle-ci nous permet de rentrer dans la boucle. Cette boucle est nécessaire afin de réaliser un trajet cohérent jusqu'à la ville d'arrivée qui est « m ». Tant que la ville de départ « d » est différente de notre ville d'arrivée, on continue notre boucle, et donc notre trajet (ligne 14 à 16).

Ensuite, nous effectuons une requête qui permet de sélectionner de manière aléatoire une « ville_A » (Ville de départ) qui correspond à la chaîne de caractère de la variable « d » (ligne 22 à 25). Nous stockons ensuite les données correspondantes au résultat de notre requête dans une autre table « temporaire1 » (Permet de générer un trajet plus rapidement) (ligne 36-37). Enfin, nous réunissons les différentes données récupérées de la table « temporaire1 » dans une chaîne de caractère de variable « f ». Cela facilite l'affichage. Cette variable est ensuite stockée dans un tableau dynamique. Nous effaçons ensuite le contenu de la table « temporaire1 ». Nous arrivons à la fin de la boucle, la variable « d » a donc une nouvelle valeur (qui correspond à la précédente ville d'arrivée ou « ville_B »). Cela permet donc d'avoir une succession de routes jusqu'à la ville d'arrivée.

Les variables sont :

- « a » correspond à l'identifiant de la ligne dans la table
- « b » correspond à la ville_A.
- « c » correspond au nom de la route se situant entre les deux points.
- « d » correspond à la ville_B.
- « e » correspond au fitness.

Tout cela forme donc « une route ».

Si on veut cinq listes, il faut créer cinq fonctions différentes avec des requêtes différentes et des tableaux différents. L'utilisation de SQL en c++ est fragile. En effet, la réalisation de plusieurs requêtes SQL différentes doit faire intervenir des variables, noms de requêtes et variables de résultats différents, c'est pour cela que nous avons privilégié cette structure de code, surtout si nous souhaitons comme pour notre projet, obtenir des trajets aléatoires qui vont bien d'un point A à un point B de manière cohérente.

5- L'affichage

L'affichage de nos listes est classique, avec des « boucles for ».

```
void Temporaire1::affichage () {  
  
    cout<<"-----"  
    for (int n = 0 ; n < tableau1.size() ; n++) {  
  
        cout << tableau1[n]<<endl;  
    }  
  
    cout<<"-----"  
    cout<<"-----"  
    for (int n = 0 ; n < tableau2.size() ; n++) {  
  
        cout << tableau2[n]<<endl;  
    }  
    cout<<"-----"  
    cout<<"-----"  
    for (int n = 0 ; n < tableau3.size() ; n++) {  
  
        cout << tableau3[n]<<endl;  
    }  
  
    cout<<"-----"  
    cout<<"-----"  
    for (int n = 0 ; n < tableau4.size() ; n++) {  
  
        cout << tableau4[n]<<endl;  
    }  
    cout<<"-----"  
    cout<<"-----"  
    for (int n = 0 ; n < tableau5.size() ; n++) {  
  
        cout << tableau5[n]<<endl;  
    }  
    cout<<"-----"  
}
```

6- Main.cpp

Le « main » est relativement simple également, nous appelons notre classe « temporaire1 », puis les différentes fonctions permettant la création des listes. Ensuite nous appelons l'affichage de ces dernières. Il ne faut pas oublier d'inclure le « fichier .h ».

```
1 #include <iostream>
2 #include "Temporaire1.h"
3
4
5
6
7
8
9
10 using namespace std;
11
12 int main()
13 {
14     Temporaire1 T1;
15     T1.connection();
16     T1.liste1();
17     T1.liste2();
18     T1.liste3();
19     T1.liste4();
20     T1.liste5();
21     T1.affichage();
22
23
24
25
26
27 }
28
```

7- Résultat de l'exécution

Nous testons notre programme entre Bordeaux et Périgueux, ci-dessous UN trajet parmi cinq tous différents :

```
125 Bordeaux A10-N10 Angouleme 95
37 Angouleme N141 Cognac 90
272 Cognac N141 Angouleme 90
359 Angouleme N141/N1141 Saintes 91
39 Saintes N150 Royan 91
274 Royan N150 Saintes 91
334 Saintes A10 Bordeaux 100
67 Bordeaux A63-A660 Gujan-Mestras 102
137 Gujan-Mestras N250 Arcachon 106
370 Arcachon N250 Gujan-Mestras 106
137 Gujan-Mestras N250 Arcachon 106
370 Arcachon N250 Gujan-Mestras 106
300 Gujan-Mestras A63-A660 Bordeaux 102
124 Bordeaux N89 Libourne 93
333 Libourne A89 Perigueux 101
```

On peut noter le temps d'exécution du programme à 5s500. Nous avons donc générée cinq trajets complètement aléatoires et différents en 5s500.

```
Process returned 0 (0x0)    execution time : 5.500 s
Press any key to continue.
```

IV- Remarques et Conclusion

A- Remarques

L'utilisation de la génétique dans notre programme est impossible. En effet, la création de nos routes au départ, lors du remplissage de notre BDD à complètement empêché une implémentation de la génétique par la suite, même en changeant la composition de notre BDD, le résultat serait le même. Une route se compose forcement d'un point A, d'un point B et d'un nom de route : sans cela, la création d'un trajet entre routes connectées n'est pas possible. Notre méthode est donc l'unique solution pour accéder à un trajet aléatoire entre deux points. Quel est donc le problème ? L'utilisation de la génétique pour générer un trajet optimal entre deux points n'est pas possible. Ce n'est donc pas une application à la génétique. C'est pour cela que les GPS utilisent des données de guidage en temps réels via l'existence des satellites.

B- Conclusion

Le projet est abouti à 50%. Cela n'est ni dû à une mauvaise compréhension de la génétique ou du c++ en général, mais par la non-faisabilité du projet au commencement même de celui-ci. A ce moment là, nous ne pouvions pas le savoir, car tous les éléments de notre projet ont été créé de nos mains, sans aide ni conseil extérieur. Avec du temps supplémentaire, un projet de recherche autour de la génétique dans la génération de trajet optimaux serait envisageable.