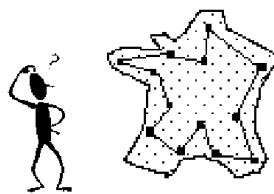


TP systèmes immunitaires artificiels

Programmation du problème du voyageur de commerce

1 Le problème du voyageur de commerce

Un voyageur de commerce doit visiter n villes données en passant par chaque ville exactement une fois. Il commence par une ville quelconque et termine en retournant à la ville de départ. Les distances entre les villes sont connues. Quel chemin faut-il choisir afin de minimiser la distance parcourue ? La notion de distance peut-être remplacée par d'autres notions comme le temps qu'il met ou l'argent qu'il dépense : dans tous les cas, on parle de coût.



1.1 Algorithmes déterministes

Il existe des algorithmes déterministes permettant de trouver la solution optimale pour n villes, mais la complexité de ces algorithmes est de l'ordre de $O(n!)$. Ces algorithmes sont très gourmands en puissance de calcul. Par exemple, il a faut plus d'une journée à un puissant super-calculateur pour trouver la solution optimale pour 2392 villes.

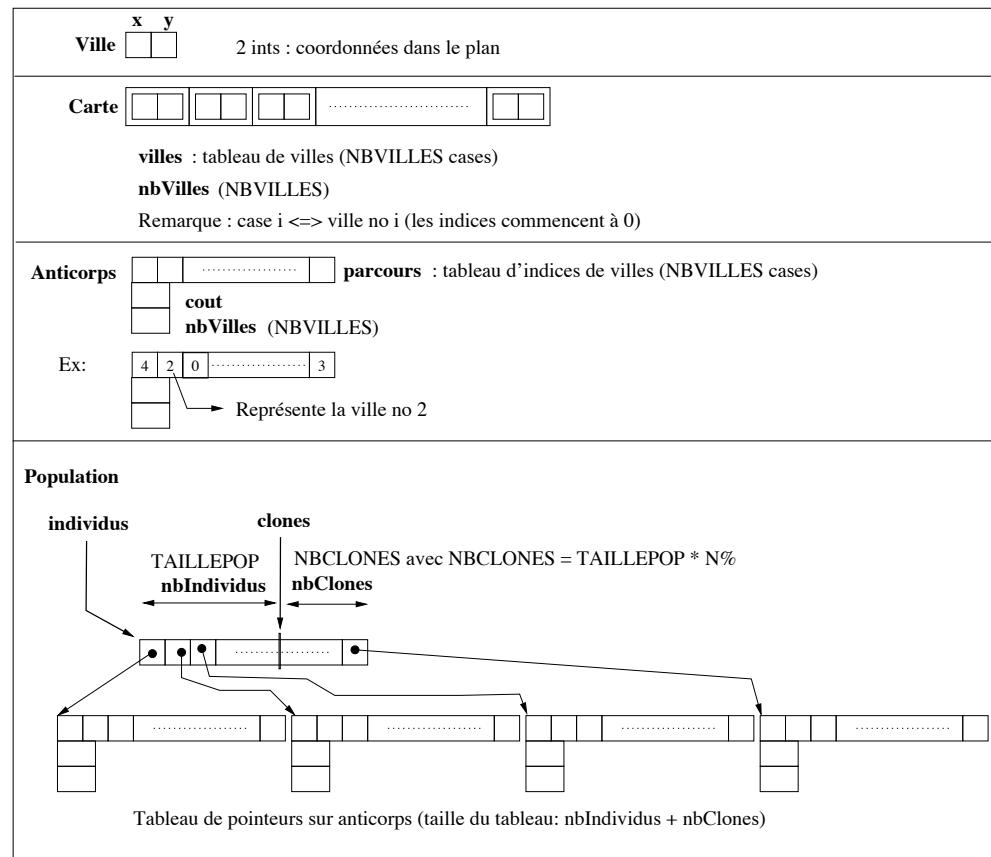
1.2 Algorithmes d'approximation

Les algorithmes d'approximation permettent de trouver une solution dont le coût est proche du coût de la solution optimale. Ils ont l'avantage de permettre en un temps raisonnable de trouver une solution. De ce fait, ils ne sont à utiliser que dans les cas où une solution aprochée est acceptable.

Il existe de nombreux algorithmes d'approximation pour résoudre le problème du voyageur de commerce. Ainsi, il existe des méthodes utilisant les algorithmes génétiques, le recuit simulé, les colonies de fourmis, ... et les systèmes immunitaires artificiels.

Nous nous proposons ici de programmer une résolution du problème du voyageur de commerce à l'aide d'un système immunitaire artificiel utilisant l'algorithme par sélection clonale.

2 Structure de données proposée



3 Travail à effectuer

Un squelette de programme est donné. Il comporte les fichiers .h/.c suivants :

- geo.h/geo.c : gestion de la géographie, c'est-à-dire des villes et de la carte.
- anticorps.h/anticorps.c : gestion des anticorps.
- population.h/population.c : gestion de la population.
- ais.c : le programme principal avec la déclaration de la carte et de la population.
On trouve aussi la boucle d'itérations (générations).

- `params.h` : fichier de configuration où l'on trouve le nombre de villes (`NBVILLES`).

On trouve également dans ce fichier des valeurs par défaut de certains paramètres :

- ◊ `TAILLEPOP` : taille de la population (sans les clones)
- ◊ `N` : lors de la sélection des meilleurs individus pour le clonage, `N` représente le % de la population considérée.
 ⇒ `NBCLOONES`
- ◊ `D` : lors de l'injection de nouveaux individus, `D` représente le % de la population remplacée.
 ⇒ `NBNOUVEAUX`
- ◊ `NBGENERATIONS`, le nombre de générations (le nombre de tours de boucle!).
- ◊ `NBGENERATIONSINJECTION`, le nombre de tours de boucle après lequel il faut faire une injection de sang frais (injection de nouveaux individus).

Remarque :

les valeurs (par défaut) de `TAILLEPOP`, `N`, `D`, `NBGENERATIONS` et `NBGENERATIONSINJECTION` peuvent être remplacées par les valeurs des arguments passés au niveau de la ligne de commande.

Ainsi, si on a dans `params.h`:

```
#define TAILLEPOP      100
#define N              50
#define D              20
#define NBGENERATIONS   100
#define NBGENERATIONSINJECTION 20
```

Exemples de lancement du programme `ais`:

```
$ ./ais
...
nbIndividus=100
N=50 => nbClones=50
D=20 => nbNouveaux=20
nbGenerations=100
nbGenerationsInjection=20
...

$ ./ais30 300 80 20 500 30
...
nbIndividus=300
N=80 => nbClones=240
D=20 => nbNouveaux=60
nbGenerations=500
nbGenerationsInjection=30
...
$
```

- `random.h/random.c` : quelques fonctions pour la génération de nombres aléatoires.

- `gnuplot.h/gnuplot.c` : quelques fonctions pour envoyer des ordres d'affichage à `gnuplot`.

3.1 Compléter le code de certaines fonctions

Les codes de certaines fonctions clés de l'algorithme ne sont pas donnés.

Ainsi, il faut compléter :

- `muteAc` ; dans le fichier `anticorps.c`
- `mutationClones`, `selectionMeilleursEtClonesMutes`, `mutationMonBons` et `remplacementMauvaisParNouveaux` ; dans le fichier `population.c`
- la boucle d'itération dans le `main` ; dans le fichier `ais.c`

3.2 Evaluation de l'influence des paramètres

Avec votre programme (ou avec les exécutables `ais8`, `ais16` et `ais30`), évaluer l'influence des différents paramètres (nombre d'individus, nombre de générations, etc...).

3.3 Rédaction d'un compte rendu + Fichiers

Un compte rendu de votre travail devra être rendu au format pdf. Il faudra également rendre le code informatique de votre travail.

Pour créer une archive d'un répertoire REP :

```
$ cd REP
$ pwd
.../REP
$ make clean          # permet d'enlever les .o et l'executable ais
$ cd ..
$ tar cvf REP.tar REP
$ gzip -9 REP.tar      # creation de REP.tar.gz
$
```

Fonctions à faire:

Dans `ais.c`, la boucle principale.
 Dans `anticorps.c`, `muteAc`.
 Dans `population.c`, `mutationClones`.
 Dans `population.c`, `selectionMeilleursEtClonesMutes`.
 Dans `population.c`, `mutationMoinsBons`.
 Dans `population.c`, `remplacementMauvaisParNouveaux`.

Et ensuite (ou bien avant) tester:

`ais8`, `ais16`, `ais30`
 pour évaluer l'influence des paramètres.

```
# Fichier makefile pour ais
#-----
CC=cc

TARGET=ais
FILES=ais.c anticorps.c geo.c population.c random.c gnuplot.c
OBJECTS=$(FILES:.c=.o)
CLEANING=rm -f $(OBJECTS) $(TARGET) core a.out

#-----

$(TARGET): $(OBJECTS)
    $(CC) $(OBJECTS) -o $(TARGET) -lm

#-----

ais.o: params.h ais.c
    $(CC) -c ais.c -o ais.o

anticorps.o: params.h anticorps.c
    $(CC) -c anticorps.c -o anticorps.o

geo.o: params.h geo.c
    $(CC) -c geo.c -o geo.o

population.o: params.h population.c
    $(CC) -c population.c -o population.o

#-----

.c.o :
    $(CC) -c $< -o $*.o

clean:
    $(CLEANING)
clear:
    $(CLEANING)
```

```
/* gnuplot.h */

#ifndef GNUPLOT_H
#define GNUPLOT_H

#include <stdio.h>

/*********************************************************
***** Affichage avec gnuplot et un tube *****/
/*********************************************************/



typedef struct { float x,y; } PointGnuplot;

extern FILE* openGnuplot(char *fileName);
extern void closeGnuplot(FILE* flot);

extern void setAutoscaleGnuplot(FILE* flot);

extern void setRangeGnuplot(FILE* flot,
                           float xmin, float xmax, float ymin, float ymax);

extern void beginPointsToGnuplot(FILE* flot,
                                 char *style); /* "lines" , "linespoint" ,
                                              "points"
                                              */

extern void pointsToGnuplot(FILE* flot,
                           PointGnuplot tabPoint[], int nbPoints);

extern void endPointsToGnuplot(FILE* flot);

extern void vectorGnuplot(FILE* flot,
                          float x1, float y1, float x2, float y2);

#endif /* GNUPLOT_H */
```

```
/* random.h : generation de nombres aleatoire */

#ifndef RANDOM_H
#define RANDOM_H

/*********************************************************
***** Generation de nombres aleatoires *****/
/*********************************************************/



extern void initRandom(void); /* Initialisation generateur */
extern int myRandomMinMax(int min, int max); /* Generation dans [min,max] */
extern double myRandom01(void); /* Generation dans [0.0,1.0] */

#endif /* RANDOM_H */
```

```
#ifndef PARAMS_H
#define PARAMS_H

#define NBVILLES 30
#define COTECARTE 10

/* DANS LA SUITE, NE PAS TOUCHER:
   TAILLEPOP, N, D, NBGENERATIONS, NBGENERATIONSINJECTION !!!
   ==> agir sur les parametres de la ligne de commande */

#define TAILLEPOP 100
#define N 50 /* Selection: En % de TAILLEPOP (Max 100%) */
#define D 20 /* Injection: En % de TAILLEPOP (Max 100%) */

#define NBGENERATIONS 100
#define NBGENERATIONSINJECTION 20

#define NBCLONES (N*TAILLEPOP)/100
#define NBNOUVEAUX (D*TAILLEPOP)/100

/*********************************************
***** Verification contraintes sur les defines *****/
/********************************************/


#if NBVILLES>COTECARTE*COTECARTE
#error "Attention: NBVILLES>COTECARTE*COTECARTE"
#endif

#if N>100
#error "Attention: N trop grand"
#endif

#if D>100
#error "Attention: D trop grand"
#endif

#define ERROR_PARAM 0
#if TAILLEPOP!=100
#error "Attention: params.h, ne pas toucher TAILLEPOP !"
#error "==> agir sur les parametres de la ligne de commande"
#endif
#if N!=50
#error "Attention: params.h, ne pas toucher N !"
#error "==> agir sur les parametres de la ligne de commande"
#endif
#if D!=20
#error "Attention: params.h, ne pas toucher D !"
#error "==> agir sur les parametres de la ligne de commande"
#endif
#if NBGENERATIONS!=100
#error "Attention: params.h, ne pas toucher NBGENERATIONS !"
#error "==> agir sur les parametres de la ligne de commande"
#endif
#if NBGENERATIONSINJECTION!=20
#error "Attention: params.h, ne pas toucher NBGENERATIONSINJECTION !"
#error "==> agir sur les parametres de la ligne de commande"
#endif

#endif /* PARAMS_H */
```

```
/* geo.h : la geographie */

#ifndef GEO_H
#define GEO_H

/* On utilise NBVILLES (genereCarte) */

/*********************************************
***** Les Villes *****/
/********************************************/


typedef struct { int x;
                 int y;
             } Ville;

extern void genereVille(Ville *ville, int coteCarte);
extern void printVille(const Ville *ville);
extern void dessineVille(FILE* flot, const Ville *ville);
extern void dessineUneSeuleVille(FILE* flot, const Ville *ville);
extern double distanceVilles(const Ville *ville1, const Ville *ville2);

/*********************************************
***** Une Carte *****/
/********************************************/


typedef struct { Ville villes[NBVILLES];
                  int nbVilles;
              } Carte;

extern void genereCarte(Carte *carte, int coteCarte);
extern void printCarte(const Carte *carte);
extern void dessineCarte(FILE* flot,const Carte *carte);

#endif /* GEO_H */
```

```

/* anticorps.h : les anti-corps */

#ifndef ANTICORPS_H
#define ANTICORPS_H

#include "params.h"

/*
   Dans anticorps.c, il y a un passage de parametre pas tres joli!
   extern Carte carte; (calculCoutAc, dessineParcoursAc)
*/

/* On utilise NBVILLES (genereAc) */

/********************* Les Anti-corps **********************/

typedef struct { int parcours[NBVILLES];
                 double cout;
                 int nbVilles;
             } Ac;

extern void genereAc(Ac *ac);
extern void calculCoutAc(Ac *ac);
extern void printAc(const Ac *ac);
extern void printParcoursAc(const Ac *ac);
extern void printCoutAc(const Ac *ac);
extern void dessineAc(FILE* flot,const Ac *ac);
extern void dessineParcoursAc(FILE* flot,const Ac *ac);
extern void cloneAc(const Ac *ac,Ac *nouvelAc);
extern void muteAc(Ac *ac, int nbMutations);

#endif /* ANTICORPS_H */

```

```

/* anticorps.c : les anti-corps */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
/* Pour fabs */
/* Pour malloc, free */
/* Pour memmove */

#include "params.h"
#include "geo.h"
#include "anticorps.h"
#include "random.h"
#include "gnuplot.h"

extern Carte carte; /* Un passage de parametre pas tres joli ! */
/* Pour calculCoutAc et dessineParcoursAc */

/* On utilise NBVILLES (genereAc) */

/********************* Les Anti-corps **********************/

/*
typedef struct { int parcours[NBVILLES];
                 double cout;
                 int nbVilles;
             } Ac;

/* Generation aleatoire du parcours d'un Anti-Corps */
void genereAc(Ac *ac)
{ [REDACTED] }

/* Calcul du cout du parcours d'un Anti-Corps */
void calculCoutAc(Ac *ac)
{ [REDACTED] }

/* Affichage d'un Anti-Corps */
void printAc(const Ac *ac)
{ [REDACTED] }

/* Affichage du parcours d'un Anti-Corps */
void printParcoursAc(const Ac *ac)
{ [REDACTED] }

/* Affichage du cout d'un Anti-Corps */
void printCoutAc(const Ac *ac)
{ [REDACTED] }

/* Dessin d'un Anti-Corps */
void dessineAc(FILE* flot,const Ac *ac)
{ [REDACTED] }

/* Dessin du parcours d'un Anti-Corps */
void dessineParcoursAc(FILE* flot,const Ac *ac)
{ [REDACTED] }

/* Clonage d'un Anti-Corps */
void cloneAc(const Ac *ac,Ac *nouvelAc)
{ *nouvelAc = *ac; }

/* Mutation d'un Anti-Corps */
void muteAc(Ac *ac,int nbMutations)
{
    (void)ac; (void)nbMutations;
}

/* A completer ... */
}
```

```
/* population.h : la population d'anticorps */

#ifndef POPULATION_H
#define POPULATION_H

#include "params.h"
#include "anticorps.h"

/********************* Une Population ********************/
/********************* Une Population ********************/
/********************* Une Population ********************/

typedef struct { Ac* *individus; /* Tableau alloue dynamiquement d'Ac* */
                 Ac* *clones; /* Pointeur vers la zone des clones */
                 int nbIndividus;
                 int nbClones;
} Population;

extern void generePopulation(Population *population,
                             int nbIndividus,int nbClones);
extern void supprimePopulation(Population *population);

/* Affichage des meilleurs individus de la Population */
/* Pour afficher toute la population => nbIndividus=0 */
extern void printIndividusPopulation(Population *population,int nbIndividus);
/* Dessin des meilleurs individus de la Population */
/* Pour dessiner toute la population => nbIndividus=0 */
extern void dessineIndividusPopulation(FILE* flot,
                                         Population *population,int nbIndividus);

/* Tri d'une Population avec un COUT DECROISSANT =>
   les moins bon a gauche, les meilleurs sont a droite. */
extern void triPopulation(Population *population);

/** Il faut un tri avant (meilleur individu le plus a droite). */
extern Ac meilleurIndividu(Population *population);

/** Il faut un tri avant (les meilleurs doivent etre a droite). */
extern void clonageMeilleurs(Population *population);

/** Il faut un clonage avant .... */
extern void mutationClones(Population *population);

/** Les meilleurs et les clones sont comparares */
/** Chaque couple (meilleur,clone) est evalue...
   Et on garde le meilleur des deux */
extern void selectionMeilleursEtClonesMutes(Population *population);

/** Les moins bons doivent etre a gauche (apres un tri par exemple) */
extern void mutationMoinsBons(Population *population);

/** Les mauvais doivent etre a gauche (apres un tri par exemple) */
extern void remplacementMauvaisParNouveaux(Population *population,
                                             int nbNouveaux);

#endif /* POPULATION_H */
```

```

/* population.c : la population d'anticorps */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <unistd.h>

#include "params.h"
#include "anticorps.h"
#include "population.h"

/*********************************************************/
/* Une Population */
/*********************************************************/

/*
typedef struct { Ac* *individus; // Tableau alloue dynamiquement d'Ac*
    Ac* *clones; // Pointeur vers la zone des clones
    int nbIndividus;
    int nbClones;
} Population;

/* Generation d'une Population */
void generePopulation(Population *population,
                      int nbIndividus,int nbClones)
{ [REDACTED] }

/* Destruction d'une population: individus + clones */
void supprimePopulation(Population *population)
{ [REDACTED] }

/* Affichage des meilleurs individus de la Population */
/* Pour afficher toute la population => nbIndividus=0 */
void printIndividusPopulation(Population *population, int nbIndividus)
{ [REDACTED] }

/* Dessin des meilleurs individus de la Population */
/* Pour dessiner toute la population => nbIndividus=0 */
void dessineIndividusPopulation(FILE* flot,
                                 Population *population, int nbIndividus)
{ [REDACTED] }

/*-----*/
/* Tri d'une Sous Population et calcul du meilleur avec */
/* un COUT DECROISSANT => */
/* les moins bons sont a gauche, les meilleurs a droite */
static void triSousPopulation(Population *population,
                               int indiceDebut, int nbIndividusSousPopulation)
{ [REDACTED]
/*-----*/
void triPopulation(Population *population)
{
    triSousPopulation(population,0,population->nbIndividus);
}

/** Il faut un tri avant */
/* Recuperation du meilleur individu */
Ac meilleurIndividu(Population *population)
{ [REDACTED]
}

```

```

/** Il faut que la population soit triee ***/
/** (les meilleurs doivent etre a droite). ***/
void clonageMeilleurs(Population *population)
{
    int i = 0;
    int nbClones=population->nbClones;

    for(i=0;i<nbClones;i++)
    {
        int indiceAc;
        int indiceClone;

        indiceAc=population->nbIndividus-nbClones+i;
        indiceClone=i;

        cloneAc(population->individus[indiceAc],
                population->clones[indiceClone]);
    }
}

/** Il faut un clonage avant .... */
void mutationClones(Population *population)
{
    /* Plus un clone est a droite, meilleur il est.
     * On peut ainsi faire varier le nombre de mutations a effectuer.
     * Dans un premier temps, le nombre de mutations peut etre fixe. */

    (void)population;

    /* A completer ... */
}

void selectionMeilleursEtClonesMutes(Population *population)
{
    /** Les meilleurs et les clones sont compares */
    /** Chaque couple (meilleur,clone) est evalue...
     * Et on garde le meilleur des deux
     * Autre possibilite: tri des meilleurs et des clones (ensemble)
     * Et on garde les meilleurs...
     * LES DEUX VERSIONS SONT A FAIRE ET DOIVENT ETRE COMPAREES */

    (void)population;

    /* A completer ... */
}

/** Les moins bons doivent etre a gauche (apres un tri par exemple) ***/
void mutationMoinsBons(Population *population)
{
    /* Plus un mauvais est a droite, meilleur il est!
     * On peut ainsi faire varier le nombre de mutations a effectuer.
     * Dans un premier temps, le nombre de mutations peut etre fixe. */

    (void)population;

    /* A completer ... */
}

/** Les moins bons doivent etre a gauche (apres un tri par exemple) ***/
void remplacementMauvaisParNouveaux(Population *population, int nbNouveaux)
{
    (void)population; (void)nbNouveaux;

    /* A completer ... */
}

```

**Exemple de gestion
des indices**

```

#include <stdio.h>
#include <math.h>          /* Pour fabs */
#include <float.h>          /* Pour DBL_MAX */
#include <unistd.h>          /* Pour sleep */
#include <stdlib.h>          /* Pour exit */

#include "params.h"
#include "geo.h"
#include "anticorps.h"
#include "population.h"
#include "random.h"
#include "gnuplot.h"

/*********************************************************/
/* ***** DEBUT ***** */
/*********************************************************/
#endif NBVILLES==8

Carte carte={{ {5,2},{7,3},{8,5},{7,7},{5,8},{3,7},{2,5},{3,3} }, 8};

Ac BestOf={{ 0 }, 17.888544, NBVILLES};

#ifndef NBVILLES==16

Carte carte={{ {5,2},{6,2},{7,3},{8,4},
{8,5},{8,6},{7,7},{6,8},
{5,8},{4,8},{3,7},{2,6},
{2,5},{2,4},{3,3},{4,2} }, 16};

Ac BestOf={{ 0 }, 19.313708, NBVILLES};

#endif NBVILLES==30

Carte carte={{ {0,0},{3,3},{4,1},{1,9},{7,6},
{2,1},{9,8},{3,5},{4,6},{5,9},
{3,9},{0,4},{8,5},{2,6},{6,1},
{7,8},{5,2},{3,6},{6,5},{1,8},
{7,1},{7,0},{7,3},{1,1},{3,1},
{5,1},{6,0},{8,4},{1,4},{1,6} }, 30};

Ac BestOf={{ 0 }, 46.371631, NBVILLES}; /* Meilleur cout ???? */

#else
Carte carte; /* Un passage de parametre pas tres joli ! */
#endif /* Voir anticorps.h/anticorps.c */

/* Et encore des passsages de paramatres pas jolis (NBVILLES) ! */

void visualiserCout(FILE *fd, char *fileName)
{ [ ] }

void ecrireCout(FILE *fdCout, int abscisse, double cout)
{ [ ] }

static int compareAc(const void *acl,const void *ac2) /* Ne pas utiliser */
{ [ ] }

/*********************************************************/
/* ***** Lecture et Traitement des arguments ***** */
/*********************************************************/

void recuperParams(int argc, char **argv,
int *nbIndividus, int *nN, int *dD,
int *nbGenerations, int *nbGenerationsInjection,
int *nbClones, int *nbNouveaux)
{ [ ] }

```

```

int main(int argc, char** argv)
{
    int nbIndividus=TAILLEPOP;
    int nN=N;
    int dD=D;
    int nbGenerations=NBGENERATIONS;
    int nbGenerationsInjection=NBGENERATIONSINJECTION;

    int nbClones=(nN * nbIndividus)/100;
    int nbNouveaux=(dD * nbIndividus)/100;

    Population p;
    Ac meilleure, LeMeilleur={{ 0 }, DBL_MAX, NBVILLES};

    int tour=0;

    FILE* fdCout;           char* fileNameCout="Cout";
    FILE* fdGnuplot;
    FILE* fdGnuplotCout;
    FILE* fdBest;           char* fileNameBest="Best";

    /*****************************************************************/
    /* ***** Traitement des arguments ***** */
    /*****************************************************************/
    if (argc!=1) { /* Si il y a des arguments */
        recuperParams(argc,argv,&nbIndividus,&nN,&dD,
                      &nbGenerations,&nbGenerationsInjection,
                      &nbClones,&nbNouveaux);
    }

    fprintf(stderr,"*****\n");
    fprintf(stderr,"nbIndividus=%d\n",nbIndividus);
    fprintf(stderr,"N=%d => nbClones=%d\n",nN,nbClones);
    fprintf(stderr,"D=%d => nbNouveaux=%d\n",dD,nbNouveaux);
    fprintf(stderr,"nbGenerations=%d\n",nbGenerations);
    fprintf(stderr,"nbGenerationsInjection=%d\n",nbGenerationsInjection);
    fprintf(stderr,"*****\n");

    /*****************************************************************/
    /* ***** Initialisation ***** */
    /*****************************************************************/

    fdCout=fopen(fileNameCout,"w"); /* Ouverture du fichier pour les couts */
    if (fdCout==NULL) {
        fprintf(stderr,"Probleme sur fopen(\"%s\",\"w\")\n",fileNameCout);
        fprintf(stderr,"=> Arret du programme\n");
        exit(EXIT_FAILURE);
    }

    fdGnuplot=openGnuplot(NULL); /* pipe + fork pour visu */
    if (fdGnuplot==NULL) {
        fprintf(stderr,"Probleme sur openGnuplot => Arret du programme\n");
        fclose(fdCout);
        exit(EXIT_FAILURE);
    }

    setRangeGnuplot(fdGnuplot,
                    -1,COTECARTE,-1,COTECARTE); /* Rappel : les coordonnees */
                                         /* des villes vont */
                                         /* de 0 a COTECARTE-1 */

    fdGnuplotCout=openGnuplot(NULL); /* pipe + fork pour visu */
    if (fdGnuplotCout==NULL) {
        fprintf(stderr,"Probleme sur openGnuplot => Arret du programme\n");
        fclose(fdCout);
        closeGnuplot(fdGnuplot);
        exit(EXIT_FAILURE);
    }
}

```

```

fdBest=openGnuplot(fileNameBest);           /* Avec gnuplot faire : */
if (fdGnuplotCout==NULL) {             /* gnuplot > load "Best" */
    fprintf(stderr,"Probleme sur openGnuplot=> Arret du programme\n");
    fclose(fdCout);
    closeGnuplot(fdGnuplot);
    closeGnuplot(fdGnuplotCout);
    exit(EXIT_FAILURE);
}

initRandom();

#if NBVILLES!=8 & NBVILLES!=16 && NBVILLES!=30
genereCarte(&carte,COTECARTE);
#endif

dessineCarte(fdGnuplot,&carte);
sleep(3);
printCarte(&carte);

/* Les choses serieuses commencent */
/* Parametres: &p(p:population) */
generePopulation(&p,nbIndividus,nbClones); /* nbIndividus,nbClones */

/* Tri d'une Population avec un COUT DECREMENTAL =>
/* les moins bons sont a gauche, les meilleurs sont a droite */

triPopulation(&p);

tour=0;
while (tour!=nbGenerations)
{
    tour++;

    /* Par construction, en debut de boucle, la population est deja triee... */

    /* A completer ... */

    /* ... */

    /* Fin a completer !! */

    triPopulation(&p);          /* Pour trouver le meilleur, la */
    meilleur=meilleurIndividu(&p); /* population doit etre deja triee */

    if (meilleur.cout<LeMeilleur.cout)
    {
        cloneAc(&meilleur,&LeMeilleur);

        dessineParcoursAc(fdGnuplot,&LeMeilleur);
        printCoutAc(&LeMeilleur);

        ecrireCout(fdCout,tour,LeMeilleur.cout);
        visualiserCout(fdGnuplotCout,fileNameCout);
    }

#if NBVILLES==8 || NBVILLES==16 || NBVILLES==30
    if (compareAc(&LeMeilleur,&BestOf)==0) {
        printf("Stop! (meilleure solution trouvée... a priori)\n");
        break;
    }
#endif
}

```

Mettre ici le test
de la fonction
muteAc

```

/*
***** A la fin de la boucle, on affiche Le Meilleur ****
****

printf("\n*** Voici la meilleure solution trouvée ***\n");
dessineParcoursAc(fdGnuplot,&LeMeilleur);
dessineParcoursAc(fdBest,&LeMeilleur);
printCoutAc(&LeMeilleur);
printf("\n*** Avec %s %d %d %d %d\n",argv[0],nbIndividus,
nN,dD,nbGenerations,
nbGenerationsInjection);

#if NBVILLES==8 || NBVILLES==16 || NBVILLES==30
if (compareAc(&LeMeilleur,&BestOf)<=0)
{
    printf("Meilleure solution obtenue en %d generations\n",tour);
    if (compareAc(&LeMeilleur,&BestOf)!=0)
    {
        printf("ET EN PLUS LE COUT (%f)"
              " EST MEILLEUR QUE LA SOLUTION PREVUE!!\n",LeMeilleur.cout);
    }
else
{
    printf("Meilleure solution NON obtenue en %d generations\n",tour);
}
#endif

#if 1
sleep(2);                                /* Dessin des
dessineIndividusPopulation(fdGnuplot,&p,5); /* meilleurs individus */
#endif

supprimePopulation(&p);

fclose(fdCout);

closeGnuplot(fdGnuplot);
closeGnuplot(fdGnuplotCout);
closeGnuplot(fdBest);

exit(EXIT_SUCCESS);
}

```