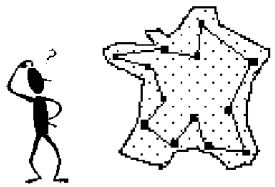
TP systèmes immunitaires artificiels
Programmation du problème du voyageur de commerce

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.



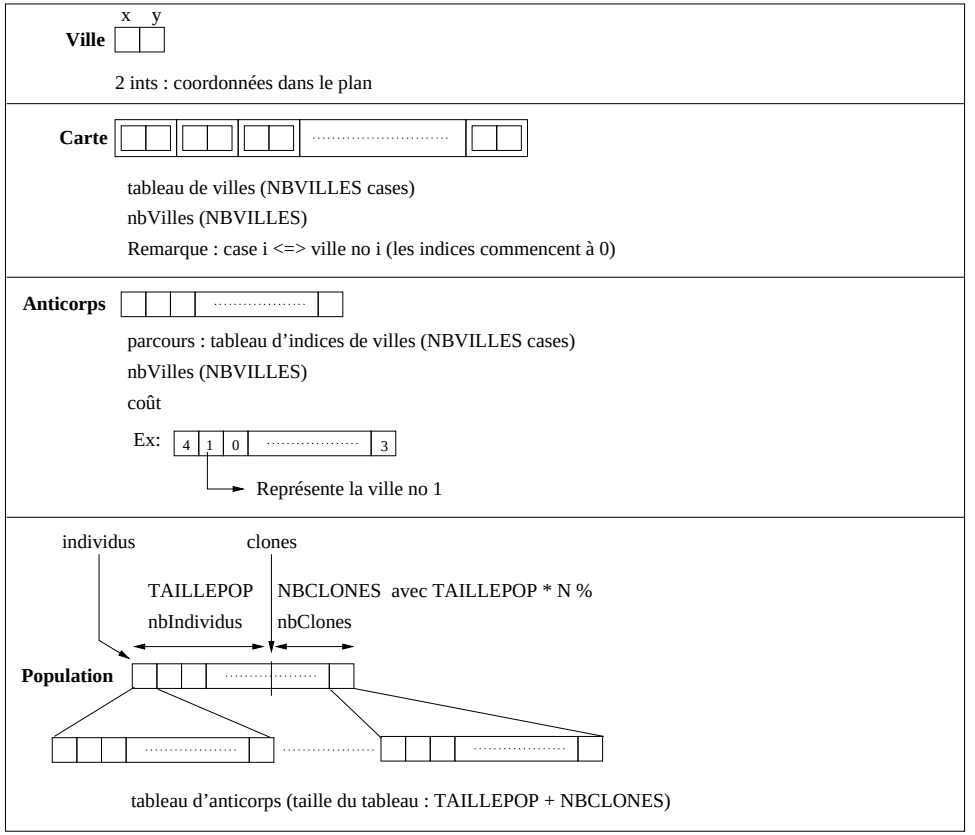
1.1 Algorithmes déterministes

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.
    $\Longrightarrow$ NBCLONES
  - ◇ `D` : lors de l'injection de nouveaux individus, `D` représente le % de la population remplacée.
    $\Longrightarrow$ 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
TAILLEPOP=100
N=50 => nbClones=50
D=20 => nbNouveaux=20
NBGENERATIONS=100
NBGENERATIONSINJECTION=20
...
$ ais 300 80 20 500 30
TAILLEPOP=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`, `remplacementMoinsBons` et `mutationMonBons` ; 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. 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
$
```