Correction 5 –

a. Phase de modélisation

Le problème du voyageur de commerce (PVC) peut se modéliser à l'aide d'un graphe complet de n sommets dont les arrêtes sont pondérées par un coût strictement négatif. Pour construire la matrice de coût.

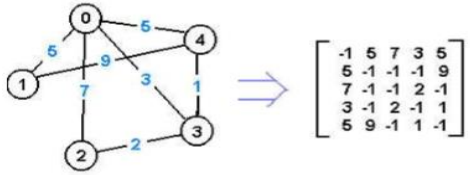

Figure : modélisation du graphe sous forme de matrice

1. L'espace de recherche :

L'espace de recherche est l'ensemble des permutations de {1, 2, ..., n }. Un point de cet espace de recherche est représenté par une de ces permutations.

2. Codage des points de l'espace de recherche :

Une première idée serait de coder chaque permutation (i.e : chaque point de l'espace de recherche) par une chaîne de bits.

Par exemple, pour n = 10 : 0011 0111 0000 0100 0001 0010 0101 0110

représente la permutation : 3 7 0 4 1 2 5 6

On s'aperçoit bien que chaque élément de l'ensemble de permutation est codé sur :

i = E (ln (n) / ln (2)) bits , E étant la fonction de partie entière.

Sachant qu'une permutation est de taille n, la chaine de bits sera alors de taille n * i.

3. Représentation d'une solution :

PVC doit revenir à son point de départ et passer par toutes les villes une fois et une seule.

- Codé une solution par une structure de données comptant autant d'éléments qu'il y a de villes.

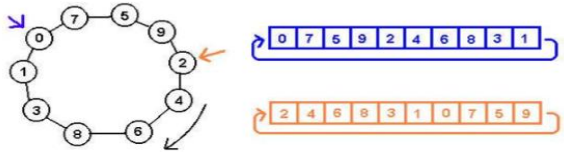

Figure : codage d'une solution (ensemble de villes) dans un tableau

4. Calcul de fitness ou dite fonction d'évaluation :

- Une solution non correcte sera de fitness négative et dans l'autre cas sera alors positive.
- Choisir comme valeur d'adaptation, l'inverse de la longueur du parcours.

On s'aperçoit bien alors que cette algorithme est aléatoire (ou dit approximatif) dans le sens ou elle se base sur des méthodes de calculs non déterministes.

b. Phase d'évolution

1. Sélection :

- Sélection par roulette. On calcule d'abord la valeur moyenne de la fonction d'évaluation dans la population :

$$\overline{f} = \frac{1}{m}\sum_{i=0}^{m-1} f(P_i),$$

où p_i est l'individu i de la population et m la taille de la population. La place d'un individu p_i dans la rou $\frac{\overline{f}}{f(P_i)}$ st proportionnel à .

On sélectionne alors m/2 individus pour la reproduction.

2. Croisement :

Etant donné deux parcours il faut combiner ces deux parcours pour en construire deux autres. Nous pouvons suivre la méthode suivante :

1. On choisi aléatoirement deux points de découpe.
2. On interverti, entre les deux parcours, les parties qui se trouvent entre ces deux points.
3. On supprime, à l'extérieur des points de coupe, les villes qui sont déja placées entre les points de coupe.
4. On recense les villes qui n'apparaissent pas dans chacun des deux parcours.
5. On remplit aléatoirement les trous dans chaque parcours.

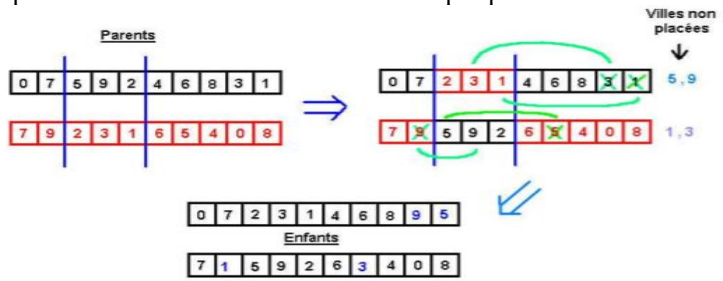

Figure : exemple de croisement.

3. Mutation :

Il s'agit ici de modifier un des éléments d'un point de l'espace de recherche, soit d'une permutation. Dans notre cas, cela correspond donc à une ville. Quand une ville doit être mutée, on choisit aléatoirement une autre ville dans ce problème et on intervertit les deux villes.

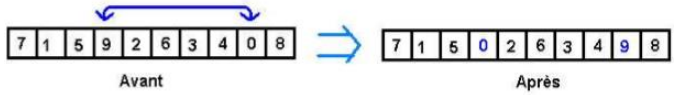

Figure : exemple de mutation

ALGORITHME ABSTRAIT :

a. Croiser(G g1,G g2){ //résultat du croisement

G r; //l'idée est tout simplement pour chaque variable de choisir aléatoirement la variable de g1 ou de g2.

Pour chaque variable v de r

Si un nombre aléatoire de 0 à 99 est inférieur à 50 alors

copier la variable correspondante à v de g1 dans v

Sinon

copier la variable correspondante à v de g2 dans v

```
            Fin Si
        Fin Pour
    Retourner r; }

    b. Muter(G g){
        Pour chaque variable v de g
            Si un nombre aléatoire de 0 à 99 est inférieur à un certain nombre
            entre 0 et 99 // On echange deux variables car on ne peut avoir 2
                          villes semblables  dans le meme parcours.
                    Echanger v et vAutre;
        Fin Pour }

c. SelectionNaturelle(G g[],N){    // On trie les éléments à la fitness la plus
                                   grande vers ceux elle est la plus  petite.
     Trier g par les g[i] dans l'ordre décroissant.
     Retourner les N premiers éléments de g }

AlgoGene(G g[N]){      // Tant qu'il y a des parcours .
    Tant que (g[i] != NULL)      // N pour gi, N-1 pour gj .
        G r[N*(N-1)];
        k=0;
        Pour chaque gi dans g[N]
            Pour chaque gj dans g[N]
                Si gi!=gj alors
                    r[k]=Croiser(gi,gj);
                    Muter(r[k++]);
                Fin Si
            Fin Pour
        Fin Pour
        g=SelectionNaturelle(r,N);
    Fin Tant Que }
```

Comparaison de complexité :

Ce problème est un représentant de la classe des problèmes NP-complets. Les algorithmes pour résoudre le problème du voyageur de commerce peuvent être répartis en deux classes :

- les algorithmes déterministes qui trouvent la solution optimale
- les algorithmes d'approximation qui fournissent une solution presque optimale

Complexité via une méthode de résolution classique (méthode déterministe) :

Un calcul rapide de la complexité montre qu'elle est en O(n!) où n est le nombre de villes. En supposant que le temps pour effectuer un trajet est d' 1 µs, le tableau 1 témoigne de l'explosion combinatoire.

NB VILLES	*NB POSSIBILITES*	*TEMPS DE CALCUL*
5	120	120µs
10	181440	0.18 ms
15	43 MILLIARDS	12 h
20	60 E +15	1928 ans
25	310 E + 21	9,8 Milliards A.

Figure : Nombre de possibilités de chemins et temps de calcul en fonction du nombre de villes

complexité via une méthode de résolution avec algorithmes génétiques (méthode approximative) :

Un calcul de complexité de l'algorithme abstrait nous montre qu'elle est en O(n^2) ou n est le nombre de villes.

Détail du calcul : O (Algo Gene) = O(n^2) + O (Muter) + O (Croiser) + O(SelectionNaturelle)
= O (n^2) + O(n) + O (2n) + O(n)
= O (n^2)

NB VILLES	*NB POSSIBILITES*	*TEMPS DE CALCUL*
5	25	25 µs
10	100	100 µs
15	225	225 µs
20	400	400 µs

Figure : Nombre de possibilités de chemins et temps de calcul en fonction du nombre de villes

=> Un temps exponentiel (explosion combinatoire) et un temps polynomial (quadratique) par rapport au nombre de villes.