Section 7

Application au problème du voyageur de commerce

Nous allons maintenant nous interesser à une application plus concréte : le problème du voyageur de commerce. Cet exemple est un classique appartenant à la classe des problèmes NP-complets.

Il consiste à visiter un nombre N de villes en un minimum de distance sans passer deux fois par la même ville. Il s'agit donc d'optimiser le coût d'un parcours dans un graphe complet possédant un certain nombre de sommets, en passant une et une seule fois par chacun. Des méthodes déterministes existent déjà pour résoudre le problème, mais le temps de calcul est très long : elles reviennent à parcourir toutes les solutions possibles et à déterminer la moins coûteuse.

Le but sera ici de montrer comment modéliser le problème à partir d'algorithmes génétiques et des divers opérateurs que nous avons à disposition, qui ont été définis antérieurement.
Ceci nous conduira ensuite à montrer les avantages de cet outil par rapport à une résolution de type déterministe.

Représentation du problème :

Le problème du voyageur de commerce peut se modéliser à l'aide d'un graphe complet de *n*

sommets dont les arrêtes sont pondérées par un coût strictement négatif. Pour l'implantation de notre algorithme, l'instance sera modélisée comme distance euclidienne sur *n* points du plan, pour construire la matrice de coût.

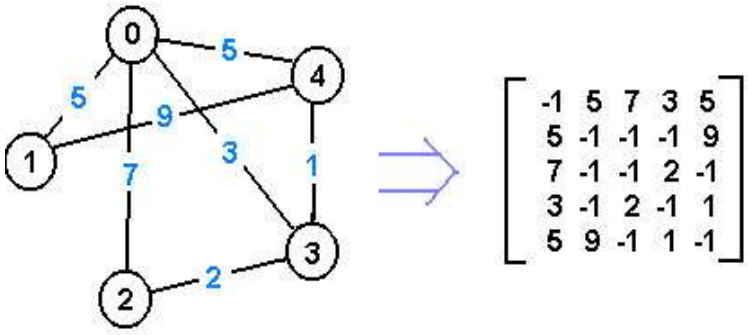

Figure 15 : modélisation du graphe sous forme de matrice

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.

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.

Représentation d'une solution :

Comme nous l'avons déja dit le voyageur de commerce doit revenir à son point de départ et passer par toutes les villes une fois et une seule. Nous avons donc codé une solution par une structure de données comptant autant d'éléments qu'il y a de villes, c'est à dire une permutation. Chaque ville y apparait une et une seule fois. Il est alors évident que selon la ville de départ que l'on choisit on

peut avoir plusieurs représentations différentes du même parcours.

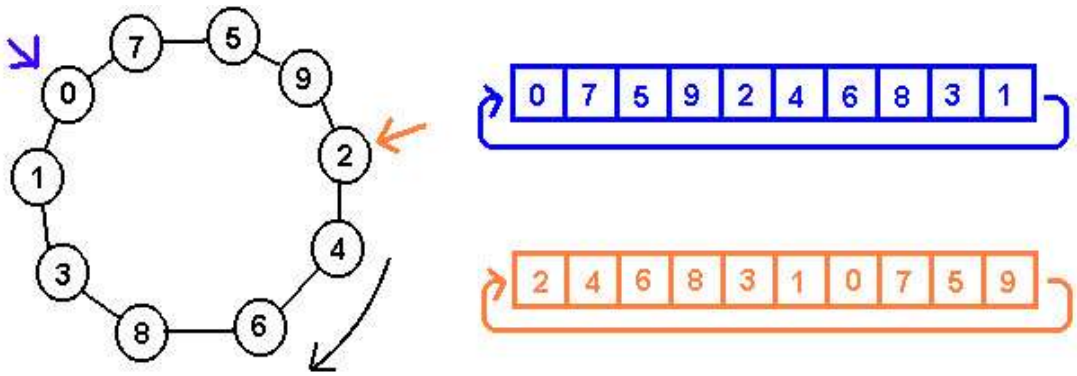

Figure 16 : codage d'une solution (ensemble de villes) dans un tableau .

Sélection :

Nous utilisons ici la méthode de 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 roulette est proportionnel à $\frac{\overline{f}}{f(P_i)}$. On sélectionne alors $m/2$ individus pour la reproduction. Il y a aussi la possibilité d'avoir une politique d'«élitisme». C'est à dire qu'à chaque étape de sélection le meilleur chromosome est automatiquement sélectionné (cf. partie 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.

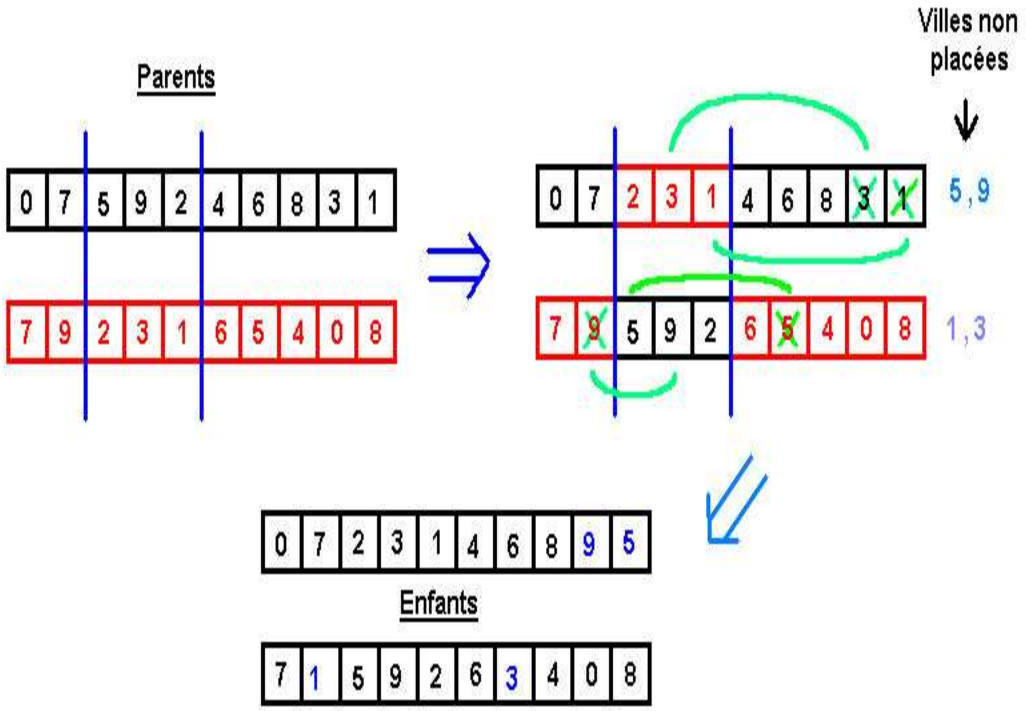

Figure 17 : exemple de croisement.

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.

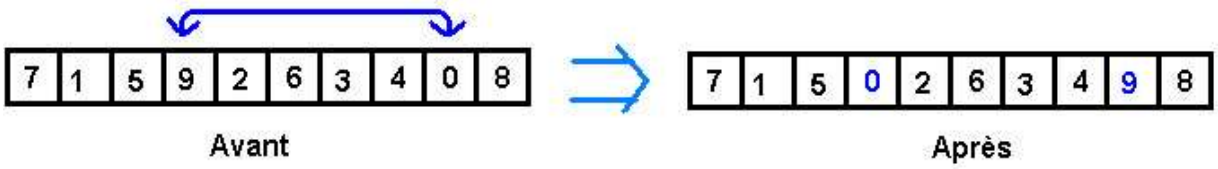

Figure 18 : exemple de mutation

Calcul de fitness ou dite fonction d'évaluation :

Le seul impératif sur la valeur d'adaptation est qu'elle soit croissante avec l'adaptation de la solution au problème. Un parcours valide renverra une valeur supérieur à un individu (solution potentielle) qui n'est pas solution au problème. On pourra décider par la suite qu'une solution non correcte sera de fitness négative et dans l'autre cas sera alors positive.

Lorsque cette dernière éventualité se produit, il faut que le comportement suivant soit respecté, à

savoir : plus notre chemin sera court, plus la fitness qui lui est associée sera forte.

On en déduit alors rapidement qu'il faut que la valeur d'adaptation varie dans le sens inverse de la distance correpondante au parcours. Par souci de simplicité, on pourra éventuellement 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. L'expérience (i.e la programmation) montre que les résultats obtenus sont convaincants, nous parvenons à obtenir un ensemble de bonnes solutions en un temps raisonnable.

ALGORITHME ABSTRAIT :

Il s'agit dans un premier temps d'établir un ensemble de gênes G. Ceci étant fait, il faut alors définir deux fonctions, qui respectivement permettront de croiser deux ensembles de gênes et de réaliser une mutation sur deux ensembles de gênes.

```
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;
}
```

Et ci dessous la procedure de mutation :

```
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
}
```

Dès lors, il nous faut une procédure qui puisse « noter » les génomes ou permutations dénotant un parcours (dans le cas de notre exemple du voyageur de commerce).

Voici donc une procédure sélection qui renvoie les éléments les plus interessants d'un génomes (ce sera dans le cas du voyageur de commerce, les éléments de fitness la plus grande). Le choix est sinon laissé au programmeur.

```
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
}
```

A présent que nous avons tous les éléments dont nous avons besoin, nous pouvons construire l'algorithme final :

```
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. L'existence d'un algorithme de compléxité polynomiale reste inconnue.

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