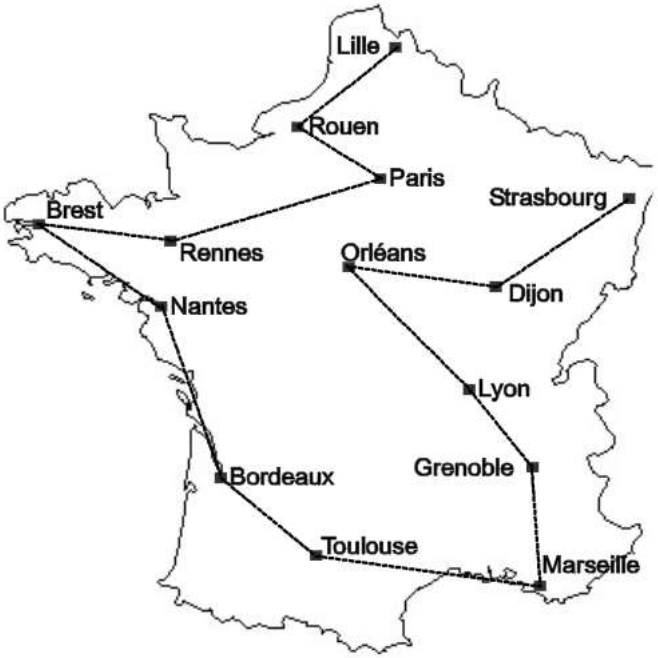
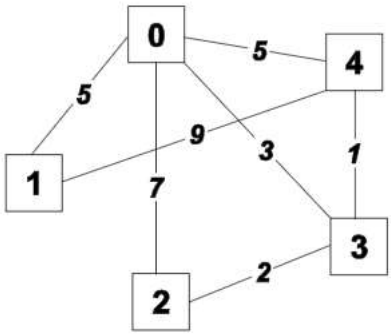
Problème du voyageur de commerce par algorithme génétique

1 Problème du voyageur de commerce

Le problème du voyageur de commerce, consiste en la recherche d'un trajet minimal permettant à un voyageur de visiter n villes. En règle générale on cherche à minimiser le temps de parcours total ou la distance totale parcourue.



On se donne n villes ainsi que les distances (longueur ou temps) entre chaque ville. On peut représenter le problème sous forme d'un graphe et d'une matrice représentant le graphe (les villes sont indicées de 0 à $n-1$) :



$$\begin{bmatrix} -1 & 5 & 7 & 3 & 5 \\ 5 & -1 & -1 & -1 & 9 \\ 7 & -1 & -1 & 2 & -1 \\ 3 & -1 & 2 & -1 & 1 \\ 5 & 9 & -1 & 1 & -1 \end{bmatrix}$$

De façon générale, on peut considérer des graphes où certains arcs (chemins) n'existent pas. Dans la matrice représentant le graphe, les arcs inexistants ont une distance strictement négative (-1). Par ailleurs, on ne considère pas les arcs reliant une ville à elle même.

Le voyageur de commerce devant revenir dans la ville de départ, on peut représenter une "tournée" par un vecteur de dimension n, contenant la liste des villes visitées dans l'ordre :

Une tournée est définie à une permutation près. Une fois la ville de départ fixée, elle est par contre unique.

Le problème du voyageur rentre dans la catégorie des problèmes *NP -complets*, à savoir que sa complexité n'est pas polynomiale. Il a été abordé pour la première fois au $19^{ème}$ siècle par Hamilton et Kirkman. Au $20^{ème}$ siècle, de nombreux auteurs s'y sont intéressés et l'avénement des ordinateurs a vu le développement de nombreux algorithmes qui ont permis de traiter des cas de plus en plus grands (1959 : 49 villes par Dantzig, 1975 : 100 villes, 1987 : 2392 villes, 1998 : 13509 villes, 2001 : 15112 villes).
Il existe deux grandes classes d'algorithme : les algorithmes déterministes qui déterminent la solution optimale (complexité $n!$) et les algorithmes d'approximation qui se "contentent" de trouver une bonne solution (non optimale) dont le coût est voisin du coût minimal. Les algorithmes génétiques font partie de cette seconde catégorie.
On trouvera sur le Web énormément de ressources sur ce problème (voyageur commerce sur Google par exemple).

2 Algorithme génétique

Les algorithmes génétiques s'appuient sur un principe de sélection des individus d'une population qui présentent des caractéristiques se rapprochant au mieux de ce que l'on recherche; cette population évoluant par ailleurs selon des critères d'évolution génétique à choisir. Dans le contexte du problème du voyageur de commerce, un individu est une tournée, un chemin et une population un ensemble de tournées. Il s'agit maintenant de définir un critère de sélection ainsi que des règles d'évolution de la population. En la matière, il y en a de nombreuses et nous n'en avons retenu que les principales. Il faut toutefois garder à l'esprit qu'il y a très peu de résultats théoriques sur les algorithmes génétiques et que le choix des critères et règles est donc heuristique.
Sachant que l'on a à un instant donné k une population P_k constituée de p individus, comment générer une population P_{k+1} à l'instant suivant toujours constituée de p individus mais qui présente de meilleurs caractéristiques. Il y a donc deux étapes : la génération d'une nouvelle population et la sélection des "meilleurs" individus.

2.1 Fonction d'adaptation

Il s'agit de caractériser l'adaptation d'un individu au critère visé. Pour le problème du voyageur de commerce, on peut simplement utiliser la distance de la tournée comme fonction d'adaptation d'un individu, étant entendu que plus cette distance est petite meilleure est son adaptation.

2.2 Génération de nouveaux individus

Le principe consiste à générer un nouvel individu à partir de deux individus de la population P_k en utilisant des règles inspirées de la biologie.

2.2.1 L'hybridation (ou crossover)

On considère deux individus $I = (i_0, i_1, ...i_{n-1})$ et $J = (j_0, j_1, ...j_{n-1})$.

- On tire au hasard un indice d'hybridation $\ell \in \{0, 1, .., n-1\}$
- On génère un fils $IJ = (i_0, ...i_\ell, j_{\ell+1}, ..j_{n-1})$
- On génère un fils $JI = (j_0, ...j_\ell, i_{\ell+1}, ..i_{n-1})$

2.2.2 La mutation

On génère un nouvel individu à partir d'un individu $I = (i_0, i_1, ...i_{n-1})$ en permutant aléatoirement deux éléments (k, ℓ) :

$$I = (i_0, ..., i_k, ...i_\ell, ...i_{n-1}) \rightarrow J = (i_0, ..., i_\ell, ...i_k, ...i_{n-1})$$

Cette mutation modifiant énormément la fonction d'adaptation, on préfère souvent utiliser la mutation (flip) qui consiste à permuter les éléments $(k, k+1, \ell-1, \ell)$:

$$[..., k, k+1, ..., \ell-1, \ell, ...] \rightarrow [..., \ell, \ell-1, ..., k+1, k, ...]$$

2.2.3 Reproduction

On génère une population de reproducteurs $\widetilde{P}_k$ de taille p en tirant des individus de la population P_k suivant un critère de sélection (un individu pouvant apparaître plusieurs fois). Les p individus reproducteurs se croisent deux à deux par hybridation de façon aléatoire pour générer p nouveaux individus à qui on applique une mutation. On obtient ainsi une population enfants $\widehat{P}_{k+1}$.

2.3 Sélection

La sélection intervient à deux stades : lors du choix des reproducteurs et lors de la fabrication de la population finale P_{k+1} à partir des populations $\widehat{P}_{k+1}$ et P_k.

2.3.1 Sélection des reproducteurs

Il s'agit de tirer les individus suivant une loi qui favorise les les individus les mieux adaptés :

- Sélection par roulette :
  - On calcule la somme S de toutes les fonctions d'adaptation d'une population
  - On génère aléatoirement suivant une loi uniforme un nombre $r \in \{0, ..., S\}$
  - On somme les fonctions d'adaptation des individus jusqu'à atteindre la valeur r, le dernier individu qui a contribué à cette somme est sélectionné.
- Sélection par rang :
  - On commence par trier les individus suivant leur fonction d'adaptation et on affecte un rang à chaque individu (1 pour le plus mauvais)
  - On effectue la sélection par la méthode précédente en remplaçant la fonction d'adaptation par la fonction de rang
- La sélection par tournoi :
  - On forme n paires d'individus d'une population de n individus
  - On donne une probabilité de victoire lors d'un match (entre 70% et 100% de chance de victoire si un individu a une fonction d'adaptation supérieure)
  - On génère n individus en appliquant cette règle de victoire
- L'eugénisme : on n'utilise que les meilleurs individus pour générer la population suivante

2.3.2 Sélection de la polulation finale

Il s'agit de générer la population P_{k+1} qui survivra à l'instant $k+1$ à partir de la population des parents P_k et de la population des enfants $\widehat{P}_{k+1}$. Pluseurs stratégies sont envisageables :

- Seuls les enfants survivent : $P_{k+1} = \widehat{P}_{k+1}$
- Les meilleurs parents survivent (élitisme): on se fixe un nombre q de parents qui doivent survivre; la population P_{k+1} est alors constituée des q individus de P_k qui ont la meilleure fonction d'adaptation et de $n-q$ individus de $\widehat{P}_{k+1}$ qui ont la meilleure fonction d'adapation.

2.4 Algorithme

En résumé, on a l'algorithme suivant :

Génération d'une population initiale P_0
Itération $k \geq 0$ (P_k connu)
- Génération d'une population enfants $\widehat{P}_{k+1}$
  Choix d'une population de reproducteurs
  Génération par hybridation et mutation des enfants
- Génération de la population P_{k+1}
- Test d'arrêt

Compte-tenu des différentes statégies que l'on peut mettre en place et des différents paramètres qui interviennent dans ces stratégies et qu'il n'y a pas de résultats de convergence, l'efficacité d'un tel algorithme repose sur une bonne analyse des causes d'échecs et pas mal de bon sens. De façon générale, le problème que l'on cherche à résoudre présente beaucoup de minima locaux et le danger est de se retrouver piéger dans un tres mauvais minimum local. C'est souvent du au fait qu'il n'y a pas assez de variabilité et d'évolution dans les populations générées. Si on augmente trop la variabilité en forçant le cross-over par exemple, on risque d'avoir un comportement chaotique car les populations varient trop d'une itération à l'autre.
L'algorithme est également tres sensible à la population de départ. Le choix de parcours aléatoire n'est pas tres bon. Une assez bonne méthode d'initialisation consiste à construire un chemin initial en prenant le plus proche voisin du point où on se trouve.

3 Programme de travail

De façon générale, on choisira d'apdopter des structures de données et des fonctionnalités assez générales qui permettent d'interchanger rapidement des stratégies. A ce titre, on dissociera bien le problème à résoudre (le voyageur de commerce) et la méthode de résolution (algorithme génétique). L'interaction entre les deux devant être limitée.

3.1 Problème du voyageur de commerce

Il s'agit essentiellement de construire une classe représentant la matrice du **graphe**, une classe représentant un **chemin** et proposant des fonctions d'évaluation des chemins (globale ou remise à jour partielle).
On pourra éventuellement développer une classe représentant les villes (nom,coordonnées) dont on génèrera le graphe. Cette classe permettra de générer un fichier de sortie représentant un chemin que l'on visualisera par exemple à l'aide de Matlab.

3.2 Algorithme génétique

Il s'agit de proposer une implémentation de l'algorithme a priori indépendante du problème à résoudre. On introduira à cet effet une classe d'**Individu**, une classe de **Population** et les fonctionnalités dont on a besoin (crossover de deux individus, mutation, sélection, ...). L'interaction avec le problème du voyageur de commerce se concentre au niveau de la classe **Individu :** évaluation de la fonction d'adaptation, croos-over et mutation. Il y a plusieurs façons de gérer cette interaction :

- par héritage : un **chemin** dérivant de la classe **individu** (classe abstraite), la classe **chemin** devant proposer les méthodes attendues par la classe **individu**
- par généricité : la classe **individu** etant générique, le type de **individu** pouvant être un **chemin** (**individu<chemin>**)

Je vous suggère d'utiliser la méthode par héritage.
Il sera utile de prévoir une une fonction permettant de générer un fichier qui contiendra le coût minimal obtenu à chaque itération, ce qui permettra de représenter graphiquement le comportement de l'algorithme dans Matlab.

3.3 Plan de travail

- Proposer une structure des classes avec les structures de données et le fonctions membres (à faire ensembe)
- Implémenter les classes en validant toutes les fonctionnalités élémentaires (se répartir le travail)
- Valider le programme complet sur des petits cas tests (à faire ensemble)
- Valider sur des cas plus significatifs (se répartir les cas)
- Comparer différentes stratégies en essayant de trouver la meilleure (se répartir les cas)