Projet de IA-IHM

Résolution du problème du voyageur de commerce à l'aide de l'algorithme du recuit simulé

1. Sujet. Problème du voyageur de commerce

Le problème du voyageur de commerce (ou *TSP* pour *Travelling Salesman Problem*) est un célèbre problème d'algorithmique. Un voyageur de commerce résidant dans une ville v_0 part faire la tournée de ses clients. Supposons qu'il doive rencontrer n clients c_1, ..., c_n résidant dans n villes $v_1...v_n$ telles que c_i habite à v_i pour $i = 1...n$. On suppose que quels que soient i et j tels que $0 \leq i < j \leq n$, v_i et v_j sont distinctes et il existe une route reliant v_i et v_j (Le graphe v_0, ..., v_n est complet). Le voyageur part de sa ville de résidence v_0, réalise la tournée de ses clients puis revient à v_0. Pour optimiser le trajet total parcouru, le voyageur ne parcourt jamais deux fois la même route.
Quel itinéraire doit-il emprunter pour minimiser la distance totale parcourue ?
Ce problème n'a, dans le cas général, pas de solution de complexité polynomiale connue. En effet, le seul algorithme exact connu consiste à évaluer les $n!/2$ itinéraires possibles. Cette solution n'est pas réaliste pour n grand ($10!/2 > 10^6$).
Remarque : Tout chemin fermé passant exactement une seule fois par chaque sommet v_0, ..., v_n est dit circuit hamiltonien du graphe complet(v_0,...,v_n).

2. Algorithme du recuit simulé

De nombreux algorithmes heuristiques ont été développés pour résoudre le *TSP* (une méthode heuristique est un algorithme de complexité polynomiale qui produit une approximation de la solution). L'objectif de ce projet est d'utiliser l'algorithme du recuit simulé pour produire une solution approchée de ce problème.

3. Hypothèses

Nous faisons les hypothèses suivantes. Le graphe traité par l'algorithme du recuit simulé représente une carte routière simplifiée où les routes sont des segments. Un sommet est défini par un nom et une position (x,y) et une arête par un coût (comme vu en TP). Si le graphe à étudier n'est pas complet, il peut être complété par des arêtes de longueur infinie (elles seront de toute façon écartées par l'algorithme).

4. Contraintes

Le projet **DOIT** utiliser les méthodes/classes vues en TP, il doit donc :

1) nécessairement modéliser le graphe à explorer à l'aide des classes *GElement*, *PElement<T>*, *Sommet<T>*, *Arete<S,T>* et *Graphe<S,T>* (cf. TP 1, TP 2 et TP.3)

2) utiliser la méthode générique *recuitSimule(...)* (cf. TP sur le Recuit Simulé) comme elle a été écrite.

5. Paramètres de la fonction recuitSimule(...) : l'espace d'exploration S

L'espace d'exploration *S* est l'ensemble des circuits hamiltoniens du graphe. La représentation en C++ des éléments de *S* est laissée à votre initiative.

6. Paramètres de la fonction recuitSimule(...) : la solution initiale

La solution initiale est choisie aléatoirement.

7. Paramètres de la fonction recuitSimule(...) : fonction coût

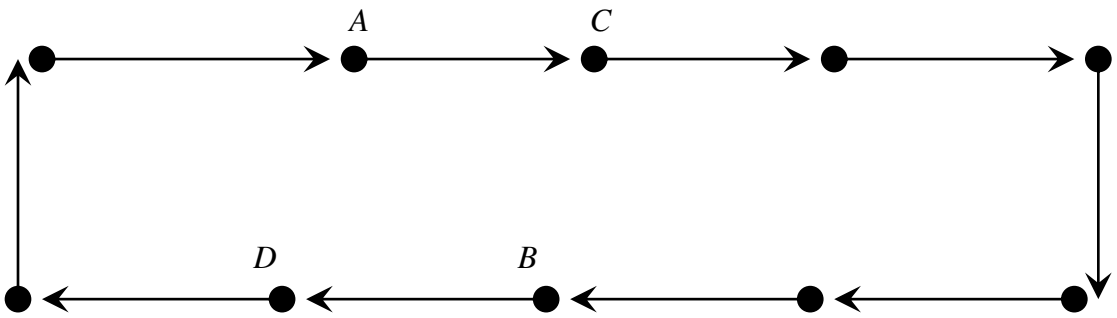
Le coût d'une solution est bien sûr la longueur totale du cycle eulérien défini par cette solution.

8. Paramètres de la fonction recuitSimule(...) : fonction changementAléatoire

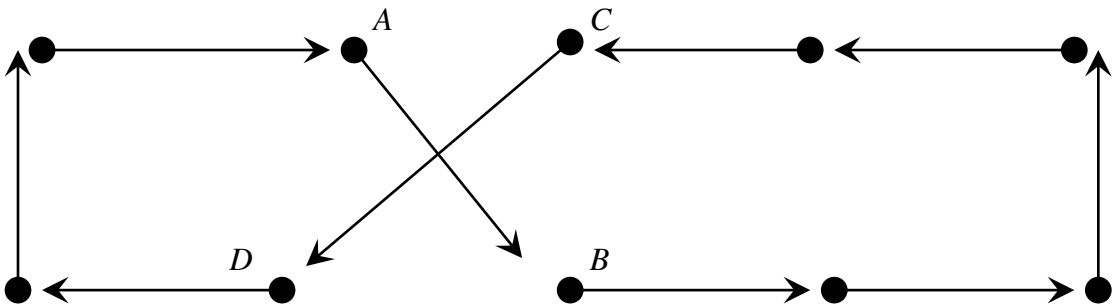
La déformation aléatoire d'un cycle eulérien est, dans ce projet, l'opération la plus importante à réaliser. Elle peut être effectuée en utilisant la procédure suivante :
Supposons le cycle arbitrairement orienté.

1. On choisit deux sommets non consécutifs *A* et *B* au hasard dans ce cycle. Notons C et *D* les sommets respectivement suivants de *A* et de *B*.
2. On remplace l'arête $A \rightarrow C$ par l'arête $A \rightarrow B$.
3. On change le sens du chemin $C \rightarrow B$.
4. On remplace l'arête $B \rightarrow D$ par l'arête $C \rightarrow D$.

Exemple, à partir du cycle eulérien suivant :



On obtient le nouveau cycle eulérien :



9. Dessin du graphe et de la solution trouvée

Dessiner l'ensemble (graphe-carte routière simplifié, chemin) est une fonctionnalité qui peut être prise en charge par *bsplines*, une autre application (écrite en JAVA et fournie avec ce sujet). Les données nécessaires aux opérations de dessin sont fournies à *bsplines* sous la forme de fichiers texte respectant un certain format (un exemple est fourni avec le sujet). Pour utiliser *bsplines*, l'application C++ du projet est donc tenue de produire, pour chaque dessin à réaliser, un tel fichier texte.

10. Degrés de liberté

La flexibilité est assurée par le choix des paramètres des différents templates : il est possible de définir librement les classes-paramètres *S*, *T* des classes génériques *Sommet<T>, Arete<S,T>* et *Graphe<S,T>*.

De la même manière, les trois fonctions *cout*(), *changementAleatoire*() et *succ*(), paramètres de la fonction *recuitSimule*(), peuvent être définies librement. Le contrat associé à ces méthodes est fixé mais pas la façon dont il est réalisé. Elles peuvent être écrites librement.

11. Qualité du code source produit

La programmation est faite en C/C++.
Les notions vues en cours doivent être exploitées au maximum afin de simplifier le code produit.
Comme toujours, il faut éviter les redondances de code (copiés-collés) (leur présence signifie qu'on a oublié d'écrire une fonction) et préférer aux maladroites instructions *if/else* et *switch* le mécanisme des fonctions virtuelles.
La gestion de la mémoire dynamique doit être assurée (constructeurs de copie, destructeurs, destructeurs virtuels).

☞ La notation tiendra compte du strict respect de ces consignes.

12. fonction main(...)

Le but de la fonction *main(...)* de l'application est d'assurer les tâches consécutives suivantes :

1. Construire une carte routière simplifiée à l'aide d'un *Graphe<S,T>*,
2. Rechercher une solution au *TSP* sur ce graphe à l'aide de la fonction *recuitSimule*(),
3. Afficher en mode console le graphe et la solution
4. Dessiner le graphe et la solution

Il n'est pas demandé d'écrire un interface graphique, une application console est suffisante. De même, un menu n'est pas demandé. Le graphe d'essai utilisé pour la soutenance est laissé à votre initiative. Il doit être au moins aussi complexe que celui construit en TP (cf. TP3, paragraphe 4).

13. Extensions possibles

Les dessins de la solution, du graphe et surtout de la progression de l'algorithme du recuit simulé peuvent être assurés par une mise en place d'un client C++-serveur de dessin JAVA à l'instar de ce qui avait été programmé lors du projet de synthèse du 1er semestre. Si cette solution est mise en oeuvre, elle apportera des points supplémentaires.

14. Organisation du travail

Les étudiants peuvent se grouper par équipes de 1 à 2 pour réaliser le travail.

15. Documents à rendre

1) Rapport :
   Un rapport (électronique) (d'au plus 5 pages) expliquant les stratégies utilisées pour résoudre les problèmes posés par le sujet.

2) Exécutable de l'application et programmes sources en C++

16. Date de remise

Tous les documents concernant le projet sont à rendre au plus tard le dimanche (soir) 5 mars 2017 à D. Michel. N'oubliez pas d'indiquer : noms et prénoms des membres de l'équipe, nom de la filière et nom du projet. Le projet doit être rendu par courrier électronique à l'adresse suivante : domic62@hotmail.com.

17. soutenance

Les soutenances seront organisées à partir du mardi 7 mars 2017. Elles se dérouleront à l'UFR MIM. Les lieux de soutenance seront communiqués dans la semaine du lundi 6 mars 2017. Les créneaux disponibles pour placer ces soutenances seront affichés sur la porte du bureau de D. Michel (E206) à partir du lundi 30 janvier 2017. Les étudiants sont invités à s'y inscrire eux même pour prendre RDV. Tous les membres de l'équipe devront être présents. La soutenance dure environ 40 minutes.

18. Notation

La notation prendra en compte :

- le rapport
- la qualité du code source et la rigueur de programmation
- le fonctionnement de l'application

N'hésitez pas à nous poser des questions ou à solliciter des RDVs pour discuter de points de programmation.

Bon travail