Projet CAA : Voyageur de commerce

Université de Bordeaux, Master-2 d'Informatique, 2014–2015

1er semestre

1 Travail à fournir

Le projet est à réaliser par groupes de deux ou trois étudiants (les groupes de 4 ne seront pas acceptés). Dans le cadre de ce projet il faut fournir :

1. Un rapport, de 10 à 15 pages (en `pdf`) présentant les algorithmes, les estimations de leur complexité et les résultats d'expérimentation.
2. Les programmes réalisant tous les algorithmes, avec le `Makefile`, et un fichier `README` expliquant comment compiler et utiliser les programmes.
3. Les soutenances des projets se tiendront vers fin janvier 2015.

Les archives du type `tar` contenant le rapport et les programmes doivent être rendues avant le début des vacances de Noël. Il faut les envoyer par mail à Paul Dorbec, Philippe Duchon et Laurent Simon [1].

2 Contexte du projet

Le sujet du projet porte sur le problème dit "du voyageur de commerce" et ses variantes.

Un voyageur de commerce doit visiter un certain nombre de clients et, à la fin de son périple, retourner à son point de départ, en un temps total le plus court possible. Il dispose, pour chaque paire de clients, du temps (un entier positif) qu'il lui faut pour aller de l'un à l'autre (le temps est censé être le même pour aller de A à B que pour aller de B à A) ; pour cette description, le point de départ et d'arrivée peut être considéré comme un client comme

1. dorbec@labri.fr, duchon@labri.fr, lsimon@labri.fr

les autres. La question est de savoir quel est le trajet (vu comme la liste des sites à visiter, dans l'ordre, respectant la condition de passer par chaque site au moins une fois) qui minimise le temps total. On se ramène naturellement à un problème de décision en ajoutant un entier T ; on cherche alors seulement à savoir s'il existe une solution (un trajet) dont le temps total de trajet soit au plus égal à T.

Dans une variante "discrétisée" du problème, les temps de trajet entre sites sont seulement classés en "courts" ou "longs", et le voyageur souhaite trouver un trajet qui utilise le moins possible de trajets "longs" ; idéalement, qui n'utilise que des trajets "courts". Cette variante se ramène simplement à un problème sur des graphes que l'on précisera. De nouveau, on a une variante décisionnelle : étant donné, en plus du graphe, un entier k, existe-t-il une solution (un trajet) empruntant au plus k trajets "longs".

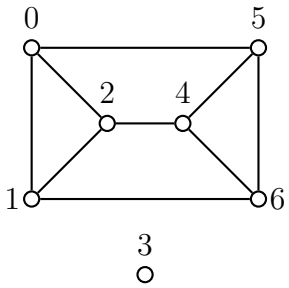
Pour ce projet, vous implémenterez plusieurs algorithmes pour résoudre le problème, et les testerez consciencieusement. Le format des données qui doit être lu par le programme en entrée est décrit dans la section 3. Pour les graphes, il correspond au format généré par le programme de Cyril Gavoille, disponible sur sa page. La section 4 décrits les types d'algorithmes que l'on attend de vous.

3 Format des données

Selon la variante du problème que l'on considère, On a besoin de représenter deux types d'instances différents : des graphes (simples), ou des matrices de distances.

Un graphe sera décrit dans un fichier texte ayant une ligne par sommet du graphe : chaque ligne commence par le numéro du sommet (entre 0 et n) suivi du symbole ' :', et contient enfin la liste des voisins du sommet séparés par des espaces. Afin de simplifier la lecture des fichiers, on ajoutera en première ligne le nombre de sommets, sous la forme `N=<valeur>`. Voici un exemple de graphe :

```
N=7
0: 1 2 5
1: 0 2 6
2: 0 1 4
3:
4: 2 5 6
5: 0 4 6
6: 1 4 5
```



Pour la version générale (pondérée) du problème, on utilisera également un format texte formé, en plus d'une ligne indiquant le nombre de sites comme pour les graphes, d'une ligne par site, listant, dans l'ordre de tous les sites, les temps de trajet individuels. Voici un exemple correspondant à une transformation à partir du graphe précédent : les arêtes donnent une distance 1, les autres distances étant de 0 (systématique) d'un sommet à lui-même ou 10 (entre deux sommets non reliés par une arête).

```
N=7
0: 0 1 1 10 10 1 10
1: 1 0 1 10 10 10 1
2: 1 1 0 10 1 10 10
3: 10 10 10 0 10 10 10
4: 10 10 1 10 0 1 1
5: 1 10 10 10 1 0 1
6: 10 1 10 10 1 1 0
```

4 Algorithmes demandés

Voici une liste d'algorithmes qu'il est demandé d'implanter et de tester. La liste n'est pas exhaustive : il est possible, et même conseillé, d'être créatif. En revanche, il est important de ne pas faire n'importe quoi : pour chaque algorithme que vous proposez, il est important de décrire l'algorithme en pseudo-code et de dire explicitement ce que vous pouvez prouver sur ses performances dans le rapport.

4.1 Algorithme de backtracking

Pour les deux variantes du problèmes, vous fournirez un algorithme (de complexité non polynomiale) qui, par recherche exhaustive ou non, donne

une solution toujours optimale au problème.

4.2 Approximation : arbre et couplage

Pour le problème d'optimisation général, vous implanterez un algorithme 2-approché à partir d'un calcul d'arbre couvrant de poids minimal.

Vous pourrez chercher à implanter un algorithme offrant un meilleur ratio d'approximation (3/2), utilisant, en plus d'un calcul d'arbre couvrant de poids minimal, une recherche de couplage parfait de poids minimal dans un sous-graphe.

4.3 Utilisation d'un solveur SAT

Dans cette section, on se cantonne au problème "discrétisé" sur des graphes, ou à un autre cas particulier du problème général, dans lequel les temps de trajet sont de petits entiers (par exemple : uniquement des entiers de 1 à 4).

En utilisant une réduction vers SAT et un solveur SAT, vous déterminerez, dans ces cas, l'existence ou la non existence de solutions exactes (optimales) aux instances étudiées.

Sur le problème discrétisé, on pourra chercher les solutions quasi optimales n'employant qu'un petit nombre (0 à 3 par exemple) de trajets "longs".

5 Expérimentation et analyse des algorithmes

Vous testerez vos algorithmes avec différents échantillons de données.

Pour le problème général, vous pourrez prendre des matrices de distances entre villes réelles. Pour le problème discrétisé sur les graphes, vous serez attentifs à chercher des échantillons de graphes permettant de tester la complexité pratique de vos algorithmes.

Il est important, lorsque vous disposez d'une estimation théorique de la complexité de vos algorithmes, de la confronter à la pratique, sur des instances aussi grandes que possibles.