Algorithmique avancée

Épreuve pour le bloc 2 : algorithmes *online*

Le concours consiste à concevoir et à implémenter un algorithme *online* (déterministe ou/et randomisé) pour un problème de logistique et calculer expérimentalement son ratio de compétitivité.

Le problème à traiter est celui de **k-serveurs** qui peut être décrit comme suit :
- k techniciens (serveurs) mobiles, $\{s_1, s_2, \ldots, s_k\}$, assurent le service de maintenance auprès de n clients,
- les techniciens traitent des demandes émanant des sites des clients ; ces demandes arrivent en série de longueur N, $(r_1, r_2, \ldots, r_N)$,
- une demande doit être traitée aussitôt (impossible de la mettre en attente),
- des déplacements des techniciens sont instantanés.

L'objectif est de traiter toutes les N demandes en minimisant la somme des distances parcourues par tous les k techniciens.

Nous primerons les algorithmes montrant un beau ratio de compétitivité pour des instances de concours.

La nécessité de calculer le ratio de compétitivité implique l'organisation de cette épreuve en deux étapes :

implémentation d'un algorithme *offline* connu : Nous vous proposons un algorithme *offline* optimal présenté dans [1]. Cet algorithme est décrit dans l'annexe à cet énoncé,

vos algorithmes *online* : conception, implémentation et évaluation de leur performance par rapport à l'algorithme exact.

Pour l'épreuve, nous supposons que :
- les sites de clients sont repartis sur une carte "grille" : $[\![0, 99]\!] \times [\![0, 99]\!]$;
- une liste de demandes est constituée des sites demandeurs de service,
- algorithme *online* reçoit des demandes une par une (il ne voit jamais plus qu'un seul élément, une demande à traiter au moment donné),
- la distance issue de la norme Manhattan $||.||_1$.
- initialement, tous les techniciens/serveurs dans le point $(0, 0)$ sur la carte.

Références

[1] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. In *Proc. of SODA*, 1990.

Annexe

Les auteurs de [1] proposent de modéliser le problème de k-serveurs *offline* par le problème du flot max du coût min qui est résoluble d'une manière exacte en temps polynomial. L'algorithme du problème du flot max du coût min consiste à modifier l'algorithme de Ford-Fulkerson pour trouver le flot max vu en cours d'Algorithmique en 1A : en plus de capacités, des arcs du réseau sont étiquetés par leur coût ; l'objectif est de faire couler par le réseau le flot de la valeur max en payant le moins cher pour ce service. Son implémentation `max_flow_min_cost` toute prête est offerte par la bibliothèque `NetworkX`.

La construction du graphe à flots $G = (V, E)$:

$$V = \{s, t\} \cup \{s_1, s_2, \ldots, s_k\} \cup \{r_1, r_2, \ldots, r_N\} \cup \{r'_1, r'_2, \ldots, r'_N\},$$

Nous prenons deux sommets fictifs s (source) et t (*terminus*, puits), tous les serveurs et toutes les requêtes "doublées".

Les arcs ont la capacité et le coût associés. **La capacité est unitaire partout.**
- les arcs (s, s_i), $i \in [\![1, k]\!]$ de coût nul (pour irriguer le réseau),
- les arcs (s_i, r_j), $i \in [\![1, k]\!]$, $j \in [\![1, N]\!]$ de coût égal à la distance entre le serveur i dans sa position initiale et le site de la requête j (pour pouvoir envoyer un serveur servir sa première requête),
- les arcs (r_j, r'_j), $j \in [\![1, N]\!]$ de coût $-K$ qui est très négatif et entier (pour qu'un serveur puisse aller servir une autre demande après avoir terminé de s'occuper de la requête r_j),
- les arcs (r'_i, r_j), $i, j \in [\![1, N]\!]$ et $i < j$ de coût égal à la distance entre le site demandeur i et le site demandeur j (pour qu'un serveur puisse aller servir la requête r_j depuis l'endroit où il a effectué l'intervention r_i),

— les arcs (r'_j, t), $j \in [\![1, N]\!]$ de coût nul pour évacuer le flot après avoir servi les requêtes,
— les arcs (s_i, t), $i \in [\![1, k]\!]$ de coût nul (au cas où il y a des serveurs qui ne servent aucun client).

Le graphe G est très expressif et il n'est pas difficile de remarquer qu'il y aura une chaîne augmentante passant par chaque s_i (le flot max est certainement k) et cette chaîne établit le planning pour s_i. La minimisation du coût de transfert du flot doit accompagner l'augmentation de la valeur du flot (`max_flow_min_cost` de `NetworkX`). L'algorithme a la complexité en temps en $\mathcal{O}(kN^2)$.