TD12: Correction

MP2I Lycée Pierre de Fermat

Exercice 1. *Collection de problèmes*

Pour modéliser un problème, il faut réfléchir à ce qu'est une solution (pas forcément optimale), et au score que l'on attribue à une solution.

Question 1. Pour le problème `Subset-Sum`, étant donnée (E, t) une instance, une solution serait un sous-ensemble $F \subseteq E$ dont la somme est inférieure à t, et le score à maximiser est la somme en question. D'où :

$$\max_{F \in S} \sum_{x \in F} x$$

avec

$$S = \{F \subseteq E \mid \sum_{x \in F} x \leq t\}$$

Question 2. Une solution est problème est une **manière d'assigner les personnes à des évènements**. On peut par exemple donner un p-uplet d'ensembles $(A_1, \dots A_p)$, où A_k donne l'ensemble des évènements auxquels la personne numéro k assiste. Chaque A_k doit être un ensemble d'évènements deux à deux disjoints, comme dans le problème à une personne étudié en cours. On peut poser S_1 l'ensemble des solutions valides pour une personne :

$$S_1 = \{A \subseteq [\![1, n]\!] \mid \forall i \neq j \in A, E_i \cap E_j = \emptyset\}$$

On veut alors maximiser le nombre total d'évènements, soit :

$$\max_{(A_1, \dots A_p) \in S} |A_1 \cup \dots \cup A_p|$$

avec

$$S = \{(A_1, \dots A_p) \mid \forall k \in [\![1, p]\!] A_k \in S_1\}$$

Question 3. Pour $p \in \mathbb{N}$, notons S_p l'ensemble défini à la question précédente, c'est à dire l'ensemble des manières différentes qu'ont p personnes d'assister à un festival. Pour le problème de cette question, on cherche essentiellement le plus petit p pour lequel on peut trouver une solution dans S_p qui couvre **tous** les évènements. Une solution sera donc un tuple $(p, A_1, \dots A_p)$ tel que $(A_1, \dots, A_p)$ est une solution valide du problème précédent, et tel que $A_1 \cup \dots A_p = [\![1, n]\!]$. D'où :

$$\min_{(p, A_1, \dots A_p) \in S} p$$

avec

$$S = \{(p, A_1, \dots A_p) \mid p \in \mathbb{N}, (A_1, \dots, A_p) \in S_p, |A_1 \cup \dots \cup A_p = [\![1, n]\!]\}$$

Question 4. Une solution à ce problème est l'ordre de visite des différentes villes. Puisque le trajet est cyclique, la ville de départ n'a pas d'importance.
Notons P_n l'ensemble des permutations de $[\![0, n-1]\!]$. Une permutation $\sigma \in P_n$ représentera l'ordre de visite : le voyageur part de la ville $\sigma(0)$, puis va à la ville $\sigma(1)$, etc... jusqu'à atteindre la ville $\sigma(n-1)$, après quoi on revient en $\sigma(0)$.

Notons d_{ij} la distance euclidienne entre les villes V_i et V_j. Alors, la distance totale du trajet défini par σ est :

$$d_{\sigma(0)\sigma(1)} + d_{\sigma(1)\sigma(2)} + \cdots + d_{\sigma(n-2)\sigma(n-1)} + d_{\sigma(n-1)\sigma(0)}$$

C'est à dire :

$$\sum_{i=0}^{n-1} d_{\sigma(i)\sigma(i+1 \bmod n}$$

Ainsi, le problème TSP peut s'exprimer :

$$\min_{\sigma \in S} \sum_{i=0}^{n-1} d_{\sigma(i)\sigma(i+1 \bmod n}$$

avec

$$S = P_n$$

Question 5. Une solution au problème est simplement un emplacement où placer le centre du parc d'éoliennes, c'est à dire des coordonnées (x, y). On veut minimiser le nombre de maisons se situant à moins de r de (x, y), soit :

$$\min_{(x,y) \in S} |\{i \in [\![0, n-1]\!] | (x - x_i)^2 + (y - y_i)^2 \leq r^2\}|$$

avec

$$S = [0; 1]^2$$

Question 6. Pour une position du parc donnée (x, y) donnée, on s'intéresse à la distance maximale entre le parc et une maison, c'est à dire à :

$$\max_{i=0}^{n-1} \sqrt{(x - x_i)^2 + (y - y_i)^2}$$

On veut minimiser cette quantité, le problème d'optimisation est donc :

$$\min_{(x,y) \in S} \max_{i=0}^{n-1} \sqrt{(x - x_i)^2 + (y - y_i)^2}$$

avec

$$S = [0; 1]^2$$

Question 7. Une solution est problème est la donnée :

— des entrepôts ayant été ouverts ;
— de la manière dont les entrepôts sont affectés aux villes.

Formellement, on peut donc considérer un sous-ensemble $I \subseteq [\![1, m]\!]$ d'entrepôts ouverts, et $f : [\![1, n]\!] \to I$ la fonction telle que $f(i)$ dit quel entrepôt est relié à la ville V_i. Il n'y a aucune contrainte, et le coût s'exprime alors simplement : on a ouvert $|I|$ entrepôts, et la ville V_i est reliée à l'entrepôt à la location $L_{f(i)}$ pour un coût $c_{i,f(i)}$. Le problème est donc :

$$\min_{(I,f)\in S} c|I| + \sum_{i=0}^{n-1} c_{i,f(i)}$$

avec

$$S = \{(I, f) \mid I \subseteq [\![1, m]\!], f : [\![1, n]\!] \to I\}$$

Exercice 2.

Banquet

Pour le problème `Subset-Sum`, on peut essayer de prendre les éléments de E du plus grand au plus petit, et de les ajouter à la solution s'ils rentrent. Par exemple, si $E = \{1, 2, 5, 8, 9, 17, 26, 41\}$ et $t = 39$, l'algorithme va construire la solution $\{26, 9, 2, 1\}$, pour une valeur de 38. Pseudo-code :

Algorithme 1 : `glouton_subset`(L, t)

Entrée(s) : E un ensemble d'entiers, t un entier
Sortie(s) : liste d'éléments de E de somme $\leq t$

```
1 L ← liste des éléments de E, dans l'ordre décroissant;
2 S ← [];
3 s ← 0// somme actuelle
4 pour x dans L faire
5     si s + x ≤ t alors
6         Ajouter x à S;
7         s ← s + x;
8 retourner S
```

La complexité est $\mathcal{O}(n \log n)$ à cause du tri initial. Si on dispose déjà de la liste triée en entrée, la boucle For seule coûte $\mathcal{O}(n)$.
Ce n'est pas optimal, par exemple si $E = \{3, 4, 6\}$ et $t = 7$: l'algorithme glouton va choisir 6 seul, mais il existe une solution exacte en prenant 3 et 4.

Exercice 3.

Bin-packing

Question 1. On a :

$$\texttt{Bin-Packing} : \min_{(p,I_1,\dots I_p)\in S} p$$

où :

$$S = \{(p, I_1, \dots I_p) \mid p \in \mathbb{N} \text{ et les } (I_k)_{k\in[\![1,p]\!]} \text{ forment une partition de } [\![1, n]\!]\}$$

Chaque I_k correspond à un disque : c'est l'ensemble des fichiers que l'on stocke sur ce disque.

Question 2. Si l'on a une solution utilisant p disques, alors on sait que l'intégralité des fichiers peut rentrer sur un espace pC, donc $p \geq \frac{S}{C}$. C'est vrai en particulier pour une solution optimale : la valeur optimale est minorée par $\frac{S}{C}$.

Question 3. Après application de l'algorithme, on a utilisé 6 disques, contenant les fichiers suivants (seules les tailles sont indiquées) :

$$[3,2],[7],[4,5,1],[6],[5,4],[5]$$

La somme des tailles fait 42, d'après la question précédente il est nécessaire d'utiliser au moins 4.2 disques, donc au moins 5 disques (on doit utiliser un nombre entier de disques !). Il existe bien une solution en 5 disque : on peut l'obtenir en regroupant le premier et le dernier disque de la solution gloutonne.

Question 4. Soit p le nombre de disques générés par Next-Fit. Notons A_i la taille effective utilisée dans le disque numéro i. Supposons par l'absurde qu'il existe $i \in [\![1, p-1]\!]$ tel que $A_i + A_{i+1} \leq C$ ont, au total, des fichiers de taille $\leq C$. Alors, en notant t la taille du premier fichier ayant été ajouté au disque $i+1$, on a $t > C - A_i$, car l'algorithme Next-Fit a été obligé de créer le disque $i+1$. C'est absurde, car on a alors $A_{i+1} \geq t > C - A_i$, soit $A_i + A_{i+1} > C$. Donc $A_i + A_{i+1} > C$.

Question 5. Reprenons les notations précédentes. On sait que les disques doivent contenir tous les disques, donc $A_1 + \cdots + A_{K_g} = S$. De plus, on peut regrouper les disques deux à deux : A_1 avec A_2, A_3 avec A_4, etc... et former $\lfloor \frac{K_g}{2} \rfloor$ couples complets, avec un éventuel disque solitaire si K_g est impair.
Ainsi, $S = A_1 + \cdots + A_{K_g} > \lfloor \frac{K_g}{2} \rfloor C$, soit $\lfloor \frac{K_g}{2} \rfloor < \frac{S}{C} \leq K^*$. Les deux côtés de l'inégalité étant des entiers, on peut en déduire que :

$$\lfloor \frac{K_g}{2} \rfloor + 1 \leq K^*$$

et donc que $\frac{K_g}{2} \leq \lfloor \frac{K_g}{2} \rfloor + 1 \leq K^*$.
L'algorithme Next-Fit renvoie toujours une solution utilisant au pire deux fois plus de disques qu'une solution optimale !