Chapitre

IV

Problèmes de cheminement

Partie A. Classification des problèmes mathématiques

Des algorithmes plus perfectionnés sont présentés dans "Graphes et algorithmes" de M.Gondran et M.Minoux. Les algorithmes sont présentés en pseudo code.

Les problèmes sont répartis en trois classes selon leurs difficultés. Un problème est :

- de classe P si l'algorithme pour le résoudre s'exécute en un "temps polynomial", c'est à dire si le nombre d'opérations est toujours plus petit qu'une puissance du nombre de données ;
- de classe NP si toute solution proposée peut être vérifiée en un "temps polynomial" (Cf. ci dessus) ;
- de classe NP complet si l'on savait résoudre un problème NP complet en un "temps polynomial" alors on saurait résoudre tous les problèmes NP en un "temps polynomial".

Actuellement, on connaît des problèmes NP que l'on ne sait pas résoudre en un "temps polynomial" donc la classe P est strictement incluse dans la classe NP. On connaît également plusieurs centaines de problèmes NP complets (par exemple : la recherche d'un chemin hamiltonien) qui revêtent donc un intérêt tout particulier.

Partie B. Recherche des composantes connexes

1. Présentation des objectifs

La vérification de la connexité d'un graphe est un des premiers problèmes de la théorie des graphes. En effet, on se ramène généralement à un graphe connexe en ne considérant qu'une composante connexe à la fois.

De nombreux algorithmes permettent de trouver l'ensemble des sommets appartenant à une même composante connexe. On se limitera, dans ce document, à montrer une version simple, non récursive : *l'algorithme de recherche de la composante connexe contenant un sommet particulier" (Trémeaux 1882 Tarjan 1972).*

Principe de l'algorithme :
Cet algorithme a pour objectif de *rechercher la composante connexe contenant un sommet particulier*. Cette recherche est effectuée par création d'une*arborescence* dont la *racine* est le sommet choisi. La recherche s'effectue en profondeur d'abord.

Si le *nombre de sommets "couverts"* est égal à l'ordre du graphe, alors il n'y a qu'une seule composante connexe et le graphe est connexe. Dans le cas, contraire, on recommence avec un sommet (qui n'appartient pas à cette composante connexe). On en déduit donc le nombre de connexité du graphe.

Pour pouvoir atteindre tous les sommets d'une même composante connexe, on va considérer le graphe orienté associé au multigraphe associé au graphe de départ. Concrètement, *on parcourt* non pas le graphe d'origine mais un *graphe associé et symétrique*: soit $G(X,U)$ on cree $G^*(X,U^*)$ tel que

$u=(x,y)\in U \Rightarrow u\in U^*$ et $u^*=(y,x)\in U^*$

2. Algorithme de Trémeaux-Tarjan

Principe de l'algorithme :
Cet algorithme est un exemple de recherche en profondeur[1] On essaie à partir de chaque sommet du graphe $G*$ de construire le plus grand chemin possible (sans cycle) dans $G*$. Lorsqu'on a "exploré" pour un sommet donné tous ses voisins dans G (ses successeurs dans H), on remonte au prédécesseur de ce sommet dans H . A partir de ce prédécesseur, on essaie alors de "redescendre" dans $G*$.
[1 Depth First Search]

Structure des données :
On suppose que les sommets de G sont numérotés de 1 à $|X|$. Soit H l'arborescence couvrant la composante connexe de $G*$ recherchée. Le graphe étant fini, on peut définir les variables suivantes :

- i et j : sommets du graphe G donc de $G*$ et par conséquent de H
- nbvois : tableau de nombres ; $nbvois_i$ est le nombre de voisins de i dans G

$\left(\text{nbvois}_i = \left|\Gamma_G(i)\right| = \left|\Gamma^+_{G*}(i)\right|\right)$;

- nbexpl : tableau de nombres ; $nb\exp l_i$ est le nombre de voisins de i dans G (successeurs de i dans $G*$) qui ont été atteints en partant de i .
- ordreH : l'ordre de H donc le cardinal de la composante connexe ;
- numH : tableau de rangs dans H ; $numH_i$ est le rang du sommet i dans H ;
- predH : tableau de prédécesseurs ; $predH_i$ est le numéro du prédécesseur de i dans H ;

On suppose que l'on dispose également de la fonction suivante qui doit avoir été écrite auparavant :

- vois : $vois(k, i)$ est le k$^{\text{ième}}$voisin de i dans G (successeur de i dans $G*$) en supposant qu'ils ont été triés par numéros.

Initialisation :

Soit S le sommet dont on recherche la composante connexe. Au départ, on pose que le prédécesseur de S dans H c'est S (convention pour l'algorithme). L'ordre de H est 1 car $S \in H$ et le premier sommet de H donc :

$\forall\ i \in G,\ \text{nbvois}_i \leftarrow \left|\Gamma_G(i)\right|$; $\text{nbexpl}_i \leftarrow 0$; $\text{numH}_i \leftarrow 0$; $\text{predH}_i \leftarrow 0$;

$i \leftarrow S$; $\text{pred}_i \leftarrow S$; $\text{ordreH} \leftarrow 1$; $\text{numH}_i \leftarrow \text{ordreH}$;

Itérations :

tant que $(i \neq S \ \text{ ou } \ \text{nbexpl}_i < \text{nbvois}_i)$

si $\text{nbexpl}_i = \text{nbvois}_i$ *alors*on a "exploré" tous les voisins de i dans G donc on "remonte" dans H et le prédécesseur de i (dans H) devient le sommet courant.

$i \leftarrow \text{pred}_i$

sinon on "descend" si possible (dans H) vers le prochain voisin de i dans G (successeur de i dans $G*$)

$\text{nbexpl}_i \leftarrow \text{nbexpl}_i + 1$; $j \leftarrow \text{vois}(\text{nbexpl}_i, i)$;

si $\text{pred}_j = 0$ *alors* j n'a pas encore de prédécesseur dans H donc on a trouvé un nouveau sommet de H

$\text{pred}_j \leftarrow i$; $\text{ordreH} \leftarrow \text{ordreH} + 1$; $\text{numH}_j \leftarrow \text{ordreH}$;

ce sommet devient le sommet courant ("parcours en profondeur d'abord")

$i \leftarrow j$;

fin "si"

fin "si"

fin "tant que"

Arrêt des itérations :

si $\text{ordreH} = \left|X\right|$ alors on a construit une arborescence maximale de $G*$, c'est à dire un arbre couvrant de G et par conséquent G *est connexe*.

sinon H est de la forme $H(X_H, T)$ avec $X_H \subset X$ et $T \subset U*$,

où $X_H = \{\ i \in X / 1 \leq \text{numH}_i \leq \text{ordreH}\ \}$

et la composante connexe de G trouvée est $G_{X_H}(X_H, U_{X_H})$
et on recommence avec un sommet de $X - X_H$
fin "si"

Partie C. Recherche du plus court chemin

1. Présentation des conditions

Soit un graphe G, on associe à chaque arc u de G une *longueur* $l(u)$ telle que $\forall\, u \in U,\ l(u) \geq 0$. Soit a et b deux sommets de G, il s'agit de trouver un chemin $\mu(a, b)$ tel que $l(\mu) = \sum_{u \in \mu} l(u)$ soit le plus petit possible. $\mu(a, b)$ est alors appelé le *plus court chemin* de a à b.

2. Algorithme de Moore-Dijkstra

Principe de l'algorithme :
Soit un graphe $G(X, U)$ *connexe* / $X = \{1, 2, ..., N\}$ [1]. L'algorithme de Moore 1957 et Dijkstra 1959 permet d'obtenir le plus court chemin d'un sommet 1 à tous les autres. Cet algorithme fonctionne en au plus $|X| - 1$ itérations. On suppose au départ que la longueur du plus court chemin de 1 à $i \neq 1$ vaut $+\infty$ .On construit une arborescence H de racine 1 qui couvre X.
[1][Si G n'est pas connexe alors on remplace par la suite X par la composante connexe de G contenant 1]

Structure de données :
On dispose de la longueur $l(i, j) \geq 0,\ \forall\ (i, j) \in U$. On définit π_i^* comme la longueur minimum des chemins de 1 à i et $\pi_1^* = 0$. On partitionne X en deux sous ensembles S et $\bar{S} = X - S$ et on attribue à chaque sommet i du graphe une étiquette π_i définie par :

- si $i \in S,\ \pi_i = \pi_i^*$
- si $i \in \bar{S} = X - S,\ \pi_i = \min_{\forall k \in S/(k,i) \in U} \left(\pi_k^* + l(k, i) \right)$

On utilise un tableau predH de prédécesseurs dans H : predH_i est le prédécesseur de i dans H.

Initialisation :
$\forall\, i \in \Gamma_G^+(1),\ \pi_i \leftarrow l(1, i)$; $\text{predH}_i \leftarrow 1$;
$\forall\, i \in X - \Gamma_G^+(1),\ \pi_i \leftarrow \infty$; $\text{predH}_i \leftarrow 0$;
$\bar{S} \leftarrow \{2, \cdots, N\}$; $\pi_1 \leftarrow 0$; $\text{predH}_1 \leftarrow 0$;

Itérations :
tant que $\left(\left| \overline{S} \right| \neq 0 \right)$
on recherche un sommet de $\overline{S}$ le plus proche des sommets de S
$\forall\, i \in \overline{S},\ si\ \pi_j = \min_{\forall i \in \overline{S}} (\pi_i)\ alors\ j \leftarrow i\ fin\ "si"$;
on retire j de la liste des sommets restant
$\overline{S} \leftarrow \overline{S} - \{ j \}$
si $\left| \overline{S} \right| \neq 0$ *alors* il reste des éléments dans $\overline{S}$, on met à jour l'étiquette de chaque sommet restant
$\forall\ i \in \overline{S} \bigcap \Gamma_G^+ (i)\ ,\ si\ \pi_i \geq \pi_j + l(j , i)\ alors$
$\pi_i \leftarrow \pi_j + l(j , i)$; $predH_i \leftarrow j$; ;
fin "si"
fin "si"
fin "tant que"

Arrêt des itérations :
$\left| \overline{S} \right| = 0 \Rightarrow \forall\ i \in X , \pi_i^* \neq +\infty$ donc on a trouvé la longueur du plus court chemin de 1 à tout autre sommet i . On peut parcourir le plus court chemin de 1 à i en construisant la liste inverse des prédécesseurs i de H .

Partie D. Recherche d'un arbre de poids extrémum

1. Présentation des objectifs

Soit un graphe $G(X , U)$. A chaque arc $u \in U$, on associe un nombre $w(u)$ appelé le *poids de*l'arc u . Pour tout graphe partiel $G'(X , U')$ avec $U' \subset U$ de G , on défini le *poids de* G' comme le nombre :

$$w(G') = \sum_{u' \in U'} w(u')$$

On suppose par la suite que G est *connexe* [1] On recherche un arbre de poids *maximum* (respectivement minimum) comme étant un arbre $\mathfrak{I}^* (X , T)$ de G tel que :

[1 sinon on appliquera l'algorithme à chaque composante connexe prise séparément.]

$w(\mathfrak{I}^*) = \max_{\forall \mathfrak{I} \text{ arbre de } G} (w(\mathfrak{I}))$ $\left(\text{respectivement } w(\mathfrak{I}^*) = \min_{\forall \mathfrak{I} \text{ arbre de } G} (w(\mathfrak{I})) \right)$

2. Algorithme de Kruskal 1956

Structure de données :
On crée une liste $(Lu_k)_{1 \le k \le |U|}$ d'arcs triés dans l'ordre des poids *décroissants* :

$1 \le k \le k' \le |U| \Rightarrow w(Lu_k) \ge w(Lu_{k'})$
(respectivement croissants : $1 \le k \le k' \le |U| \Rightarrow w(Lu_k) \le w(Lu_{k'})$)

On connaît bien sûr pour chaque arc les numéros de ses extrémités[1]. Soit k une variable de type rang prenant ses valeurs dans l'intervalle $[1 , |U|]$
[1 On peut par exemple créer la matrice SIF donnant pour chaque arc son sommet initial et son sommet final (Cf. Partie Matrices associées à un graphe).]

Initialisation :
$T \leftarrow \{ Lu_1 \}$; on a déjà utilisé Lu_1 donc $k \leftarrow 2$: ;

Itérations :
tant que $(k \le | U |)$
si Lu_k "ne forme pas de cycle avec les arcs de " *alors* l'arc Lu_k est conservé
$T \leftarrow T \cup \{ Lu_k \}$
fin "si"
$k \leftarrow k + 1$;
fin "tant que"

Arrêt des itérations :
$| T | \ge 1$ et $\mathfrak{I}(X , T)$ est un arbre de poids *maximum* (respectivement minimal).