Chapitre 4

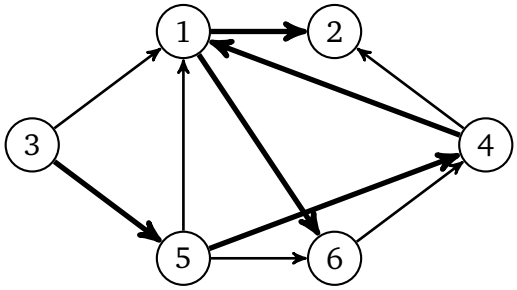
Algorithmes de plus court chemin

4.1 Définitions

Définition 4.1 (racine, arborescence) *Un sommet i_0 est* racine *d'un graphe G, ssi pour tout sommet i de $X \setminus \{i_0\}$, il existe au moins un chemin $\mu_{i_0 i}$ dans G allant de i_0 à i.*

Une arborescence *est un arbre qui admet une racine.*

Exemple 21 *Le graphe suivant admet une arborescence de racine 3.*



Définition 4.2 (poids d'un chemin, chemin ij-minimal) *Soient $G = (X = [\![1, n]\!], U)$ un graphe orienté pondéré, c'est-à-dire que chaque arc (i, j) est affecté d'un poids l_{ij}, le* poids d'un chemin *μ dans G est égal à la somme du poids de ses arcs sera noté $l(\mu_{i_0 i})$).*

Un chemin ij-minimal *est un chemin de i à j de poids minimum parmi tous les chemins de i à j.*

On suppose que i_0 est racine de G, et on s'intéresse à déterminer pour tout sommet i de $X \setminus \{i_0\}$, un plus court chemin de i_0 à i, c'est à dire un chemin de poids minimum, ou encore chemin $i_0 i$-minimal.

Propriété 4.1 *Si les l_{ij} sont positives alors tout chemin ij-minimal est soit élémentaire (ne passant pas deux fois par le même sommet) soit contient un circuit de poids nul.*

4.2 L'algorithme de Bellman

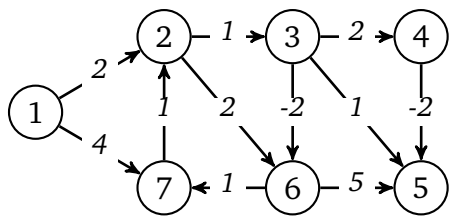
L'algorithme de Bellman-Kalaba 1958 (encore appelé Bellman-Ford ou Bellman ou Ford) calcule la longueur d'un plus court chemin à partir d'un sommet i_0 d'un graphe valué quelconque. Le principe de l'algorithme est de considérer d'abord les chemins de 1 arc (longueur λ_1) entre i_0 et tous les autres sommets puis 2 arcs (longueur λ_2) et ainsi de suite. Si le graphe ne contient pas de circuit de longueur négative il existe un plus court chemin qui est élémentaire qui a donc au plus n-1 arcs. Si G n'a pas de circuit de longueur négative $\lambda_{n-1}(j)$ est la longueur du plus court chemin de i_0 à j.

Algorithm 2 L'algorithme de Bellman-Kalaba-Ford

procedure BELLMAN-KALABA-FORD
Données : $G = (X, U)$ un graphe connexe orienté pondéré par l (l_{ij} : poids de l'arc (i, j))
Résultat : Si G a un circuit de longueur négative pas de solution sinon pour chaque sommet $i \neq i_0$, distance du plus court[long] chemin de i_0 à i ou ∞
Variables : j : sommet courant, k : étape courante

$\lambda_0(i_0) \leftarrow 0$
for $i \in X - \{i_0\}$ **do** $\lambda_0(i) = \infty$ $[-\infty]$
$k \leftarrow 1$
while $k \leq n$ **do**
  for $j \in X$ **do** $\lambda_k(j) = \min[\max]_{i \in \Gamma^-(j)}(\lambda_{k-1}(i) + l_{ij})$
  if $\lambda_k = \lambda_{k-1}$ **then**
    STOP
  else
    $k \leftarrow k + 1$
if $k = n + 1$ **then**
  G admet un circuit < 0 $[> 0]$
else
  $\forall j \in X \setminus \{i_0\}, \lambda_k(j)$ est la longueur d'un chemin $i_0 j$-minimal [maximal]

Exemple 22 *Soit $G = (X = [\![1, 7]\!], U)$ le graphe orienté suivant, on désire connaître les plus courts chemins d'origine ① :*



k	*Sommets*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0	λ_0	*0*	∞	∞	∞	∞	∞	∞
1	λ_1	*0*	*2*	∞	∞	∞	∞	*4*
2	λ_2	*0*	*2*	*3*	∞	∞	*4*	*4*
3	λ_3	*0*	*2*	*3*	*5*	*4*	*1*	*4*
4	λ_4	*0*	*2*	*3*	*5*	*3*	*1*	*2*
5	λ_5	*0*	*2*	*3*	*5*	*3*	*1*	*2*
STOP								

La complexité de l'algorithme de Bellman-Kalaba est de l'ordre de n^3 puisqu'elle est en $o(nm)$: au pire, on regarde tous les prédecesseurs de tous les sommets (m arcs), et ceci n fois. Lorsque le graphe est sans circuit, on peut diminuer grandement sa complexité. L'algorithme de Bellman pour les graphes *sans circuit* se sert des niveaux pour calculer les *plus courts chemins* en une seule passe sa complexité est en $o(m)$.

Algorithm 3 L'algorithme de Bellman pour les graphes sans circuits

procedure BELLMAN-SANS-CIRCUITS

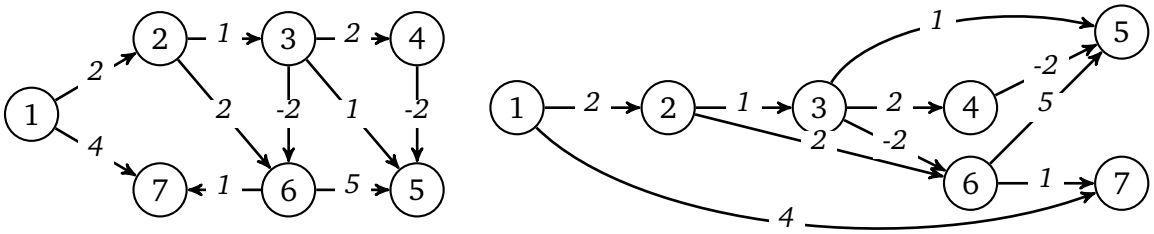
Données : $G = (X, U)$ un graphe orienté pondéré sans circuits (poids de signe quelconque), les sommets $x_1 \dots x_n$ étant ordonnées topologiquement.

Résultat : pour chaque sommet $i \neq 1$, $\lambda(i)$ distance du plus court[long] chemin du sommet 1 vers les autres sommets.

$\lambda(x_1) \leftarrow 0$
for j **in** $x_1 \,..\, x_n$ **do**
$\lambda(x_j) = \min[\max]_{x_i \in \Gamma^-(x_j)}(\lambda(x_i) + l_{x_i x_j})$

Exemple 23 *Soit $G = (X = [\![1,7]\!], U)$ le graphe orienté sans circuit suivant, on désire connaître les plus courts chemins d'origine* ①

Graphe mis en niveaux :



On obtient donc successivement, $\lambda(1) = 0$, $\lambda(2) = 2$, $\lambda(3) = 3$, $\lambda(4) = 5$, $\lambda(6) = min(\lambda(2) + l_{26}, \lambda(3) + l_{36}) = 1$, $\lambda(5) = 3$, $\lambda(7) = 2$.

Ici, grace à la mise à niveau, et comme pour l'algorithme de Dijkstra que l'on étudiera au paragraphe suivant, chaque étape nous fournit la valeur définitive d'un sommet, et on le sait. Si on s'intéresse aux chemins de valeur minimale entre le sommet de départ A et un sommet particulier B, on peut arrêter le calcul dès que B est calculé, même si tous les sommets ne sont pas calculés. On dit que la stratégie est **gloutonne**, ce qui signifie que certains résultats ne seront plus remis en question (un glouton est quelqu'un qui avale et ne peut pas revenir en arrière, c'est-à-dire ne peut pas remettre en cause ce qu'il vient de faire).

Remarque 10 *L'algorithme de Bellman-Kalaba marche pour calculer les plus longs chemins, il admet les valuations négatives et les circuits.*

4.3 Graphe orienté pondéré positivement : algorithme de Dijkstra

Les chemins mis en évidence par les algorithmes proposés ci-dessous sont élémentaires.

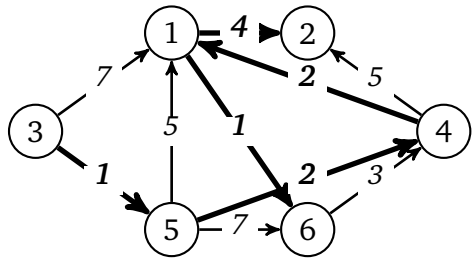
L'algorithme de Dijkstra (1971) [2] (aussi appelé Moore-Dijkstra) permet d'obtenir une arborescence partielle $H = (X, V)$ de G de racine i_0 (c'est-à-dire que H est un graphe sans cycle de racine i_0 (et donc connexe) avec $V \subseteq U$) dans laquelle, pour tout sommet i de $X \setminus \{i_0\}$, l'unique chemin allant de i_0 à i est un chemin $i_0 i$-minimal de G.

Algorithm 4 L'algorithme de Moore-Dijkstra

procedure DIJKSTRA
Données : G : un graphe connexe orienté pondéré **positivement**
i_0 : un sommet racine de G
Résultat : pour chaque sommet $i \neq i_0$, λ_i distance du plus court chemin de i_0 à i et v_i arc d'extrémité terminale i sur ce plus court chemiN
Variables : S : sommets explorés, i : sommet courant, j : successeur de i non encore exploré

$\lambda_{i_0} \leftarrow 0$
for $i = 1$ à n, $i \neq i_0$ **do** $\lambda_i \leftarrow \infty$
$S \leftarrow \emptyset$
while $S \neq X$ **do**
sélectionner i dans $X \setminus S$ tel que $\lambda_i = \min_{j \in X \setminus S} \lambda_j$
$S \leftarrow S \cup \{i\}$
for j de $\Gamma^+(i) \cap (X \setminus S)$ **do**
if $\lambda_i + l_{ij} < \lambda_j$ **then**
//ajustement de j à partir de i et m.à.j. de l'arc d'extrémité j
$\lambda_j \leftarrow \lambda_i + l_{ij}$
$v_j \leftarrow (i, j)$

Exemple 24 *Soit $G = (X = [\![1, 6]\!], U)$ le graphe orienté suivant de racine ③.*

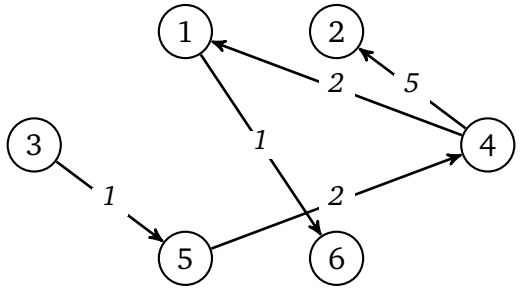


Sommets \ sélectionnés	*Sommets* 1	2	3	4	5	6
Initialisation	∞	∞	⓪	∞	∞	∞
3	*7*	∞	/	∞	①	∞
5	*6*	∞	/	③	/	*8*
4	⑤	⑧	/	/	/	*8*
1	/	*8*	/	/	/	⑥
6	/	*8*	/	/	/	/
2	/	/	/	/	/	/

Normalement, on doit noter les arcs v_j à chaque fois que l'on change le λ_j mais sur un tableau on peut s'en passer : Les nombres entourés dans la colonne d'un sommet j correspondent à la valeur définitive λ_j (distance du chemin minimal de i_0, ici ③, à j). On a entouré ce nombre

sur la ligne où cette valeur apparaît pour la première fois pour ce sommet. Ainsi ce nombre se trouve sur la ligne du sommet i à partir duquel la marque λ_j a été ajustée à cette valeur. Cette convention permet de connaître l'arc dont l'extrémité terminale est j dans l'arborescence de chemin $i_0 j$-minimaux.

Ainsi le sommet ① a obtenu sa valeur définitive à partir du sommet ④, ce qui signifie que l'arc (4,1) appartient à l'arborescence. On obtient finalement l'arborescence suivante :



Complexité de l'algorithme $o(n^2)$. Chaque étape de l'algorithme de Dijkstra nous fournit la valeur définitive d'un sommet, et on le sait. Si on s'intéresse aux chemins de valeur minimale entre le sommet de départ A et un sommet particulier B, on peut arrêter le calcul dès que B est calculé, même si tous les sommets ne sont pas calculés. La stratégie est donc **gloutonne**.

Justification de la validité de l'algorithme de Dijkstra

Montrons que l'arborescence H donnée par l'algorithme est telle que pour tout sommet j, l'unique chemin dans H de i_0 à j est un chemin $i_0 j$-minimal de G.

1. Montrons que les sommets sont sélectionnés dans l'ordre des λ croissant. Au moment où le sommet i est sélectionné, d'après le critère de sélection, λ_i est plus petit que tous les lambda des sommets restants. Soit j le sommet qui sera selectionné juste après i. Avant l'ajustement on a $\lambda_i \leq \lambda_j$, après la phase d'ajustement à partir de i, soit λ_j reste inchangée si j n'est pas un successeur de i, soit λ_j prend la valeur $\lambda_i + l_{ij}$, compte tenu de la positivité des poids, cette valeur reste supérieure à λ_i. Ce raisonnement est vrai à chaque étape.
2. Montrons que le graphe $H = (X, V = \cup_{j \in X \setminus \{i_0\}} v_j)$ donnée par l'algorithme est une arborescence de racine i_0 telle que pour tout sommet j, il existe un unique chemin dans H de i_0 à j de poids λ_j.

   Montrons qu'à toute étape, le graphe $H_S = (S, V_S = \cup_{j \in X \setminus \{i_0\}} v_j)$ est une arborescence de racine i_0 dans laquelle pour tout sommet j de $S \setminus \{i_0\}$ il existe un unique chemin de i_0 à j de poids λ_j.
   - — Au départ, considérons $H_0 = (\{i_0\}, V = \emptyset)$ la propriété est vraie car il n'y a pas de sommet dans $S \setminus \{i_0\}$.
   - — Supposons que cette propriété soit vraie après avoir sélectionné p sommets dans S_p. Montrons qu'elle est vraie après avoir sélectionné le sommet suivant. Soit $H_p = (S_p, V_p)$ le graphe contenant les arcs entrants donnés par l'algorithme après avoir

sélectionnés p sommets. Soit i le sommet non encore sélectionné de λ_i minimal, $H_{p+1} = (S_p \cup \{i\}, V_p \cup \{v_i\})$, l'arc v_i par construction est égal à (k,i) tel que k est un sommet déjà exploré donc de S_p à i et $\lambda_i = \lambda_k + l_{ki}$. Puisque k appartient à H_p, il existe un chemin de i_o à k de longueur λ_k (par hypothèse de récurrence). Donc il existe un chemin de i_0 à i de longueur $\lambda_k + l_{ki} = \lambda_i$. H_p était une arborescence de racine i_0 on a ajouté un sommet et un arc vers ce sommet, on n'a donc pas créé de cycle (puisque ce sommet n'existait pas il n'y avait pas de chemin partant de lui), H_{p+1} est donc sans cycle et de racine i_0 (puisqu'il existe un chemin de i_0 au nouveau sommet).

A chaque étape la propriété est vérifiée, elle l'est à la première étape donc elle l'est à la fin de l'algorithme.

3. Montrons que pour tout arc (i,j) de G, λ_i et λ_j donnés par l'algorithme de Dijkstra vérifient $\lambda_j - \lambda_i \leq l_{ij}$.
   — Si j est entré dans S après i alors après la phase d'ajustement à partir de i, soit $\lambda_j = \lambda_i + l_{ij}$ soit $\lambda_j \leq \lambda_i + l_{ij}$. Ensuite λ_j n'a pu que diminuer. Donc à la fin de l'algorithme, $\lambda_j \leq \lambda_i + l_{ij}$.
   — Si j est entré après i alors $\lambda_j \leq \lambda_i$ (puisque les sommets sont examinés dans l'ordre croissant des λ). Donc a fortiori $\lambda_j \leq \lambda_i + l_{ij}$.
4. Montrons que pour tout sommet $j \neq i_0$, et tout chemin non vide $\mu = [x_1, \dots x_p]$ de G tel que $x_1 = i_0$ et $x_p = j$, $\lambda_j \leq l(\mu)$.
   Calculons $l(\mu) = \sum_{k=1}^{p-1} l(x_k, x_{k+1})$. Donc $l(\mu) \geq \sum_{k=1}^{p-1} \lambda_{x_{k+1}} - \lambda_{x_k} = \lambda_j - \lambda_0 = \lambda_j$.

Remarque 11 *Dans le cas où i_0 n'est pas racine de G l'algorithme se termine avec des sommets dont la marque reste infinie, ces sommets ne sont pas des descendants de i_0.*

Dans le cas d'un graphe non orienté, l'algorithme fonctionne en considérant qu'une arête équivaut à avoir un arc dans chaque sens. On obtient donc des chaînes $i_0 j$*-minimales.*

4.4 Matrice de routage

Définition 4.3 (matrice des longueurs des chemins ij-minimaux) *La matrice des longueurs des chemins ij-minimaux associée à un graphe orienté valué d'ordre n est une matrice de dimension $n \times n$ telle que*

$$\lambda_{ij} = \textit{longueur du chemin } ij\textit{-minimal}$$

Dans le cas non-orienté on considère les chaînes ij-minimales.

Définition 4.4 (matrice de routage) *La matrice de routage associée à un graphe orienté valué d'ordre n est une matrice de dimension $n \times n$ telle que*

$$m_{ij} = \textit{sommet adjacent à } i \textit{ sur le chemin } ij\textit{-minimal}$$

Dans le cas non-orienté on considère les chaînes ij-minimales.

Exemple 25 *L'arborescence est une arborescence de chemin 3i-minimaux dans le graphe, elle permet de remplir les matrices des longueurs et de routage de la façon suivante :*

longueurs

$$\begin{pmatrix} \times & ? & ? & ? & ? & 1 \\ ? & \times & ? & ? & ? & ? \\ 5 & 8 & \times & 3 & 1 & 6 \\ 2 & 5 & ? & \times & ? & 3 \\ 4 & 7 & ? & 2 & \times & 5 \\ ? & ? & ? & ? & ? & \times \end{pmatrix}$$

routage

$$\begin{pmatrix} \times & ? & ? & ? & ? & 6 \\ ? & \times & ? & ? & ? & ? \\ 5 & 5 & \times & 5 & 5 & 5 \\ 1 & 2 & ? & \times & ? & 1 \\ 4 & 4 & ? & 4 & \times & 4 \\ ? & ? & ? & ? & ? & \times \end{pmatrix}$$

longueurs cas non-orienté

$$\begin{pmatrix} \times & ? & 5 & 2 & 4 & 1 \\ ? & \times & 8 & 5 & 7 & ? \\ 5 & 8 & \times & 3 & 1 & 6 \\ 2 & 5 & 3 & \times & 2 & 3 \\ 4 & 7 & 1 & 2 & \times & 5 \\ 1 & ? & 7 & 3 & 5 & \times \end{pmatrix}$$

routage cas non-orienté

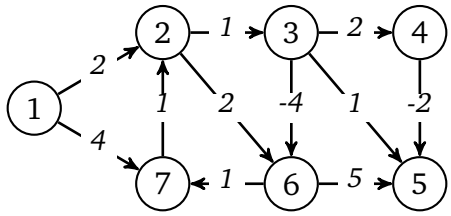
$$\begin{pmatrix} \times & ? & 4 & 4 & 4 & 6 \\ ? & \times & 4 & 4 & 4 & ? \\ 5 & 5 & \times & 5 & 5 & 5 \\ 1 & 2 & 5 & \times & 5 & 1 \\ 4 & 4 & 3 & 4 & \times & 4 \\ 1 & ? & 1 & 1 & 1 & \times \end{pmatrix}$$

Dans le cas non-orienté, la matrice des longueurs est symétrique mais pas la matrice de routage.

4.5 L'algorithme de Moore

Dans un graphe orienté valué par des poids de signe quelconque : il faut faire attention au circuit négatifs, on peut alors utiliser l'algorithme de Moore qui peut réinjecter les successeurs de i dans la liste des sommets à explorer même s'ils ont déjà été sélectionnés. L'algorithme de Moore 1957 [5] permet de détecter les circuits négatifs. Son pseudocode se trouve sur l'algorithme 5.

Exemple 26 *Soit $G = (X = [\![1,7]\!], U)$ le graphe orienté suivant de racine* ①.



Sommets \ Sommets sélectionnés λ_j, v_j	*1*	*2*	*3*	*4*	*5*	*6*	*7*	Σ
Initialisation	*0*	∞	∞	∞	∞	∞	∞	*1*
1		*2 (1,2)*	∞	∞	∞	∞	*4 (1,7)*	*2,7*
2		*(1,2)*	*3 (2,3)*	∞	∞	*4 (2,6)*	*4 (1,7)*	*3,6,7*
3		*(1,2)*	*(2,3)*	*5 (3,4)*	*4 (3,5)*	*-1 (3,6)*	*4 (1,7)*	*4,5,6,7*
6		*(1,2)*	*(2,3)*	*5 (3,4)*	*4 (3,5)*	*(3,6)*	*0 (6,7)*	*4,5,7*
7		*1 (7,2)*	*(2,3)*	*5 (3,4)*	*4 (3,5)*	*(3,6)*	*(6,7)*	*2,4,5*
circuit < 0 *STOP*								

Algorithm 5 L'algorithme de Moore

procedure MOORE
Données : G : un graphe connexe orienté pondéré (poids de signe quelconque)
i_0 : un sommet racine de G
Résultat : soit détection d'un circuit de longueur < 0, soit pour chaque sommet $i \neq i_0$, λ_i distance du plus court chemin de i_0 à i et v_i arc d'extr. i
Variables : Σ : sommets à explorer, i : sommet courant, j : successeur de i non encore exploré

for $i = 1$ à n **do** $\lambda_i \leftarrow \infty$
$\lambda_{i_0} \leftarrow 0$
$\Sigma \leftarrow \{i_0\}$
while $\Sigma \neq \emptyset$ **do**
sélectionner i dans Σ tel que $\lambda_i = \min_{j \in \Sigma} \lambda_j$
$\Sigma \leftarrow \Sigma \setminus \{i\}$
for tout j de $\Gamma^+(i)$ **do**
if $\lambda_i + l_{ij} < \lambda_j$ **then**
if l'adjonction de (i,j) à $\bigcup_{j \in X} v_j$ crée un circuit **then**
return "circuit de longueur < 0"
else
// ajustement de j à partir de i
// et mise à jour de l'arc d'extrémité terminale j
$\lambda_j \leftarrow \lambda_i + l_{ij}$
$v_j \leftarrow (i, j)$
$\Sigma \leftarrow \Sigma \cup \{j\}$

Exemple 27 *Si on prend -2 comme pondération de l'arc (3,6), on obtient :*

Sommets \ sélectionnés / *Sommets* λ_j, v_j	*1*	*2*	*3*	*4*	*5*	*6*	*7*	Σ
Initialisation	*0*	∞	∞	∞	∞	∞	∞	*1*
1		*2 (1,2)*	∞	∞	∞	∞	*4 (1,7)*	*2,7*
2		*(1,2)*	*3 (2,3)*	∞	∞	*4 (2,6)*	*4 (1,7)*	*3,6,7*
3		*(1,2)*	*(2,3)*	*5 (3,4)*	*4 (3,5)*	*1 (3,6)*	*4 (1,7)*	*4,5,6,7*
6		*(1,2)*	*(2,3)*	*5 (3,4)*	*4 (3,5)*	*(3,6)*	*2 (6,7)*	*4,5,7*
7		*(1,2)*	*(2,3)*	*5 (3,4)*	*4 (3,5)*	*(3,6)*	*(6,7)*	*4,5*
5		*(1,2)*	*(2,3)*	*5 (3,4)*	*(3,5)*	*(3,6)*	*(6,7)*	*4*
4		*(1,2)*	*(2,3)*	*(3,4)*	*3 (4,5)*	*(3,6)*	*(6,7)*	*5*
5		*(1,2)*	*(2,3)*	*(3,4)*	*(4,5)*	*(3,6)*	*(6,7)*	$\emptyset$

On constate que cet algorithme n'est pas un algorithme glouton puisqu'il peut revenir sur la valeur de sommets déjà sélectionnés.

Les graphes de jeux sont énormes puisque les sommets sont les différents états du jeux et les arcs les passage possibles entre deux états en jouant un coup. La recherche de plus courts chemins d'un sommet vers un sommet intéressant revient à rechercher une stratégie

gagnante mais vu l'énormité des graphes, on ne génère pas forcément tout le graphe. C'est pourquoi on utilise plutot des algorithmes de type A^* [3] qui est une variante de Dijkstra utilisant une fonction appelée heuristique qui estime la valeur d'un plus court chemin d'un sommet donné à la solution et une fonction qui donne les successeurs d'un sommet.

En résumé les principaux algorithmes de recherche de plus court chemin sont

— Bellman-Kalaba pour les graphes pondérés de poids quelconque (cet algorithme est aussi utilisable pour trouver les chemins maximaux contrairement à Dijkstra).
— et Bellman pour les graphes sans-circuit.
— Dijkstra qui est plus efficace que Bellman-Kalaba mais ne s'applique qu'à des graphes pondérés positivement,
— l'algorithme de Moore est une variante de Dijkstra permettant de traiter des graphes pondérés quelconques.
— Si l'on désire connaître tous les plus courts chemins possibles à partir de tous les sommets du graphe vers tous les autres sommets, il existe l'algorithme de Roy-Floyd-Warshall (1959) qui travaille sur une matrice d'adjacence. Sa complexité étant en $o(n^3)$ il est en général moins couteux en mémoire d'utiliser Dijkstra (ou ses variantes utilisant une structure spéciale pour stocker les sommets non-encore explorés en $o(n.log(n))$) en considérant chaque sommet successivement comme source du graphe.
— Si la pondération des arcs est uniforme alors le plus court chemin est le plus court en nombre d'arcs et un algorithme d'exploration en largeur d'abord est le plus efficace $o(m)$.

Bibliographie

[1] Cormen, Leiserson, Rivest, and Stein. *Introduction to Algorithms Second Edition*. McGraw-Hill, 2001.

[2] E. W. Dijkstra. A short introduction to the art of programming. Technical Report EWD316, T. H. Eindhoven, The Netherlands, August 1971.

[3] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2) :100–107, July 1968.

[4] J.B. Kruskal. On the shortest spanning sub-tree and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, 1956.

[5] E. Moore. The shortest path through a maze. In *Proceedings of the International Symposium on Theory of Switching*, 1957.

[6] R.C. Prim. Shortest connection networks and some generalisations. *Bell System Technical Journal*, 36 :1389–1401, 1957.

[7] Robert E Tarjan and Jan Van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM (JACM)*, 31(2) :245–281, 1984.