Chapitre n°6

Graphes pondérés

Table des matières

1. Présentation et intérêt

Les **graphes pondérés** forment une classe de graphes pour lesquels des valeurs, appelées « poids », sont associées aux liens constituant ces graphes. Ces poids permettent de représenter des flux, des coûts, etc. associés aux liens auxquels ils sont affectés. Par exemple, les graphes pondérés permettent de représenter :

- le réseau global des aéroports et les flux de passagers entre chaque couple d'aéroports ;
- les flux de voitures dans une ville, pour laquelle on associe un flux (en nombre de voitures par heure, par exemple) à chaque rue de la ville ;
- la circulation d'un courant électrique dans un circuit électrique, pour lequel on associe une intensité électrique à chaque fil reliant deux nœuds du réseau ;
- les flux monétaires dans un réseau financier ;
- etc.

2. Définition et représentation

Nota : afin de simplifier les notations utilisées et d'éviter d'alourdir inutilement la présentation, on utilisera ici le vocabulaire associé aux graphes pondérés orientés (arc, chemin, etc.). Tous les concepts évoqués sont transposables aux graphes pondérés non-orientés.

Définition : graphe pondéré

Un graphe pondéré $G = (S, A, p)$ est un graphe $G = (S, A)$ auquel on a associé une fonction de valuation p :

$$p : \begin{cases} A \longrightarrow \mathbb{R} \\ (s_i, s_j) \longmapsto p\left((s_i, s_j)\right) \end{cases}$$

permettant de définir un poids associé à chaque arc.

Ex : le graphe $G = (S, A, p)$ ci-contre est un exemple de graphe pondéré. Voici le tableau définissant la fonction de valuation p sur ce graphe :

Arc	$(0,1)$	$(0,2)$	$(1,3)$	$(1,4)$	$(2,3)$	$(3,0)$	$(4,1)$
Poids	$+2$	-1	-6	$+5$	$+4$	$+7$	-2

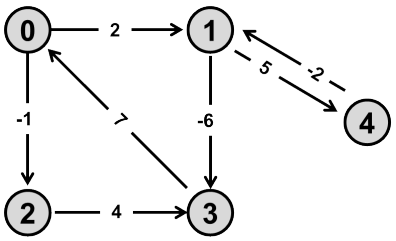


Fig. 1 – exemple de graphe pondéré

Nota : bien que ce soit – en général – nettement moins intéressant dans le cadre d'applications pratiques, il est également possible de définir une fonction de valuation sur les sommets du graphe.

Nota : dans le cadre de ce cours, on se limitera principalement à l'étude des graphes pondérés de poids positifs.

Il est possible d'utiliser la représentation matricielle des graphes afin d'implémenter un graphe pondéré :

Définition : matrice d'adjacence associée à un graphe pondéré

La matrice d'adjacence associée à un graphe pondéré $G = (S, A, p)$ est une matrice $\mathcal{M}$ dont les coefficients sont donnés par :

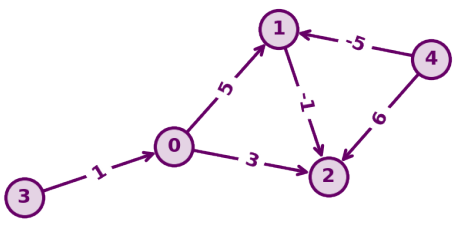
$$\mathcal{M}_{i,j} = \begin{cases} p\left((s_i, s_j)\right) & \textbf{si } (s_i, s_j) \in A \\ +\infty & \textbf{sinon} \end{cases}$$

(où l'on suppose que la numérotation des lignes et colonnes commence à 0).

Ex : la matrice suivante :

$$\begin{pmatrix} \infty & 5 & 3 & \infty & \infty \\ \infty & \infty & -1 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & \infty \\ \infty & -5 & 6 & \infty & \infty \end{pmatrix}$$

constitue une implémentation du graphe pondéré ci-contre :



3. Métrique(s) sur un graphe pondéré

3.1. Poids associé à un chemin

Pour tout chemin $\mathcal{C}_{u_0 \longrightarrow u_\ell} = (u_0 \longrightarrow u_1 \cdots u_{\ell-1} \longrightarrow u_\ell)$ de longueur ℓ d'un graphe pondéré $G = (S, A, p)$, il est possible de définir un **poids** $\mathcal{P}(\mathcal{C}_{u_0 \longrightarrow u_\ell})$ associé à ce chemin. Le poids usuel consiste à sommer les poids de chaque arête empruntée.

Définition : poids (usuel) d'un chemin dans un graphe pondéré

On définit (usuellement) le poids $\mathcal{P}(\mathcal{C}_{u_0 \longrightarrow u_\ell})$ d'un chemin $\mathcal{C}_{u_0 \longrightarrow u_\ell} = (u_0 \longrightarrow u_1 \cdots u_{\ell-1} \longrightarrow u_\ell)$ dans un graphe pondéré $G = (S, A, p)$ comme :

$$\mathcal{P}(\mathcal{C}_{u_0 \longrightarrow u_\ell}) = \sum_{k=0}^{\ell-1} p(u_k, u_{k+1})$$

Il existe d'autres façons de définir un poids associé à un chemin. On peut – par exemple – définir le poids d'un chemin comme étant le minimum des poids des arcs constituant ce chemin, ou encore comme étant la moyenne des poids des arcs constituant ce chemin, etc.

La définition ci-dessus n'a donc rien d'universel : il faut toujours tenir compte de cette fonction de poids lors de l'étude d'un graphe pondéré. À défaut, et en l'absence d'indication particulière de la part de l'énoncé, on prendra le poids usuel défini ci-dessus.

3.2. Distance entre deux sommets

Étant donnés deux sommets u_0 et u_ℓ, la **distance** entre deux sommets d'un graphe pondéré est définie à partir des différents poids $\mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_j})$ associés à chaque chemin $\mathcal{C}_{s_i \longrightarrow s_j}$ de s_i vers s_j. Ici encore, il existe plusieurs manières de définir cette distance : la distance usuelle entre s_i et s_j consiste à prendre le minimum des poids des chemins entre ces deux sommets :

Définition : distance (usuelle) entre deux sommets d'un graphe

La distance (usuelle) $\delta(s_i, s_j)$ entre deux sommets s_i et s_j d'un graphe pondéré $G = (S, A, p)$ est définie comme :

$$\delta(s_i, s_j) = \begin{cases} \min\left(\mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_j})\right) & \text{s'il existe au moins un chemin de } s_i \text{ vers } s_j \\ +\infty & \text{sinon} \end{cases}$$

c'est-à-dire la distance de s_i à s_j est le poids minimal parmis les poids des chemins de s_i vers s_j.

Attention ! Si le graphe est orienté, cette distance n'est pas symétrique : la distance de s_i vers s_j peut être plus grande que celle de s_j vers s_i.

Trouver la distance entre deux sommets s_i et s_j d'un graphe consiste à résoudre le problème dit de « **recherche de plus court chemin** » entre ces sommets.

Le problème de la détermination de la distance entre deux sommets d'un graphe – c'est-à-dire la détermination d'un plus court chemin entre ces deux sommets – est un problème central de

la théorie des graphes. Trouver une solution à ce problème permet par exemple :

- de se rendre d'un point A à un point B en un minimum de temps / coût / etc. ;
- de créer un moteur de recherche tel que Google (on recherche la page web la plus "proche" de la requête formulée) ;
- etc.

4. Algorithmes de recherche de plus court chemin

4.1. Stratégie générale

La plupart des algorithmes couramment utilisés afin de déterminer la distance entre un sommet d'origine s_i et les autres sommets s_j d'un graphe pondéré reposent sur la construction d'un coût $\lambda_{s_i}(s_j)$ associé à chaque sommet s_j du graphe. Ce coût représente une borne maximale de la distance $\delta(s_i, s_j)$, laquelle est mise à jour au fur et à mesure de l'exécution de l'algorithme selon le schéma général suivant :

- chaque coût $\lambda_{s_i}(s_j)$ est initialisé à $+\infty$ (sauf le coût $\lambda_{s_i}(s_i)$, lequel vaut 0 par construction) ;
- au fur et à mesure du parcours du graphe, on compare les poids $\mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_j})$ associés aux différents chemins $\mathcal{C}_{s_i \longrightarrow s_j}$ découverts au coût $\lambda_{s_i}(s_j)$ associé au sommet s_j du graphe pondéré. Si le poids $\mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_j})$ d'un chemin $\mathcal{C}_{s_i \longrightarrow s_j}$ est inférieur au coût $\lambda_{s_i}(s_j)$ associé au sommet s_j, le coût $\lambda_{s_i}(s_j)$ est mis à jour et on conserve ce nouveau chemin $\mathcal{C}_{s_i \longrightarrow s_j}$.
- le processus est itéré jusqu'à avoir minimisé les valeurs $\lambda_{s_i}(s_j)$ associées à chacun des sommets s_j du graphe pondéré, c'est-à-dire obtenir la valeurs $\delta(s_i, s_j)$ associée à chaque sommet s_j du graphe pondéré.

4.2. Relâchement d'un arc

Supposons que l'on connaisse un chemin $\mathcal{C}_{s_i \longrightarrow s_j}$ de poids $\mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_j})$ allant du sommet d'origine s_i jusqu'au sommet s_j – ce chemin constitue, pour l'instant, le chemin de poids minimal permettant d'aller du sommet s_i au sommet s_j, c'est-à-dire que :

$$\lambda_{s_i}(s_j) = \mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_j}) \geqslant \delta(s_i, s_j)$$

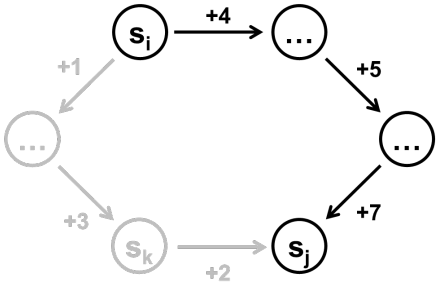


Fig. 2 – mise en évidence d'un chemin $\mathcal{C}_{s_i \longrightarrow s_j}$

Supposons que l'on connaisse également un chemin $\mathcal{C}_{s_i \longrightarrow s_k}$ de poids $\mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_k})$ allant du sommet de d'origine s_i jusqu'au sommet s_k, prédécesseur du sommet s_j.

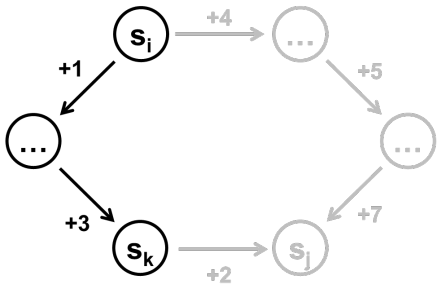


Fig. 3 – mise en évidence d'un chemin $\mathcal{C}_{s_i \longrightarrow s_k}$ où le sommet s_k est un prédécesseur du sommet s_j

On obtient ici un nouveau chemin $\mathcal{C}_{s_i \longrightarrow s_k \longrightarrow s_j}$, obtenu en ajoutant l'arc (s_k, s_j) à la fin du chemin $\mathcal{C}_{s_i \longrightarrow s_k}$. Ce chemin relie s_1 à s_j et ce avec un poids inférieur à celui du chemin $\mathcal{C}_{s_i \longrightarrow s_j}$ (ne passant pas par le sommet s_k). On va donc mettre à jour l'estimateur $\lambda_{s_i}(s_j)$ en posant $\lambda_{s_i}(s_j) = \mathcal{P}(\mathcal{C}_{s_i \longrightarrow s_k \longrightarrow s_j})$.

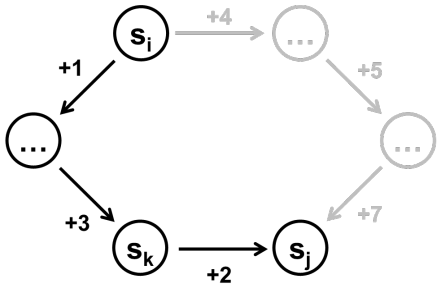


Fig. 4 – mise en évidence d'un chemin $\mathcal{C}_{s_i \longrightarrow s_k \longrightarrow s_j}$ obtenu en ajoutant l'arc (s_k, s_j) à la fin du chemin $\mathcal{C}_{s_i \longrightarrow s_k}$

4.3. Algorithme de Dijkstra

L'**algorithme de Dijkstra** est un algorithme de recherche de plus court chemin dans un graphe pondéré dont tous les arcs sont associés à des poids positifs. L'idée centrale de cet algorithme est de séparer l'ensemble S des sommets du graphe pondéré $G = (S, A, p)$ étudié en deux ensembles :

- l'ensemble T des sommets pour lesquels le plus court chemin depuis le sommet s_i initial a déjà été déterminé, c'est-à-dire l'ensemble des sommets t_k étiquetés par $\lambda_{s_i}(t_k) = \delta(s_i, t_k)$;
- l'ensemble U des sommets pour lesquels les coûts $\lambda_{s_i}(u_k)$ n'ont pas encore été minimisés.

À chaque itération de la boucle principale de l'algorithme, un sommet de l'ensemble U est (judicieusement) choisi de manière à être traité et ajouté à l'ensemble T. Le sommet u à traiter est choisi de manière à ce que $\lambda_{s_i}(u)$ soit minimal, c'est-à-dire :

$$\forall v \in U\ ,\ \lambda_{s_i}(u) \leqslant \lambda_{s_i}(v)$$

Ainsi, un chemin minimal allant de s_i à u passera forcément par des sommets t_k déjà traités. Lorsqu'un sommet est ajouté à l'ensemble T, le coût $\lambda_{s_i}(v)$ de chacun de ses successeurs (i.e. les sommets $v \in U$ tels que $(u, v) \in A$) est mis à jour.

L'implémentation suivante peut être utilisée pour mettre en application l'algorithme de Dijkstra. L'entier i représente l'index du sommet initial s_i dans le graphe G. La liste `LAMBDA` contient les coûts $\lambda_{s_i}(s_j)$ pour tout $s_j \in S$ et est donc mise à jour au cours de l'algorithme.

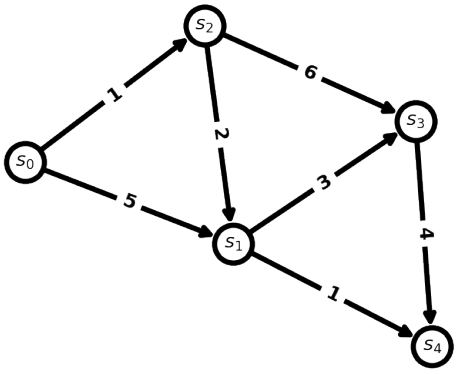
Plus précisément `LAMBDA[j]` contient le coût $\lambda_{s_i}(s_j)$. À la fin de l'exécution, `LAMBDA` contient donc les distances $\delta(s_i, s_j)$ pour tout s_j dans S.

Algorithme : algorithme de Dijkstra

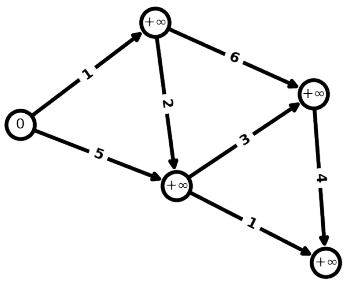
Entrée : `G`, `i`
Sortie : `LAMBDA`

```
n ⟵ ordre de G
T ⟵ ∅
U ⟵ sommets de G
LAMBDA ⟵ liste contenant n fois la valeur +∞
LAMBDA(i) ⟵ 0
Tant que U ≠ ∅ faire :
    j ⟵ indice du sommet v de l'ensemble U tel que λ_{s_i}(v) = min{λ_{s_i}(w)}_{w∈U}
    ajouter le sommet v à l'ensemble T
    retirer le sommet v de l'ensemble U
    Pour s_j dans l'ensemble des successeurs de v faire :
        Si s_j ∈ U alors :
            LAMBDA(j) ⟵ min(LAMBDA(j), LAMBDA(k) + p(v, s_j))
Retourner LAMBDA
```

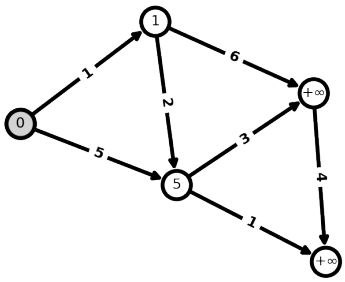
Afin d'illustrer la recherche de plus court chemin par l'algorithme de Dijkstra, on considère le graphe pondéré ci-contre :



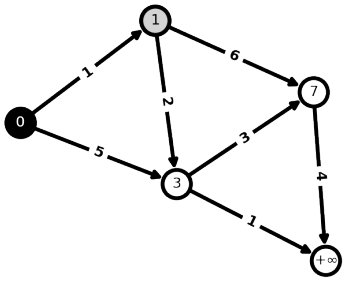
On cherche à déterminer la distance $\delta(s_0, s_j)$ de chaque sommet s_j du graphe au sommet d'origine noté s_0. À chaque itération, on remplace le numéro de chaque sommet par la valeur de $\lambda_{s_0}(s_j)$.
Lorsqu'un sommet est identifié comme étant celui pour lequel le coût $\lambda_{s_0}(s_j)$ est minimal, il est colorié en gris. Une fois que ce sommet est traité (donc il est dans U et son numéro est $\delta(s_0, s_j)$), il est colorié en noir.

Description de l'étape	Représentation du graphe
▶ initialisation : les coûts $\lambda_{s_0}(s_j)$ associées à chaque sommet s_j sont initialisés. Les sommets autre que 0 sont initialement numérotés à $+\infty$.	

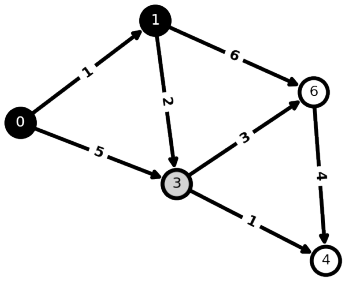
- On traite le sommet avec le coût $\lambda_{s_0}(s_j)$ le plus faible, qui est le sommet d'origine s_0 ici car tous les autres ont un coût de $+\infty$ à ce stade.
  On va donc traiter le sommet s_0 et, pour les sommets adjacents s_1 et s_2, on actualise leurs coûts pour tenir compte du fait qu'il y a des arcs pour les atteindre depuis s_0. On appelle cela **relâcher les arcs** (s_0, s_1) et (s_0, s_2).



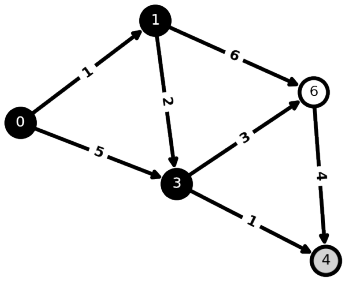
- parmi les 4 sommets non-traités, le sommet s_2 associé au coût $\lambda_{s_0}(s_2) = 1$ est celui de valeur minimale – on traite donc ce sommet. Les arcs entre le sommet s_2 et ses deux successeurs, à savoir les sommets s_1 et s_3, sont relâchés, ce qui conduit ici à une diminution des valeurs des coûts $\lambda_{s_0}(s_1)$ et $\lambda_{s_0}(s_3)$;



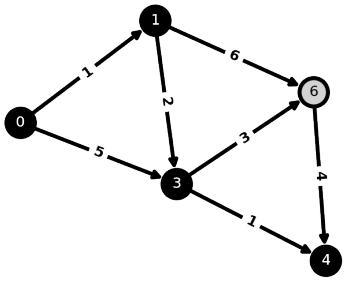
- parmi les 3 sommets non-traités, le sommet s_1 associé au coût $\lambda_{s_0}(s_1) = 3$ est celui de valeur minimale – on traite donc ce sommet. Les arcs entre le sommet s_1 et ses deux successeurs, à savoir les sommets s_3 et s_4, sont relâchés, ce qui conduit ici à une diminution des valeurs des coûts $\lambda_{s_0}(s_3)$ et $\lambda_{s_0}(s_4)$;



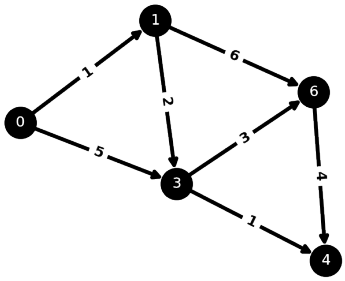
- parmi les 2 sommets non-traités, le sommet s_4 associé au coût $\lambda_{s_0}(s_4) = 4$ est celui de valeur minimale – on traite donc ce sommet. Ce sommet n'a pas de successeur : il n'y a donc aucun arc à relâcher ;



- le dernier sommet non-traité est le sommet s_3 associé au coût $\lambda_{s_0}(s_3) = 6$. Ce sommet n'a aucun successeur qui n'ait pas déjà été traité : il n'y a donc aucun arc à relâcher ;



- tous les sommets de la liste `U` ont été traités. La boucle principale de l'algorithme de Dijkstra se termine. Toutes les distances $\delta(s_0, s_j)$ ont correctement été déterminées.



Si on souhaite connaître non seulement la distance entre un sommet s_j et le sommet initial s_i, mais également un plus court chemin $\mathcal{C}_{s_i \longrightarrow s_j}$ depuis le sommet s_i vers le sommet s_j de poids $\delta(s_i, s_j)$, l'algorithme proposé précédemment peut être modifié de manière à construire la liste `Q` des prédécesseurs de chaque sommet s_k dans le plus court chemin depuis le sommet s_i vers

le sommet s_j :

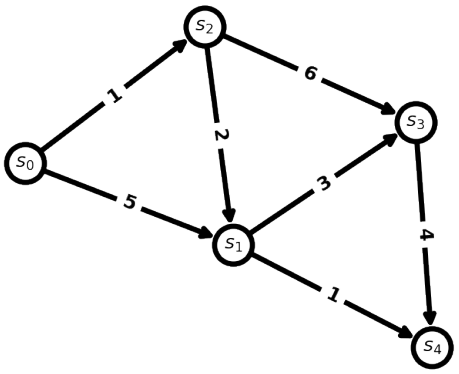
Algorithme : algorithme de Dijkstra (avec prédécesseurs)

Entrée : `G`, `i`
Sortie : `LAMBDA`, `Q`

```
1  n ⟵ ordre de G
2  T ⟵ ∅
3  U ⟵ sommets de G
4  LAMBDA ⟵ liste contenant n fois la valeur +∞
5  LAMBDA(i) ⟵ 0
6  Q ⟵ liste de n éléments non-affectés
7  Q(i) ⟵ i
8  Tant que U ≠ ∅ faire :
9      k ⟵ indice du sommet v de l'ensemble U tel que λ_{s_i}(v) = min{λ_{s_i}(w)}_{w∈U}
10     ajouter le sommet v à l'ensemble T
11     retirer le sommet v de l'ensemble U
12     Pour s_j dans l'ensemble des successeurs de v faire :
13         Si s_j ∈ U alors :
14             LAMBDA(j) ⟵ min(LAMBDA(j), LAMBDA(k) + p(v, s_j))
15             Q(j) ⟵ k
16 Retourner LAMBDA, Q
```

En appliquant ce nouvel algorithme au graphe pondéré ci-contre, on obtient la liste des prédécesseurs suivante :

$$\texttt{Q} = [\varnothing, 2, 0, 1, 1]$$



Le plus court chemin du sommet s_0 au sommet s_4 est ainsi :

$$\mathcal{C}_{s_0 \longrightarrow s_4} = (s_0 \longrightarrow s_2 \longrightarrow s_1 \longrightarrow s_4)$$

Nota : on peut démontrer que la complexité de l'algorithme de Dijkstra est en $\mathcal{O}((Card(S) + Card(A)) \times \log(Card(S))$.

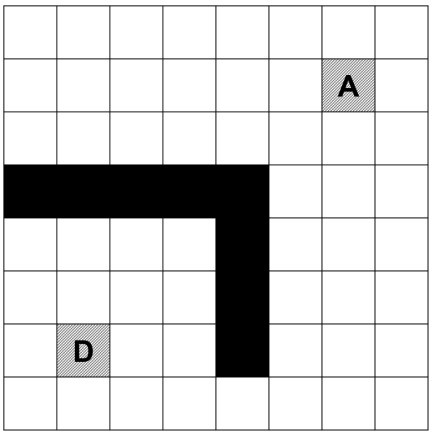
5. Recherche de solutions approchées : utilisation d'une heuristique

L'algorithme de Dijkstra présente une complexité qui peut s'avérer rapidement problématique pour des graphes comportant un nombre important de sommets. Cette complexité importante peut limiter son application à des situations pratiques, telles que l'utilisation d'un GPS. Une

autre approche consiste à rechercher un plus court chemin de manière approchée en utilisant une **heuristique**.

Une démarche heuristique consiste, face à un problème présentant une résolution exacte trop « gourmande » en termes de complexité, à choisir quelles solutions sont explorées de manière préférentielle. Ce choix dans les solutions possibles nécessite donc de courir le risque de passer à côté d'une solution exacte au problème étudié – on se contente alors d'une solution (éventuellement) approchée. Le choix des solutions à explorer se fait en utilisant une heuristique, c'est-à-dire un ensemble de règles qui vont guider la recherche d'une solution (éventuellement) approchée.

Afin d'illustrer la notion d'heuristique, on considère le graphe suivant :



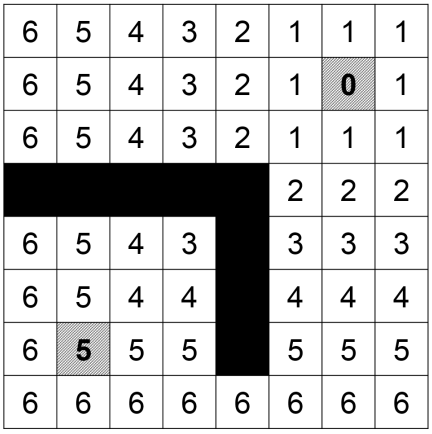
Chaque case représente un sommet d'un graphe (non-orienté). Les sommets représentés par deux cases adjacentes (horizontalement, verticalement ou en diagonale) sont liés par une arête. Les cases noires représentent une absence de sommet (elles jouent donc un rôle de « murs »). Le poids associé à chaque arête du graphe est pris égal à +1. On souhaite trouver un plus court chemin depuis le sommet d'origine D vers le sommet A (en gras et hachurés).

Une résolution de ce problème à l'aide de l'algorithme de Dijkstra conduit à l'obtention du résultat suivant :

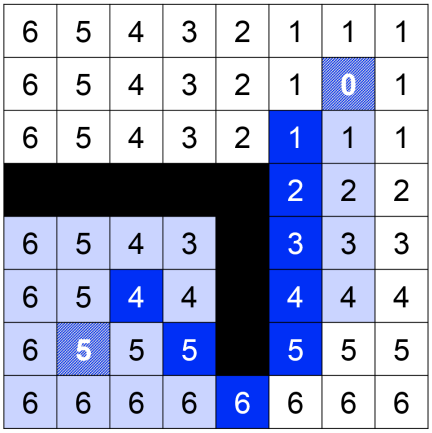
12	11	10	10	10	10	10	10
12	11	10	9	9	9	**9**	9
12	11	10	9	8	**8**	8	8
					7	7	7
2	2	2	2		**6**	6	6
1	1	1	2		**5**	5	6
1	**0**	1	2		**4**	5	6
1	1	**1**	**2**	**3**	4	5	6

Les valeurs associées à chaque sommet sont les distances $\delta(D, s_j)$ associées aux sommets s_j du graphe. Le chemin de coût minimal est matérialisé en couleur (vive) ; l'ensemble des sommets visités lors de l'exécution de l'algorithme de Dijkstra est matérialisé en couleur (pâle). On constate qu'une grande partie du graphe a été explorée avant d'obtenir un chemin de court minimal.

En partant de l'idée que la ligne droite est – probablement – la meilleure manière de procéder pour aller du sommet D au sommet A, on peut considérer la distance à vol d'oiseau au sommet A :



En cherchant à minimiser, lors de l'exploration du graphe, cette distance à vol d'oiseau, on aboutit au résultat suivant :



Les valeurs associées à chaque sommet sont les distances à vol d'oiseau $H(A, s_j)$ associées aux sommets s_j du graphe. Le chemin de coût minimal est matérialisé en couleur (vive) ; l'ensemble des sommets visités lors de l'exécution de l'algorithme est matérialisé en couleur (pâle). On constate que la proportion de sommets explorés est, ici, nettement moins importante que dans le cas de l'algorithme de Dijkstra. Le chemin retenu est, ici, de poids minimal : cependant, dans le cas général, l'approche heuristique ne conduit pas toujours à une solution optimale.