Plus court chemin dans un graphe

Haetham AL ASWAD

Références. Ces notes suivent la présentation du livre *Introuduction to Algorithms* [CLRS22] [1] pour les sections 2 à 4 et l'article original introduisant l'algorithme A^*, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths* [HNR68] [2] pour la section 5.

1 Introduction

Les problèmes liés aux graphes occupent une place centrale en informatique. Des centaines de problèmes algorithmiques sont formulés en utilisant des graphes, notamment des problèmes d'optimisation et d'aide à la décision. Dans ce cours nous allons étudier le problème de plus court chemin dans un graphe orienté et avec arêtes pondérées. Étant donné un sommet s dans un tel graphe G, on s'intéresse aux deux problèmes algorithmiques suivants :

— Trouver un plus court chemin de s à tout sommet du graphe. S'il n'existe pas un plus court chemin de s à un sommet v, alors détecter cela.

— Étant donné un sommet t, trouver un plus court chemin de s à t s'il en existe un, et détecter qu'il n'en existe pas sinon.

Deux algorithmes pour le problème *plus-court-chemin* dans un graphe.

L'algorithme de Bellman-ford (Section 4). Prend en entrée un graphe G orienté et pondéré et un sommet source s. Pour tout sommet v, il trouve un *plus-court-chemin* de s à v si un tel chemin existe, et détecte la non-existence d'un tel chemin sinon. Lorsque le graphe est donné sous forme de listes d'adjacence, l'algorithme Bellman-ford est de complexité asymptotique $O(|V|^2 + |V||E|)$ où $|V|$ est le nombre de sommets de G et $|E|$ son nombre d'arêtes.

L'algorithme A^* (Section 5). Prend en entrée un graphe G orienté et pondéré avec des poids d'arêtes *strictement positifs*, un sommet source s, un sommet cible t, et une certaine fonction dite d'estimation. Il trouve un *plus-court-chemin* de s à t. On prouvera sous certaines hypothèses que l'algorithme A^* est *optimal*.

L'algorithme de Dijkstra (Section 5.4) est un cas particulier de l'algorithme A^*. Il se généralise à un algorithme calculant un *plus-court-chemin* d'un sommet s à tous les sommets du graphe. Cette famille d'algorithmes est de complexité asymptotique $O(|V|^2)$ où V est l'ensemble des sommets du graphe.

2 Graphes et *plus-court-chemin*

Un graphe G est un couple (V, E) où V est l'ensemble des sommets (Vertices en anglais), et E est l'ensemble des arêtes (Edges en anglais). Dans ce cours tout graphe est orienté, on a $E \subset V^2$ et $(u, v) \in E$ signifie qu'il y dans G a une arête du sommet u vers le sommet v. Dans ce cours tout graphe est pondéré, c'est à dire muni d'une fonction de poids $\omega : E \longrightarrow \mathbb{R}$. Pour tout $(u, v) \in E$, le poids de l'arête (u, v) est $\omega((u, v))$ qu'on notera $\omega(u, v)$ par abus.

Il existe deux manières standards de représenter un graphe $G = (V, E)$, par listes d'adjacence, ou par matrice d'adjacence. Nous adoptons la première. La représentation par listes d'adjacence consiste en la donnée d'une liste Adj de $|V|$ listes, une pour chaque sommet dans V. Pour tout $u \in V$, la liste d'adjacence Adj$[u]$ contient tous les sommets $v \in V$ tel que $(u, v) \in E$.

1. En libre accès : `https://dl.ebooksworld.ir/books/Introduction.to.Algorithms.4th.Leiserson.Stein.Rivest.Cormen.MIT.Press.9780262046305.EBooksWorld.ir.pdf`.

2. Disponible ici : `https://ieeexplore.ieee.org/abstract/document/4082128`.

Notation par attribut. Les algorithmes présentés dans ce cours opèrent sur des valeurs qu'ils attribuent aux sommets. Nous adoptons une notation par attribut : $v.d$ désigne la valeur de l'attribut d au sommet v.

Plus court chemin. L'entrée au problème *plus-court-chemin* est un graphe $G = (V, E)$ orienté et pondéré avec une fonction de poids ω. Un *chemin* d'un sommet $v_0 \in V$ à un sommet $v_k \in V$ est une suite $p = [v_0,\ v_1,\ \ldots,\ v_k]$ tel que $(v_i, v_{i+1}) \in E$ pour tout $0 \leq i \leq k-1$. Le poids d'un chemin est la somme des poids de ces arêtes :

$$\omega(p) := \sum_{i=0}^{k-1} \omega(v_i, v_{i+1})$$

On définit le poids d'un *plus-court-chemin* $\delta(u, v)$ d'un sommet u vers un sommet v par

$$\delta(u,v) = \begin{cases} \min\{\omega(p) \mid u \overset{p}{\rightsquigarrow} v\} & \text{S'il existe un chemin de u à v,} \\ \infty & \text{Sinon.} \end{cases} \tag{1}$$

Où $u \overset{p}{\rightsquigarrow} v$ désigne un chemin p de u à v.

Un *plus-court-chemin* d'un sommet u à un sommet v est défini comme un chemin p de u à v tel que $\omega(p) = \delta(u, v)$. Le lemme suivant énonce l'optimalité des sous chemins d'un *plus-court-chemin* :

Lemme 2.1 (Un sous chemins d'un *plus-court-chemin* est un *plus-court-chemin*)**.** *Soit $p = [v_0,\ \ldots\ v_k]$ un plus-court-chemin de v_0 à v_k. Alors pour tout $0 \leq i \leq j \leq k$, $p_{ij} := [v_i,\ v_{i+1},\ \ldots\ v_j]$ est un plus court chemin de v_i à v_j.*

Démonstration. Décomposons p en $v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$ de telle sorte que $\omega(p) = \omega(p_{0i}) + \omega(p_{ij}) + \omega(p_{jk})$. Supposons par l'absurde qu'il existe un chemin p'_{ij} de i à j avec $\omega(p'_{ij})[\omega(p_{ij})$. Alors $v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p'_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$ est un chemin de v_0 à v_k de poids $\omega(p_{0i}) + \omega(p'_{ij}) + \omega(p_{jk})$ strictement inférieur au poids de p. Absurde, car p est un *plus-court-chemin* de v_0 à v_k. □

Sommet source. On s'intéresse à des *plus-court-chemin* d'un sommet source s à d'autres sommets du graphe. La lettre s désignera toujours le sommet source et il fait partie de l'entrée à notre problème.

Existence d'un *plus-court-chemin*. Il y a deux scénarios possibles pour qu'un *plus-court-chemin* d'un sommet s à un sommet v n'existe pas. Le premier est lorsque le graphe G ne contient aucun chemin de s à v, et à fortiori aucun *plus-court-chemin*. Dans ce cas on a $\delta(s, v) = \infty$. Le deuxième scénario se produit lorsqu'il existe un chemin de s à v et qu'il existe un cycle de poids strictement négatif accessible depuis s. Dans ce cas on a $\delta(s, v) = -\infty$.

3 Principe de relaxation

Cette section définit et étudie un procédé général utilisé par une grande famille d'algorithmes traitant des graphes. On définit deux attributs sur l'ensemble des sommets V :

— Attribut d dit d'estimation où $v.d$ est une borne supérieur de $\delta(s, v)$ pour tout $v \in V$.

— Attribut π dit de prédécesseur où $v.\pi$ désigne un autre sommet de V ou le pointeur null Nil.

Remarque 3.1 (Saut dans le futur)**.** *À la terminaison d'un algorithme, les attributs d et π donnent respectivement le poids d'un plus-court-chemin et un plus-court-chemin (lorsqu'il existe) de s à des sommets du graphe. Ainsi pour certains sommets v, l'attribut $v.d$ est amélioré (décroît) durant l'algorithme jusqu'à atteindre $v.d = \delta(s, v)$, et l'attribut $v.\pi$ s'améliore durant l'algorithme jusqu'à pointer un plus-court-chemin de s à v.*

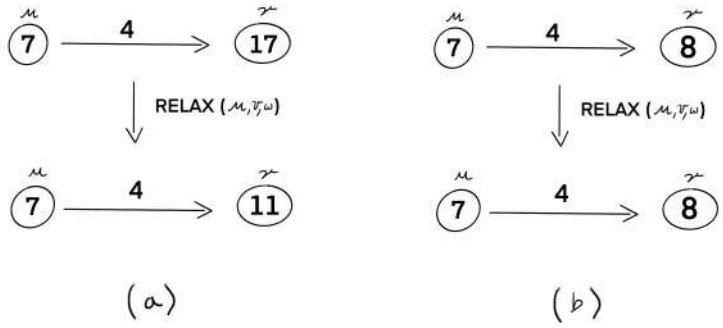

FIGURE 1 – Procédure de relaxation d'une arête (u, v). À gauche (a), la procédure de relaxation met à jour l'attribut $v.d$ à $7+4 = 11$ et l'attribut $v.\pi$ à u. À droite (b), la procédure de relaxation ne permet pas d'améliorer $v.d = 8$. les attributs $v.d$ et $v.\pi$ restent inchangés.

Initialisation et relaxations d'arêtes. Les algorithmes dans ce cours commencent par initialiser les attributs d et π. Après l'initialisation on a $s.d = 0$, et $v.d = \infty$ pour tout $v \in V \backslash \{s\}$, et $v.\pi = \text{Nil}$ pour tout $v \in V$. Algorithme INITIALIZE est un pseudo-code de cette procédure.

Le procédé de relaxer une arête (u, v) teste si le passage par le sommet u améliore le plus court chemin de s à v trouvé jusqu'à présent. Ce procédé appliqué à l'arête (u, v) peut décroître $v.d$ et mettre à jour $v.\pi$, ou ne rien faire. Algorithme RELAX est un pseudo-code de cette procédure et Figure 3 en est une illustration sur un exemple.

Tout algorithme dans ce cours fait appel à INITIALIZE puis relaxe des arêtes en faisant appel à RELAX de manière répétée. Ils différent du nombre de fois chaque arête est relaxée et de l'ordre dans lequel les arêtes sont relaxées. Les deux lemmes suivants énoncent des propriétés de l'attribut d.

INITIALIZE $(G = (V, E),\ s)$

```
1: for each v in V
2:     v.d = ∞
3:     v.π = Nil
4: s.d = 0
```

RELAX$(u,\ v,\ \omega)$

```
1: if v.d > u.d + ω(u, v)
2:     v.d = u.d + ω(u, v)
3:     v.π = u
```

Lemme 3.2 (Minoration)**.** *Après un appel à la procédure* INITIALIZE *sur G, et un nombre fini quelconque d'appels à la procédure* RELAX *sur les arêtes de G, on a $v.d \geq \delta(s, v)$ pour tout sommet $v \in V$. De plus, si $v.d$ atteint $\delta(s, v)$, il ne changera plus.*

Démonstration. À faire en exercice. □

Corollaire 3.3. *Si un sommet $v \in V$ n'est pas accessible depuis s, alors $v.d = \delta(s, v) = \infty$.*

Démonstration. Comme v n'est pas accessible depuis s, alors $\delta(s, v) = \infty$ par définition. Par le lemme de la Minoration 3.2 on a $v.d \geq \infty$. □

4 Algorithme de Bellman-ford

L'algorithme de Bellman-ford résout le problème des *plus-court-chemin* à partir d'une source s à tous les sommets du graphe, même dans le cas général où les poids des arêtes peuvent être **négatifs**.

Étant donné un graphe pondéré et orienté $G = (V, E)$ et un sommet source s, l'algorithme de Bellman-ford renvoie une valeur booléenne indiquant s'il existe un cycle de poids strictement négatif atteignable à partir de la source s. S'il y a un tel cycle, l'algorithme indique qu'aucune solution n'existe. S'il n'y a pas de cycle de ce type, l'algorithme produit des *plus-court-chemin* et leurs poids. La procédure de relaxation de Bellman-ford ajuste progressivement une estimation $v.d$ du poids d'un chemin le plus court depuis la source s vers chaque sommet $v \in V$ jusqu'à atteindre le poids d'un *plus-court-chemin* réel $\delta(s, v)$. L'algorithme renvoie VRAI si et seulement si le graphe ne contient aucun cycle de poids strictement négatif atteignable depuis la source. Algorithme BELLMAN-FORD est un pseudo-code de l'algorithme de Bellman-ford et Figure 2 illustre l'application de cet algorithme sur un exemple.

BELLMAN-FORD $(G = (V, E),\ s,\ \omega)$

```
1: INITIALIZE(G, s)
2: for i from 1 to |V| − 1
3:     for each (u, v) in E
4:         RELAX(u, v, ω)
5: for each (u, v) in E
6:     if v.d > u.d + ω(u, v)
7:         return FALSE
8: return TRUE
```

Complexité : Lorsque le graphe est donné sous forme de listes d'adjacence, la complexité de BELLMAN-FORD est $O\left(|V|^2 + |V||E|\right)$. Prouvez le.

Nous allons prouvez que l'algorithme BELLMAN-FORD est correct. Nous commençons par prouver que si le graphe ne contient pas un cycle de poids strictement négatif, alors l'algorithme trouve bien le poids d'un *plus-court-chemin* de la source s à tout sommet accessible depuis s.

Lemme 4.1. *Supposons que G ne contient pas de cycle de poids strictement négatif accessible depuis s. Alors, après les $|V| - 1$ itérations de la boucle* ***for*** *des lines 2-4 de* BELLMAN-FORD, *$v.d = \delta(s, v)$ pour tout sommet v accessible depuis s.*

Démonstration. Soit $v \in V$ un sommet accessible depuis s. Comme G ne contient pas de cycle de poids strictement négatif accessible depuis s, alors il existe un *plus-court-chemin* de s à v qui est simple (i.e., ne contient pas de cycle). Prouvez le [3]. Soit $p = [v_0, v_1, \ldots, v_k]$ un *plus-court-chemin* de $s = v_0$ à $v = v_k$ qui est simple, donc $0 \leq k \leq |V| - 1$. Prouvons par récurrence sur $0 \leq i \leq k$ qu'après la i-ième itération de la boucle de la ligne 2 de BELLMAN-FORD, on a $v_i.d = \delta(s, v_i)$.

Initialisation. Avant d'entrer dans la boucle **for** de la ligne 2, $s.d = 0 = \delta(s, s)$ par la procédure INITIALIZE.

Hérédité. Supposons $v_i.d = \delta(s, v_i)$ pour un $0 \leq i < k$. À la (i+1)-ième itération de la boucle **for**, toutes les arêtes du graphe sont relaxées, dont l'arête (v_i, v_{i+1}). La procédure RELAX appliqué à (v_i, v_{i+1}, ω) considère la quantité $v_i.d + \omega(v_i, v_{i+1})$ qui est égale à $\delta(s, v_i) + \omega(v_i, v_{i+1})$ par H.R. Par le lemme 2.1, le chemin $p_{i+1} := [v_0, v_1, \ldots, v_{i+1}]$ est un *plus-court-chemin* de $v_0 = s$ à v_{i+1}, donc $\omega(p_{i+1}) = \delta(s, v_{i+1})$. Or

$$\begin{aligned}
\delta(s, v_{i+1}) &= \omega(p_{i+1}) \\
&= \sum_{j=0}^{i-1} \omega(v_j, v_{j+1}) + \omega(v_i, v_{i+1}) \\
&= \delta(s, v_i) + \omega(v_i, v_{i+1}) \text{ car à nouveau par le lemme 2.1, } [v_0, \ldots, v_i] \text{est un } \textit{plus-court-chemin.}
\end{aligned}$$

Donc au test de la condition **if** de RELAX : **if** $v_{i+1}.d > v_i.d + \omega(v_i, v_{i+1})$, ou bien $v_{i+1}.d$ valait déjà $\delta(s, v_{i+1})$ et reste inchangé, ou bien $v_i.d + \omega(v_i, v_{i+1}) = \delta(s, v_{i+1})$ est affecté à $v_{i+1}.d$, car $v_{i+1}.d \geq \delta(s, v_{i+1})$ par le lemme de Minoration 3.2. Cela conclut la récurrence. On vient de prouver qu'après k

3. Distinguez trois cas : un cycle de poids strictement négatif, de poids strictement positif, ou de poids nul. Dans le troisième cas, pourquoi on peut se ramener à un chemin sans cycle ?

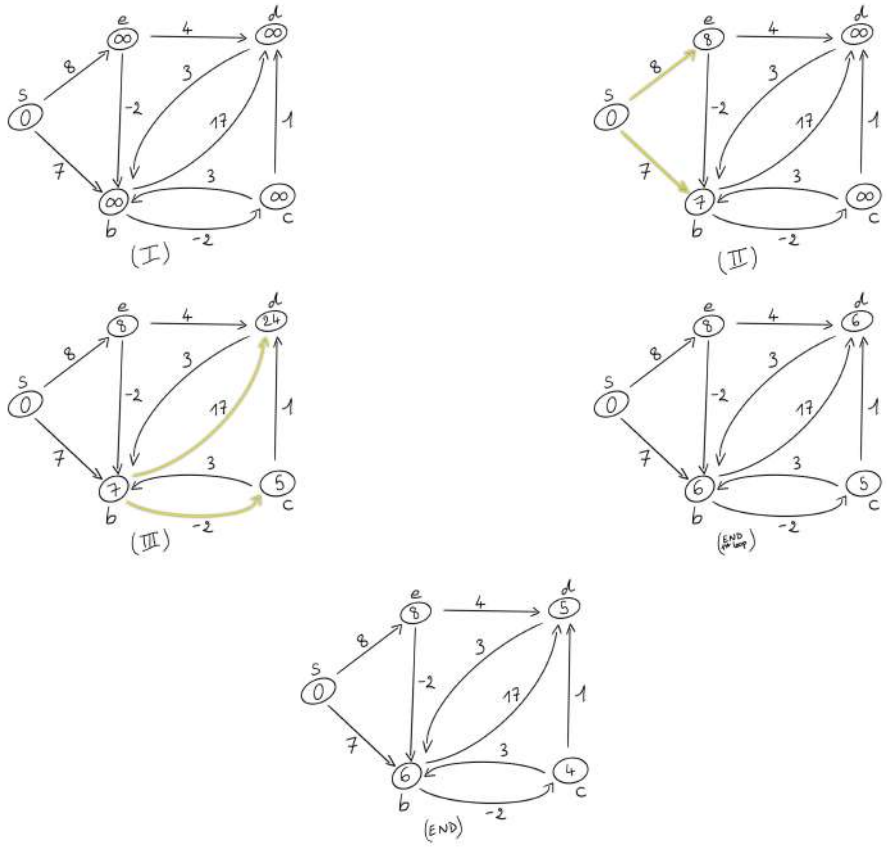

FIGURE 2 – Algorithme de Bellman-Ford. (I) Les valeurs de l'attribut d après initialisation. (II) On rentre dans le premier tour de boucle de la ligne 2. Les arêtes (s, b) et (s, e) sont relaxées : $b.d = 7$, $b.\pi = s$ et $e.d = 8$, $e.\pi = s$. (III) Toujours dans le premier tour de boucle de la ligne 2. (END FIRST LOOP) est l'état de l'attribut d une fois chaque arête a été relaxée exactement une fois, ce qui correspond à un tour de boucle de la ligne 2. Les arêtes ont été relaxé dans cet exemple dans le sens trigonométrique : Les arête sortantes de s, puis celles sortantes de b ... etc. (END) est l'état de l'attribut d une fois les 4 tours de boucle de la ligne 2 on été effectué.

itérations de la boucle **for** de la ligne 2 de BELLMAN-FORD, on a $v_k.d = \delta(s, v_k)$. Par le Lemme de Minoration 3.2, $v_k.d$ qui ne peut que décroître ne changera plus. Comme $k \leq |V| - 1$, après les $|V| - 1$ itérations de la boucle **for** de la ligne 2 de BELLMAN-FORD, on a bien $v.d = \delta(s, v)$. □

Très souvent nous ne sommes pas seulement intéressé.e.s par le poids d'un *plus-court-chemin* mais par un *plus-court-chemin*. Pensez à votre GPS qui vous calcule un chemin et ne se contente pas de vous donner la distance à votre destination. La procédure PATH-BY-π permet après application de BELLMAN-FORD de trouver un *plus-court-chemin* de s à un sommet v lorsqu'il existe. Nous prouvons cela au Lemme 4.3. Admettons le Lemme 4.3 dans un premier temps et prouvons que l'algorithme de Bellman-ford est correct.

Théorème 4.2 (L'algorithme BELLMAN-FORD est correct)**.** *Si le graphe G ne contient pas de cycle de poids strictement négatif accessible depuis s, alors l'algorithme renvoie **TRUE**, $v.d = \delta(s, v)$ pour tout sommet $v \in V$. De plus, pour tout sommet v accessible depuis s, la procédure* PATH-BY-π *appliquée à v permet de trouver en $O(|V|)$ un plus-court-chemin de s à v. Si G contient un cycle de poids strictement négatif accessible depuis s, alors l'algorithme renvoie **FALSE**.*

Démonstration. Supposons d'abord que G ne contient pas de cycle de poids strictement négatif accessible depuis s, et soit $v \in V$ un sommet. Si v n'est pas accessible depuis s, alors par le Corollaire 3.3, on a toujours $v.d = \delta(s, v) = \infty$. Si v est accessible depuis s, on a également à la terminaison de l'algorithme $v.d = \delta(s, v)$ par le Lemme 4.1, et la procédure PATH-BY-π permet de trouver un *plus-court-chemin* de s à un tel sommet en $O(|V|)$ opérations par le Lemme 4.3. Prouvons maintenant que BELLMAN-FORD renvoie **TRUE**. Par ce que nous venons de démontrer, on a à la términaison de la boucle **for** des lignes 2-4, pour toute arête $(u, v) \in E$:

$$\begin{aligned} v.d &= \delta(s, v) \\ &\leq \delta(s, u) + \omega(u, v) \quad \text{Prouvez cette inégalité triangulaire.} \\ &= u.d + \omega(u, v) \end{aligned}$$

Donc aucun des tests de la ligne 7 n'est vérifiée et l'algorithme renvoie **TRUE**.

Supposons maintenant que le graphe G contient un cycle de poids strictement négatif accessible depuis s. Notons ce cycle $c = [v_0, \ldots, v_k]$ avec $v_0 = v_k$. On a alors

$$\sum_{i=1}^{k} \omega(v_{i-1}, v_i) < 0 \tag{2}$$

Supposons par l'absurde que BELLMAN-FORD renvoie **TRUE**. Donc aucun des tests **if** n'est vérifiée, on a alors $v_i.d \leq v_{i-1}.d + \omega(v_{i-1}, v_i)$ pour tout $1 \leq i \leq k$. En sommant ces inégalités on obtient :

$$\begin{aligned} \sum_{i=1}^{k} v_i.d &\leq \sum_{i=1}^{k} (v_{i-1}.d + \omega(v_{i-1}, v_i)) \\ &= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} \omega(v_{i-1}, v_i) \end{aligned}$$

Comme $v_0 = v_k$, et que chaque sommet de c apparaît exactement une fois dans chacune des sommes $\sum_{i=1}^{k} v_i.d$ et $\sum_{i=1}^{k} v_{i-1}.d$, ces deux sont égales. Par ailleurs, comme les sommets de c sont accessibles depuis s, alors par le Lemme 4.1, $v_i.d$ est finie pour tout $1 \leq i \leq k$. En conclusion on a :

$$\sum_{i=1}^{k} \omega(v_{i-1}, v_i) \geq 0$$

contredisant l'Équation 2.

L'algorithme BELLMAN-FORD renvoie **TRUE** si G ne contient pas de cycle de poids strictement négatif accessible depuis s, et **FALSE** sinon. □

PATH-BY-$\pi(v)$

```
1: if v = s
2:     return [s]
3: if v.π = Nil
4:     return FALSE
5: p = [v]  # path containing the vertice v
6: u = v.π
7: while u ≠ Nil
8:     add u to the beginning of p
9:     u = u.π
10: return p
```

Lemme 4.3 (*plus-court-chemin* par prédécesseur)**.** *Supposons que G ne contient pas de cycle de poids strictement négatif accessible depuis s. Alors après appel à* BELLMAN-FORD $(G = (V, E),\ s,\ \omega)$*, un appel à* PATH-BY-$\pi(v)$ *pour un $v \in V$ accessible depuis s renvoie un plus-court-chemin de s à v en $O(|V|)$ opérations.*

Démonstration. Nous allons prouver les deux faits suivants. Après un appel à BELLMAN-FORD sur $(G = (V, E),\ s,\ \omega)$ on a :

Fait 1 : Si $v \in V \backslash \{s\}$ est accessible depuis s alors $v.\pi \neq$ Nil.

Fait 2 : La boucle **while** des lignes 7-9 de PATH-BY-π ne crée jamais de cycle dans le chemin p.

Avant de démontrer ces deux faits, admettons les et voyons comment on peut conclure. Si $v = s$, l'algorithme renvoie $[s]$ qui est bien un *plus-court-chemin* de s à s. Sinon, v est accessible depuis s et est différent de s. Le *Fait 1* nous assure que $v.\pi \neq$ Nil, la procédure PATH-BY-π appliqué à v rentre bien dans la boucle **while**. Par le *Fait 2*, la boucle **while** ne crée pas de cycle dans le chemin p, donc les sommets ajoutés à p sont deux à deux distincts. Comme le graphe G contient $|V|$ sommets, la boucle **while** effectue au plus $|V|$ itérations et l'algorithme renvoie un chemin $p = [v_0, \ldots, v_k]$ avec $v_k = v$. Le sommet v_0 c'est la source s, pourquoi ? D'une part, $v_0 = v_1.\pi$, donc pendant le déroulement de la procédure BELLMAN-FORD, l'arête (v_0, v_1) a été relaxée et la condition $v_1.d > v_0.d + \omega(v_0, v_1)$ était vérifiée juste avant la relaxation. Donc $v_0.d$ avait une valeur finie, et on a $v_0.d < \infty$ à la fin de BELLMAN-FORD, car il ne peut que décroître. D'où v_0 est accessible depuis s par la contraposée du Corollaire 3.3. D'autre part, par la condition d'arrêt de la boucle **while**, on a $v_0.\pi =$ Nil. D'où $v_0 = s$ par le *Fait 1*. Il rest à prouver que p est un *plus-court-chemin*. Comme $v_{i-1} = v_i.\pi$ pour tout $1 \leq i \leq k$, alors puisque les attributs d et π sont mis à jour ensemble par RELAX, on sait que $v_i.d \geq v_{i-1}.d + \omega(v_{i-1}, v_i)$ pour tout $1 \leq i \leq k$. En effet, la valeur $v_i.d$ n'a fait que décroître après la dernière relaxation de (v_{i-1}, v_i) qui a affecté $v_{i-1}.d + \omega(v_{i-1}, v_i)$ à $v_i.d$. D'où en sommant on a :

$$\begin{aligned}
\omega(p) &= \sum_{i=1}^{k} \omega(v_{i-1}, v_i) \\
&\leq \sum_{i=1}^{k} v_i.d - v_{i-1}.d \\
&= v_k.d - v_0.d \\
&= v.d - s.d \\
&= \delta(s, v) \text{ Par le Lemme 4.1}
\end{aligned}$$

Donc $\omega(p) = \delta(s, v)$ et p est bien un *plus-court-chemin* de s à v.

Prouvons le *Fait 1*. Rappelons que G est supposé ne pas contenir un cycle de poids strictement négatif accessible depuis s, donc le Lemme 4.1 s'applique. Si $v \in V \backslash \{s\}$ est accessible depuis s, alors $\delta(s, v) < \infty$ et à la fin de l'algorithme BELLMAN-FORD, $v.d = \delta(s, v) < \infty$. C'est bien que $v.d$ qui vallait ∞ après INITIALIZE(G, s) a été modifié par la procédure RELAX, et donc $v.\pi$ aussi. D'où $v.\pi \neq$ Nil.

Prouvons le *Fait 2*. Supposons par l'absurde que la boucle **while** crée un cycle $c = [v_0, \ldots v_k]$ dans le chemin p. On a donc $v_0 = v_k$ et $v_{i-1} = v_i.\pi$ pour tout $1 \leq i \leq k$. Ce cycle est accessible depuis s.

En effet chaque valeur $v_i.d$ pour $1 \leq i \leq k$ est fini car $v_i.\pi \neq \text{Nil}$, et la contraposée du Corollaire 3.3 s'applique. On va montrer que c est de poids strictement négatif. Quitte à renuméroter, supposons que le dernier sommet du cycle dont l'attribut d a décroît (durant BELLMAN-FORD) est v_k. Comme $v_k.\pi = v_{k-1}$, alors cette décroissance est dû à la relaxation de l'arête (v_{k-1}, v_k). Donc juste avant cette relaxation on a :

$$\begin{cases} v_k.d > v_{k-1}.d + \omega(v_{k-1}, v_k) \\ v_i.d \geq v_{i-1}.d + \omega(v_{i-1}, v_i) \text{ pour tout } 1 \leq i \leq k-1 \end{cases}$$

En sommant on obtient

$$\begin{aligned} \sum_{i=1}^{k} v_i.d &> \sum_{i=1}^{k} v_{i-1}.d + \omega(v_{i-1}, v_i) \\ &= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} \omega(v_{i-1}, v_i) \end{aligned}$$

Mais $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$ car $v_0 = v_k$. On obtient alors que le poids du cycle c est strictement négatif : $\sum_{i=1}^{k} \omega(v_{i-1}, v_i) < 0$. Contradiction avec l'hypothèse que G ne contient pas de cycle de poids strictement négatif accessible depuis s. □

Nous verrons en exercices comment certains problèmes algorithmiques se modélisent dans un problème de *plus-court-chemin* sur un graphe, qu'on peut alors résoudre avec l'algorithme de Bellman-Ford.

L'algorithme de Bellman-Ford nécessite de relaxer chaque arête du graphe $|V|$ fois pour garantir de trouver des *plus-court-chemin*. Si nous nous spécialisons dans les graphes dont les arêtes ont un poids strictement positif, les distances d'un réseau routier par exemple, alors on peut faire mieux. C'est l'objet de la section suivante.

5 Algorithme A^*

Vous êtes à Nancy et vous voulez vous rendre à Lyon en voiture. Il est inutile de considérer les routes qui passent par Metz (qui se situe au nord de Nancy). Votre navigateur GPS ne considère pas de telles routes, comment fait-il ? Il s'agit là d'exploiter des informations sur le graphe qui viennent du monde physique. Dans notre exemple on sait que la distance routière entre deux villes et plus grande que la distance à vol d'oiseau entre ces deux villes. Votre navigateur commence par calculer la distance entre Nancy et les villes voisines. La distance de Nancy à Lyon en passant par Metz est alors plus grande que la distance entre Nancy et Metz plus la distance à vol d'oiseau entre Metz et Lyon (que votre Navigateur a en mémoire par exemple). Il calcule de même une estimation de distance d'une route entre Nancy et Lyon qui passe par Dijon. Cette dernière estimation est plus petite, il va privilégier d'explorer cette option, et il recommence depuis Dijon. Nous venons de décrire le fonctionnement de l'algorithme A^*. Avant de présenter cet algorithme, introduisons quelques notions.

Les η-graphes. Soit $\eta > 0$ un réel strictement positif. Un graphe orienté muni d'une fonction de poids est dit un η-graphe si toute arête est de poids strictement supérieur à η. L'algorithme A^* opère sur les η-graphes pour un $\eta > 0$ fixé à l'avance. En particulier, on n'autorise pas des arêtes avec des poids négatifs.

Un sommet source s et un sommet cible t. On s'intéresse au problème de calculer un *plus-court-chemin* d'un sommet source s à un sommet [4] t. Tout au long de la présentation, s et t désignerons les sommets source et cible et font partie de l'entrée au problème. Pour faciliter la présentation on suppose que t est accessible depuis s.

4. L'algorithme A^* se généralise aisément à un algorithme qui calcule un *plus-court-chemin* entre un sommet source s et un sommet cible t le plus proche de s parmi plusieurs sommets cibles.

La fonction d'évaluation $\hat{f}$. On définit sur les sommets la fonction $f : V \longrightarrow \mathbb{R}_+$ où pour tout sommet v, $f(v) = \delta(s, v) + \delta(v, t)$. C'est le poids d'un plus court chemin de s à t contraint de passer par le sommet v. Remarquez que $f(v) = \delta(s, t)$ si et seulement si v se trouve sur un *plus-court-chemin* de s à t. Bien sûr la fonction f n'est pas connu, sinon $f(t) = \delta(s, t)$ serait connu. En revanche, on a accès à des informations supplémentaires sur le graphe, qui viennent par exemple du monde physique, et qui se modélisent par une fonction $\hat{f} : V \longrightarrow \mathbb{R}_+$ dite d'évaluation. Cette fonction est donnée en entrée et est une estimation de la fonction f. Pour tout sommet v, $\hat{f}(v)$ est une estimation du poids d'un *plus-court-chemin* de s à t contraint de passer par v. Cette fonction sera "améliorée" au cours de l'algorithme.

L'algorithme A^*. Soit $\eta > 0$ quelconque fixé. L'algorithme A^* prend en entrée un η- graphe $(G = (V, E), \omega)$, un sommet source s, un sommet cible t, et une fonction d'évaluation $\hat{f}$. Il trouve un *plus-court-chemin* de s à t ainsi que son poids. Comme les autres algorithmes de ce cours, A^* relaxe successivement des arêtes en mettant à jour les attributs d [5] et π jusqu'à trouver un *plus-court-chemin*. L'ordre des relaxations des arêtes est guidée par la fonction d'évaluation $\hat{f}$. Au cours de l'algorithme un sommet v est marqué *ouvert* ou *fermé* ou reste non marqué. Ouvert signifie qu'il est possible d'explorer ce sommet au tour d'après et fermé signifie qu'il a déjà été exploré. Un sommet v peut n'être marqué ni ouvert ni fermé ce qui est équivalent à avoir son attribut $v.d$ à ∞. Dans le reste du cours nous allons employer le terme **explorer** un sommet. Cela signifie en général la lecture de toutes les arête sortantes de ce sommet, et dans le cas de A^*, explorer un sommet signifie le marquer fermé et relaxer toutes les arêtes sortantes de ce sommet. L'algorithme suivant est un pseudo-code de l'algorithme A^*. De plus, Figure 3 illustre le fonctionnement de A^* sur un exemple, il est cependant conseillé de lire la Section 5.1 avant la figure.

$A^*\left(G = (V, E), s, t, \omega, \hat{f}\right)$

1: INITIALIZE(G, s)
2: Mark s as *open* and calculate $\hat{f}(s)$
3: Among all *open* nodes, select the node u whose value of $\hat{f}$ is smallest. Resolve ties arbitrary but always in favor of t
4: Mark u as *closed*
5: **if** u is t
6: Terminate
7: **for** v in $u.\text{Adj}$
8: RELAX(u, v, ω) and if $\hat{f}(v)$ decreased then mark v as open
9: Go back to step 3

L'algorithme commence par initialiser le graphe, il marque s ouvert et calcule $\hat{f}(s)$. À chaque étape, A^* choisi d'explorer un sommet parmi tous les sommets marqués ouverts, celui dont la valeur $\hat{f}$ est minimale (le plus prometteur). Durant l'exploration de u, un sommet v voisin à u est marqué ouvert lorsque le processus RELAX(u, v, ω) a fait décroître $\hat{f}(v)$ (equivalent à faire décroître $v.d$ comme on le verra). Dans ce cas, v est un potentiel sommet à explorer aux tours d'après. L'algorithme s'arrête dès que le sommet cible t est marqué fermé.

Le reste de la section est divisé en deux parties. Premièrement nous allons prouver que pour une fonction d'évaluation bien choisie l'algorithme A^* renvoie bien un *plus-court-chemin* de s à t. Deuxièmement nous prouverons que A^* est *optimal* dans le sens où tout algorithme *moins informé* que A^* doit explorer au moins autant de sommets que A^* pour garantir trouver un *plus-court-chemin* de s à t.

5.1 La fonction d'évaluation $\hat{f}$

La fonction f évaluée en un sommet v donne le poids d'un *plus-court-chemin* de s à t contraint de passer par v. Donc $f(v) = \delta(s, v) + \delta(v, t)$. Pour estimer cette fonction, nous allons estimer ses deux termes séparément.

5. et la fonction d'évaluation $\hat{f}$. Car comme on le verra, il dépendra de l'attribut d

Estimation de de s à v. On estime $\delta(s,v)$ par $v.d$. On sait par le lemme 3.2 qu'à tout moment de l'algorithme $v.d \geq \delta(s,v)$.

Estimation de v à t. L'estimation de $\delta(v,t)$ pour tout sommet v est une donnée de l'entrée sous forme de fonction dite *d'estimation* qu'on notera $\hat{h} : V \longrightarrow \mathbb{R}_+$. Plus précisément, on impose que $\hat{h}$ soit un minorant de $v \mapsto \delta(v,t)$, c'est à dire, $\hat{h}(v) \leq \delta(v,t)$ pour tout $v \in V$. Dans notre exemple en début de section, $\hat{h}(v)$ est la distance à vol d'oiseau entre la ville v et Lyon. On supposera toujours que l'hypothèse suivante est vérifiée par la fonction d'estimation :

Hypothèse 5.1 (Minoration)**.** *$\hat{h}$ est un minorant de la fonction $v \mapsto \delta(v,t)$.*

Notez qu'on peut prendre $\hat{h} = 0$ la fonction nulle. Cela revient à dire qu'on n'a pas d'information sur le graphe. On verra que plus la fonction $\hat{h}$ est grande (i.e., plus l'estimation est fine), moins l'algorithme A^* a besoin d'explorer de sommets pour trouver un *plus-court-chemin*.

La fonction d'évaluation $\hat{f}$. On estime le poids d'un plus court chemin de s à t contraint de passer par un sommet v par

$$\hat{f}(v) = v.d + \hat{h}(v)$$

où $\hat{h}(v)$ minore $\delta(v,t)$. Notez que durant l'algorithme A^*, la valeur $\hat{f}(v)$ décroît si et seulement si $v.d$ décroît. Comme on sait que $v.d$ ne peut que décroître, alors $\hat{f}$ ne fait que décroître durant l'algorithme.

5.2 A^* est correct

Dans cette section nous prouvons que A^* termine et trouve un *plus-court-chemin* de s à t (par application du processus PATH-BY-π). **Nous supposons toujours dans cette section que $\hat{h}$ vérifie l'hypothèse de la Minoration 5.1.** Commençons par prouver le lemme suivant :

Lemme 5.2. *Pour tout sommet v non-fermé et tout plus-court-chemin p de s à v, il existe un sommet ouvert u dans p avec $u.d = \delta(s,u)$.*

Démonstration. Si v est non accessible depuis s, le lemme est trivial. Soit $p = [s = v_0, v_1, \ldots, v_k = v]$ un *plus-court-chemin* de s à v. Si s est ouvert (i.e., A^* n'a pas encore atteint la ligne 4), alors s convient car $s.d = 0 = \delta(s,s)$. Sinon, soit Δ l'ensemble des sommets v_i qui sont fermés, qui appartiennent à p, et dont $v_i.d = \delta(s,v_i)$. Il est non-vide car $s \in \Delta$. Soit v_j l'element de Δ avec le plus grand indice j. Comme $v_k = v$ est non-fermé, il n'appartient pas Δ et donc $j < k$. Montrons que v_{j+1} (qui peut-être v_k) satisfait le lemme. La dernière fois que le sommet v_j a été fermé, l'arête (v_j, v_{j+1}) a été relaxé, et depuis $v_{j+1}.d$ n'a pu faire que décroître, d'où

$$\begin{aligned} v_{j+1}.d &\leq v_j.d + \omega(v_j, v_{j+1}) \\ &= \delta(s,v_j) + \omega(v_j, v_{j+1}) \text{ car } v_j \in \Delta \\ &= \delta(s,v_{j+1}) \text{ Par le lemme 2.1 car } p \text{ est un } \textit{plus-court-chemin} \end{aligned}$$

Donc $v_{j+1} = \delta(s,v_{j+1})$. Comme v_{j+1} appartient à p mais pas à Δ, il est non-fermé par définition de Δ. Il est donc ouvert car $v_{j+1}.d < \infty$. Le sommet v_{j+1} convient. □

Corollaire 5.3. *Supposons que A^* n'ait pas terminé. Alors pour tout plus-court-chemin p de s à t, il existe un sommet ouvert v dans p avec $\hat{f}(v) \leq \delta(s,t)$.*

Démonstration. Soit p un *plus-court-chemin* de s à t. Le sommet t n'est pas fermé car l'algorithme n'a pas terminé. Par le lemme 5.2, il existe $v \in p$ ouvert avec $v.d = \delta(s,v)$. D'où

$$\begin{aligned} \hat{f}(v) &= v.d + \hat{h}(v) \\ &= \delta(s,v) + \hat{h}(v) \\ &\leq \delta(s,v) + \delta(v,t) \text{ par propriéte de } \hat{h} \end{aligned}$$

Mais v est sur un *plus-court-chemin* de s à t, donc $\delta(s,v) + \delta(v,t) = \delta(s,t)$. □

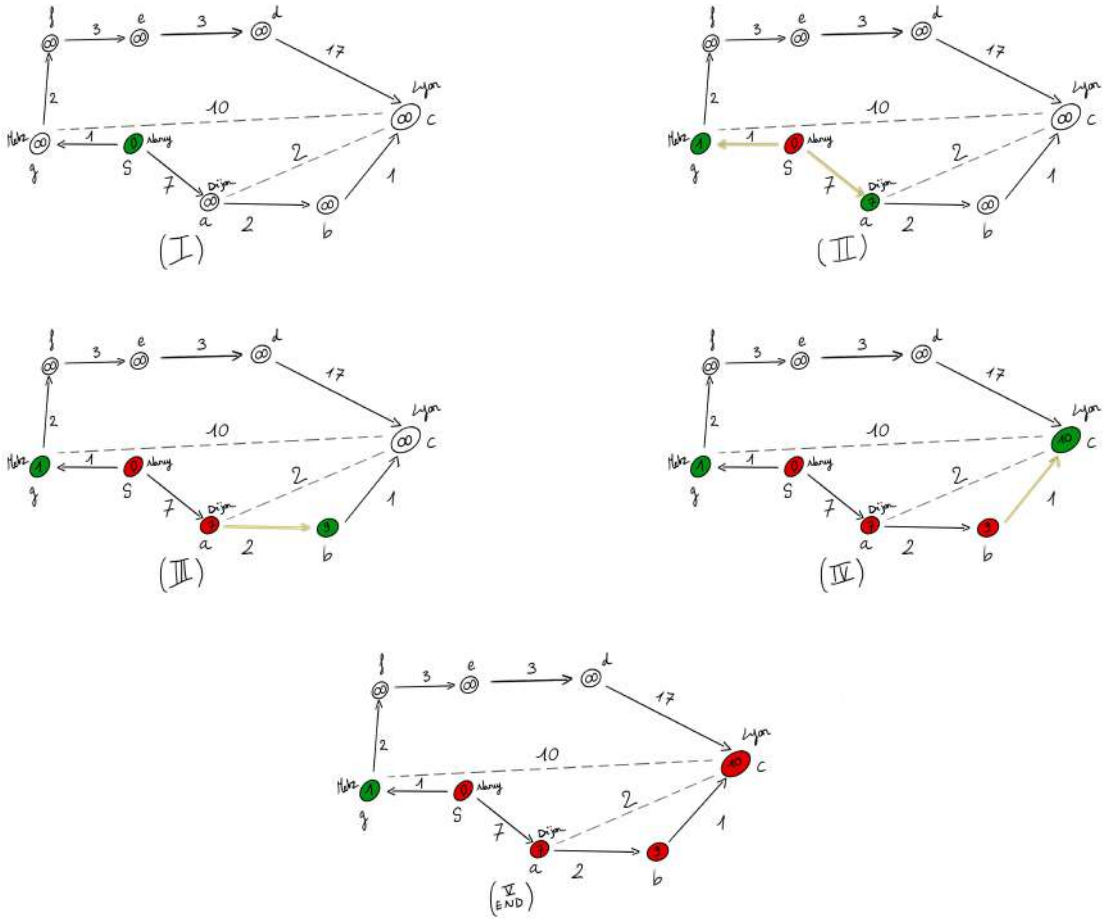

FIGURE 3 – Fonctionnement de A^* sur un exemple. Le sommet source est s et le sommet cible est c. Les valeurs à l'intérieur des sommets représentent la valeur de l'attribut d, et les valeurs au-dessus des lignes pointillées entre un sommet v et c représentent l'estimation de la distance de v à la destination c, i.e., $\hat{h}(v)$. De plus $\hat{h}(b) = 1$. Seules les valeurs $\hat{h}(a)$, $\hat{h}(b)$, et $\hat{h}(g)$ sont données ici. (I) Après initialisation, le sommet s est ouvert (vert). (II) Le sommet s est exploré, il est marqué fermé (rouge) et les arêtes (s, a) et (s, g) sont relaxées. Les sommets ouverts (verts) sont maintenant a et g et la fonction d'estimation vaut en ces sommets 9 et 11 respectivement. (III) C'est le sommet a qui est choisi pour être exploré, l'arête (a, b) est relaxée et le sommet b est ouvert. De plus, $\hat{f}(b) = 10 < \hat{f}(g)$. (IV) Le sommet b est exploré. (V) Le sommet c est marqué fermé et l'algorithme termine. Une fois vous avez lu la Section 5.4, appliquez l'algorithme de Dijkstra sur ce graphe.

On peut à présent prouver que A^* est correct.

Théorème 5.4. *Soit G un η-graphe pour un $\delta > 0$ et $\hat{h}$ une fonction d'estimation vérifiant l'Hypothèse 5.1. Alors l'algorithme A^* termine. À la terminaison on a $t.d = \delta(s,t)$ et la procédure* PATH-BY-π *appliquée à t permet de trouver en $O(|V|)$ un plus-court-chemin de s à t.*

Démonstration. Prouvons d'abord que A^* termine. Si un sommet $v \in V$ se trouve à une distance, en nombre d'arcs, supérieure strictement à $M := \delta(s,t)/\eta$, alors il ne sera jamais fermé. En effet, pour un tel sommet v on a $\hat{f}(v) \geq v.d \geq \delta(s,v) > M\eta = \delta(s,t)$ où la dernière inégalité stricte découle du fait que chaque arête est de poids supérieur à η. Or par le Corollaire 5.3 il existe un sommet ouvert u sur un *plus-court-chemin* de s à t avec $\hat{f}(u) \leq \delta(s,t)$. Donc $\hat{f}(u) < \hat{f}(v)$, l'algorithme ne choisira pas d'explorer le sommet v. On vient de prouver que l'algorithme n'explore que des sommets à distance au plus M de s en nombre d'arcs. On peut s'y restreindre. Notons $\xi(M)$ l'ensemble des sommets accessibles en au plus M pas depuis s. Tout sommet $v \in \xi(M)$ ne peut-être ouvert qu'un nombre fini de fois. En effet, l'ouverture d'un sommet v correspond à un chemin de s à v [6] et il y a un nombre fini de chemins d'au plus M pas de s à v. Donc après un nombre fini d'étapes, aucun sommet du graphe n'est ouvert, d'où par la contraposée du Corollaire 5.3 A^* termine.

Prouvons maintenant qu'à la terminaison on a $t.d = \delta(s,t)$. Supposons par l'absurde que l'algorithme termine avec $t.d > \delta(s,t)$. Mais alors juste avant la terminaison, il existe par le Corollaire 5.3 un sommet v ouvert sur un *plus-court-chemin* de s à t avec $\hat{f}(v) \leq \delta(s,t) < t.d = \hat{f}(t)$. Absurde, car l'algorithme a choisi d'explorer t et on vient de prouver $\hat{f}$ restreint au sommets ouverts n'atteint pas un minimum en t. La preuve que le processus PATH-BY-π permet de trouver un *plus-court-chemin* de s à t est identique à la preuve du Lemme 4.3. □

5.3 Optimalité de A^*

Nous allons prouver sous une hypothèse supplémentaire sur la fonction d'estimation $\hat{h}$ (Hypothèse 5.5) que A^* est optimal en nombre de sommets explorés. Plus précisément, Si A est un algorithme qui *n'est pas plus informé* sur le graphe que A^*, alors A doit explorer tout sommet exploré par A^* pour garantir trouver un *plus-court-chemin*.

Schéma de la preuve. Nous allons prouver que A^* n'explore jamais de sommets "inutiles". Tout sommet exploré par A^* est un potentiel sommet sur un *plus-court-chemin*. Supposons qu'un algorithme A n'explore pas un tel sommet v. Comme il *n'est pas plus informé* sur le graphe que A^*, il ne peut pas savoir à l'avance que v n'est pas sur un *plus-court-chemin*, et ne peut donc pas garantir de trouver un *plus-court-chemin*.

Un algorithme A n'est pas plus informé que que A^*. Cette preuve nécessite de formaliser la notion *n'est pas plus informé que*. Cependant pour faciliter la compréhension, nous allons dans un premier temps nous tenir au sens "intuitif" de cette notion en signalant précisément les endroits où cette notion est employée. Nous renvoyons le lecteur à l'Annexe A pour cette formalisation.

5.3.1 Hypothèse de consistance

Nous supposons pour le reste du cours que l'information extérieure dont nous disposons sur le graphe et qui est modélisée par la fonction $\hat{h}$ vérifie la propriété de consistance suivante, qui peut-être vu comme une "inégalité triangulaire".

Hypothèse 5.5 (Consistance)**.** *Pour tous sommets u et v :*

$$\delta(u,v) + \hat{h}(v) \geq \hat{h}(u)$$

La Figure 4 représente un graphe avec un sommet source s et un sommet destination t. Supposons que la fonction d'estimation $\hat{h}$ soit tel que $\hat{h}(s) = 8$, $\hat{h}(u) = 5$, et $\hat{h}(v) = 1$. Après exploration du sommet s, l'algorithme A^* estime $\hat{f}(u)$ à $u.d + \hat{h}(u) = 8$ et $\hat{f}(v)$ à $v.d + \hat{h}(v) = 7$. Il choisi donc "à tort" d'explorer le sommet v. Dans cet exemple l'information que la distance du sommet s au sommet t est

6. $v.d$ correspond au poids d'un chemin de s à v, celui qu'on obtient en "backtrackant" v par π.

au moins $\hat{h}(s) = 8$ a été ignorée dans l'estimation de $\hat{h}(v)$. En effet, comme on sait que v est seulement à distance 6 de s et que s est à distance au moins 8 de t, alors v est à distance au moins 2 de t. On dit que $\hat{h}(s) = 8$ et $\hat{h}(v) = 1$ sont inconsistants car :

$$\delta(s, v) + \hat{h}(v) = 7 < 8 = \hat{h}(s)$$

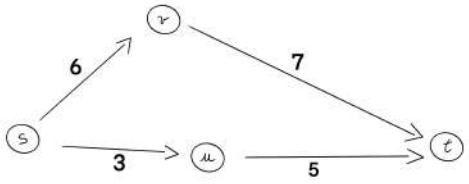

FIGURE 4 –

5.3.2 Preuve d'optimalité

Nous supposons toujours dans cette section que la fonction d'estimation vérifie les deux hypothèses, de Minoration 5.1 et de Consistance 5.5.

Lorsque A^* marque un sommet v fermé, alors A^* a déjà trouvé un *plus-court-chemin* de s à v et $v.d$ atteint sa valeur minimale. Un sommet fermé par A^* ne sera jamais réouvert, c'est l'objet du lemme suivant.

Lemme 5.6. *Soit $v \in V$ un sommet fermé par A^*, alors $v.d = \delta(s, v)$.*

Démonstration. Comme v a été fermé par A^*, alors v est acessible depuis s. considérons le graphe juste avant de fermer le sommet v et supposons par l'absurde qu'à ce moment là on a $v.d > \delta(s, v)$. Par le Lemme 5.2,il existe p un *plus-court-chemin* de s à v et un sommet $u \in p$ ouvert avec $u.d = \delta(s, u)$. Nécessairement $u \neq v$. Montrons que $\hat{f}(v) > \hat{f}(u)$. On a :

$$\begin{aligned} v.d &> \delta(s, v) \\ &= \delta(s, u) + \delta(u, v) \text{ car } u \text{ est sur un } \textit{plus-court-chemin} \text{ de } s \text{ à } v \\ &= u.d + \delta(u, v) \end{aligned}$$

En ajoutant $\hat{h}(v)$ des deux côtés on obtient :

$$\begin{aligned} \hat{f}(v) &= v.d + \hat{h}(v) \\ &> u.d + \delta(u, v) + \hat{h}(v) \\ &\geq u.d + \hat{h}(u) \text{ par l'hypothèse de la consistance 5.5} \\ &= \hat{f}(u) \end{aligned}$$

Contredit le fait que A^* ait choisi d'explorer le sommet v car u était ouvert et $\hat{f}(u) < \hat{f}(v)$. □

Le lemme suivant montre que la fonction d'évaluation $\hat{f}$ est croissante sur la suite des sommets fermés par A^*.

Lemme 5.7. *Soit $(v_1, v_2, \ldots, v_k)$ la suite des sommets fermés par A^* dans l'ordre. Si $p < q$ alors $\hat{f}(v_p) \leq \hat{f}(v_q)$.*

Démonstration. Supposons que A^* ait fermé dans l'ordre un sommet u puis un sommet v.

Supposons d'abord que u ne se trouve pas sur un *plus-court-chemin* de s à v, alors v était ouvert au moment où u a été sélectionne par A^* pour exploration, pourquoi ? L'autre possibilité serait que v ait été ouvert par l'exploration du sommet u, c'est à dire que (u, v) est une arête, et RELAX(u, v, ω) a mis

à jour $v.d$ pour qu'il devient $v.d = u.d+\omega(u,v)$. Mais comme v est choisi pour être fermé le tour d'après, alors $v.d = u.d + \omega(u,v) = \delta(s,v)$ par le Lemme 5.6. C'est bien que u est sur un *plus-court-chemin* de s à v ce qui contredit notre hypothèse. Donc si u n'est pas sur un *plus-court-chemin* de s à v, alors v était ouvert au moment ou A^* a choisi d'explorer u, d'où $\hat{f}(u) \leq \hat{f}(v)$.

Supposons maintenant que u se trouve sur un *plus-court-chemin* de s à v. Alors :

$$\begin{aligned}
\hat{f}(v) &= v.d + \hat{h}(v) \\
&= \delta(s,v) + \hat{h}(v) \text{ par le Lemme 5.6 car } v \text{ est fermé} \\
&= \delta(s,u) + \delta(u,v) + \hat{h}(v) \text{ car } u \text{ est sur un } \textit{plus-court-chemin} \text{ de } s \text{ à } v \\
&\geq \delta(s,u) + \hat{h}(u) \text{ par l'hypothèse de la consistance 5.5} \\
&= u.d + \hat{h}(u) \text{ par le Lemme 5.6 car } u \text{ est fermé} \\
&= \hat{f}(u)
\end{aligned}$$

□

Nous pouvons à présent prouver l'optimalité de A^*. Si un algorithme A *n'est pas plus informé* par A^*, alors tout sommet exploré par A^* est exploré par A. On commence par prouver le théorème dans le cas particulier où aucune égalité sur les valeurs de $\hat{f}$ n'a lieu durant l'algorithme et généralisons ce résultat ensuite.

Théorème 5.8. *Soit A un algorithme qui n'est pas plus informé que A^* et supposons que les hypothèses de Minoration 5.1 et de Consistance 5.5 sur $\hat{h}$ utilisé par A^* sont satisfaites. Soit G un η-graphe pour un $\eta > 0$ et supposons que pour tous sommets $u \neq v$ ouverts durant l'algorithme A^*, on a $\hat{f}(u) \neq \hat{f}(v)$. Alors si un sommet v a été exploré par A^*, alors il a été exploré par A.*

Démonstration. Supposons par l'absurde l'existence d'un sommet $u \in V$ tel que u ait été exploré par A^* mais pas par A. Comme u a été fermé par A^* avant le sommet cible t, alors par le Lemme 5.7 on a à la fin de l'execution de A^* :

$$\hat{f}(u) < \hat{f}(t) = \delta(s,t) \tag{3}$$

Où l'inégalité est stricte par notre hypothèse sur $\hat{f}$.

Soit le graphe G' identique à G où l'on rajoute l'arête (u,t) avec un poids $\omega(u,t) := \hat{h}(t)$. On notera δ_G respectivement $\delta_{G'}$ pour les fonctions "poids de *plus-court-chemin*" dans G respectivement dans G'. Donc l'inégalité (3) s'écrit $\hat{f}(u) < \delta_G(s,t)$. On va montrer que A ne trouve pas un *plus-court-chemin* de s à t dans G'.

D'une part, comme l'algorithme A n'a pas exploré le sommet u (i.e., n'a pas lu les arêtes sortantes de u), il renvoie la même réponse que sur G lorsque G' est donné en entrée, une réponse correspondante à un chemin p de poids $\delta_G(s,t)$. D'autre part, si G' est donné en entrée à A^*, alors comme pour G, l'algorithme A^* explore le sommet u. En effet, avant l'exploration du sommet u, A^* n'a pas lu l'arête (u,t) qui différencie G' de G. Donc u est fermé par A^* et on a $u.d = \delta_{G'}(s,u) = \delta_G(s,u)$ par le Lemme 5.6. De plus, dans le graphe G', un *plus-court-chemin* de u à t est $[u,t]$. En effet, dans G' on a $\delta_{G'}(u,t) = \omega(u,t) := \hat{h}(u)$, car le poids de l'arête (u,t) qui est $\hat{h}(u)$ est plus petit que le poids de tout chemin de u à t dans G par l'hypothèse de minoration 5.1 sur $\hat{h}$. On en déduit que dans G', le poids du chemin de s à u trouvé par A^* qu'on complète par l'arête (u,t) : $s \rightsquigarrow u \to t$ est :

$$\delta_{G'}(s,u) + \omega(u,t) = u.d + \hat{h}(u) = \hat{f}(u)$$

On a donc dans G' un chemin de s à t de poids $\hat{f}(u)$. Par l'inégalité (3), ce chemin est de poids strictement plus petit que $\delta_G(s,t)$ et pourtant A a renvoyé sur l'entrée G' un chemin de poids $\delta_G(s,t)$.

Comme A *n'est pas plus informé* sur G que A^*, il ne pouvait pas éliminer l'existence de l'arête (u,t) que nous avons défini sans explorer le sommet u. Il ne garanti donc pas de trouver un *plus-court-chemin* de s t. Contradiction avec l'hypothèse que A résout le problème de *plus-court-chemin*. □

Nous venons de prouver un résultat d'optimalité de A^* dans le cas où aucune égalité des valeurs de $\hat{f}$ ne se produit durant l'algorithme. Généralisons ce résultat d'optimalité. Dans un cas d'égalité, c'est à dire lorsque deux sommets ou plus tous ouverts $v_1, \ldots, v_k$ vérifient $\hat{f}(v_1) = \cdots = \hat{f}(v_k) < \hat{f}(v)$

pour tout autre sommet ouvert v, l'algorithme A^* choisit arbitrairement un sommet parmi les v_i pour l'explorer [7].

Soit $\mathfrak{A}^*$ l'ensemble des algorithmes qui agissent de la même manière que A^* s'il n'y a pas d'égalité des valeurs de $\hat{f}$, et qui choisissent de manière différente les sommets à explorer en cas d'égalité. Pour tout choix possible de sommet à explorer en cas d'égalité, il existe un algorithme dans $\mathfrak{A}^*$ qui applique ce choix en cas d'égalité.

Le théorème suivant énonce que pour tout algorithme A résolvant *plus-court-chemin* et qui *n'est pas plus informé que* les algorithmes dans $\mathfrak{A}^*$, il existe un algorithme A^* dans $\mathfrak{A}^*$ tel que tout sommet exploré par A^* est aussi exploré par A.

Théorème 5.9. *Soit A un algorithme qui n'est pas plus informé que que les algorithmes dans $\mathfrak{A}^*$ et supposons que les hypothèses de Minoration 5.1 et de Consistance 5.5 sur $\hat{h}$ utilisé par les algorithmes dans $\mathfrak{A}^*$ sont satisfaites. Pour tout η-graphe G, avec $\eta > 0$, il existe $A^* \in \mathfrak{A}^*$ tel que tout sommet exploré par A^* est aussi exploré par A.*

Démonstration. Soit A_1^* un algorithme quelconque de $\mathfrak{A}^*$. Si tout sommet exploré par A_1^* est aussi exploré par A, alors A_1^* satisfait le théorème. Sinon, nous allons modifier la manière dont A_1^* règle les égalités pour construire un algorithme $A^* \in \mathfrak{A}^*$ qui satisfait le théorème.

Soit L l'ensemble des sommets explorés par A, et notons $p = [v_0 = s, v_1, \ldots, v_k = t]$ le *plus-court-chemin* trouvé par A. Tant que A_1^* explore des sommets dans L on ne fait rien. Soit u le premier sommet exploré par A_1^* qui n'appartient pas à L. D'une part, on sait par le Lemme 5.7 que $\hat{f}(u) \leq \hat{f}(t) = \delta(s,t)$. De plus, par la preuve du Theorème 5.8, on ne peut pas avoir $\hat{f}(u) < \delta(s,t)$ auquel cas A ne garantirait pas de trouver un *plus-court-chemin* car u a été exploré par A_1^* mais pas par A. On a donc $\hat{f}(u) = \delta(s,t)$. D'autre part, au moment où A_1^* a choisi d'explorer u le sommet cible t n'était pas fermé (sinon A_1^* aurait terminé), donc par le Corollaire 5.3, il existe un sommet v ouvert sur p avec $\hat{f}(v) \leq \delta(s,t)$. On en déduit que $\hat{f}(v) \leq \hat{f}(u)$. Mais comme A_1^* a choisi d'explorer u, alors $\hat{f}(u) \leq \hat{f}(v)$. Donc $\hat{f}(u) = \hat{f}(v)$. Soit $A_2^* \in \mathfrak{A}^*$ identique à A_1^* sauf pour la manière dont les égalités des valeurs de $\hat{f}$ sont traités, de sorte que A_2^* choisi d'explorer v plutôt que u. En répetant cet argument, on construit un algorithme A_i^* qui n'explore que des sommets dans L, i.e., des sommets explorés par A. L'algorithme A_i^* satisfait le théorème. □

5.4 Algorithme de Dijkstra

L'algorithme de Dijkstra est un cas particulier de l'algorithme A^* lorsque aucune information n'est disponible sur le graphe, i.e., lorsque la fonction d'estimation $\hat{h}$ est la fonction nulle, et donc la fonction d'évaluation $\hat{f}$ est égale à l'attribut d. Algorithme Dijkstra est un pseudo-code de l'algorithme de Dijkstra et Figure 5 en est une illustration sur un exemple.

Dijkstra $(G = (V, E), s, t, \omega)$

1: INITIALIZE(G, s)
2: Mark s as *open*
3: Among all open nodes, select the node u whose value of $u.d$ is smallest. Resolve ties arbitrary but always in favor of t
4: Mark u as *closed*
5: **if** u is t
6: Terminate
7: **for** v in $u.\mathrm{Adj}$
8: RELAX(u, v, ω) and if $v.d$ decreased then mark v as open
9: Go back to step 3

Remarque 5.10. *Appliquez l'algorithme de Dijkstra sur l'exemple de la Figure 3 et comparez l'efficacité de de Dijkstra et de A^*.*

7. Si le sommet t fait parti des v_i, alors c'est le sommet choisi par A^*.

Dijkstra ciblant tous les sommets. On peut modifier l'algorithme de Dijkstra pour qu'il calcule un *plus-court-chemin* de s à tous les sommets du graphe plutôt qu'à un sommet cible t, comme le fait l'algorithme de Bellman- Ford [8]. Il suffit de changer la condition d'arrêt de Dijkstra en remplaçant "**if** u is t" par "**if** there are no more open nodes". Cette considérantion est identique pour l'algorithme A^* mais est moins pertinente. En effet, la fonction d'estimation $\hat{h}$ permet de mieux diriger la recherche d'un *plus-court-chemin* vers un sommet (ou un ensemble de sommets) cible t. Si nous cherchons un *plus-court-chemin* vers tous les sommets, alors la fonction d'estimation n'a plus d'intêret et on peut appliquer l'algorithme de Dijkstra.

Complexité. Nous allons prouver en exercices que lorsque le graphe est donné sous dorme de listes d'adjacence, l'algorithme de Dijkstra est de complexité asymptotique $O(|V|^2 + |E|) = O(|V|^2)$ où V est l'ensemble des sommets du graphe et E l'ensemble de ses arêtes. Rappelons que la complexité asymptotique de l'algorithme de Bellman-Ford est $O\left(|V|^2 + |V||E|\right)$.

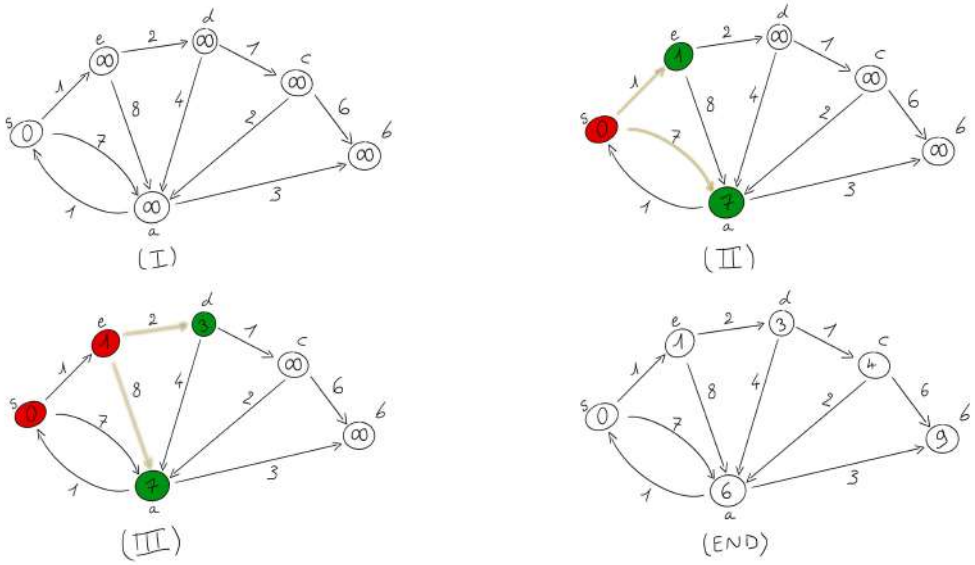

FIGURE 5 – Algorithme de Dijkstra sur un exemple. (I) le graphe est initialisé. Le sommet s est le seul sommet ouvert. (II) Le sommet s est exploré, donc le sommet s est marqué fermé (rouge) et les arêtes (s, a) et (s, e) sont relaxées. Les valeurs $a.d$ et $e.d$ sont mis à 7 et 1 respectivement et les sommets correspondants sont marqués ouverts (vert). (III) Comme $e.d = 1 < 7 = a.d$, l'arête e est exploré. (END) À la terminaison de l'algorithme, tous les sommets ont été fermé.

A Formalisation de la notion "*n'est pas plus informé que*"

La fonction d'estimation $\hat{h}$ est une borne inférieure de la fonction "distance restante jusqu'au sommet cible" : $v \mapsto \delta(v, t)$. Pour un sommet v, la fonction $\hat{h}$ sur v nous dit qu'il n'existe pas de chemin de v à t de poids strictement plus petit que $\hat{h}(v)$, et ce sans avoir besoin d'explorer le sommet v. Si nous prenons $\hat{h} = 0$ la fonction nulle, alors en tout sommet v, on n'a aucune information sur les chemins possibles de v à t. Le sommet v peut-être arbitrairement proche de l'arrivée t. Très souvent on a de l'information sur le graphe qui vient du monde physique et qui contraint les chemins possibles du sommet v au sommet cible t. Dans l'exemple des routes entre villes, on sait sans explorer les routes partant d'une ville v qu'il n'y a pas de chemin routière entre v et la ville destination t de distance strictement plus petite que la distance à vol d'oiseau entre les deux villes. Cette information restreint les chemins possibles d'un sommet donné au sommet cible.

On suppose qu'à chaque sommet v, on a de l'information, du monde physique par exemple, qui restreint les chemins possibles $\{p_{v,i}\}_{i \in \Omega_v}$ de v à t, où Ω_v est un ensemble d'indices indexant les chemins

8. Contrairement à Bellman-Ford, Dijkstra ne s'applique que sur des graphes dont les arêtes ont un poids strictement positif

possibles. Supposons que cette information restreint l'ensemble Ω_v à un ensemble plus petit $\Theta_v \subset \Omega_v$. Si on reprend notre exemple de routes entre ville. Prendre $\hat{h} = 0$ la fonction nulle est équivalent à considérer qu'à chaque sommet v on a $\Theta_v = \Omega_v$, c'est à dire qu'il peut y avoir des chemins arbitrairement courts entre v et t. Prendre $\hat{h}$ la distance à vol d'oiseau est équivalent à considérer pour tout sommet v, Θ_v un sous ensemble strict de Ω_v qui permet de décrire les chemins de v à t dont le poids est supérieur à la distance à vol d'oiseau entre les deux villes. Avec cette formalisation de l'information disponible sur le graphe, on définit la fonction d'estimation $\hat{h}$ en un sommet v comme la borne inférieure des poids de chemins possibles de v à t :

$$\hat{h}(v) := \inf_{\theta \in \Theta_v} \text{Poids}\,(p_{v,\theta}) \tag{4}$$

Où $\text{Poids}(p_{v,\theta})$ est le poids du chemin possible $p_{v,\theta}$. Nous supposons que cette borne est atteinte pour un certain $\theta \in \Theta_v$. Insistons que $p_{v,\theta}$ n'est pas nécessairement un chemin du graphe mais c'est en quelque sorte un chemin qui peut exister depuis v, on n'a pas d'information pour infirmer ou confirmer son existence sans explorer le sommet v. Cette formalisation de l'information disponible sur le graphe permet de parler des algorithmes qui utilisent cette information.

Définition A.1. *Soit A un algorithme résolvant le problème de plus-court-chemin. Soit G un graphe, et notons pour tout sommet v de G, Θ_v^A, respectivement, $\Theta_v^{A^*}$ les ensembles d'indices utilisés par l'algorithme A, respectivement A^*. On dit que A n'est pas plus informé que A^* si $\Theta_v^{A^*} \subset \Theta_v^A$ pour tout sommet v.*

Exemple A.2. *Sur les routes entre villes en France, l'algorithme de Dijkstra, ou encore l'algorithme de Bellman-Ford, ou encore un algorithme A_1^* utilisant la fonction d'estimation "distance à vol d'oiseau divisé par 2", n'est pas plus informé qu'un algorithme A^* utilisant la fonction d'estimation "distance à vol d'oiseau".*

Références

[CLRS22] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, fourth edition.* MIT Press, 2022.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2) :100–107, 1968.

Remerciements. Je tiens à exprimer ma gratitude envers Pauline Thomas pour la réalisation des figures qui enrichissent ce polycopié.