Recherche Opérationnelle

Chapitre 3 : Programmation dynamique
Plus court chemin dans un graphe

J.-F. Scheid
Institut Elie Cartan de Lorraine/Télécom Nancy
Université de Lorraine

2025-26 (version du 21/10/25)

Table des matières

I. Graphes

De nombreuses phénomènes/situations peuvent être modélisés par des graphes. Quelques exemples :

- réseau routier : les sommets sont les intersections des routes, les arêtes représentent les routes. Problème du voyageur de commerce (TSP).
- cheminement dans un réseau informatique.
- web modélisé par un graphe. Les sommets sont les pages Web et les arêtes sont les liens hypertexte entre ces différentes pages.

Définition 1.

Un graphe *orienté* $G = (E, \Gamma)$ est constitué d'un ensemble fini E de *sommets* et d'un ensemble fini Γ de couples ordonnés (i, j) avec $i, j \in E$. Les éléments de Γ sont appelés les **arêtes** du graphe.

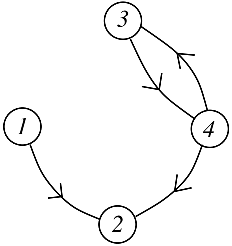
Notation (les flèches indiquent l'orientation des arêtes) :

Remarques :

- Pour un graphe orienté, les arêtes (i, j) et (j, i) sont distinctes.
- Pour un graphe non-orienté, ces arêtes sont confondues et sont représentées par un seul arc sans flêche ; l'arête est notée $\{i, j\}$.
- Dans un graphe orienté, un sommet i peut avoir une arête sur lui-même, i.e. une arête (i, i) ce qu'on appelle une *boucle*. Un graphe orienté sans boucle est dit *simple*.

Exemple de graphe orienté :

$E = \{1, 2, 3, 4\}$; $\Gamma = \{(1, 2), (3, 4), (4, 2), (4, 3)\}$



Définition 2.

L'ensemble $G = (E, \Gamma, c)$ est un **graphe valué** si (E, Γ) est un graphe auquel on associe une fonction positive $c : \Gamma \to \mathbb{R}^+$ appelée **valuation** ou **capacité**. La capacité de l'arête (i, j) est notée c_{ij}.

Exemple : La capacité c_{ij} représente par exemple :

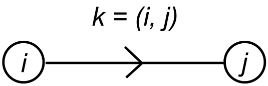
- la distance ou longueur du tronçon de route (i, j) entre deux villes i et j (cf. TSP)
- le nombre maximal de voitures par unité de temps entre deux villes i et j
- le débit maximal entre deux points i et j d'approvisionnement en eau$/$gaz.
- ...

Représentation d'un graphe.

a) Matrice d'incidence sommet-arête

Soit un graphe avec n sommets et m arêtes, **sans boucle** c-à-d sans arête (i, i). On définit A la **matrice d'incidence** de taille $n \times m$:

$$a_{ik} = \begin{cases} -1 & \text{si le sommet } i \text{ est l'extrémité } \textbf{initiale} \text{ de l'arête } k \\ +1 & \text{si le sommet } i \text{ est l'extrémité } \textbf{terminale} \text{ de l'arête } k \\ 0 & \text{sinon} \end{cases}$$



$$a_{ik} = -1$$
$$a_{jk} = +1$$

Exemple : $E = \{1, 2, 3, 4\}$; $\Gamma = \{(1, 2), (2, 3), (3, 1), (2, 4)\}$

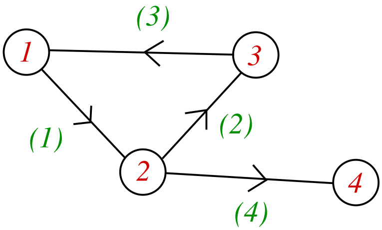


Tableau des correspondances entre les sommets (lignes) et les arêtes (colonnes) et matrice d'incidence A :

	(1,2)	(2,3)	(3,1)	(2,4)
1	-1	0	$+1$	0
2	$+1$	-1	0	-1
3	0	$+1$	-1	0
4	0	0	0	$+1$

$$A = \begin{pmatrix} -1 & 0 & 1 & 0 \\ 1 & -1 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

b) Matrice d'adjacence sommet-sommet

Le graphe est représenté par une matrice booléenne A de taille $n \times n$ (n sommets) dont les coefficients a_{ij} sont définis par

$$a_{ij} = \begin{cases} 1 & \text{si l'arête } (i,j) \text{ existe dans le graphe} \\ 0 & \text{sinon} \end{cases}$$

Exemple : $E = \{1,2,3,4\}$; $\Gamma = \{(1,2),\ (2,3),\ (3,1),\ (2,4)\}$

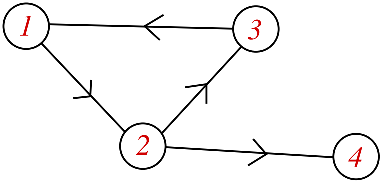


Tableau des correspondances entre les sommets et matrice d'adjacence A :

	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	1	0	0	0
4	0	0	0	0

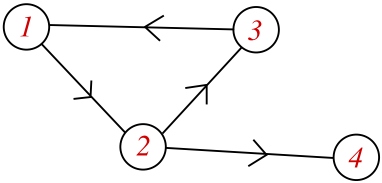
$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

c) Listes d'adjacence : successeurs et prédécesseurs

Pour chaque sommet i du graphe, on définit

- la liste de ses **successeurs** $S(i)$: liste des sommets j tq l'arête (i, j) existe dans le graphe.
- la liste de ses **prédécesseurs** $P(i)$: liste des sommets j tq l'arête (j, i) existe dans le graphe.

Exemple : $E = \{1, 2, 3, 4\}$; $\Gamma = \{(1, 2), (2, 3), (3, 1), (2, 4)\}$



sommet	successeur S	prédécesseur P
1	2	3
2	3, 4	1
3	1	2
4	–	2

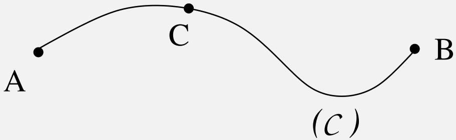
II. Programmation dynamique

1. Principe d'optimalité de Bellman

Inventée par Bellman ($\sim$ 1954) pour résoudre des pb de chemins optimaux (longueur max. ou min.) Méthode de construction d'algorithme très utilisée en optimisation combinatoire ($\rightarrow$ recherche de solution optimale dans un ensemble **fini** de solutions **mais très grand**).

Principe d'optimalité de Bellman.

Un chemin optimal est formé de sous-chemins optimaux : Si $(\mathcal{C})$ est un chemin *optimal* allant de A à B et si C appartient à $(\mathcal{C})$ alors les sous-chemins de $(\mathcal{C})$ allant de A à C et de C à B sont *optimaux*.

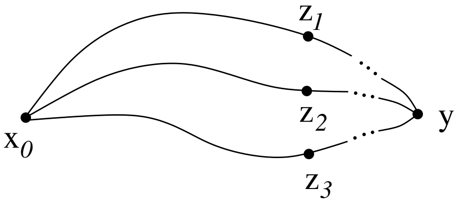


Démonstration par l'absurde.

Ce principe appliqué de façon *séquentielle* fournit des **formules récursives** pour la recherche de chemins optimaux.

Un exemple.

On cherche le chemin le plus court allant de x_0 fixé à y quelconque dans un graphe valué.



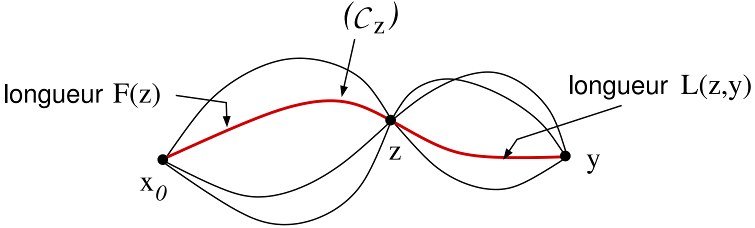
On note $F(y)$ la longueur *minimale* de tous les chemins allant de x_0 à y.

Formule récursive

$$F(y) = \min_{z \neq y} \left(F(z) + L(z, y) \right) \quad (1)$$

où $L(z, y)$ est la longueur *minimale* de tous les chemins allant de z à y.

Démonstration de la formule récursive (1). Soit un point z fixé dans le graphe. On considère un chemin *minimal* $(\mathcal{C}_z)$ allant de x_0 à y et passant par z. Par le principe d'optimalité de Bellman, le sous-chemin allant de x_0 à z est minimal et a pour longueur $F(z)$. De même, le sous-chemin allant de z à y est minimal et a pour longueur $L(z, y)$.



Donc la longueur du chemin $(\mathcal{C}_z)$ est $F(z) + L(z, y)$. On conclut en prenant le minimum sur $z \;\Rightarrow F(y) = \min_z (F(z) + L(z, y))$. $\square$

La programmation dynamique appliquée à un problème donné, consiste à trouver une *formulation récursive* du problème. En procédant ensuite à un découpage étape par étape, on obtient une *formule de récurrence*.

2. Programmation dynamique pour le plus court chemin dans un graphe

(a) Optimisation "passé $\rightarrow$ futur" - algorithme de Bellman

On cherche les plus courts chemins de x_0 fixé (racine du graphe) à y quelconque. On note $F(y)$ la longueur de ces chemins **minimaux**. D'après le principe de Bellman,

$$F(y) = \min_{z \in P(y)} \left(F(z) + L(z, y)\right) \tag{2}$$

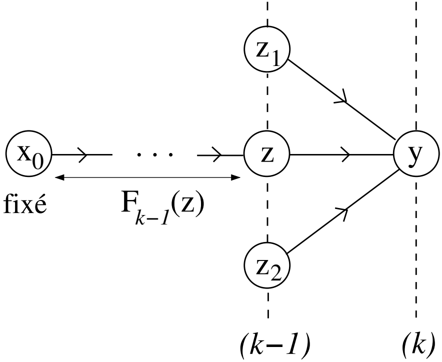
où $L(z, y)$ est la longueur de l'arête allant de z à y

$P(y)$ est l'ensemble des *sommets prédécesseurs* de y.

Par récurrence, on obtient la formule suivante.

Formule de récurrence (passé→futur)

$$\boxed{\begin{array}{l} \bullet \quad F_0(x_0) = 0, \quad F_0(y) = +\infty, \ y \neq x_0 \\ \bullet \quad F_k(y) = \min\limits_{z \in P(y)} \left(F_{k-1}(z) + L(z, y)\right) \end{array}} \tag{3}$$



Interprétation.

- $F_k(y) = +\infty$ s'il n'y a pas de chemin entre x_0 et y avec au plus k arêtes.
- $F_k(y)$ représente la longueur *minimale* entre x_0 et y, de tous les chemins qui vont de x_0 à y et ayant au plus k arêtes.

Chemin minimal. Pour déterminer le *chemin minimal*, on considère l'ensemble des sommets

$$P_k(y) = \{z \in P(y) \text{ tel que } F_k(y) = F_{k-1}(z) + L(z, y)\}$$

C'est l'ensemble des prédécesseurs de y qui réalisent le minimum.

On utilise cet ensemble pour déterminer le chemin minimal.

Chemin minimal

$$\boxed{\begin{array}{l} x_N = y; \\ x_k \in P_{k+1}(x_{k+1}) \quad \text{(jusqu'à la racine } x_0\text{)} \end{array}} \tag{4}$$

Algorithme de Bellman.

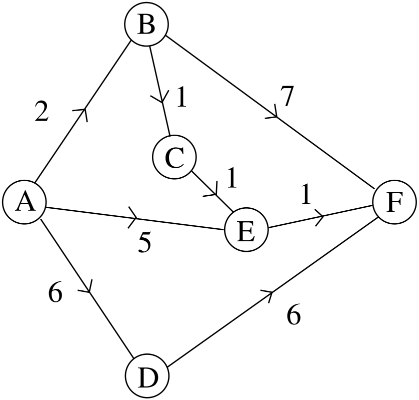
$P(y)$ désigne l'ensemble des prédécesseurs du sommet y **au sens large** i.e. $y \in P(y) \Rightarrow F_k$ est toujours décroissante.

```
N : nb de sommets
x0 : racine du graphe
P : ens. des prédécesseurs (sens large)
l(z,y) : longueur de l'arête (z,y)

F0(x0)=0, F0(y) = +∞ pour y ≠ x0
Pour k de 1 à N-1
  | Pour tout sommet y
  |   F1(y)= min(F0(z) + l(z,y); z ∈ P(y))
  | Fin pour
  | F0=F1
Fin pour
```

Exemple.

Déterminer dans le graphe suivant le plus court chemin de A à F (les valuations sur les arêtes sont les distances entre sommets) avec l'algorithme de Bellman.



Calcul des F_k.

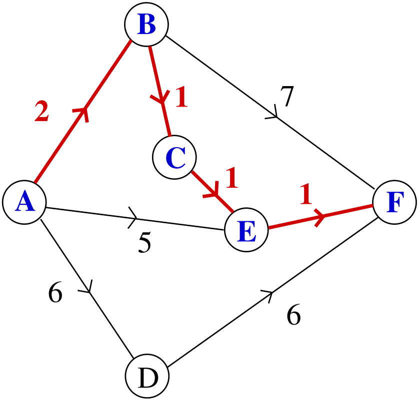
	A	B	C	D	E	F
F_0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
F_1	0	**2**	$+\infty$	**6**	**5**	$+\infty$
F_2	0	2	**3**	6	5	**6**
F_3	0	2	3	6	**4**	6
F_4	0	2	3	6	4	**5**

- $F_1(B) = \mathbf{2}$; $P_1(B) = \{A\}$
  $F_1(D) = \mathbf{6}$; $P_1(D) = \{A\}$
  $F_1(E) = \mathbf{5}$; $P_1(E) = \{A\}$
- $F_2(C) = F_1(B) + 1 = \mathbf{3}$; $P_2(C) = \{B\}$

  $$\begin{aligned} F_2(F) &= \min\left(F_1(F),\ F_1(B)+7,\ F_1(D)+6,\ F_1(E)+1\right) \\ &= \min(+\infty, 9, 12, \mathbf{6}) = \mathbf{6}; \quad P_2(F) = \{E\} \end{aligned}$$
- $$\begin{aligned} F_3(E) &= \min\left(F_2(E),\ F_2(A)+5,\ F_2(C)+1\right) \\ &= \min(5, 5, \mathbf{4}) = \mathbf{4}; \quad P_3(E) = \{C\} \end{aligned}$$
- $$\begin{aligned} F_4(F) &= \min\left(F_3(F),\ F_3(B)+7,\ F_3(D)+6,\ F_3(E)+1\right) \\ &= \min(6, 9, 12, \mathbf{5}) = \mathbf{5}; \quad P_4(F) = \{E\} \end{aligned}$$

Chemin optimal.

$$
\begin{array}{lcl}
x_4 = F & & \\
x_3 = P_4(F) = E & & \\
x_2 = P_3(E) = C & \Rightarrow & \text{chemin optimal } (A, B, C, E, F) \\
x_1 = P_2(C) = B & & \text{de longueur minimale 5} \\
x_0 = P_1(B) = A & &
\end{array}
$$



(b) Optimisation "futur $\rightarrow$ passé" - algorithme de Dijkstra

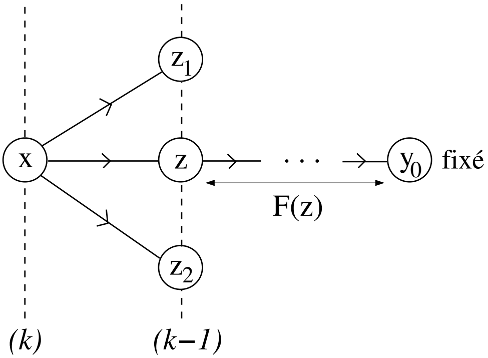
On cherche les plus courts chemins de x quelconque à y_0 fixé (antiracine du graphe). On note $F(x)$ la longueur de ces chemins **minimaux**. D'après le principe de Bellman,

$$F(x) = \min_{z \in S(x)} \left(F(z) + L(x,z)\right)$$

où $L(x,z)$ est la longueur de l'arête allant de x à z
$S(x)$ est l'ensemble des *sommets successeurs* de x.

Formule de récurrence (futur$\rightarrow$passé)

$$\boxed{\begin{array}{l} \bullet \quad F_0(y_0) = 0, \quad F_0(x) = +\infty,\ x \neq y_0 \\ \bullet \quad F_k(x) = \min\limits_{z \in S(x)} \left(F_{k-1}(z) + L(x,z)\right) \end{array}} \tag{5}$$



Interprétation.

- $F_k(y) = +\infty$ s'il n'y a pas de chemin allant de x à y_0 avec au plus k arêtes.
- $F_k(y)$ représente la longueur *minimale* entre x et y_0, de tous les chemins qui vont de x à y_0 et ayant au plus k arêtes.

Algorithme de Dijkstra

```
E : ens. des sommets du graphe
x0 : racine du graphe
S : ens. des successeurs
l(z,y) : longueur de l'arête (z,y)

F0(x0)=0, F0(y) = +∞ pour y ≠ x0
T=E-{x0};  y=x0
Tant que T ≠ ∅
  | Pour tout sommet z ∈ S(y)∩T
  |   F0(z)= min(F0(z), F0(y) + l(y,z))
  | Fin pour
  | y est le sommet de T tq F0(y)= min(F0(z); z∈ T)
  | T=T-{y}
Fin tant que
```

Pour déterminer le chemin minimal, on peut utiliser le tableau `Pred` défini de la façon suivante : Pour tout sommet `z ∈ S(y)∩T`, si `F0(y) + l(y,z)) < F0(z)` alors `Pred(z)=y`.

Exemple.

On reprend le graphe de l'Exemple 18 et on applique l'algorithme de Dijkstra pour déterminer un plus court chemin. Dans le tableau ci-dessous, les valeurs de `F0` sont affichées pour les sommets du graphe à chaque itération.

A	B	C	D	E	F	y
0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	A
	2	$+\infty$	6	5	$+\infty$	B
		3	6	5	9	C
			6	**4**	9	E
			6		**5**	F
			6			D

Les valeurs en gras dans le tableau correspondent aux distances minimales obtenues pour chaque sommet (à partir de A). On obtient aussi Pred(B)=A, Pred(C)=B, Pred(D)=A, Pred(E)=C, Pred(F)=E, ce qui donne à partir de F le chemin minimal (dans l'ordre inverse) F,E,C,B,A.

(c) Remarques et comparaisons des algorithmes

• Dans les formules de récurrence précédentes, les valuations peuvent être de signes **quelconques**.

• Attention aux graphes *avec circuit* : un **circuit** dans un graphe est un chemin dont les extrémités sont confondues. La **valeur du circuit** est alors définie comme la somme de toutes les valuations des arêtes du circuit.

- Si un circuit a une valeur *strictement positive*, on ne peut pas boucler sur le circuit car sinon on ne fait qu'augmenter la longueur du chemin.
- Si un circuit a une valeur *strictement négative*, alors la longueur du chemin n'est plus minorée et on boucle...

Les différents algorithmes de programmation dynamique utilisant les formules de récurrences précédentes pour la recherche de chemins *minimaux* sont valables pour des graphes **sans circuit de valeur strictement négative**.

Plusieurs algo de programmation dynamique pour le plus court chemin.

1. *Algorithme de Bellman*. Valuations de signes **quelconques** sur un graphe sans circuit de valeur négative. Nécessite une numérotation topologique des sommets : arêtes (i, j) avec $i < j$. Complexité en $\mathcal{O}(nm)$ avec n sommets, m arêtes.
2. *Algorithme de Dijkstra (1956)*. Valuations **positives** sur un graphe qui peut comporter des circuits (de valeurs positives). Complexité en $\mathcal{O}((n+m)\log(n))$ avec un *tas* pour gérer l'ensemble T par une structure de donnée "file de priorité" efficace pour l'opération de retirer le sommet y de T.
3. *Algorithme* A^* (cf. section suivante)

Tous ces algorithmes diffèrent essentiellement sur la façon de parcourir les sommets. Par exemple,

- Bellman : on choisit un sommet dont tous les prédécesseurs ont déjà été traités.
- Dijkstra : on choisit le sommet avec la plus petite distance.

III. Algorithme A^*

Proposé par Hart, Nilsson, Raphael (1968). Algorithme *heuristique* de programmation dynamique qui fournit généralement une solution *approchée*. Très utilisé pour sa rapidité (jeux vidéo, itinéraire...)

Algorithme de recherche d'un plus court chemin dans un graphe allant de x_0 à x_F.

- Il est basé sur une **évaluation heuristique** à chaque sommet pour *estimer* le meilleur chemin qui y passe.
- Les sommets sont parcourus en fonction de cette évaluation heuristique : on retient le sommet où l'évaluation est la plus petite.

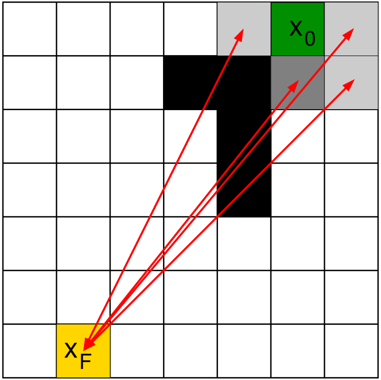
Différence Dijkstra/A^* :

• Dijkstra : recherche exhaustive parmi tous les sommets ; on calcule la distance minimale de chaque sommet à x_F.
• A^* : on réduit l'ensemble des sommets à explorer.

1. Plus court chemin dans un graphe

Recherche un plus court chemin dans un graphe allant de x_0 à x_F.
Grille avec obstacles (cases noires sur la figure ci-dessous) modélisée par un graphe :

- sommets : cases de la grille
- arêtes : déplacements possibles d'une case à une case voisine (déplacement vertical/horizontal/diagonal...).



Soit x un sommet du graphe. On définit alors les quantités suivantes.

$F(x)$ la longueur du plus court chemin allant de x_0 au sommet x.
$L^*(x)$ la longueur du plus court chemin allant du sommet x à x_F.

Pour tout sommet x, on suppose qu'on sait calculer **une approximation minorante** $L(x)$ de la longueur minimale $L^*(x)$ i.e. $L^*(x) \geq L(x)$. L est une évaluation *heuristique* de L^*.

- On construit progressivement une liste de sommets S telle que pour tout $x' \in S$, la longueur $F(x')$ a déjà été calculée.
- A chaque étape, on ajoute à une autre liste $\overline{S}$ le sommet x tel que

$$\phi(x) = F(x) + L(x) = \min_{x' \in S} \left(F(x') + L(x')\right) \tag{6}$$

  La valeur $\phi(x)$ représente donc une valeur approchée de la longueur minimale pour aller de x_0 à x_F en passant par x.
- On utilise les successeurs de x pour continuer
- On arrête dès que le sommet x_F est rencontré

Evaluation heuristique $L(x)$: distance de x à x_F. Par exemple, dans le cas de la grille avec obstacle, l'heuristique L peut être une distance "à vol d'oiseau" (distance euclidienne) sans tenir compte des obstacles.

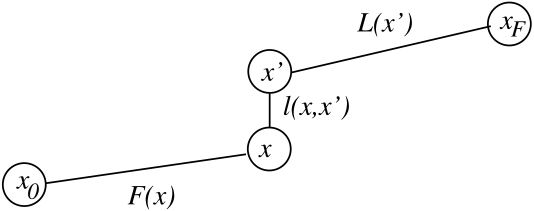
On construit deux listes :

S *liste ouverte* : elle contient les sommets (successeurs) **à examiner**.

$\overline{S}$ *liste fermée* : elle contient tous les sommets **déjà examinés**, qui appartiennent au chemin solution.

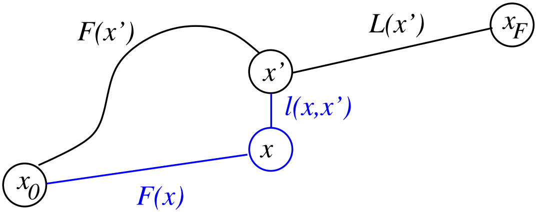
On commence par le sommet $x = x_0$.

1. On regarde tous les successeurs (voisins) x' de x
2. Si un sommet x' *n'a jamais été rencontré* (i.e. x' n'est ni dans la liste S, ni dans la liste $\overline{S}$), alors on l'ajoute dans S.



On calcule $\phi(x') = F(x) + l(x, x') + L(x')$, où $l(x, x')$ désigne la distance de x à son successeur x'.

3. Si un sommet x' *a déjà été rencontré* (i.e. x' est soit dans S ou soit dans $\overline{S}$) on dispose alors d'une évaluation précédente de la distance $\phi(x')$. On détermine la nouvelle évaluation de la distance associée à x' et si cette nouvelle distance est plus petite que $\phi(x')$ alors on met à jour la distance $\phi(x')$.



Si $\phi(x') = F(x') + L(x') > F(x) + l(x,x') + L(x')$ (nouvelle évaluation de la distance) alors
on met à jour $F(x') = F(x) + l(x,x')$ et $\phi(x') = F(x') + L(x')$.

De plus, si $x' \in \overline{S}$ alors on l'enlève de $\overline{S}$ et on l'ajoute à S pour que x' soit à nouveau un sommet à examiner.

4. On détermine le meilleur sommet x de toute la liste ouverte S en résolvant (6) (si $S = \emptyset$ alors arrêt, pas de solution).
5. On met x dans la liste fermée $\overline{S}$ et on le supprime de la liste S.
6. On itère avec le sommet courant x (retour en 1)

Algorithme A^*

$S = \{x_0\}$; $\overline{S} = \emptyset$; $F(x_0) = 0$.

Tant que $S \neq \emptyset$

- Déterminer $x \in S$ tq $\phi(x) = F(x) + L(x) = \min_{x' \in S}(\phi(x'))$
- $S = S \setminus \{x\}$; $\overline{S} = \overline{S} \cup \{x\}$
- Si $x = x_F$ alors arrêt (solution trouvée x)

  Pour tout $x' \in$ Successeur(x)

    Si $x' \notin S \cup \overline{S}$

      $S = S \cup \{x'\}$

      $F(x') = F(x) + l(x, x')$; $\phi(x') = F(x') + L(x')$

    Sinon Si $F(x) + l(x, x') + L(x') < \phi(x')$

      $F(x') = F(x) + l(x, x')$; $\phi(x') = F(x') + L(x')$

      Si $x' \in \overline{S}$ alors $S = S \cup \{x'\}$ et $\overline{S} = \overline{S} \setminus \{x'\}$

    Fin si

  Fin pour

Fin tant que

Le tableau *pred* permet de récupérer les sommets constituant le chemin trouvé.

```
x = x_F ;    i = 1
Tant que x ≠ x_0
    x = pred(x)
    c_opt(i) = x
    i = i + 1
Fin Tant que
```

Le tableau c_{opt} contient en sortie les sommets du chemin solution trouvé par l'algorithme A^* (en allant de x_F à x_0).

Un exemple.

Recherche d'un plus court chemin dans une grille avec obstacle. Les déplacements autorisés sont horizontaux et verticaux.

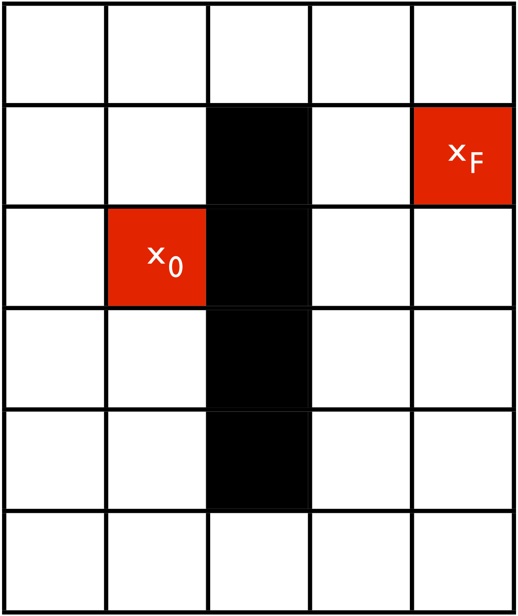
Choix de l'heuristique :

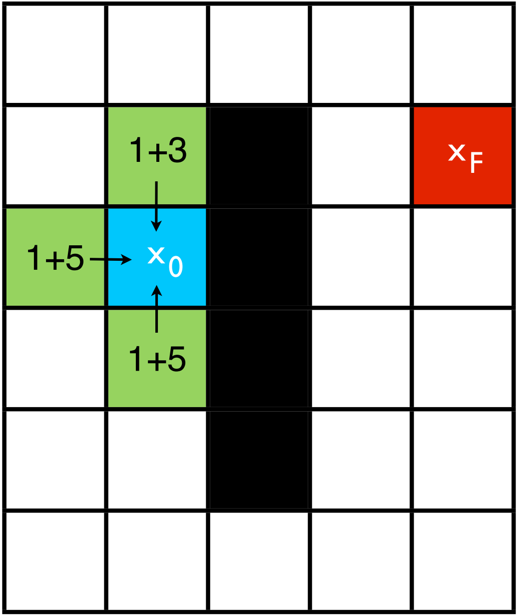
$$\begin{aligned} L(x) &= \text{distance de Manhattan du sommet } x \text{ au sommet d'arrivée } x_F. \\ &= \text{nombre de déplacements horizontaux et verticaux pour aller} \\ & \text{de } x \text{ à } x_F. \end{aligned}$$

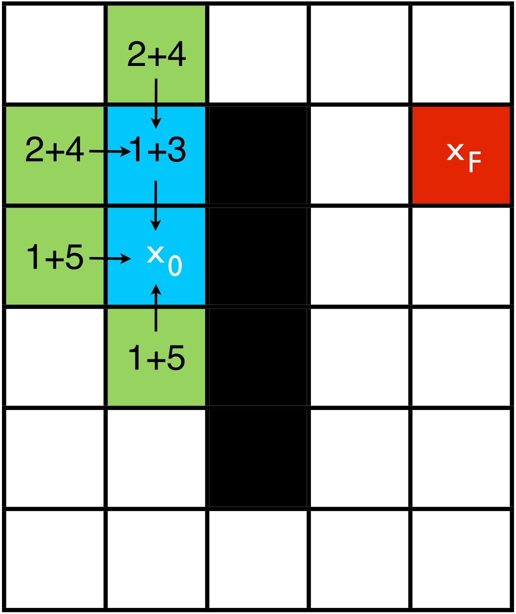
On désigne par $F(x)$ la distance (de Manhattan) de x_0 à x et on calcule $\phi(x) = F(x) + L(x)$ (valeurs numériques qui apparaissent dans les cases).

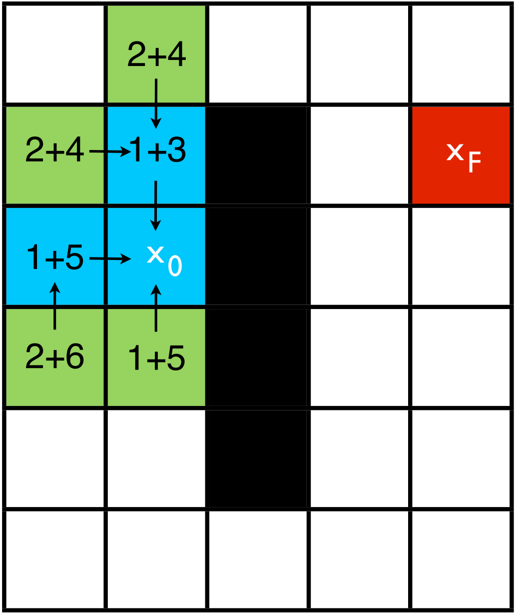
On notera

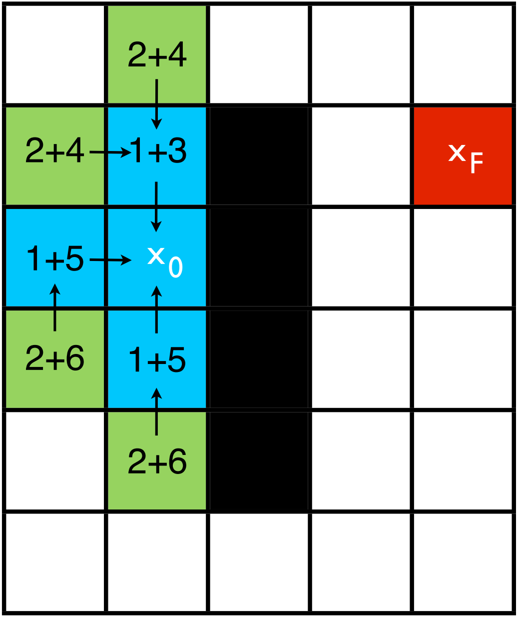
- en bleu les cases correspondant aux sommets déjà examinés (dans $\overline{S}$).
- en vert les cases correspondant aux sommets à examiner (dans S).

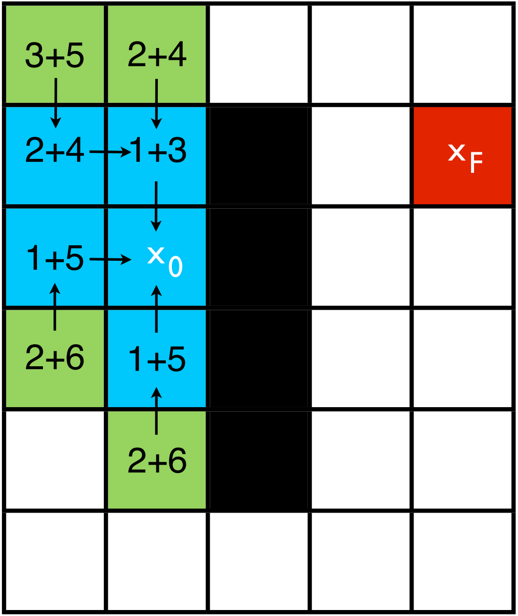

x F
x 0

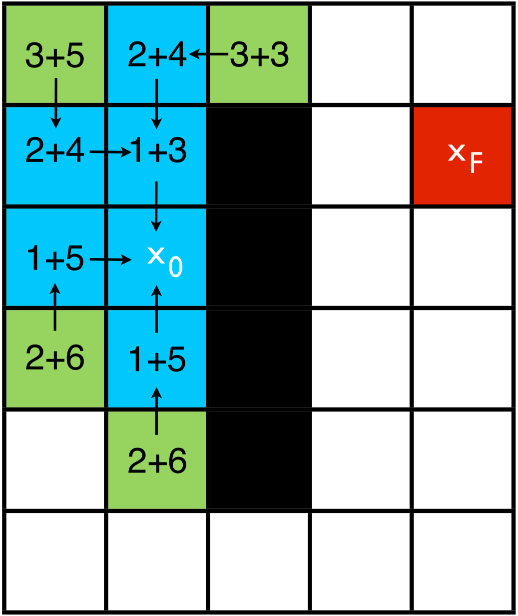

1+3
x_F
1+5
x_0
1+5

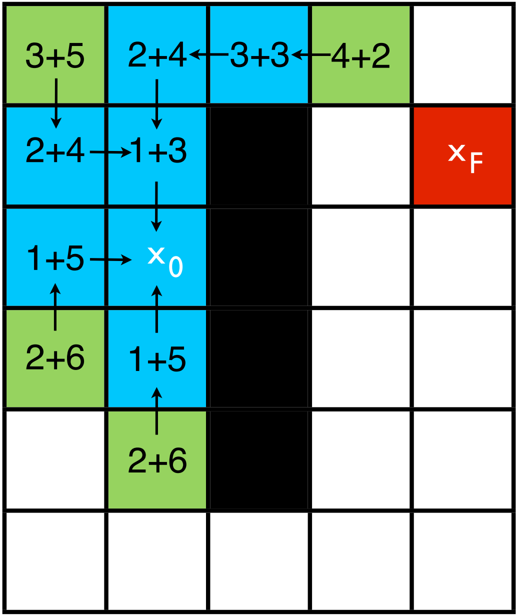

2+4
2+4
1+3
x F
1+5
x 0
1+5

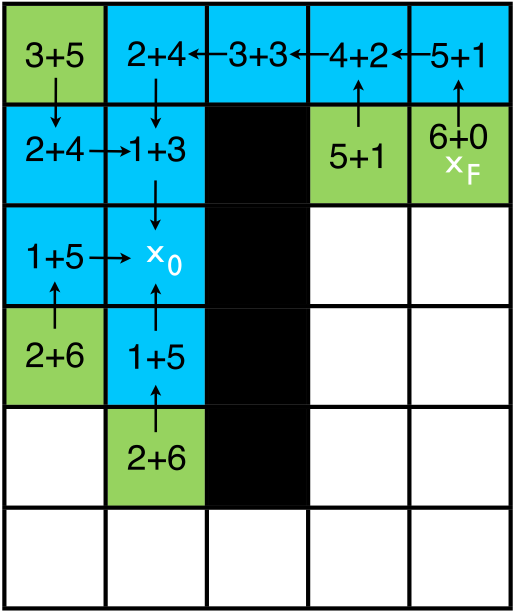

2+4
2+4
1+3
x F
1+5
x 0
2+6
1+5

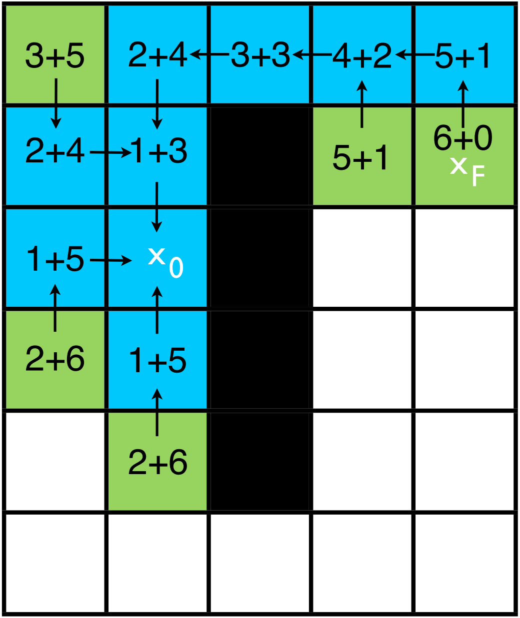

2+4
2+4
1+3
x_F
1+5
x_0
2+6
1+5
2+6

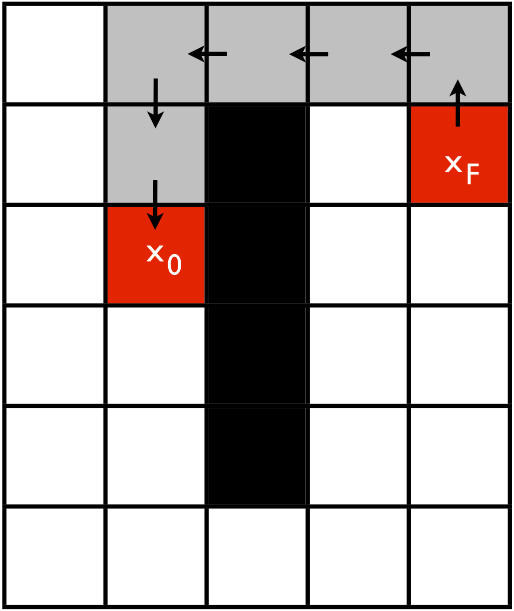

3+5
2+4
2+4
1+3
x F
1+5
x 0
2+6
1+5
2+6


3+5
2+4
3+3
2+4
1+3
x F
1+5
x 0
2+6
1+5
2+6


3+5
2+4
3+3
4+2
2+4
1+3
x_F
1+5
x_0
2+6
1+5
2+6


3+5
2+4
3+3
4+2
5+1
2+4
1+3
5+1
6+0
x_F
1+5
x_0
2+6
1+5
2+6


3+5
2+4
3+3
4+2
5+1
2+4
1+3
5+1
6+0
x F
1+5
x 0
2+6
1+5
2+6


x F
x 0

2. Propriétés de A^*

On note $l(x, y)$ la distance de x à son successeur y et $L^*(x)$ est la distance minimale de x à x_F.

- *Heuristique consistante* : L est une heuristique *consistante* si

$$L(x) \leq L(y) + l(x, y) \quad \forall x, \forall y \text{ successeur de } x.$$

- *Heuristique admissible* : L est une heuristique *admissible* si

$$L(x) \leq L^*(x) \quad \forall x.$$

Proposition 1.

- Si L est admissible alors l'algorithme A^* trouvera toujours le chemin optimal.
- Si L est consistante alors
  - L est admissible
  - l'algorithme A^* a une complexité linéaire

Sous les hypothèses de la Proposition 1, A^* est a priori un algorithme *heuristique* (avec une solution optimale approchée), mais il fournit en fait la solution *optimale*. Il est très efficace grâce à sa complexité linéaire.

Extension 3D : recherche d'un chemin minimal sur une surface.