Recherche dans les arbres de jeu

Plan

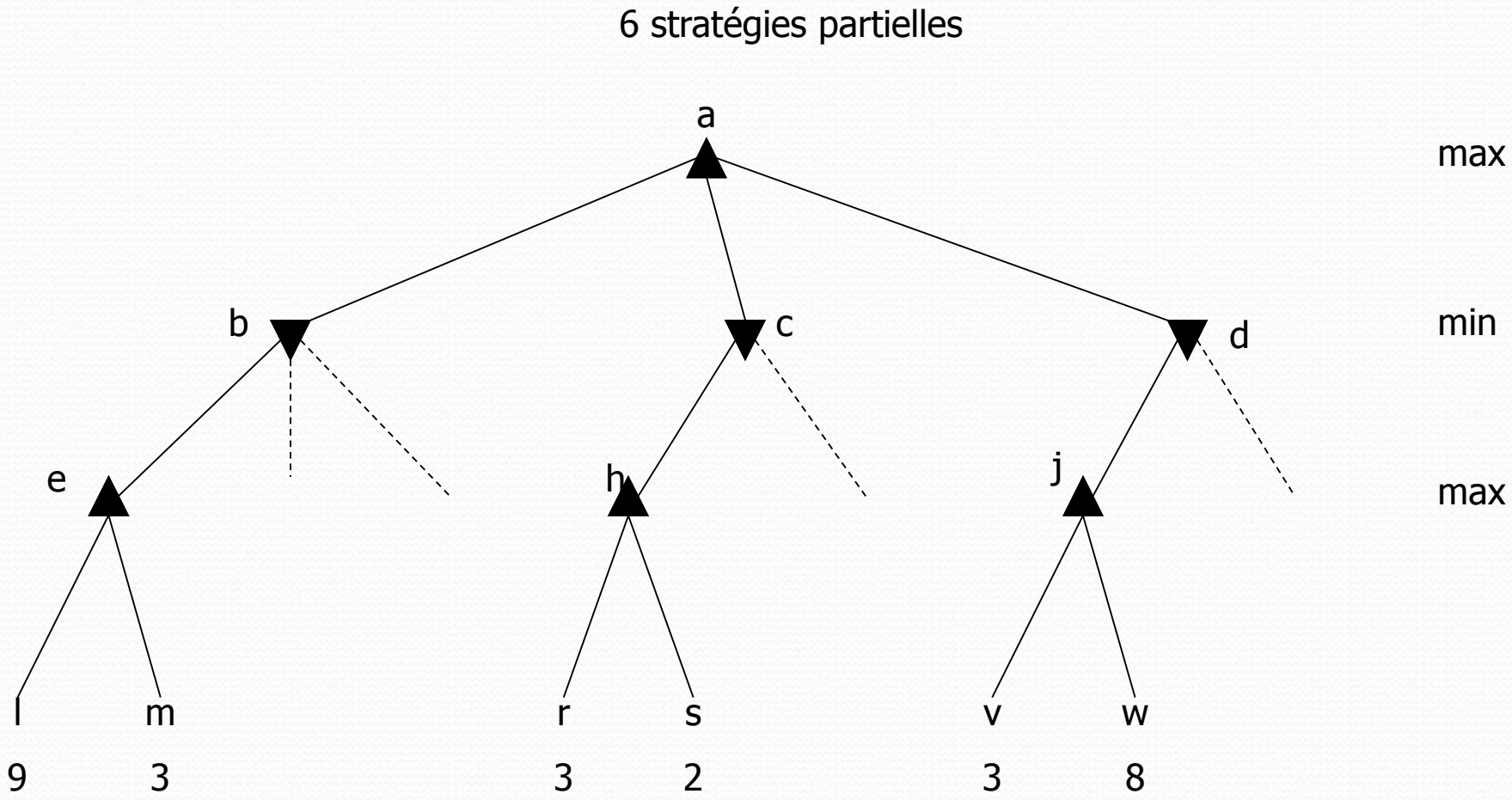
- Introduction
- Algorithmes de recherche du meilleur coup
  - Principe
  - Minimax
  - Alpha-beta
  - SSS*

L'algorithme SSS* (State Space Search)

- Une stratégie complète pour J1 est un sous-arbre de l'arbre de jeu A qui :
  - contient la racine de A
  - dont chaque noeud Max a exactement un fils
  - dont chaque noeud Min a tous ses fils

- Une stratégie partielle pour J1 est un sous-arbre de l'arbre de jeu A qui :
  - contient la racine de A
  - dont chaque noeud Max a au plus un fils (0 à 1)
  - dont chaque nœud Min a au plus tous ses fils (0 à k)

SSS* : stratégies partielles

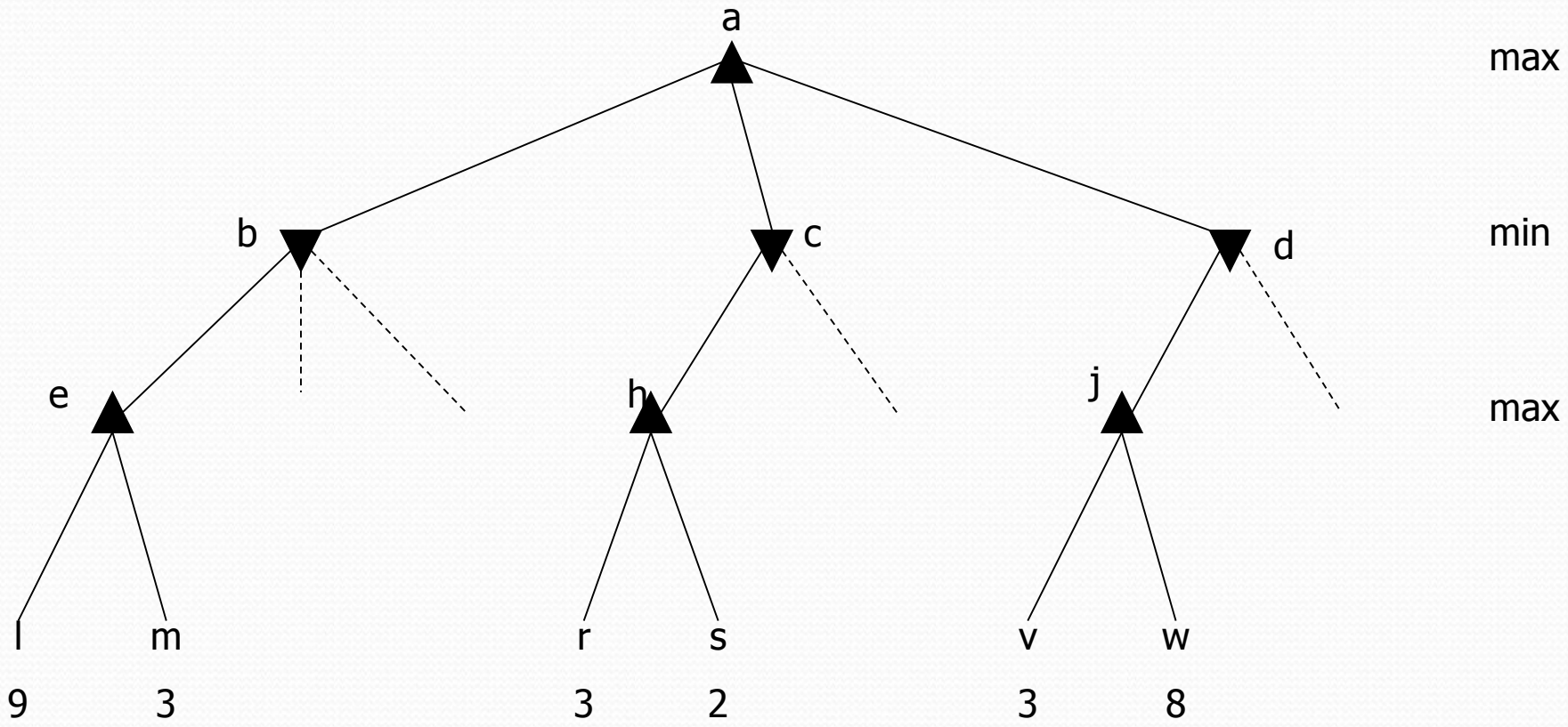


L'algorithme SSS*

- Une stratégie partielle S représente implicitement toutes les stratégies complètes Cs auxquelles on aboutit en développant S
- La valeur d'une stratégie complète est le minimum des valeurs des feuilles (car 1 seul successeur pour max)
  - Une stratégie complète est gagnante si sa valeur est la valeur max (+inf)
- La valeur d'une stratégie partielle S donne une borne supérieure pour toutes les stratégies complètes Cs qu'elle représente (implicitement)

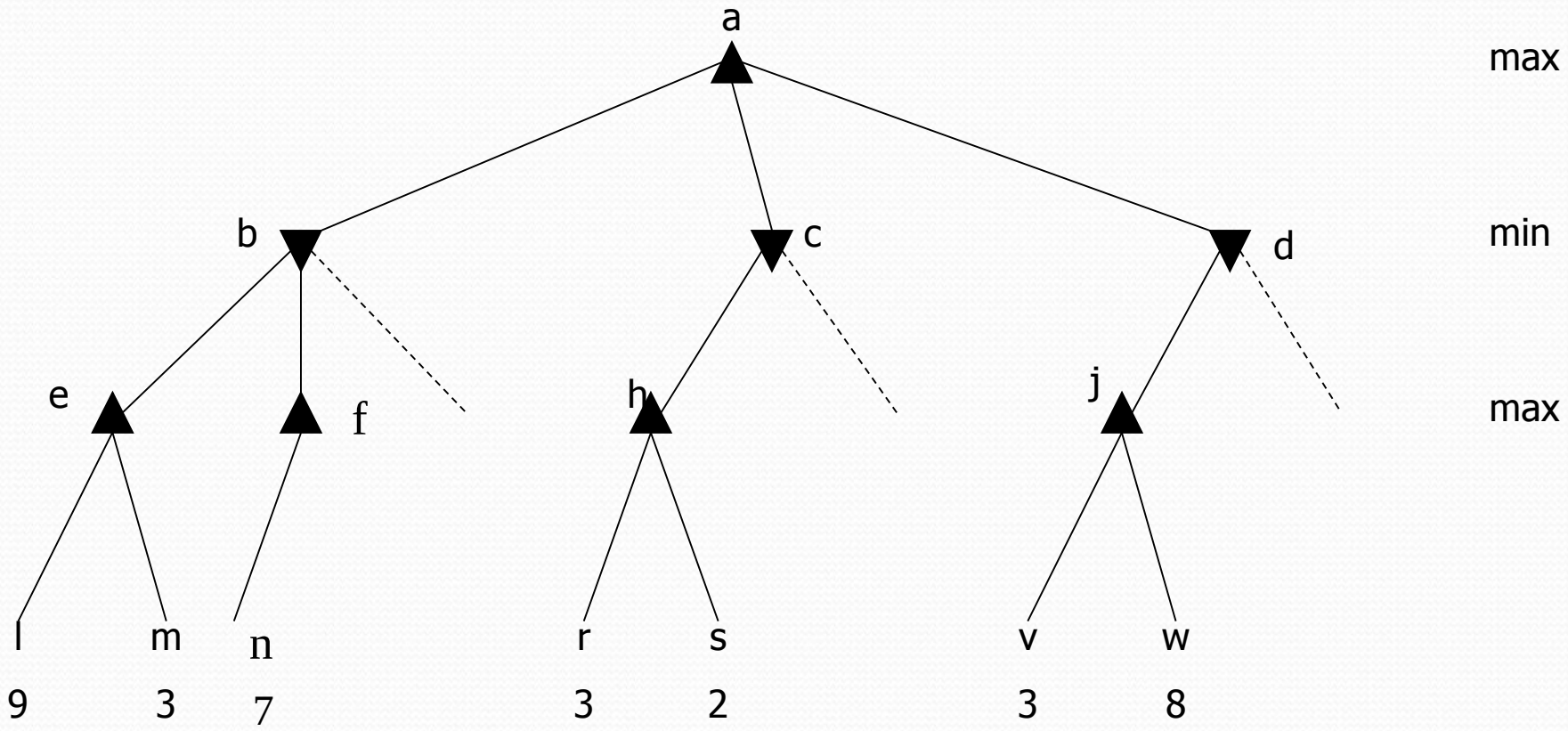
SSS* : stratégies partielles

6 stratégies partielles de valeurs 9, 3, 3, 2, 3, 8
(bornes sup des stratégies complètes qu'elles représentent)



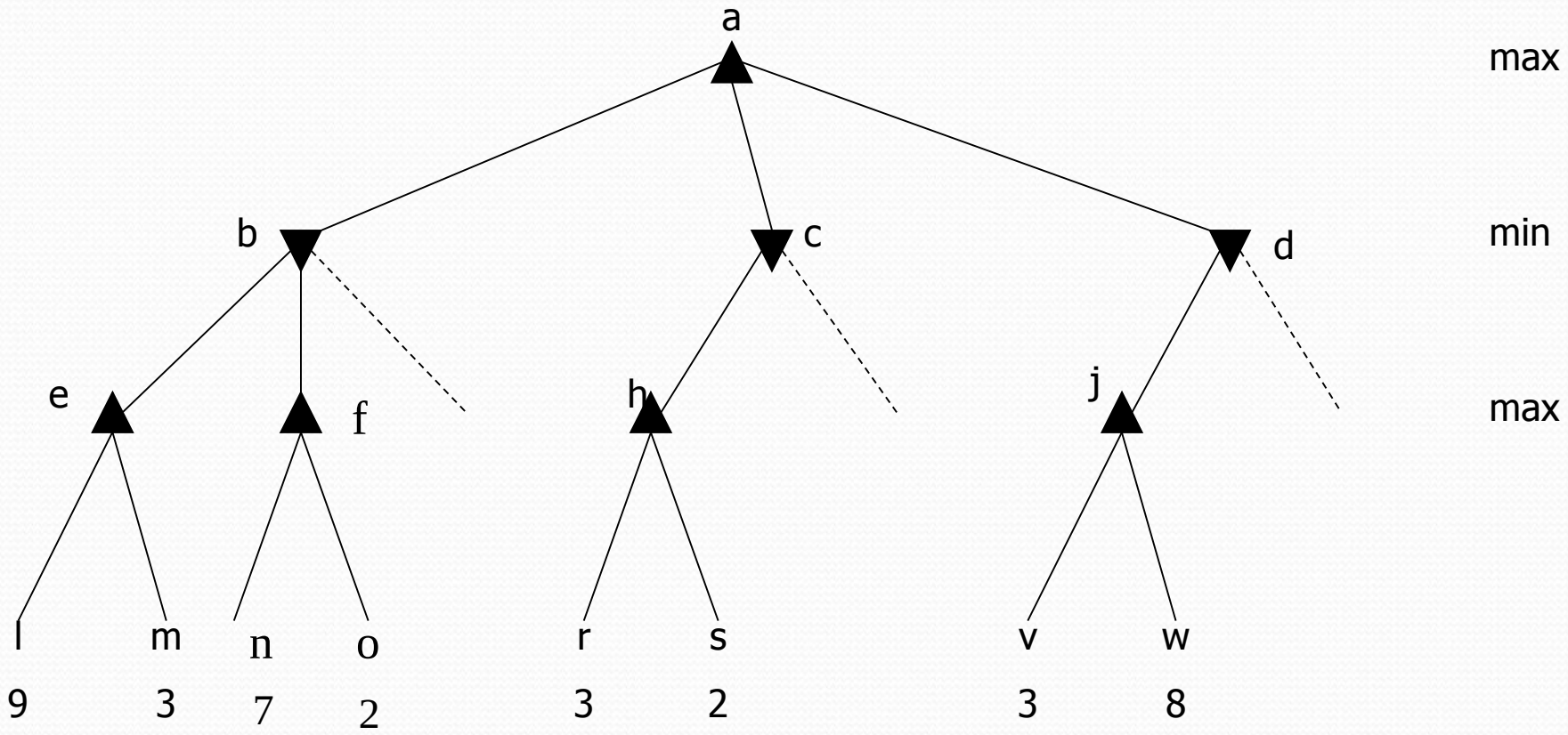
SSS* : stratégies partielles

~~6 stratégies partielles de valeurs **9, 3**, 3, 2, 3, 8~~
6 stratégies partielles de valeurs **7, 3**, 3, 2, 3, 8



SSS* : stratégies partielles

~~6 stratégies partielles de valeurs **7, 3**, 3, 2, 3, 8~~
8 stratégies partielles de valeurs **7, 2, 3, 2**, 3, 2, 3, 8



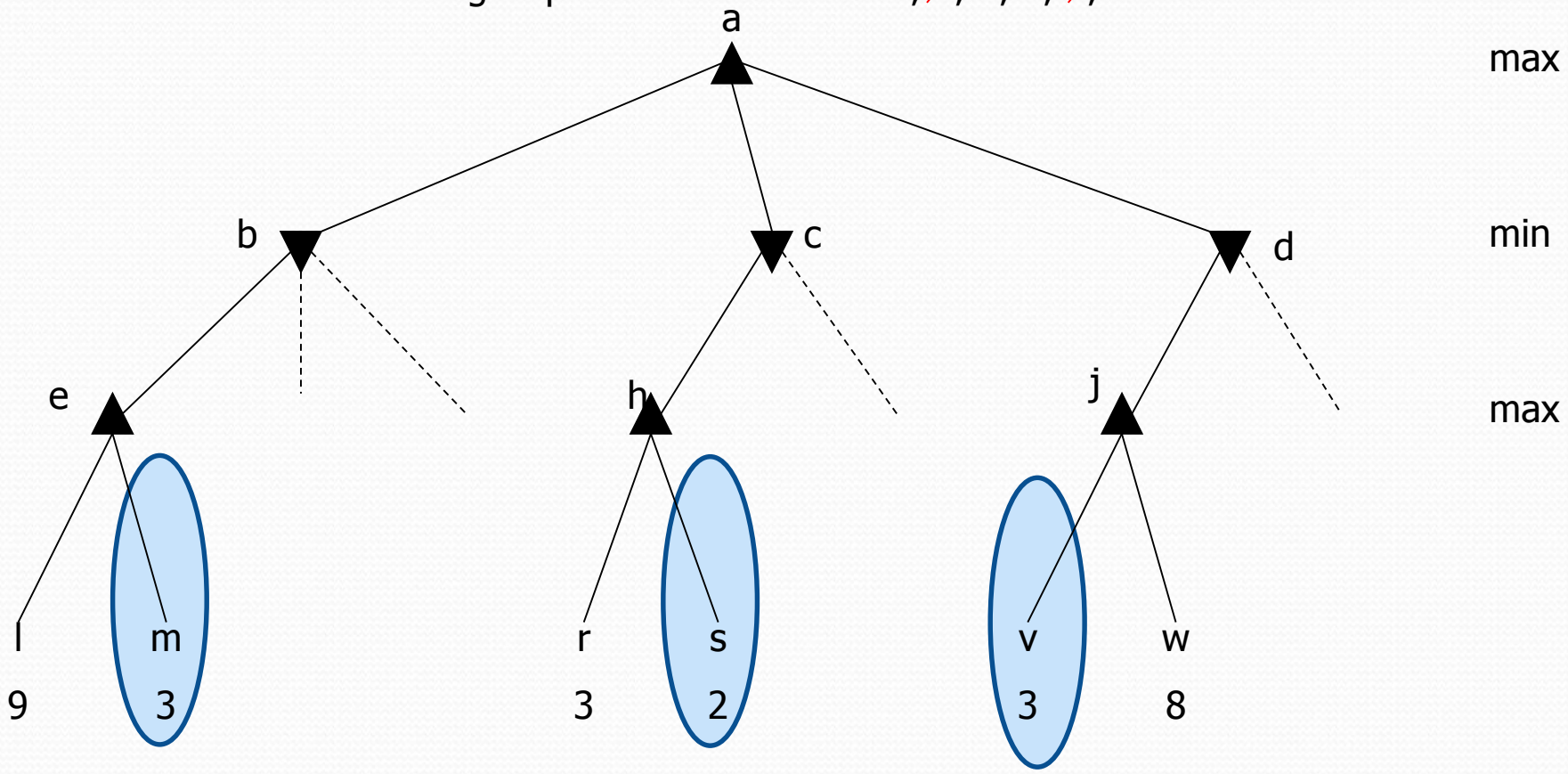
L'algorithme SSS*

- Une stratégie partielle S représente implicitement toutes les stratégies complètes Cs auxquelles on aboutit en développant S
- La valeur d'une stratégie complète est le minimum des valeurs des feuilles (car 1 seul successeur pour max)
- La valeur d'une stratégie partielle S donne une borne supérieure pour toutes ses stratégies complètes Cs qu'elle représente (implicitement)
- Une fois qu'une stratégie complète a été trouvée à partir d'un nœud x, (nœud Max) on ne considère plus les fils de x de valeur moindre.

SSS* : stratégies partielles

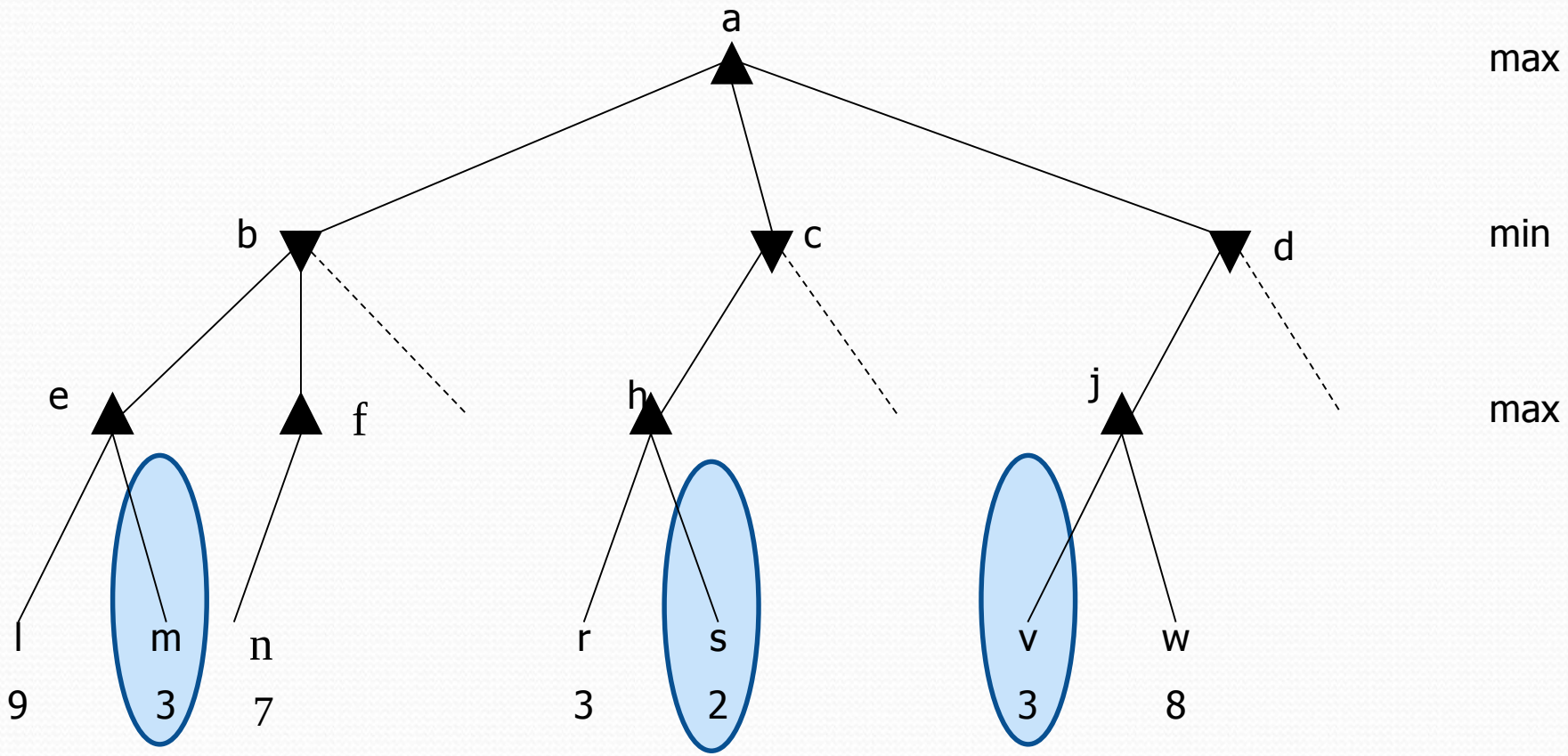
Une fois qu'une stratégie complète a été trouvée à partir d'un nœud x, (nœud Max), on ne considère plus les fils de x de valeur moindre. Ici, e, h, j sont concernés

~~6 stratégies partielles de valeurs 9, 3, 3, 2, 3, 8~~
3 stratégies partielles de valeurs 9, ~~3~~, 3, ~~2~~, ~~3~~, 8



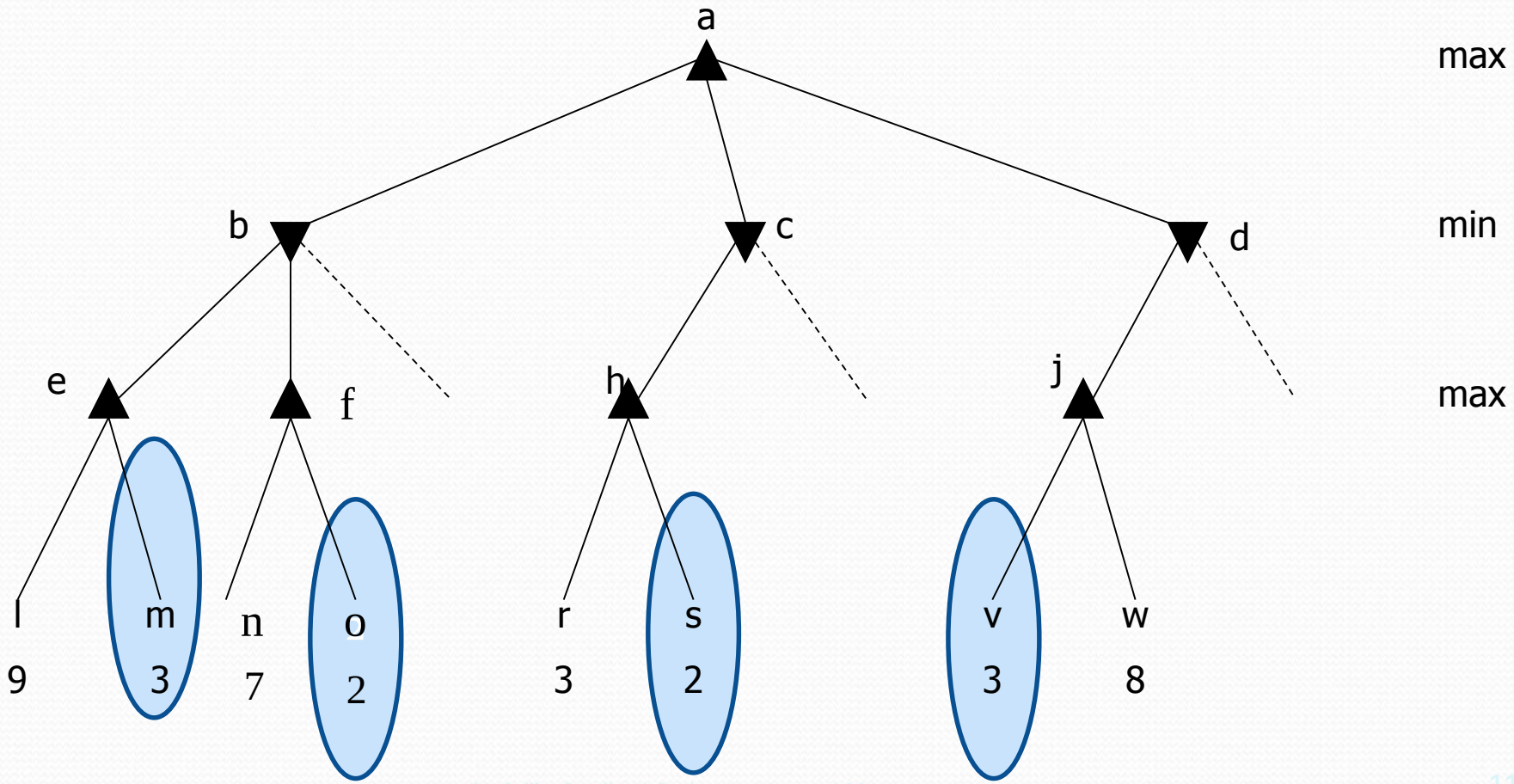
SSS* : stratégies partielles

~~3 stratégies partielles de valeurs **9,** 3, 8~~
3 stratégies partielles de valeurs **7,** 3, 8



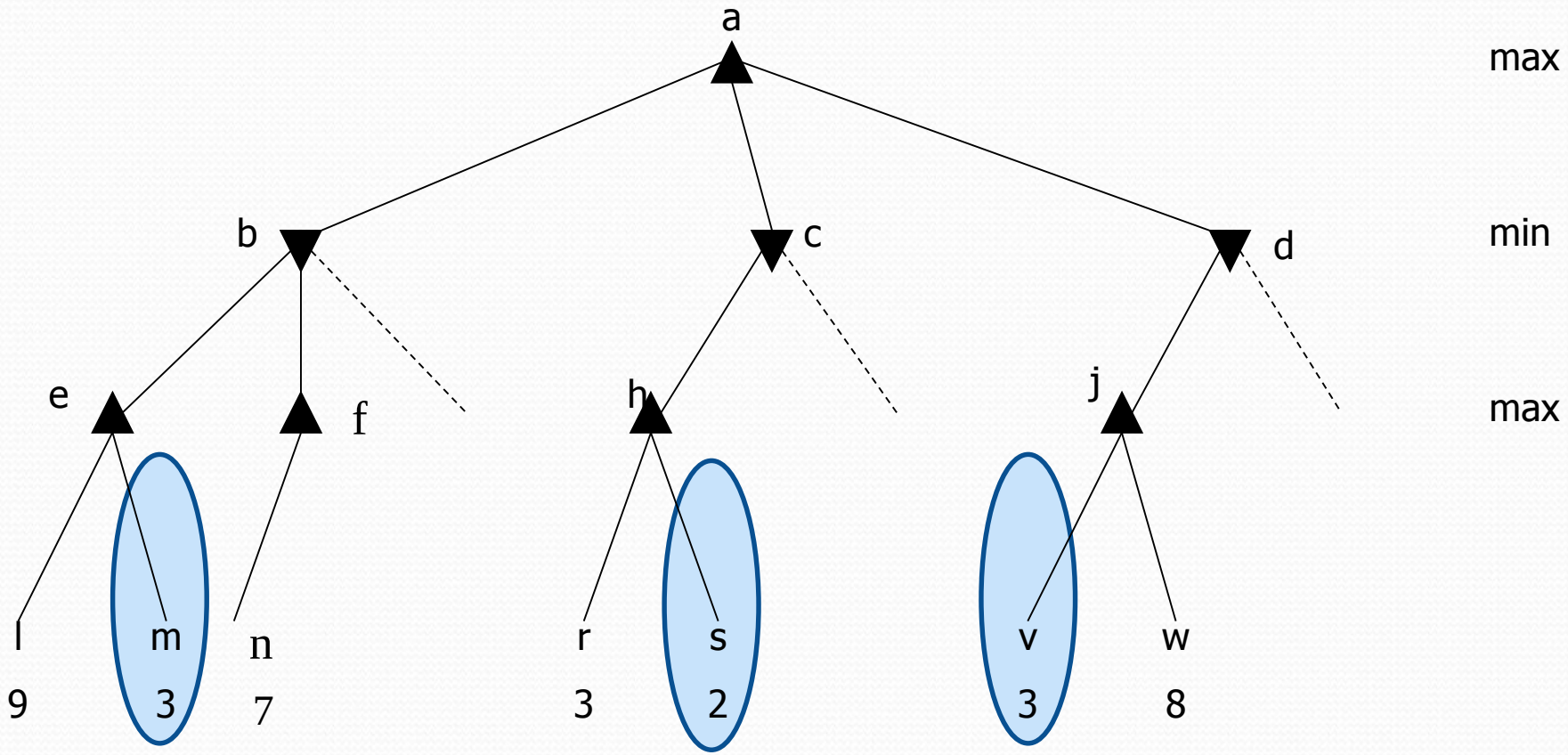
SSS* : stratégies partielles

~~3 stratégies partielles de valeurs **7,** 3, 8~~
~~4 stratégies partielles de valeurs **7, 2**, 3, 8~~
3 stratégies partielles de valeurs **7,** 3, 8



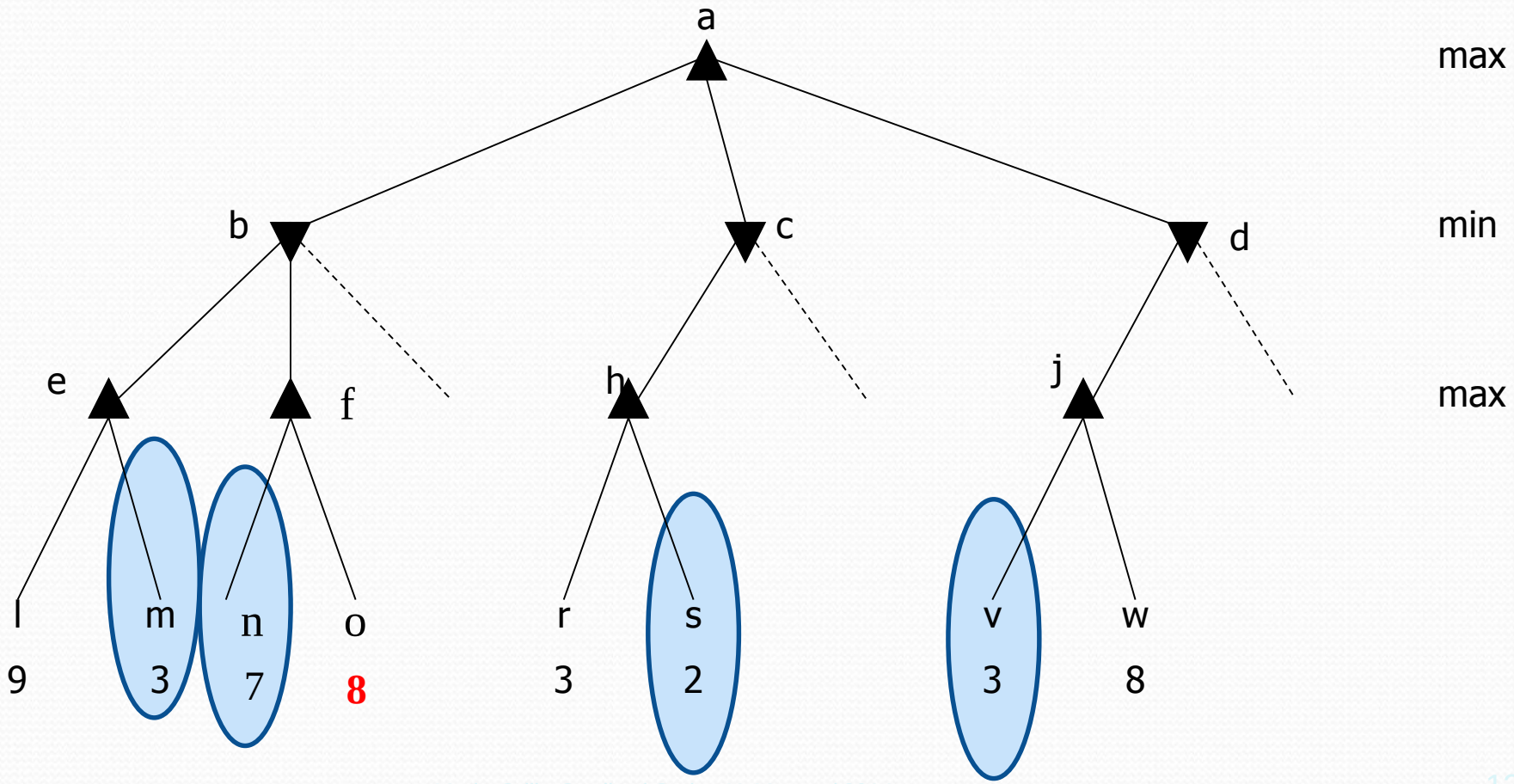
SSS* : stratégies partielles

~~3 stratégies partielles de valeurs **9,** 3, 8~~
3 stratégies partielles de valeurs **7,** 3, 8



SSS* : stratégies partielles

~~3 stratégies partielles de valeurs **7,** 3, 8~~
3 stratégies partielles de valeurs **8,** 3, 8



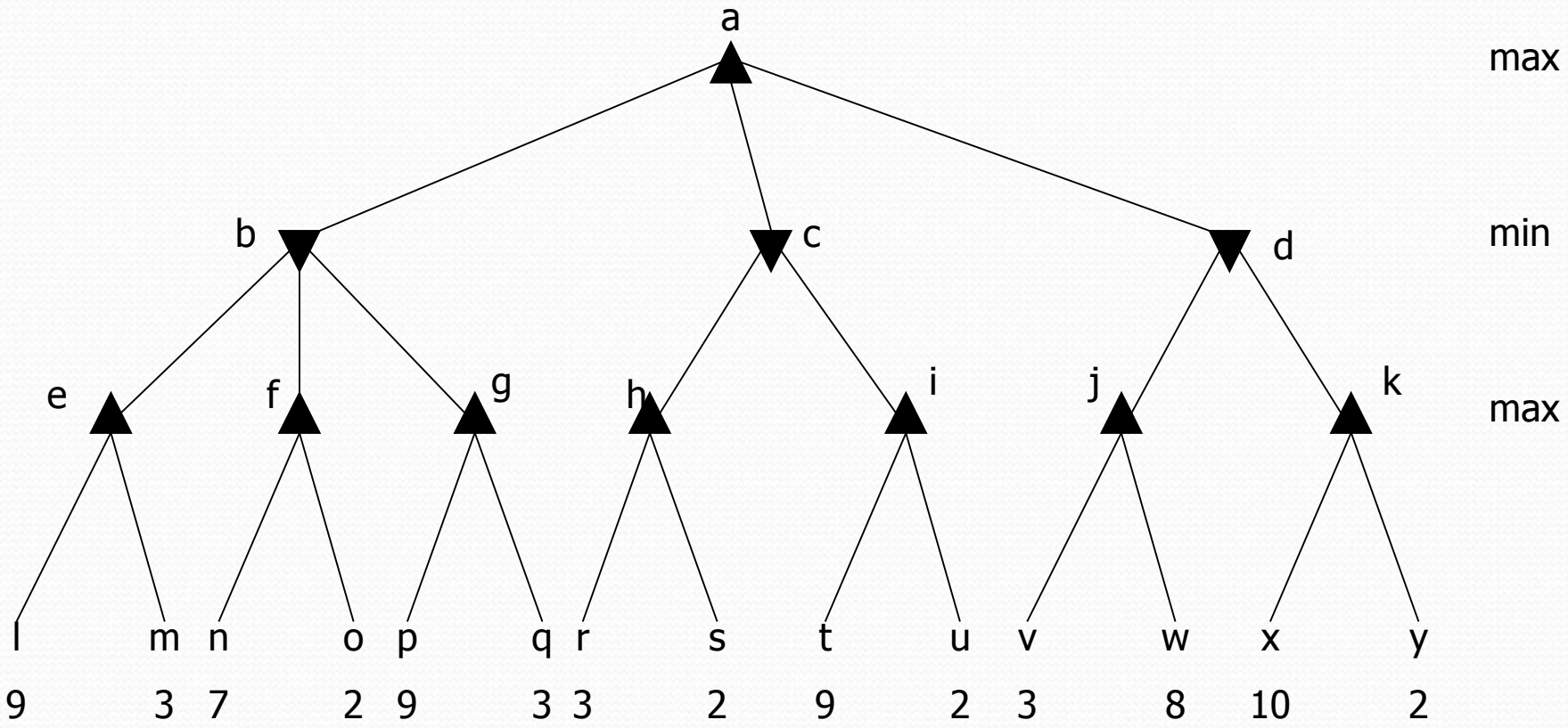
L'algorithme SSS*

- Une stratégie partielle S représente implicitement toutes les stratégies complètes Cs auxquelles on aboutit en développant S
- La valeur d'une stratégie complète est le minimum des valeurs des feuilles (car 1 seul successeur pour max)
- La valeur d'une stratégie partielle S donne une borne supérieure pour toutes ses stratégies complètes Cs qu'elle représente (implicitement)
- **Idée**: On cherche a compléter les stratégies partielles suivant leur valeur courante et en commençant par les « meilleures » et donc:
- Une fois qu'une stratégie complète a été trouvée à partir d'un nœud x, (nœud Max) on ne considère plus les autres fils de x car la stratégie est optimale pour x. Pour un nœud Min, la stratégie optimale n'est trouvée qu'après résolution de tous ses fils.

L'algorithme SSS*

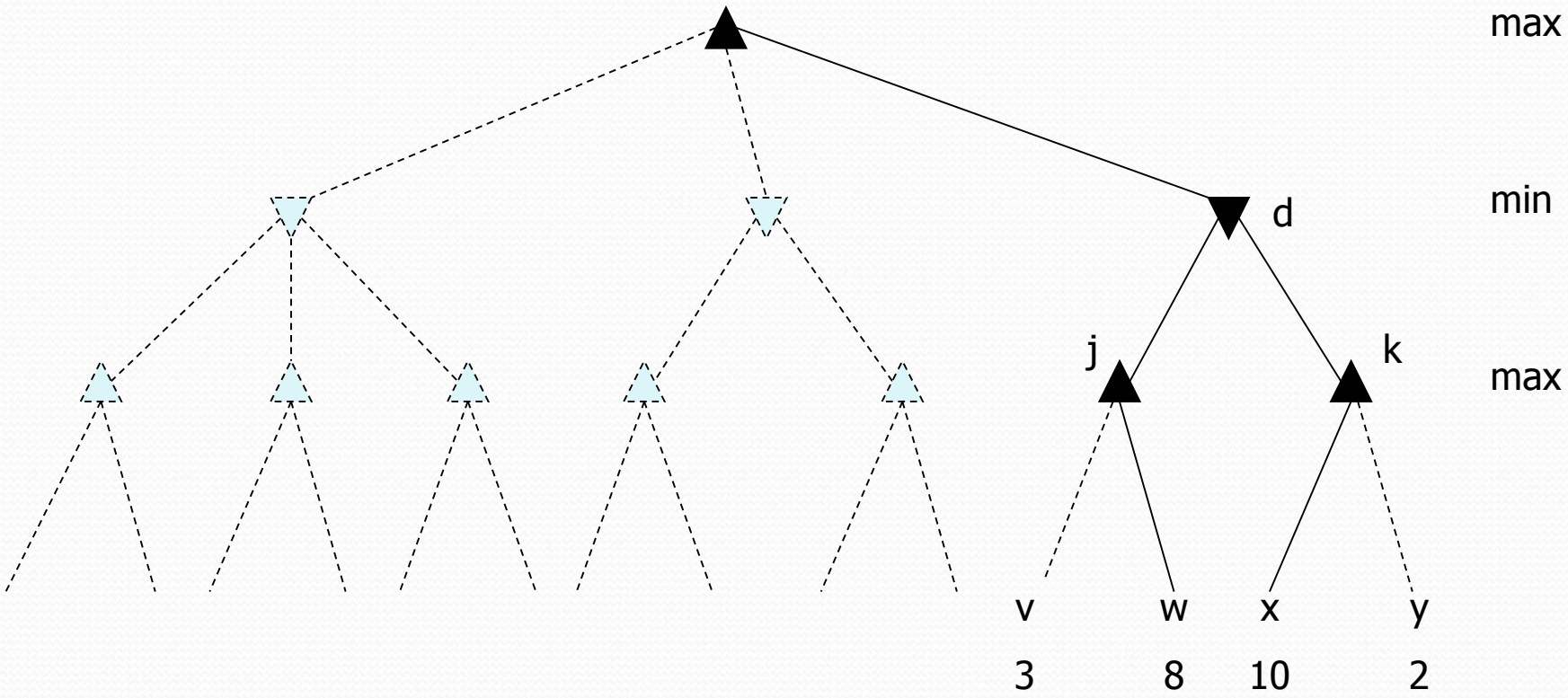
- SSS* est une recherche du meilleur d'abord
- SSS* maintient une liste OUVERT avec les descripteurs des nœuds
- Les descripteurs sont rangés dans l'ordre de leur valeur *h*
- Un descripteur (n,s,h) contient
  - un identifiant de nœud n
  - Un état s qui appartient à l'ensemble {vivant, resolu}
    - VIVANT : *n* est toujours non développé et h est une borne supérieure de la vraie valeur *h*
    - RESOLU : *h* est la vraie valeur
  - Une valeur *h*

L'algorithme SSS*



SSS* deux phases de recherche – bottom-up

La meilleure est la suivante et vaut 8 :



L'algorithme SSS*

Empiler/dépiler :
dans une file de priorité, selon la valeur h

```
/*On initialise OUVERT à (racine,VIVANT,∞)*/

SSS*(OUVERT)

Tant que premier(OUVERT) n'est pas = (racine,
          RESOLU,h) alors
          traiter(OUVERT)
ftant que
```

- La stratégie générale est de type « meilleur d'abord, de gauche à droite si égalité » :
  - On empile tous les fils d'un nœud Max (autant de stratégies partielles que de fils)
  - On empile un seul fils d'un nœud Min (une stratégie partielle)

```
Traiter(OUVERT)
<nœud,état,h> ← dépiler ()
si état == VIVANT alors
        si feuille(nœud) alors
                empiler(<noeud, RESOLU, min(eval(noeud),h)>)
        sinon si nœud-MIN(nœud) alors
                empiler (<fils(nœud), VIVANT, h>)
        sinon si nœud-MAX(nœud) alors
                pour tout les nœuds fils
                        empiler (<fils(nœud), VIVANT, h>)
                fpour
        fsi
sinon si état == RESOLU alors
        si racine(nœud) alors
                retourner h
        sinon si nœud-MIN(nœud) alors
                empiler (<père(noeud), RESOLU, h>)
                supprimer tous les successeurs de père(nœud)
        sinon si nœud-MAX(nœud) alors
                si il existe un frère non examinés alors
                        empiler (<frère(noeud), VIVANT, h>)
                sinon
                        empiler (<père(noeud), RESOLU, h>)
                fsi
        fsi
fsi
```

L'algorithme SSS*

```
/*On initialise OUVERT à (racine,VIVANT,∞)*/

SSS*(OUVERT)

Tant que premier(OUVERT) n'est pas  = (racine,
          RESOLU,h) alors
          traiter(OUVERT)
ftant que
```

- La stratégie générale est de type « meilleur d'abord, de gauche à droite si égalité »
- Elagage quand on dépile un nœud Min : le père est Max, et résolu, donc les frères peuvent être élagués

```
Traiter(OUVERT)
<nœud,état,h> ← dépiler ()
si état == VIVANT  alors
          si feuille(nœud) alors
                    empiler(<noeud, RESOLU, min(eval(noeud),h)>)
          sinon si nœud-MIN(nœud) alors
                    empiler (<fils(nœud), VIVANT, h>)
          sinon si nœud-MAX(nœud) alors
                    pour tout les nœuds fils
                              empiler (<fils(nœud), VIVANT, h>)
                    fpour
          fsi
sinon si état == RESOLU alors
          si racine(nœud) alors
                    retourner h
          sinon si nœud-MIN(nœud) alors
                    empiler (<père(noeud), RESOLU, h>)
                    supprimer tous les successeurs de père(nœud)
          sinon si nœud-MAX(nœud) alors
                    si il existe un frère non examinés alors
                              empiler (<frère(noeud), VIVANT, h>)
                    sinon
                              empiler (<père(noeud), RESOLU, h>)
                    fsi
          fsi
fsi
```

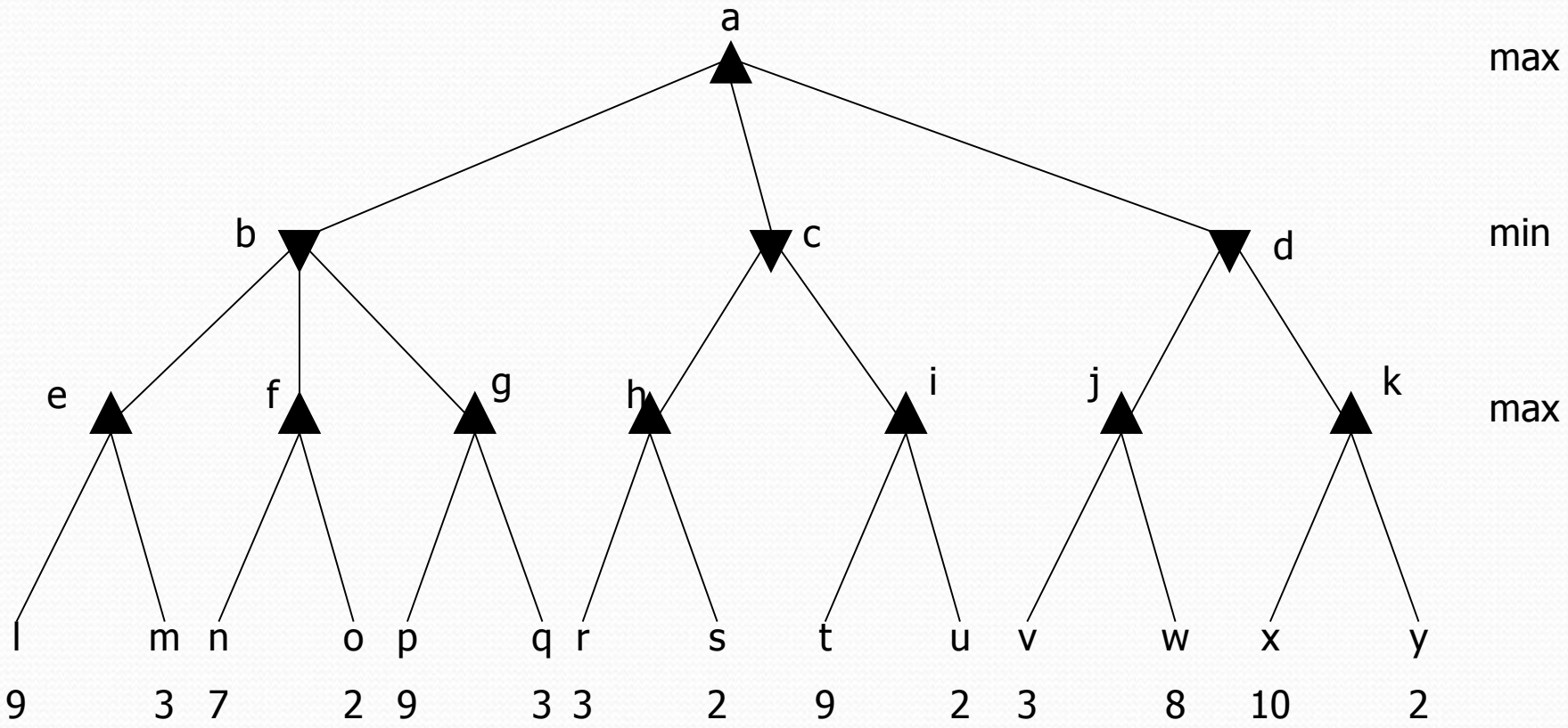
L'algorithme SSS*

```
/*On initialise OUVERT à (racine,VIVANT,∞)*/

SSS*(OUVERT)

Tant que premier(OUVERT) n'est pas = (racine,
          RESOLU,h) alors
          traiter(OUVERT)
ftant que
```

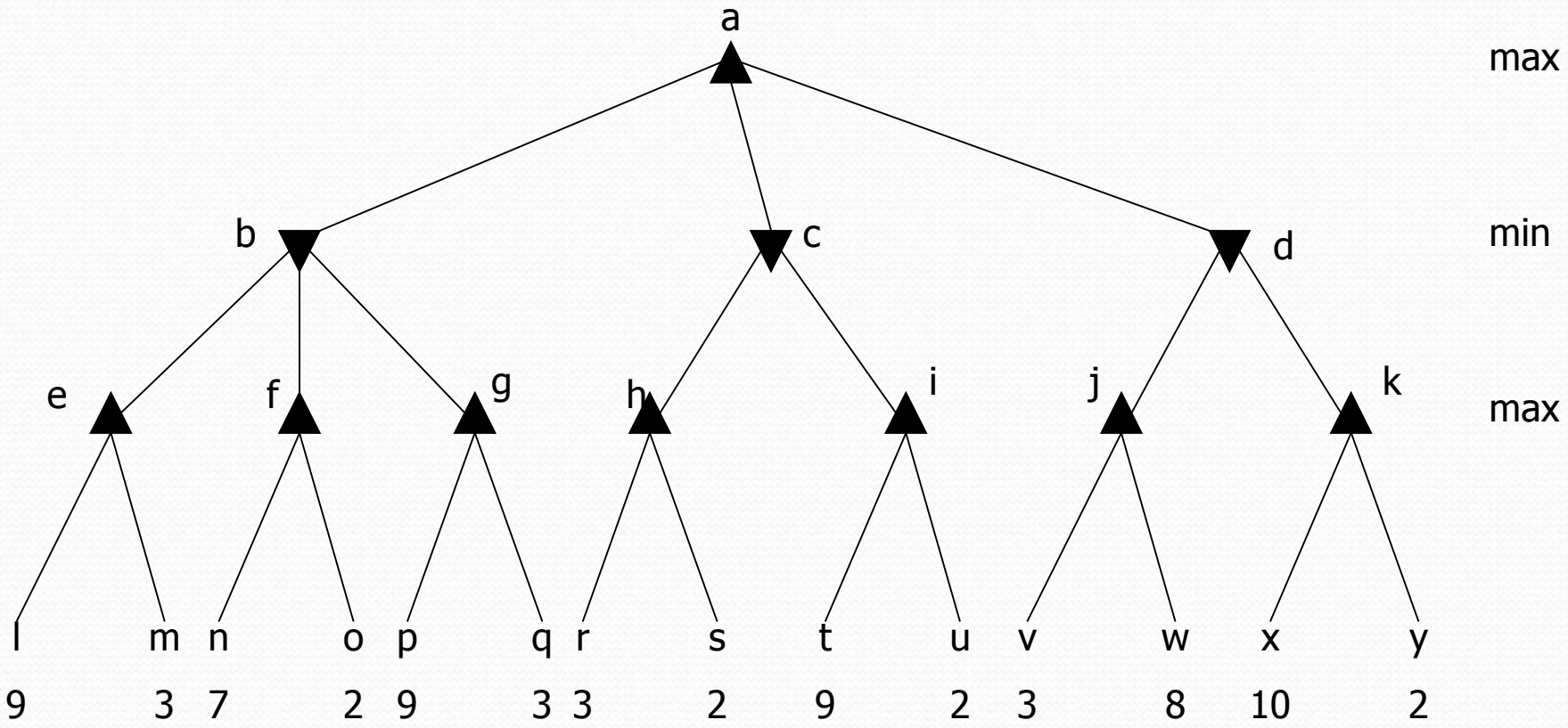
- La stratégie générale est de type « meilleur d'abord, de gauche à droite si égalité »
- Elagage quand on dépile un nœud Min : le père est Max, et résolu, donc les frères peuvent être élagués
- Quand on dépile un nœud Max, le père est Min, et les frères non examinés doivent être empilés

```
Traiter(OUVERT)
<nœud,état,h> ← dépiler ()
si état == VIVANT alors
          si feuille(nœud) alors
                    empiler(<noeud, RESOLU, min(eval(noeud),h)>)
          sinon si nœud-MIN(nœud) alors
                    empiler (<fils(nœud), VIVANT, h>)
          sinon si nœud-MAX(nœud) alors
                    pour tout les nœuds fils
                              empiler (<fils(nœud), VIVANT, h>)
                    fpour
          fsi
sinon si état == RESOLU alors
          si racine(nœud) alors
                    retourner h
          sinon si nœud-MIN(nœud) alors
                    empiler (<père(noeud), RESOLU, h>)
                    supprimer tous les successeurs de père(nœud)
          sinon si nœud-MAX(nœud) alors
                    si il existe un frère non examinés alors
                              empiler (<frère(noeud), VIVANT, h>)
                    sinon
                              empiler (<père(noeud), RESOLU, h>)
                    fsi
          fsi
fsi
```

L’algorithme SSS*

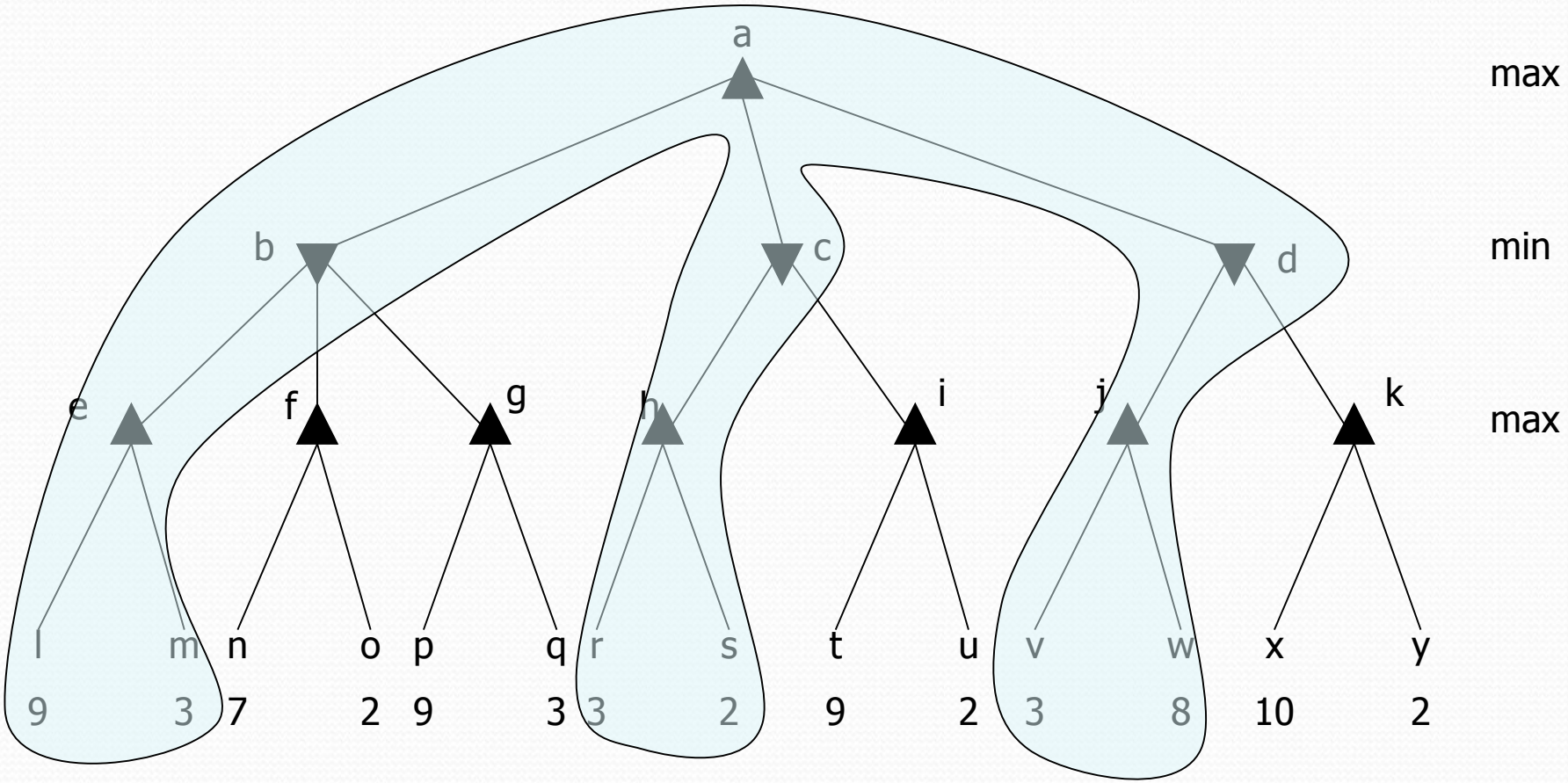


L'algorithme SSS*



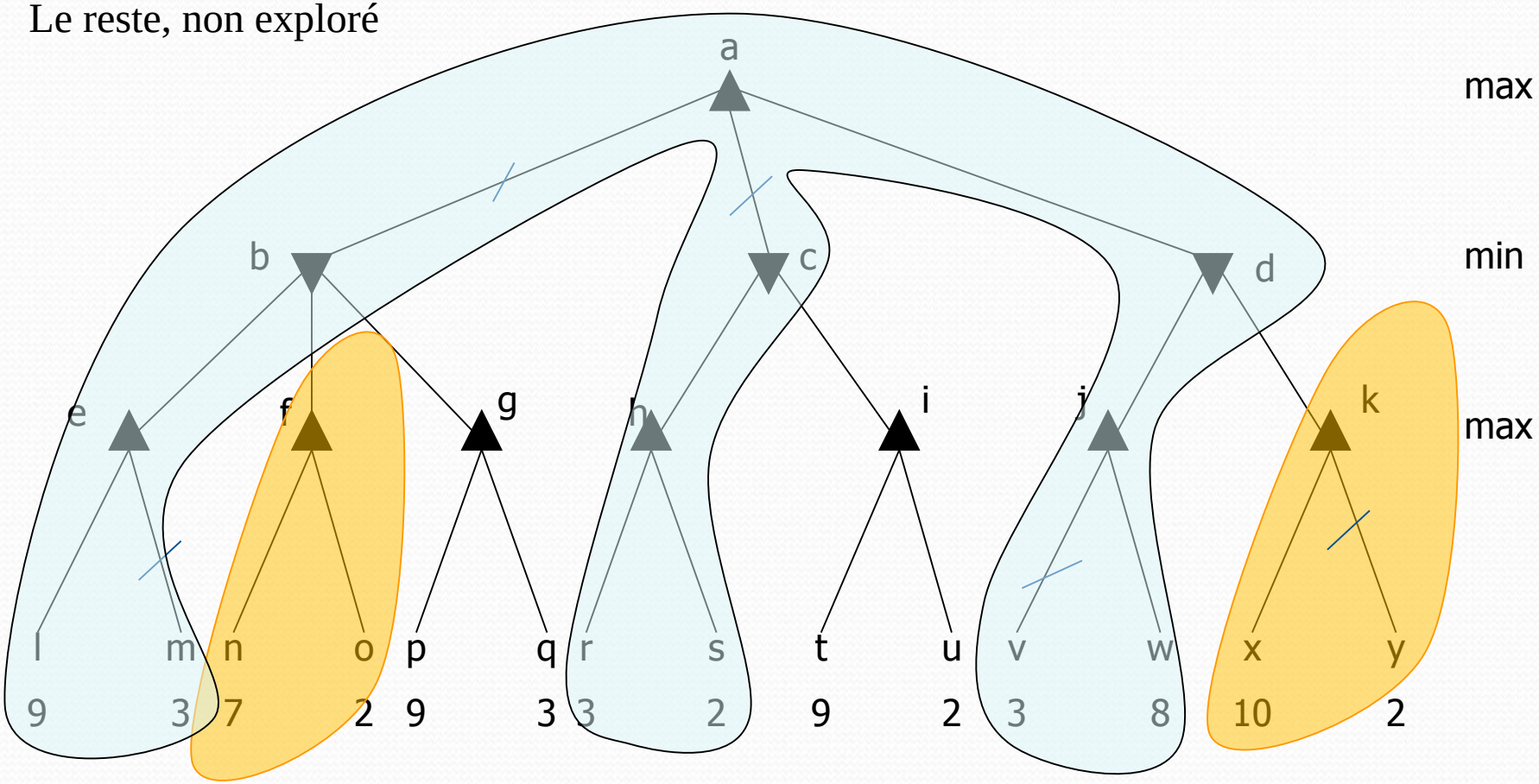
L'algorithme SSS*

Phase top-down : six stratégies partielles évaluées et ordonnées



L'algorithme SSS*

En bleu clair, première étape (top-down)
En orange, étape bottom-up
Le reste, non exploré



L'algorithme SSS*

Au départ, tous les noeuds sont vivants et ont la même valeur

étape	0	1	2	…	…
Ouvert	{a/∞}	{b/∞, c/∞, d/∞}	{e/∞, c/∞, d/∞}	{c/∞, d/∞, l/9, m/3}	{l/9, w/8, m/3, r/3, v/3, s/2}

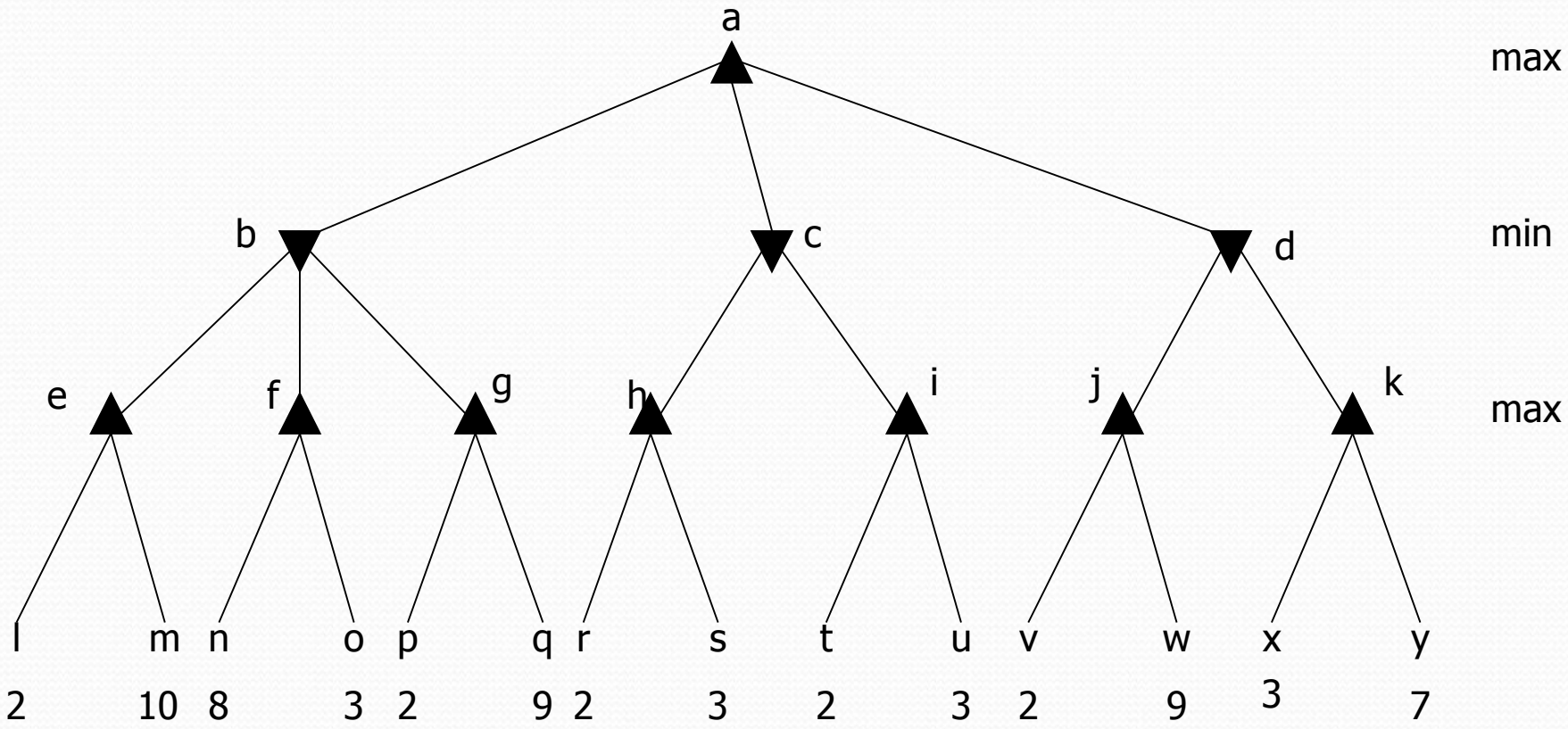
Arrivé aux feuilles, la résolution commence avec les vrais valeurs de feuille

étape		…	On change de branche	…	…	…	…
Ouvert	{e/9, w/8, ~~m/3,~~ r/3,v/3, s/2}	{n/9,o/9, w/8, r/3, v/3,s/2}	{w/8,n/7, r/3, v/3, s/2, o/2}	{k/8, n/7, r/3,s/2, o/2}	{x/8, y/8, n/7, r/3,s/2 ,o/2}	{k/8, ~~y/8,~~ n/7, r/3,s/2 ,o/2}	{d/8, n/7, r/3,s/2 ,o/2}

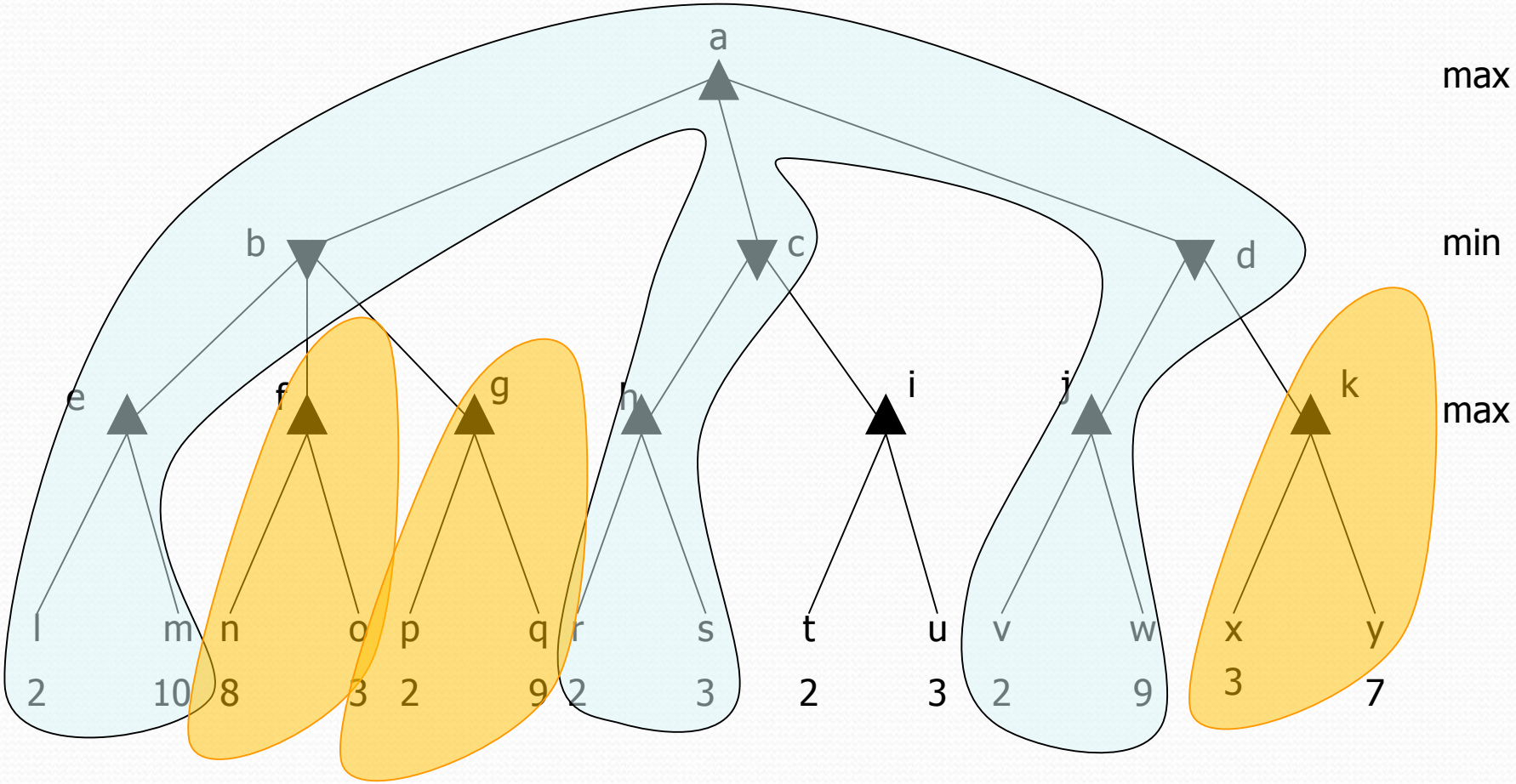
Sous arbre de stratégie optimale -> {a,d,j,k,v,w,x,y} de valeur 8 (on est sûr d'arriver au moins à 8 en prenant la branche de d)

On empile de gauche à droite

L'algorithme SSS* : à faire

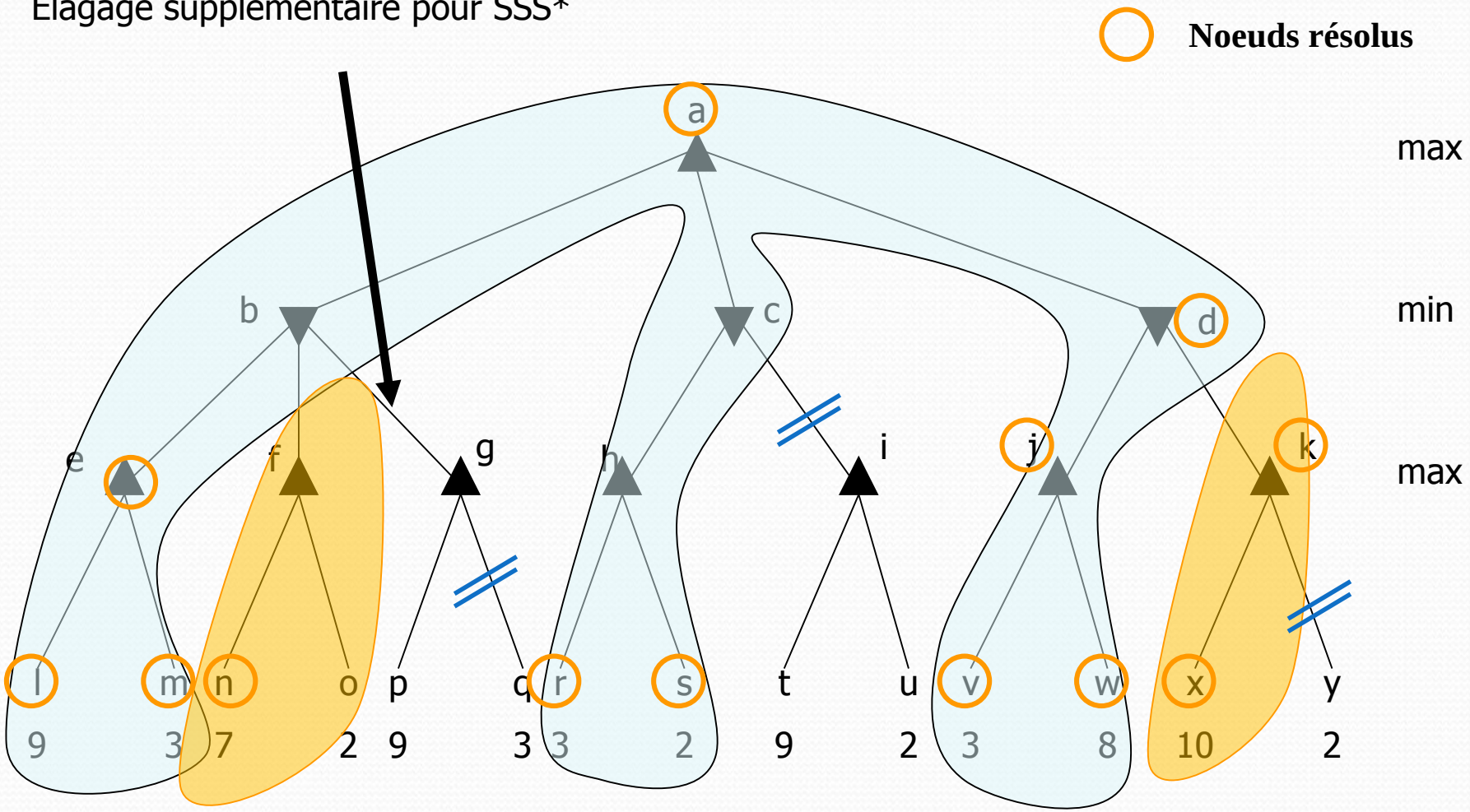


L'algorithme SSS* : à faire



SSS* - alpha-bêta ( )

Élagage supplémentaire pour SSS*



L'algorithme SSS* - remarques

- SSS* explore moins de nœuds qu'alpha-beta mais est plus gourmand en mémoire

=> On peut limiter la taille mémoire et faire un mix entre SSS* et alpha-beta