Intelligence Artificielle

Recherche Aveugle
(*Uninformed Search*)

Stephane Marchand-Maillet

Contenu

Algorithmes de recherche:

- Largeur
- Profondeur
- Profondeur Limitée
- Approfondissement itératif (IDS)
- Bidirectionnelle
- Coût uniforme

→ principes, complexité

Introduction

- On applique le principe de recherche de solution sans utiliser de connaissance a priori (aveugle – *uninformed search*)

- Essentiellement, cela revient à prendre la fonction de voisinage (les successeurs d'un état donné) ou la liste (catégorie B) sans ordre particulier

- L'alternative est la recherche heuristique (*informed search*) – prochain chapitre

Formalisation

Le graphe d'états donne une formalisation pour la résolution de problèmes. Un problème s'énonce formellement par:

Graphe G

- L'espace des états S
- Une fonction de transition Γ (avec ou sans cout)

Instance

- Un état initial s_I
- Un état final s_G

→ Une solution est un chemin de s_I à s_G dans G

Algorithme general de recherche

liste ← vide ; liste.push(s_I)
repeat
    $s_{courant}$ ← liste.pop()
    if ($s_{courant}$ == s_G)
        break
    list.push($\Gamma(s_{courant})$) ← expansion de $s_{courant}$
until liste.len() == 0

Recherche

if ($s_{courant}$ == s_G)
    backtrack solution
else
    pas de solution

Explicitation de la solution

→ Tout est dans la stratégie de gestion de la liste et l'expansion des noeuds (états)

Algorithme general de recherche

- Categorie A: Noeuds deja visités
  - sortis de la liste
- Catégorie B: Noeuds pas encore visités avec voisins visités (en A)
  - Noeuds dans la liste
- Catégorie C: noeuds pas encore visités (ni en B)
  - Noeuds jamais passés dans la liste

Un état fera le trajet C-B-A

Recherche en Largeur

- La liste est une file FIFO (type file d'attente)
- On progresse en explorant toutes les stratégies à la fois

→ Expansion du nœud le moins profond de la liste

Facteurs de complexité:

- Nombre maximum (moyen) de successeurs d'un etat: b = facteur de branchement
- Profondeur de la solution dans l'arbre : d

Recherche en Largeur

- Complet: garantit de trouver la solution
- Optimal: trouve la solution la plus simple (en nombre d’actions)

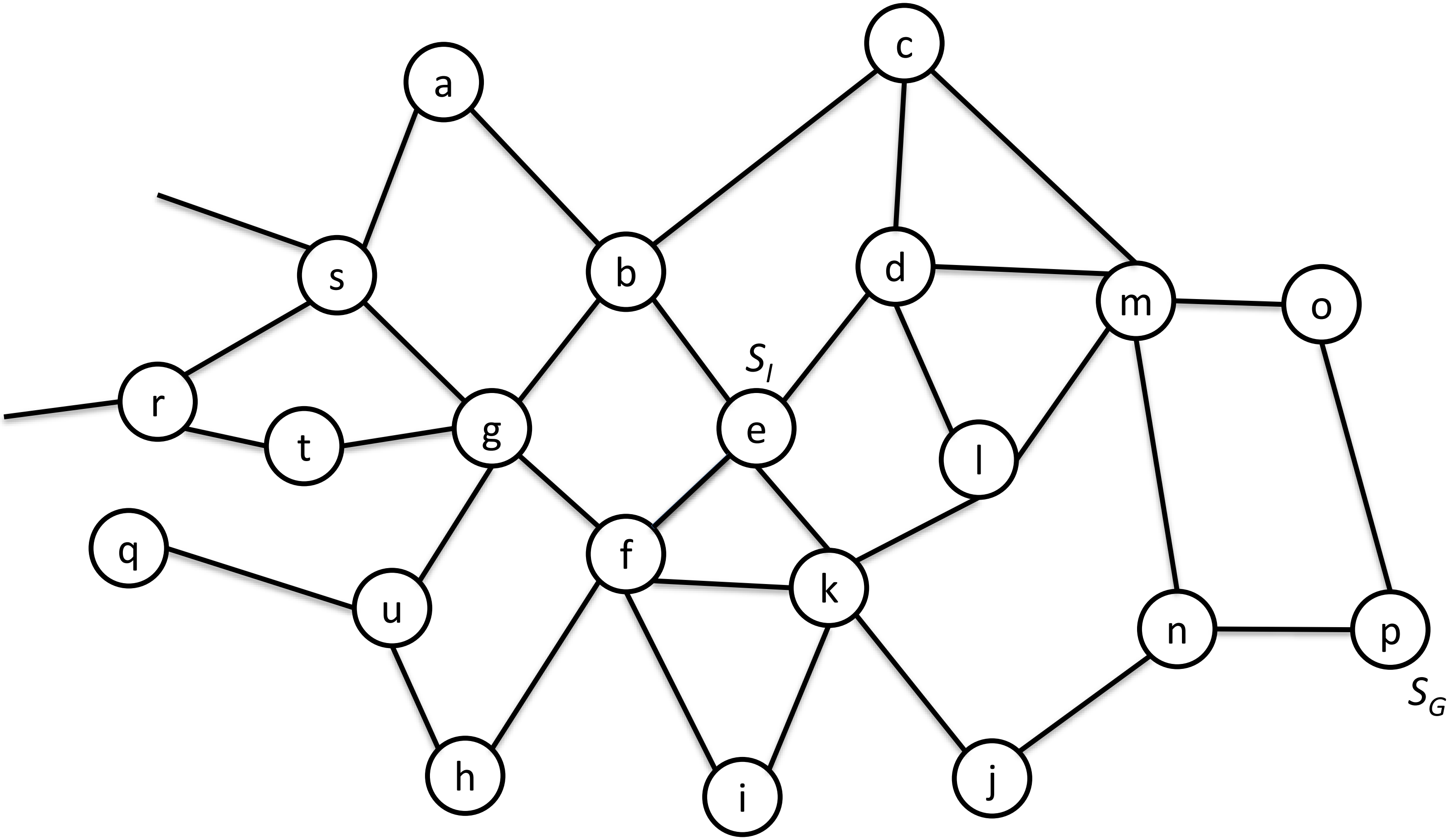
Nombre de nœuds de l’arbre produits:

$$N=b^0+ b^1+ b^2+\ldots+ b^d= (b^{(d+1)}-1)/(b-1) = O(b^d)$$

→Complexité:

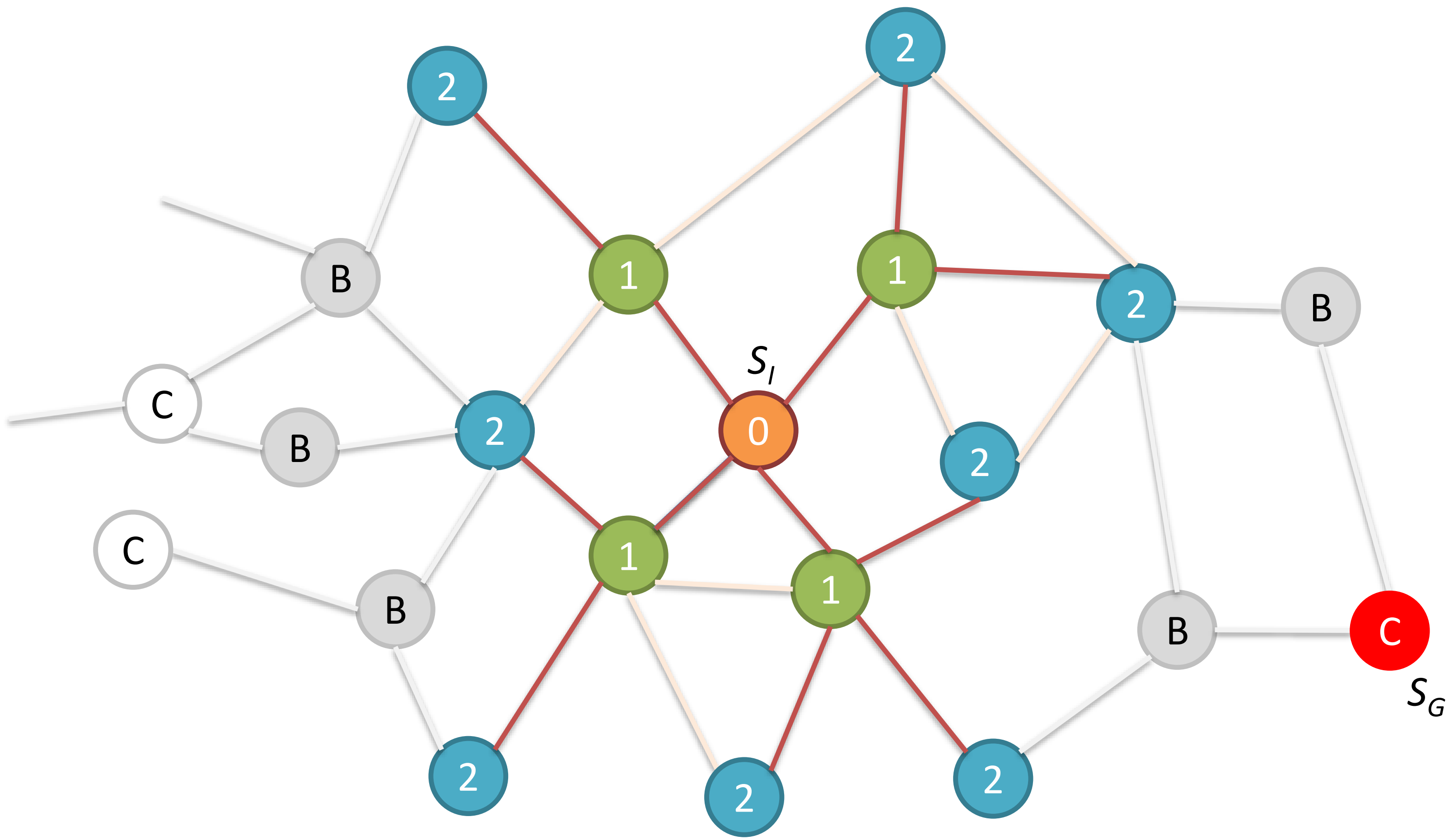
- Temps: $O(b^d)$
- Espace: $O(b^d)$

Recherche en Largeur (BFS)



Graphe d'états

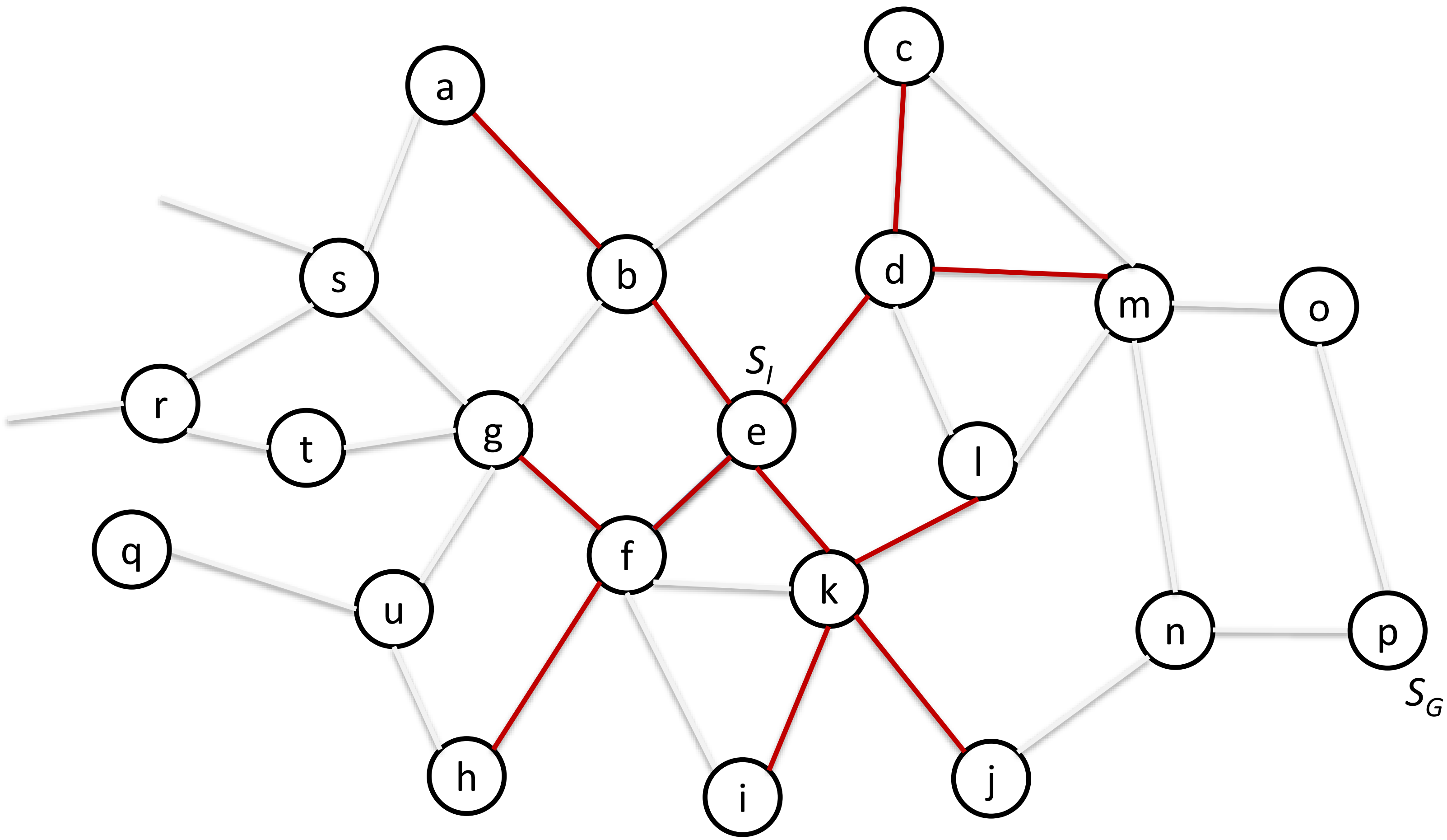
Recherche en Largeur (BFS)



Progression du BFS

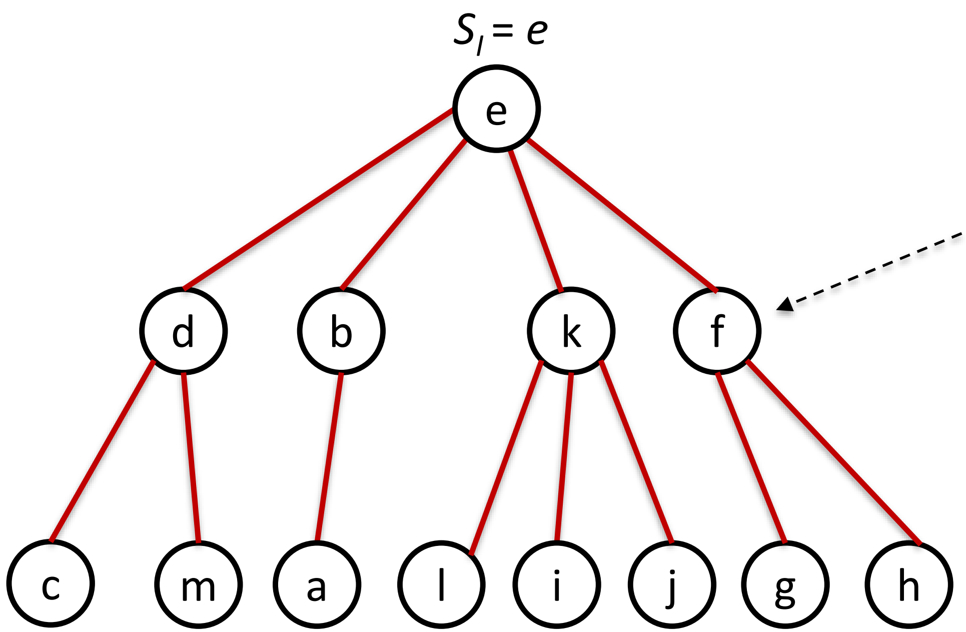
Voir aussi: https://visualgo.net/en/dfsbfs

Recherche en Largeur (BFS)



Transitions sélectionnées dans le graphe d'états

Recherche en Largeur (BFS)



Nœud v
State(v) = f
Parent(v) = racine
d = 1

Arbre de recherche correspondant

Remarque sur la complétude

- En cas de cycle dans le graphe, l'algorithme d'exploration court le risque de re-visiter des états déjà visités et donc ne pas se terminer
- L'exploration peut elle-même contenir un mécanisme pour éviter les re-visites (organisation de la catégorie « B »)
- Sinon on s'assurera de maintenir une liste globale (exple: hash-table) des nœuds visités (marquage de la catégorie « A »)

Recherche en Profondeur

- La liste est une file LIFO (type Pile)
- On explore chaque stratégie jusqu'au bout

→ Expansion du nœud le plus profond de la liste

Facteurs de complexité:

- Nombre maximum (moyen) de successeurs d'un état: b = facteur de branchement
- Profondeur de la solution dans l'arbre: d
- Profondeur maximum d'une feuille: m

Recherche en Profondeur

- Complet si l'arbre est fini
- Non optimal (en général)

Complexité:

- Temps: $b^0 + b^1 + \ldots + b^m = O(b^m)$

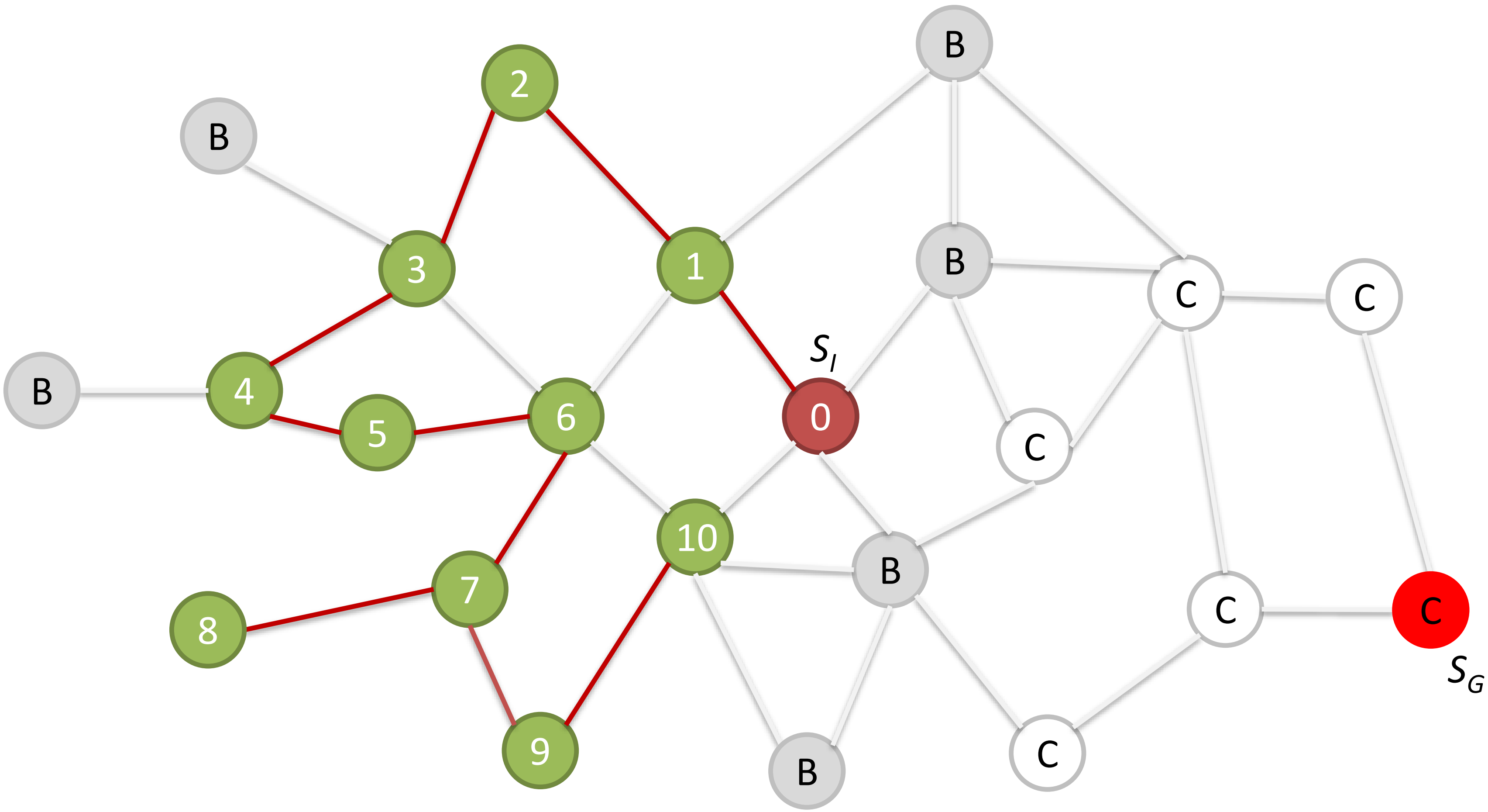
  (m peut être >> d)

- Espace: O($b.m$)

  Lineaire !

m est le facteur clé

Recherche en Profondeur (DFS)



Voir aussi: https://visualgo.net/en/dfsbfs

Recherche en Profondeur Limitée

Comme m est un facteur de complexité, on borne m par M (donné)

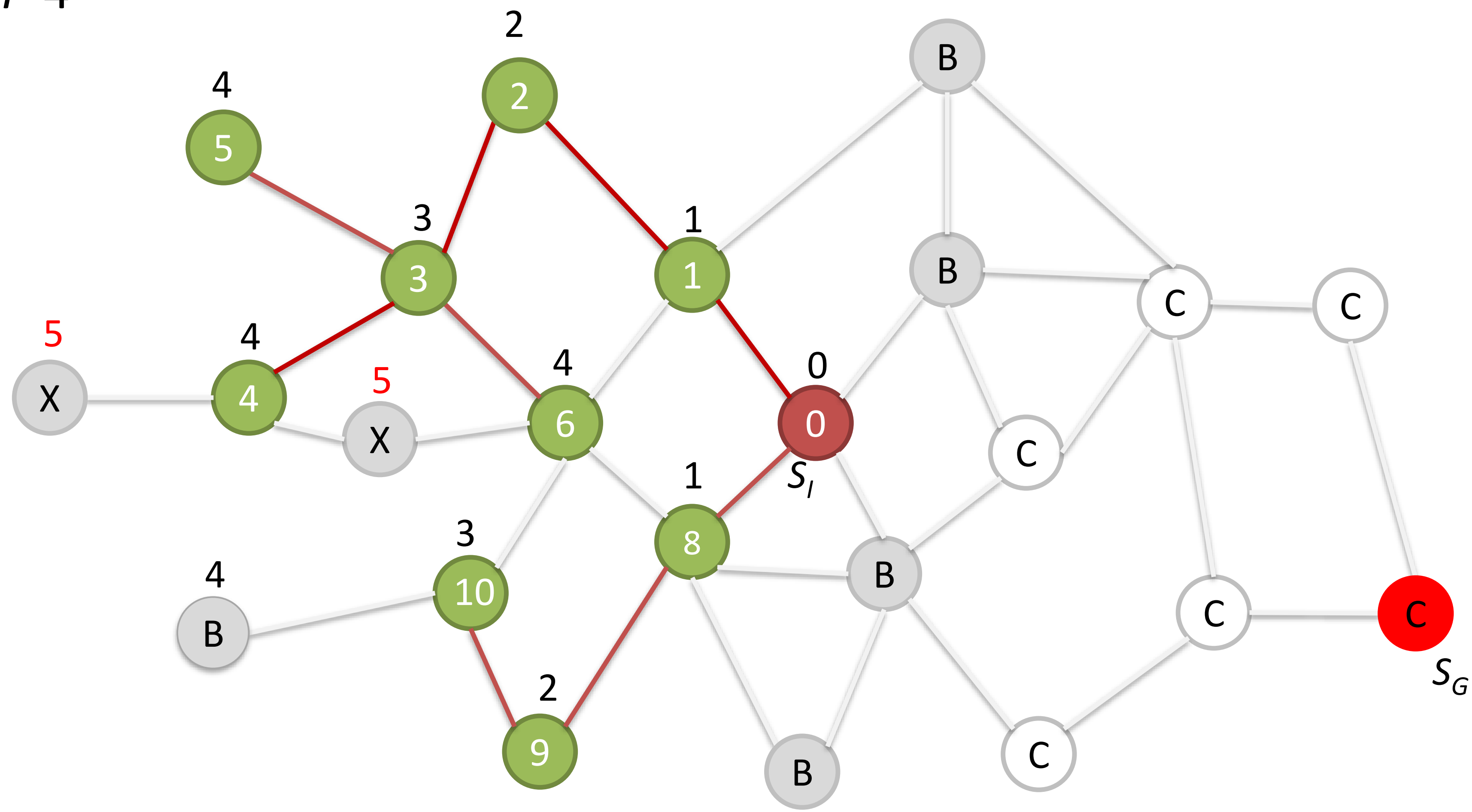
- Complet si d<=M
- Pas de garantie d'optimalité

Complexité:

- Temps: $b^0 + b^1 + \ldots + b^M = O(b^M)$
- Espace: $O(b.M)$

Recherche en Profondeur Limitée

M=4



Exercice: Completer l'exploration avec M=4

Approfondissement itératif (IDS)

Recherche en profondeur Limitée:

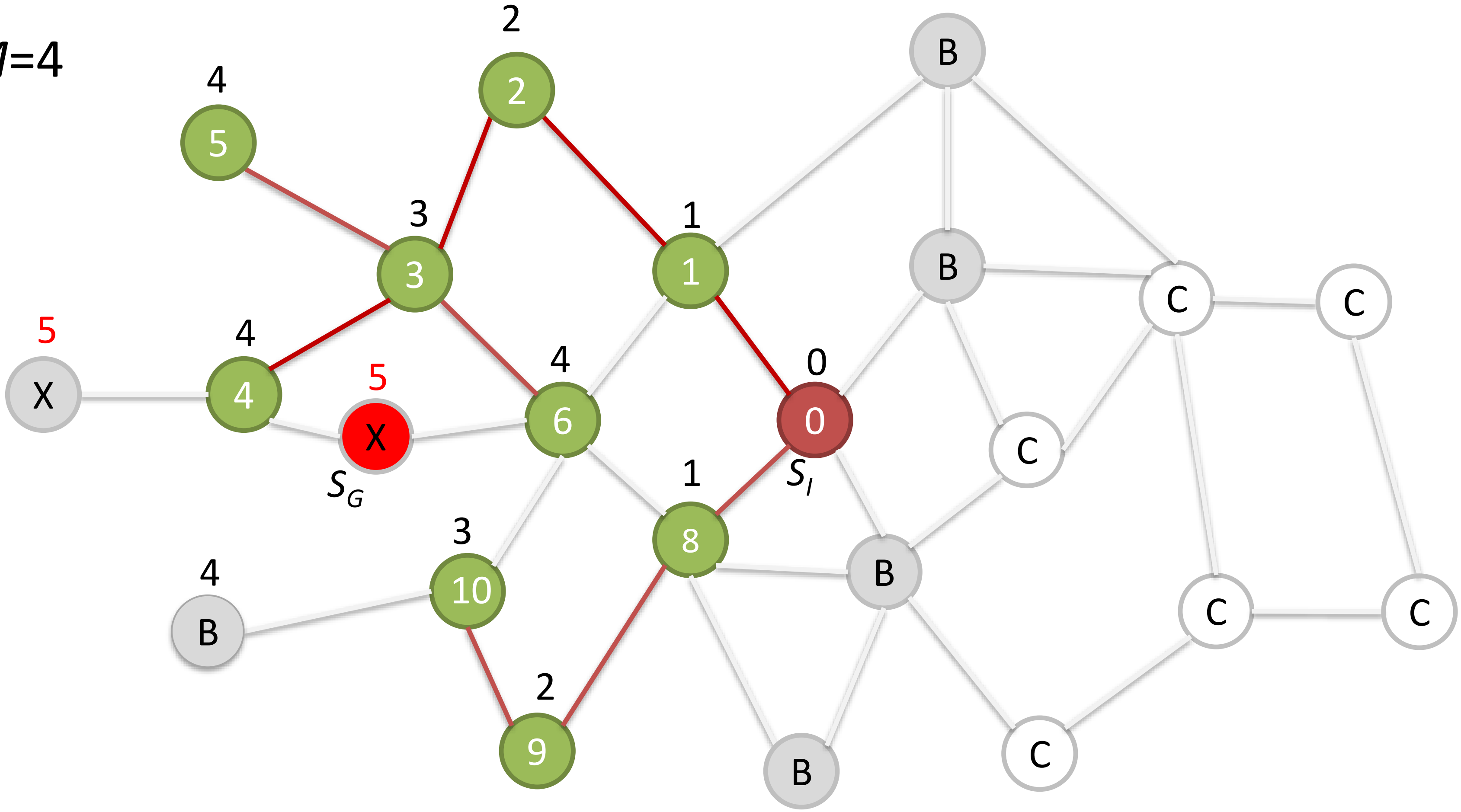
- Fournit une solution approximée
  - Solution si $d<=M$
  - Indécidable si $d>M$

→On itère sur *M*: approfondissement itératif:

(*IDS: Iterative Depth Search*)

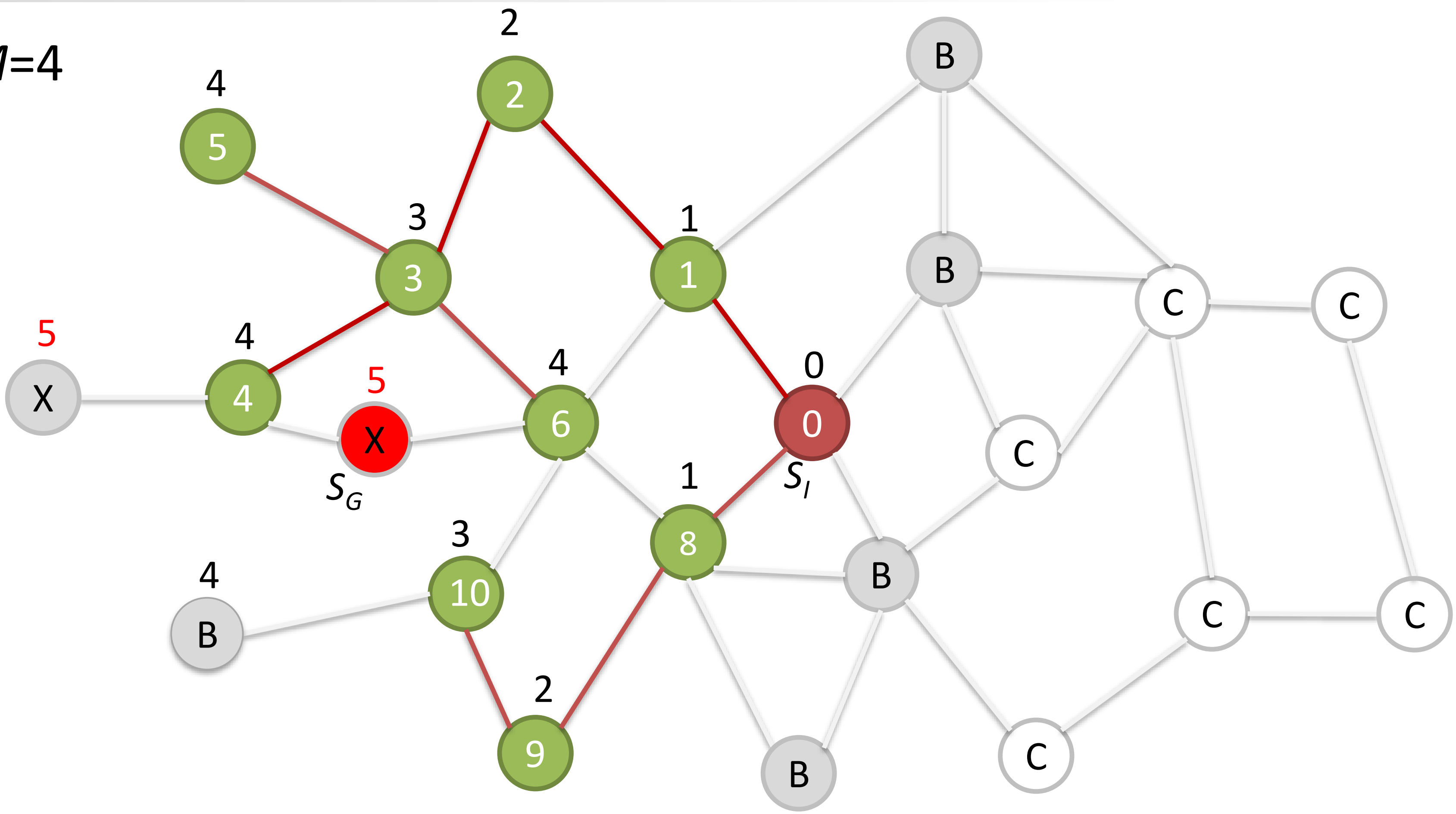
Recherche en Profondeur Limitée

M=4



Recherche en Profondeur Limitée

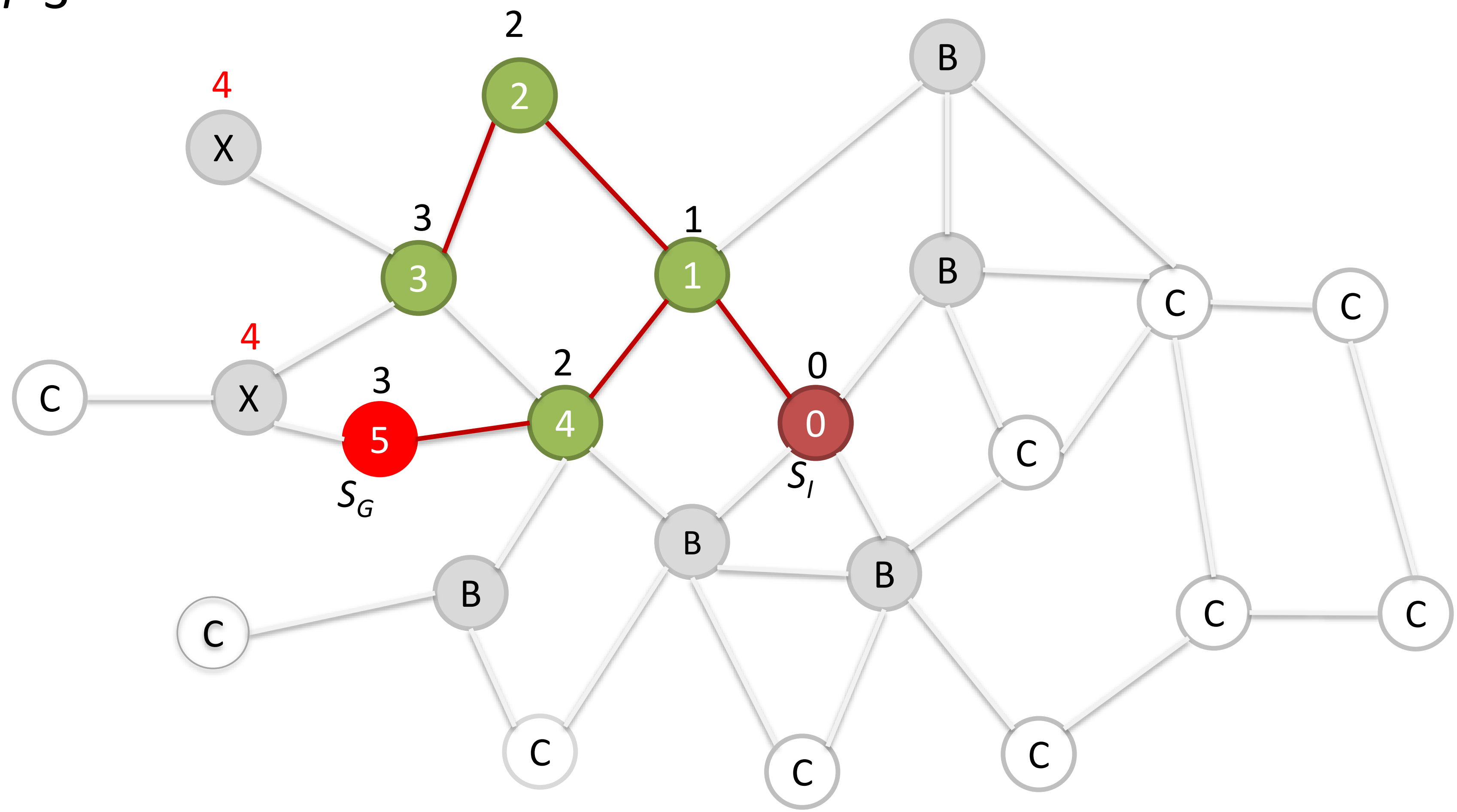
M=4



Note: Il semblerait que cette solution ne soit pas trouvée, mais en fait elle serait trouvée à l'étape *M*=3 (cf slide suivant)

Recherche en Profondeur Limitée

M=3



S_G est atteint → l'algorithme s'arrête

Approfondissement itératif (IDS)

- Complet: car on explore éventuellement toutes les solutions
- Optimal: car on trouve la solution la plus simple (profondeur *d*) avant les autres

Complexité:

- Temps: $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Espace: $O(b.d)$

Approfondissement itératif (IDS)

- Les recherches répétées sont celles des niveaux inférieurs (0, 1,…, *d*-1), que l'on répète de moins en moins ((*d*+1) fois, *d* fois, (*d*-1) fois,…etc)
- La croissance exponentielle

$$a_0 b^0 + a_1 b^1 + a_2 b^2 + \ldots + a_d b^d$$

rend la somme niveaux inferieurs aussi chère que le niveau lui-même:

$$b^0 + b^1 + \ldots + b^j = O(b^{j+1})$$

Approfondissement itératif (IDS)

Exemple: b=10

Profondeur d'abord:

$$n_{\mathrm{DFS}} = b^0 + b^1 + \ldots + b^d$$

IDS:

$$n_{\mathrm{IDS}} = (d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d$$

D	**5**	**8**	**10**	**15**
n_{DFS}	11'111	11'111'111	$1\text{x}10^{11}$	$1\text{x}10^{16}$
n_{IDS}	23'456	23'456'789	$2\text{x}10^{11}$	$2\text{x}10^{16}$

Recherche Bidirectionnelle

On exploite le fait que

$$b^{d/2} << b^d$$

On développe des chemins par la recherche en largeur à partir de s_I et s_G

À leur intersection, on joint le chemin qui mène de s_I à s_G

$\boxed{\text{Condition: existence de } \Gamma^{-1}}$

On doit connaitre explicitement Γ^{-1} pour développer le chemin à partir de s_G

Recherche Bidirectionnelle

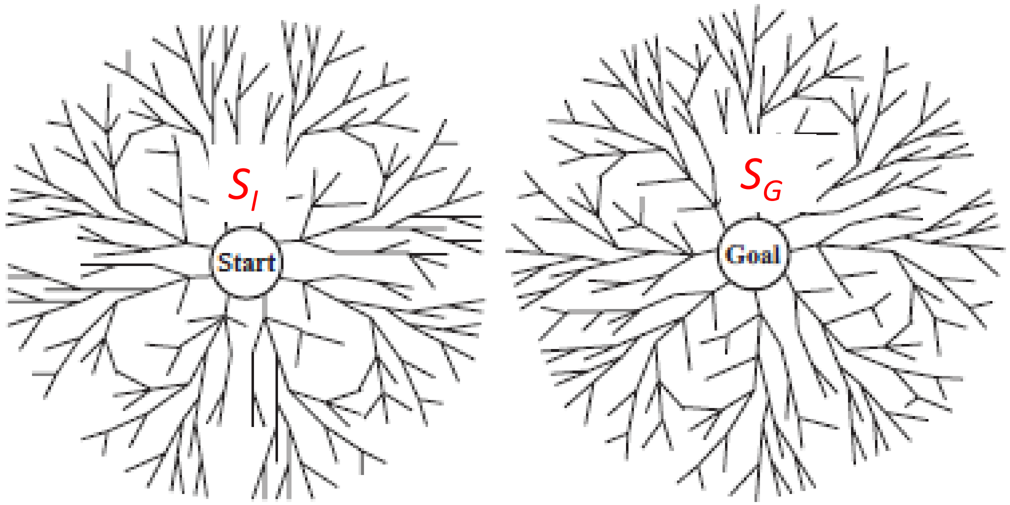


Figure 3.20 A schematic view of a bidirectional search that is about to succeed when a branch from the start node meets a branch from the goal node.

Adapted rom AIMA: Section 3.4 – page 91

Recherche Bidirectionnelle

- Complet: car les recherches se rejoignent si une solution existe
- Optimal: car on trouve la solution la plus simple à la jointure des chemins

Complexité (recherche en Largeur pour *d*/2):

- Temps: $O(b^{d/2})$
- Espace: $O(b^{d/2})$

Recherche en coût uniforme

Similaire à la recherche en Largeur

$g(v)$: coût du chemin de la racine au nœud v

$g(v)$ représente la somme des coûts de transition entre les états $c(s,s')$ le long du chemin

→Expansion du nœud le moins coûteux de la liste

→La liste est un Tas-min des coûts

Analogue au plus court chemin de Dijkstra

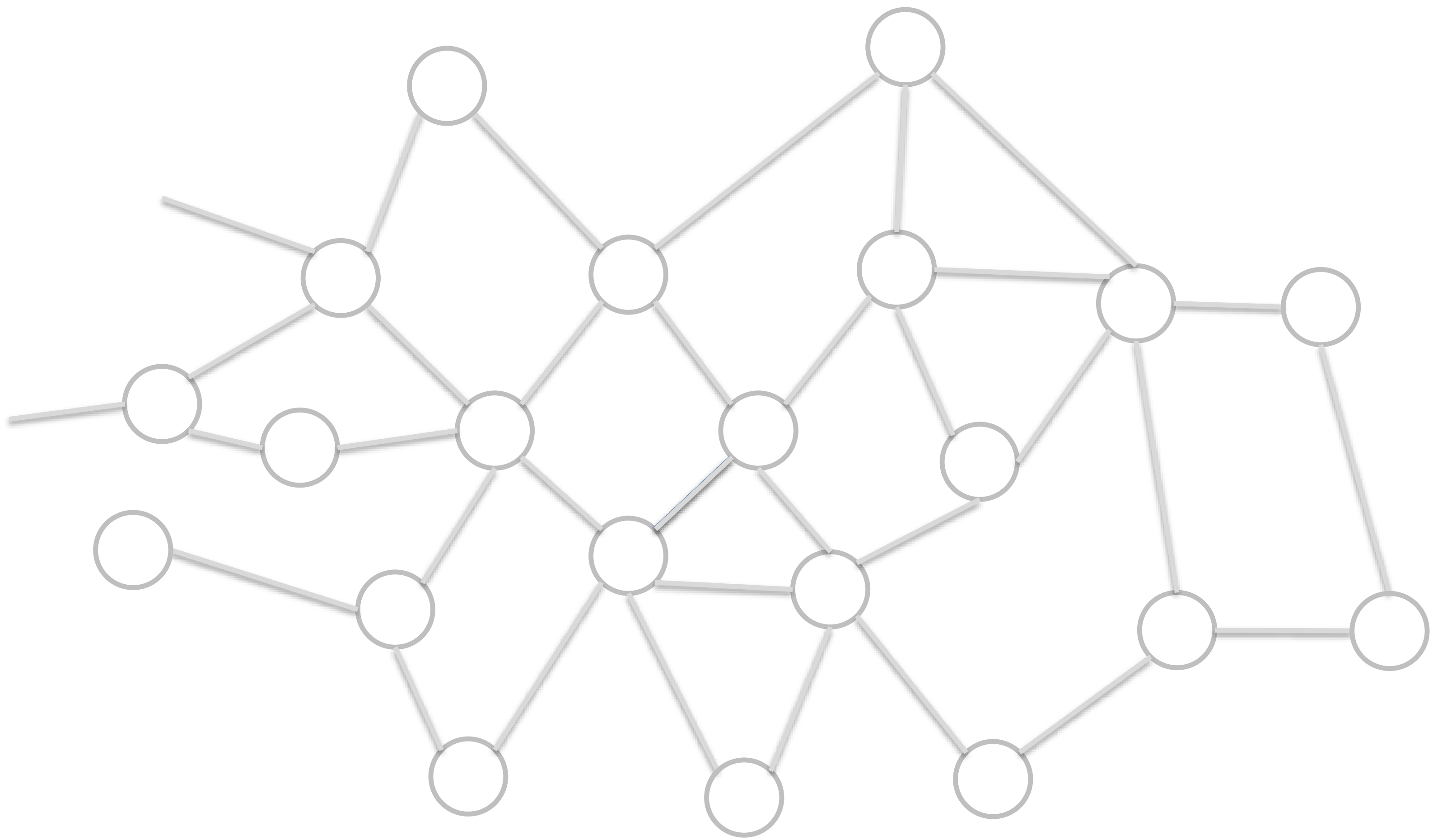
Recherche en coût uniforme

- Complet: garantit de trouver la solution
- Optimal: trouve la solution la moins coûteuse

Complexité:

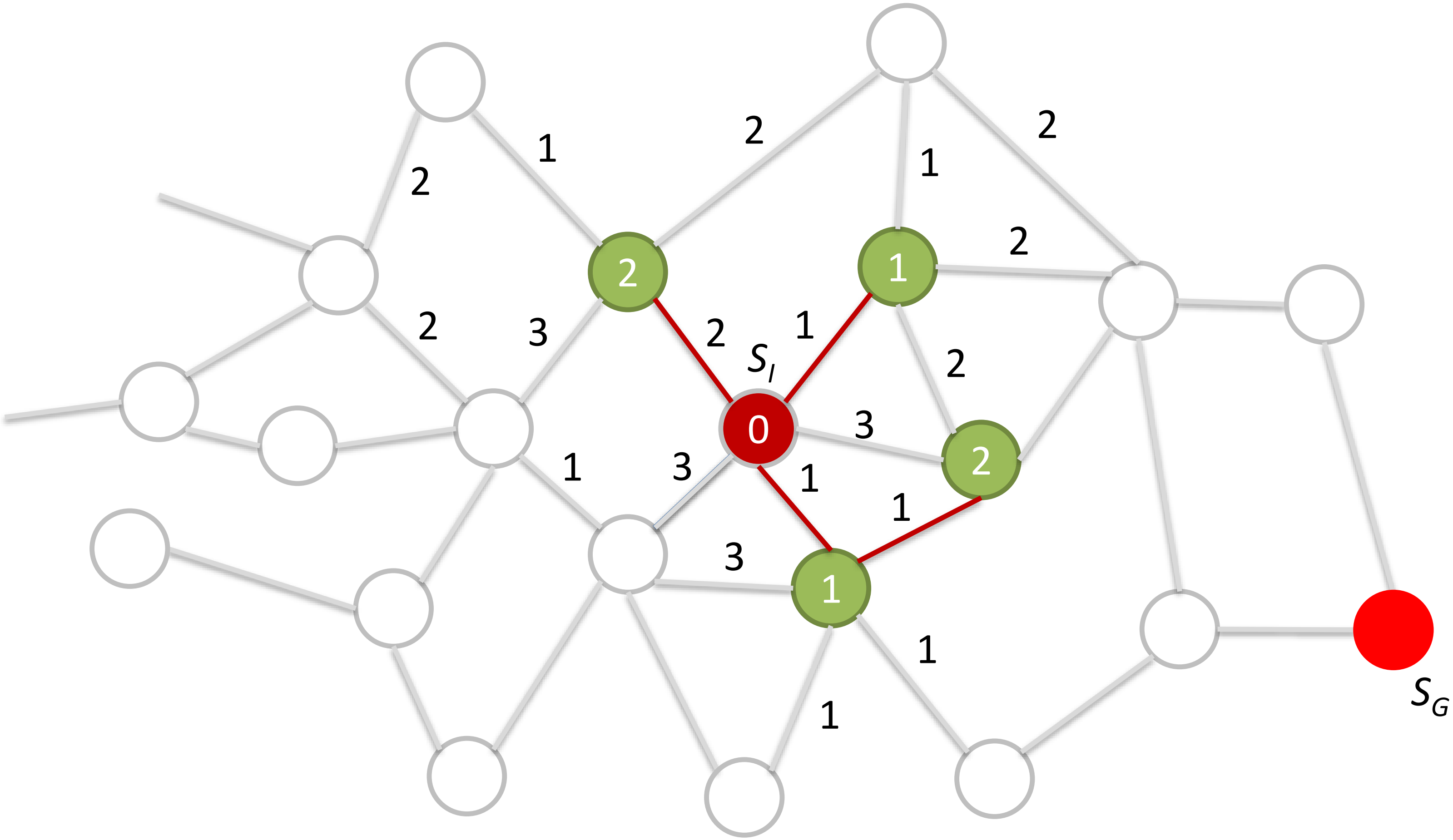
- Temps: O(b^d)
- Espace: O(b^d)

Recherche en Coût Uniforme



Exercice: Attribuez des coûts et développez l’algorithme

Recherche en Coût Uniforme



Voir aussi: https://visualgo.net/en/sssp

Résumé

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

AIMA: Section 3.4 – page 91