Intelligence Artificielle

Méthodes de Recherche

Stéphane Marchand-Maillet

Contenu

- Problèmes de recherche
- Etats et Transitions
- Graphe d'Etats
- Parcours du Graphe

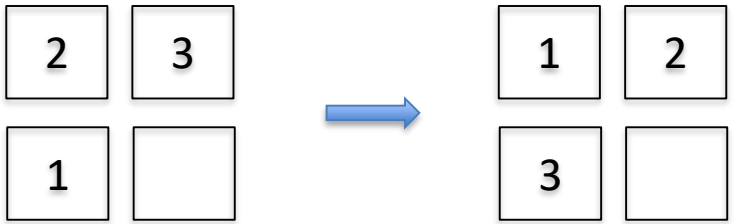
Problèmes de Recherche

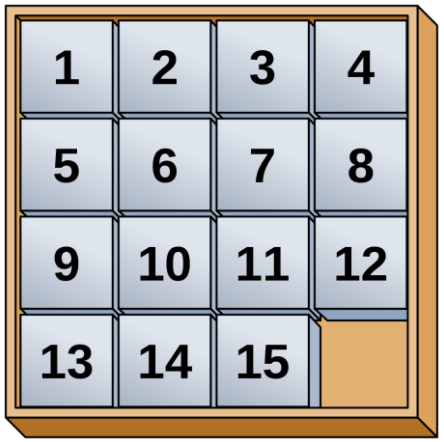
But: résoudre un problème donné

Méthode: par étapes, selon «ce qui est possible»

Exemple: Jeu du Taquin

But: remettre les pièces dans l'ordre en les déplaçant (gauche-droite, haut-bas)

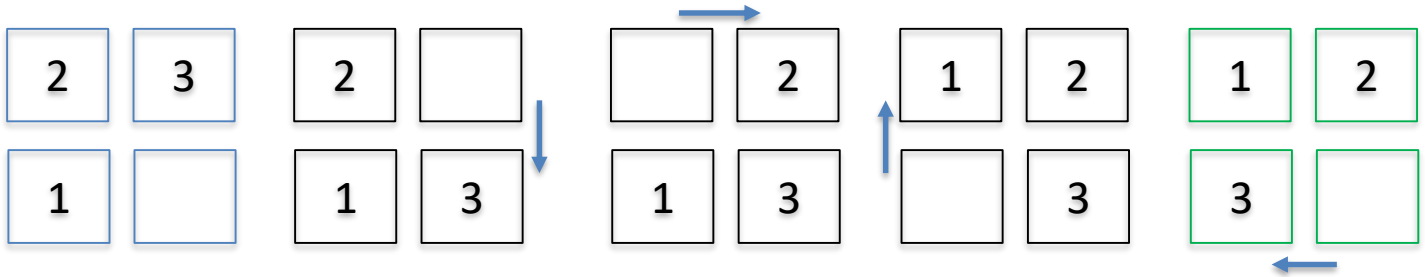


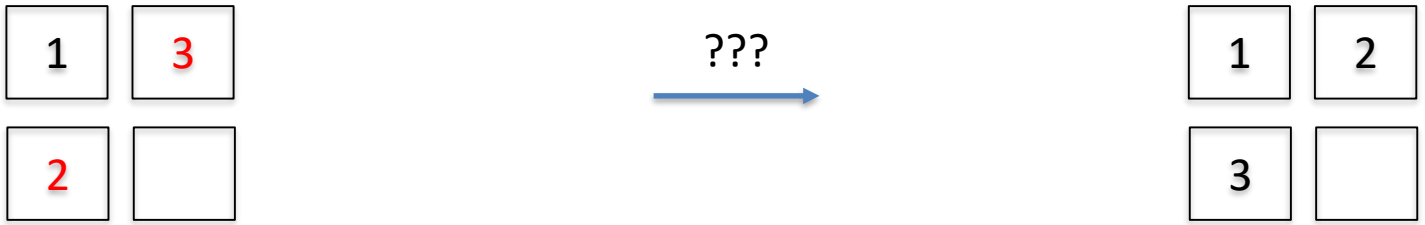


Taquin-15

Source: wikipedia

Analyse de l'exemple





Pas de solution (exercice)

Formalisation

Comment formaliser:

- Le problème (ou classe de problèmes)
- Le fait de chercher une solution
- La «difficulté» du problème
- La solution elle-même
- Le fait qu'il y ait (ou pas) de solution

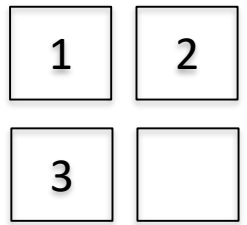
?

Formalisation

Comment formaliser:

- Le problème (ou classe de problèmes)
  - Modèle, catégorie
- Le fait de chercher une solution
  - Algorithme
- La «difficulté» du problème
  - Complexité
- La solution elle-même
  - Convergence
- Le fait qu'il y ait (ou pas) de solution
  - Propriétés: existence, unicité, optimalité

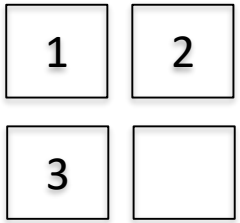
Analyse de l'exemple



Catégorisation:

- 1 seul joueur/acteur
- But bien défini
- Actions «discrètes»
- Actions certaines

Analyse de l'exemple



Catégorisation:

- 1 seul joueur/acteur
  - Dépend seulement de nos actions
- But bien défini
  - Test de convergence exact
- Actions «discrètes»
  - Possibilités énumérables (dénombrables)
- Actions certaines
  - Contexte déterministe (prédictible)

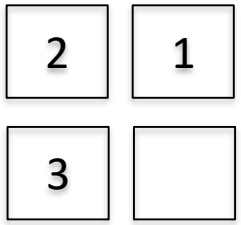
Formalisation: Etats

On définit la notion d'état du système:

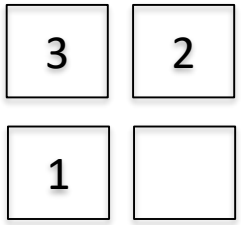
Def: Un état *s* (*state*) est une configuration d'un système.

Un état peut être observable ou non.

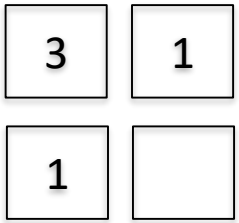
Exemples:



Un état s_i



Un état s_j



Ceci n est pas un état

Notion d'état

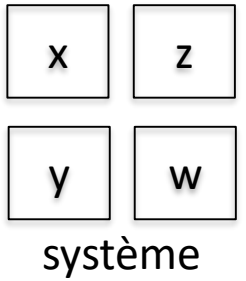
Un état est un ensemble de valeurs des paramètres d'un système.

On parle de *vecteur d'état*

Exple: Véhicule: position, vitesse, direction,…

Dans notre cas:

- Paramètres: *x,y,z,w*
- Valeurs: permutations de {1,2,3,« »}

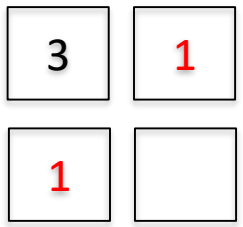


Espace d'états

Def: L'espace d'états (*state space*) *S* d'un système est l'ensemble de tous les états possibles du système.

C'est la couverture de toutes les valeurs possibles de chaque paramètre

Dans notre cas: $S = \Pi(\{1,2,3,« »\})$



Ceci n est pas un état

Taille de l'espace d'états

Dans le cas discret (notre cas), on compte (énumère) les états, en général en utilisant la combinatoire du problème.

La taille de S peut être une mesure de la complexité du problème

Dans notre cas:

- 4 choix pour la 1ere case
- 3 choix pour la 2eme case
- 2 choix pour la 3eme case
- 1 choix pour la 4eme case

→ 4! = 24 états possibles = $|\Pi(\{1,2,3,\text{« »}\})|$

Note: espace d'états infini

On peut distinguer la notion d'espace d'états continu ou dénombrables

Espace continu: position d'un véhicule

Espace dénombrable: «infini, non borné mais discret», «labelisable»

Espace d'états infini

Conjecture (AIMA 3.2.1 p 73):

A partir de la valeur 4 et étant donné un entier *n*, on peut générer une séquence d'opérations factorielles et de racines carrées pour obtenir une valeur dont la partie entière est *n*.

Symboliquement, il existe *a,b* t.q qquesoit *n:*

$$n = \text{int}(\text{sqrt}^a(\text{fact}^b(4)))$$

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}} \right\rfloor = 5$$

(la puissance sur les opérateurs indique la composition)

Espace d'états infini

$$n = \text{int}(\text{sqrt}^a(\text{fact}^b(4)))$$

Un état est une paire (a,b), on va organiser l'énumération de ces états.

Il n'y a pas forcément de limite sur a et b. On doit donc explorer toutes les paires (a,b), ce qui crée un espace d'état infini.

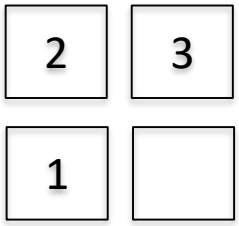
→ On est donc jamais sûrs qu'une solution n'existe pas, même si la recherche reste infructueuse

Formalisation: instanciation

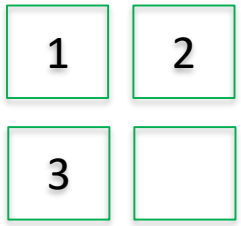
Pour définir un problème on doit définir:

- Un état initial: s_I
- Un état final: s_G

Dans notre cas:



s_I



s_G

Les états initiaux ou finaux sont décrits de façon explicite (exacte) ou implicite (règle)

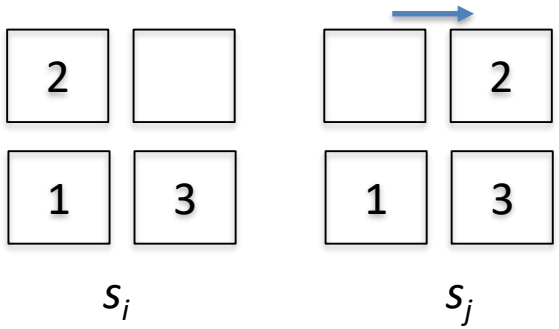
Exemple: “le 3 doit être en dessous du 1”

Transition entre états

L'espace d'états rassemble tous les états. Mais les états sont liés entre eux par des actions permettant de passer d'un état à un autre (*state transition*).

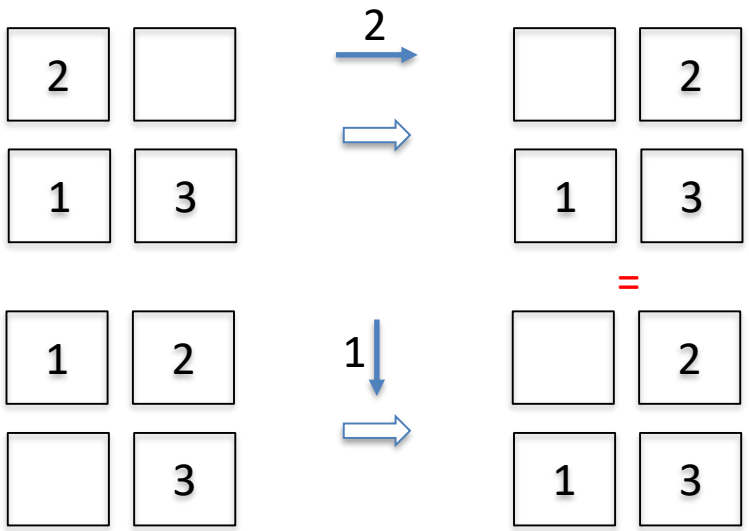
Pour définir un problème on doit aussi définir les actions possibles

Exple: Action = déplacer une case vers la droite



Transition entre états

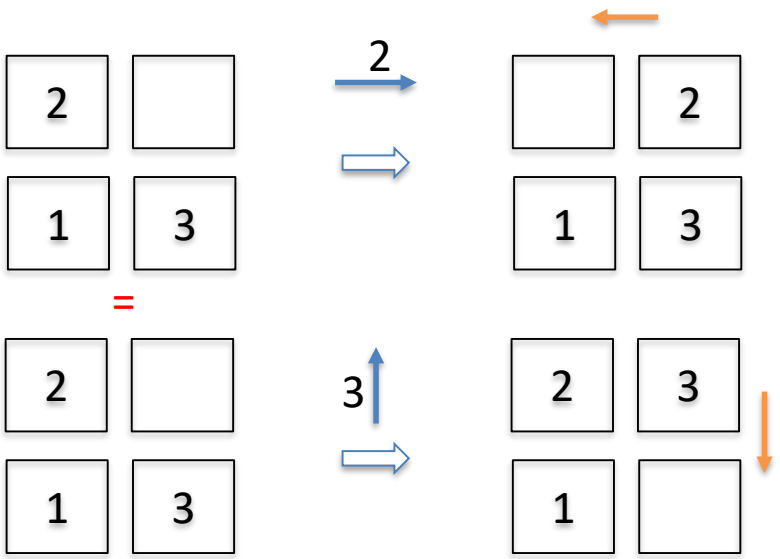
Dans notre cas, un état ne dépendra jamais de son passé. Il est indépendant des transitions (actions) qui ont mené a lui



Transitions

Dans notre contexte, les actions possibles sur un état dépendent de cet état

Exple:



Note: plutôt que de symboliser le déplacement d'une case particulière (1,2,3), on peut visualiser le déplacement de la case vide

Actions = {2D,3H} = {G,B}

Transitions

- Actions discrètes: transitions dénombrables, états dénombrables
- Une transition peut avoir un coût (positif)
  - Sinon on considère son coût égal a 1 (on compte les transitions)

- Plus tard (contexte probabiliste):

$$G(s',a,s) = P(s' \mid s,a)$$

ici (actions déterministes):

T(s',s) = matrice de transition (binaire)

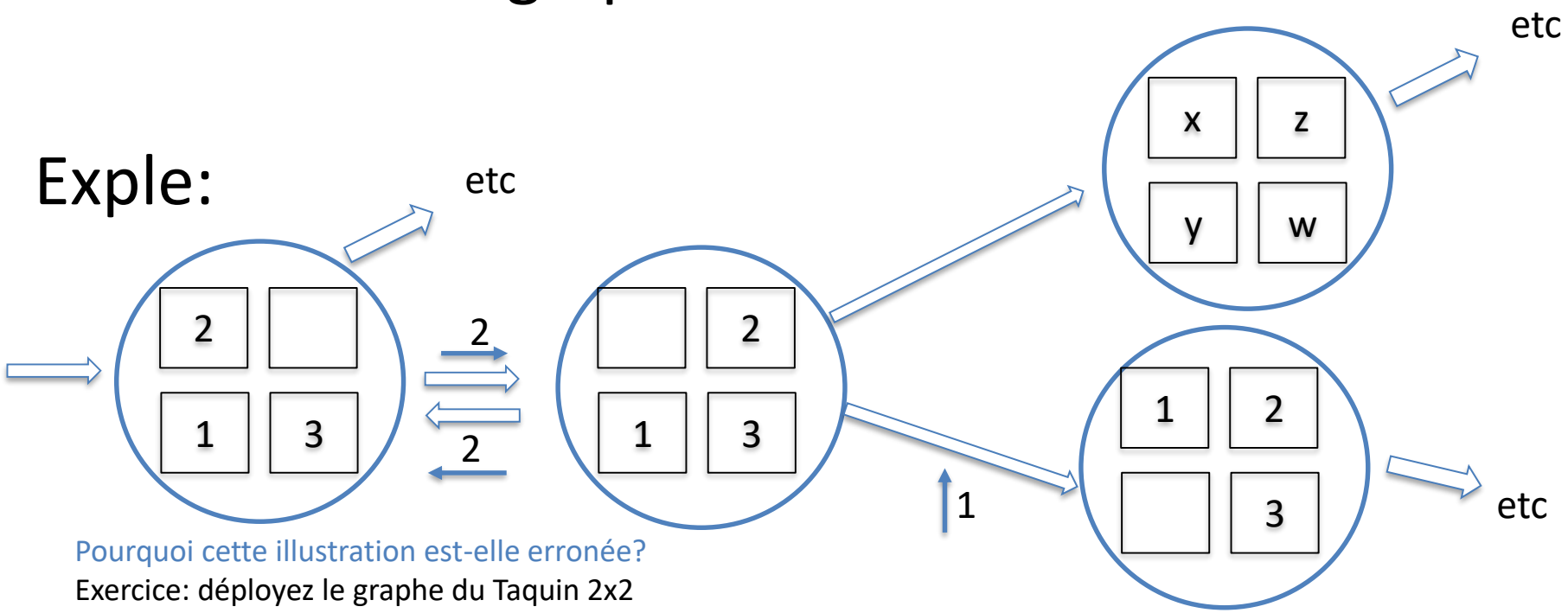
(en fait, liste d'adjacence Γ)

Graphe d'états

On définit donc le graphe d'états à partir de la fonction de transition:

- Les nœuds du graphe sont les états
- Les arêtes du graphe sont les transitions

Exple:



Pourquoi cette illustration est-elle erronée?

Exercice: déployez le graphe du Taquin 2x2

Graphe d'états

Les propriétés du graphe d'états sont autant d'indicateurs sur la complexité du problème:

- Degré, degré moyen
- Connexité

Dans notre cas, le graphe d'états est fixe dans le temps (les règles ne changent pas)

Plutôt que de l'expliciter (le développer), on le représente (localement) par la fonction multivoque de voisinage (fonction successeur):

$$\Gamma : S \longrightarrow \mathcal{P}(S)$$

qui peut aussi invoquer un coût c(s,s') par arête (s,s')

Application multivoque

- v_j est le successeur de v_i dans G si il existe
$$e_l = (v_i, v_j)$$
alors v_i est le prédécesseur de v_j dans G
- l'ensemble des successeurs d'un sommet v_i dans G est noté $N_G^+(v_i)$
- l'ensemble des prédécesseurs d'un sommet v_i dans G est noté $N_G^-(v_i)$
- L'application Γ qui à tout sommet de V fait correspondre l'ensemble de ses successeurs est une application multivoque. On a:

$$\Gamma : \quad V \to \mathcal{P}(V) \qquad v_i \mapsto N_G^+(v_i) \quad \Longrightarrow \quad \Gamma^{-1} : \quad V \to \mathcal{P}(V) \qquad v_i \mapsto N_G^-(v_i)$$

donc, Γ^{-1} est l'application qui a tout sommet de V fait correspondre l'ensemble de ses prédécesseurs

Formalisation

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

- L'espace des états S
- Une fonction de transition Γ (avec ou sans cout)
- Un état initial s_I
- Un état final s_G

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*

Un mot sur l'observabilité

- Notion liée formellement à la théorie des Systèmes Dynamiques
- Essentiellement liée à la réversibilité de la fonction de transition

- Ici on considère un contexte observable
- On a une vue globale du système
  - Cf aussi plus tard (*Partially Observable*…)

Exemples de problèmes

- Taquin NxN
- Rubik's cube
- Reines sur échiquier
- Actions d'un robot
- Déplacement de véhicule (path planning)
- Plus court chemin
- Voyageur de Commerce
- …

Types de problèmes

Taquin, Rubik's cube: on cherche à minimiser le nombre de coups (déplacements) pour arriver à un but explicite (image finie, nombres classés)

- Le chemin vers la solution est important

Reines sur l'échiquier: on cherche à construire une solution dont on est pas sûr qu'elle existe

- La solution elle-même est importante (nombre maximum pour une taille d'échiquier donnée)

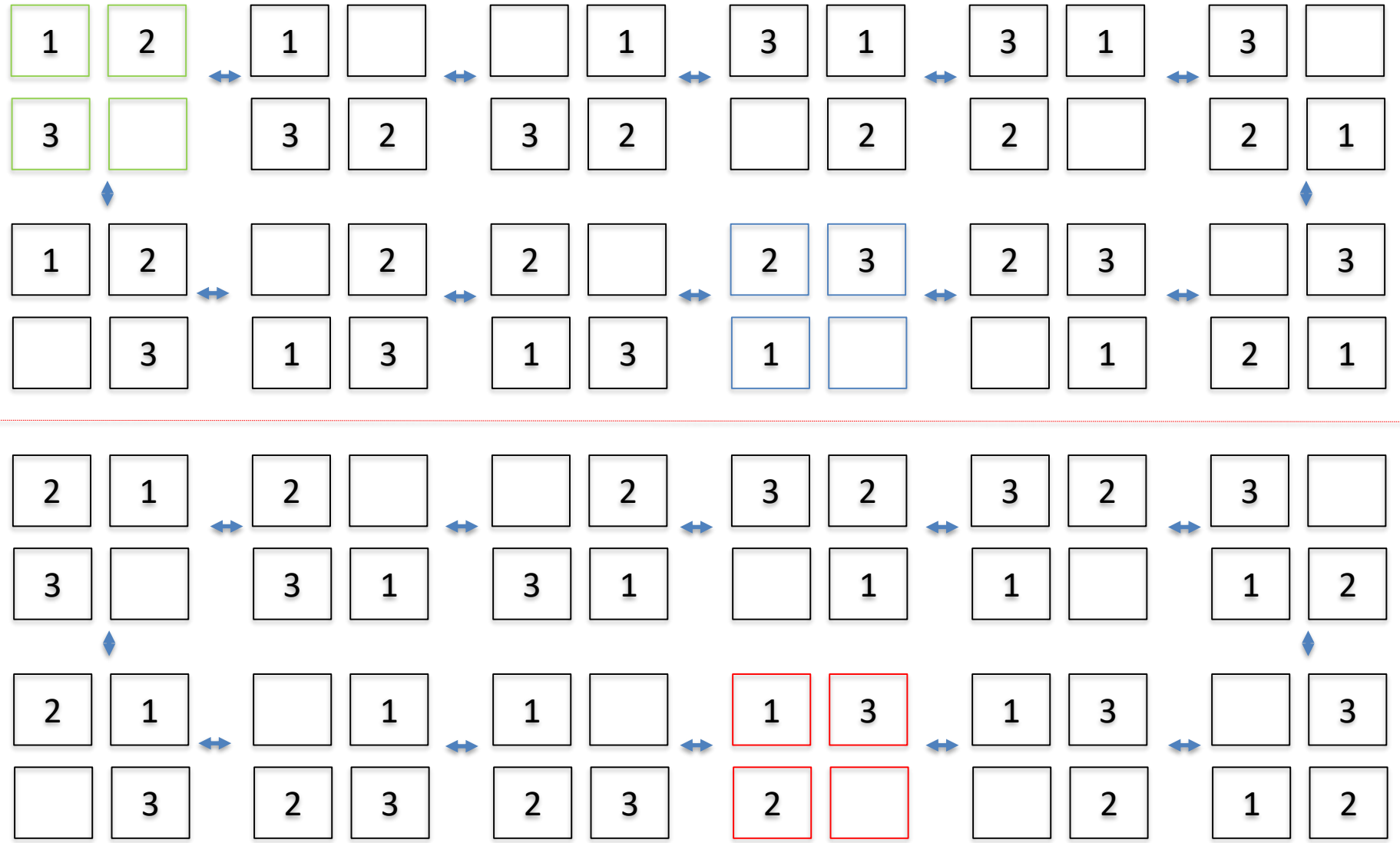
Résolution

Une solution est un chemin de l'état initial s_I à l'état final s_G dans le graphe d'états G

L'existence de la solution est liée à la connexité du graphe

L'optimalité de la solution est liée à son coût

Connexité du graphe d'etats



Connexité du graphe d'etats

Taquin 3x3: $3^2!$ = 9! = 362'880 états. Impossible de déployer le graphe

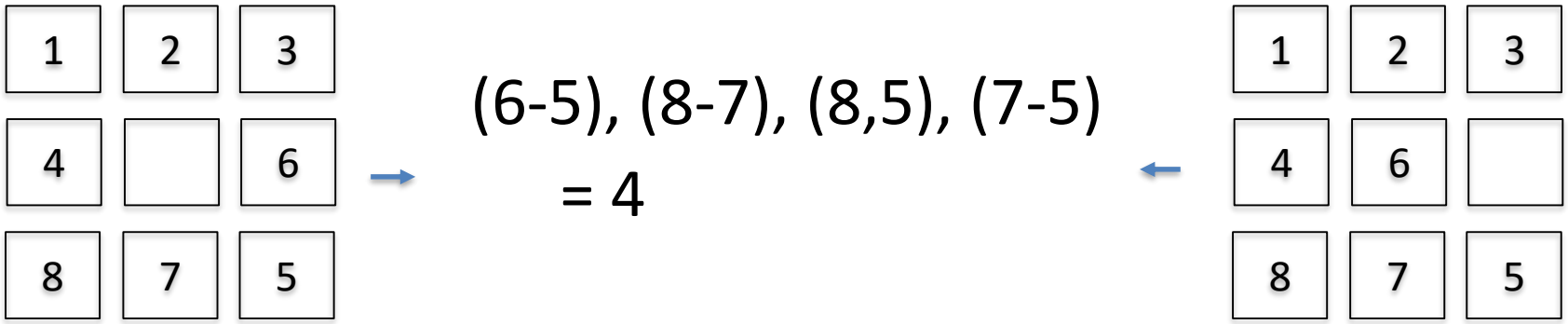
Taquin NxN: on voudrait étudier le problème en general

- Dans ce cas on peut…

Taquin *N*x*N*: connexité

On se donne une mesure de la qualité d'un état (comme fonction de la distance à l'état solution):

– On lit les valeurs en séquence et on compte le nombre d'inversions



On remarque qu'un mouvement horizontal ne change pas le résultat

Taquin *N*x*N*: connexité

- On peut identifier le taquin avec l'espace des permutations $\Pi(\{1,\ldots,N^2\text{-}1\})$
- La mesure est liée à la signature des permutations
- Les mouvements ne changent pas la parité de la mesure (resp. la signature)

→2 composantes connexes (pair/impair - +1/-1)

(cf document annexe sur Moodle pour preuve formelle)

Résolution

On représente le graphe par une matrice d'adjacence T (cf *Transitions*)

On sait que $(Id-T)^{-1}$ nous liste les chemins entre chaque noeud du graphe

Sauf que: c est impraticable, souvent le nombre d'états croît <u>exponentiellement</u> avec la taille du problème

Nombre d'états du taquin NxN = $(N^2!)$

N	**2**	**3**	**4**	**…**	**7**
Etats	24	362'880	~21 x 10^{12}		~6 x 10^{62}

Résolution

On contre la croissance exponentielle des parametres par des definitions et actions implicites (recursives, de proche en proche):

- Algorithme de recherche de chemin dans un graphe: BFS, DFS,…
  - exploration de proche en proche

→Construction d'un arbre de recherche

- Depend de la stratégie de recherche

Arbre de recherche

But: Explorer le graphe à partir de s_I

→Trace de la stratégie d'exploration

Nœuds: contiennent les états visités

Racine: correspond a l'état initial s_I

Définition récursive:

Les enfants(v) sont construits à partir de Γ(state(v))

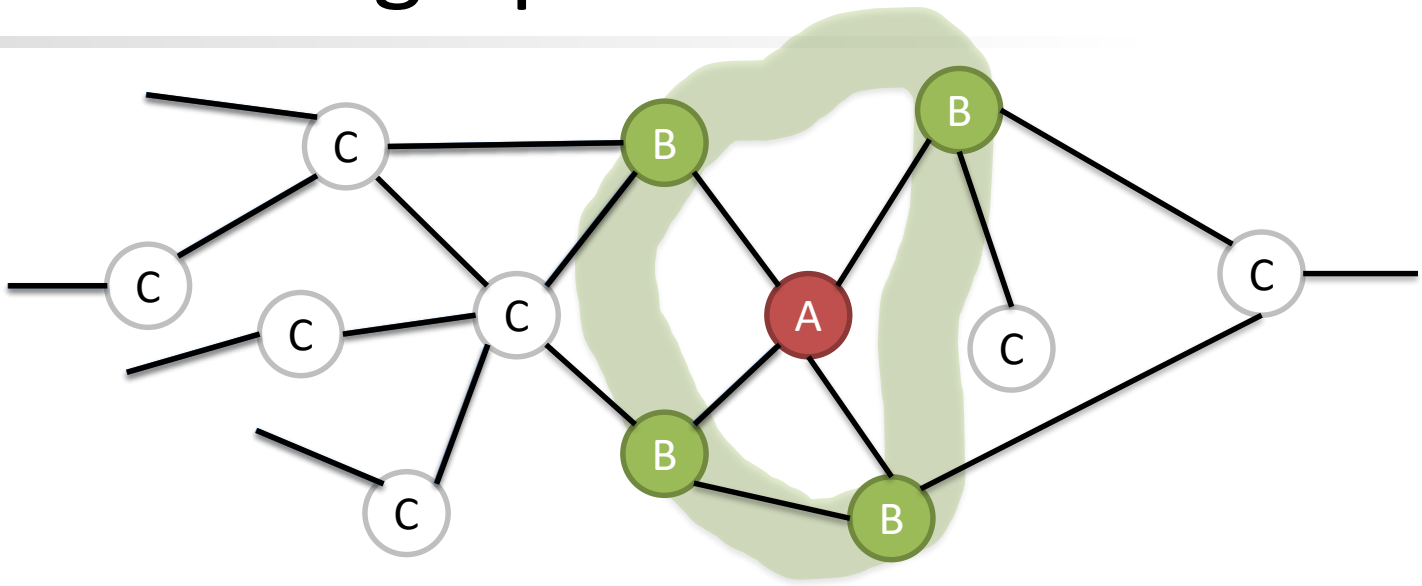
Cycle → arbre infini (même si graphe fini)

Stratégie de construction: exploration type DFS, BFS..

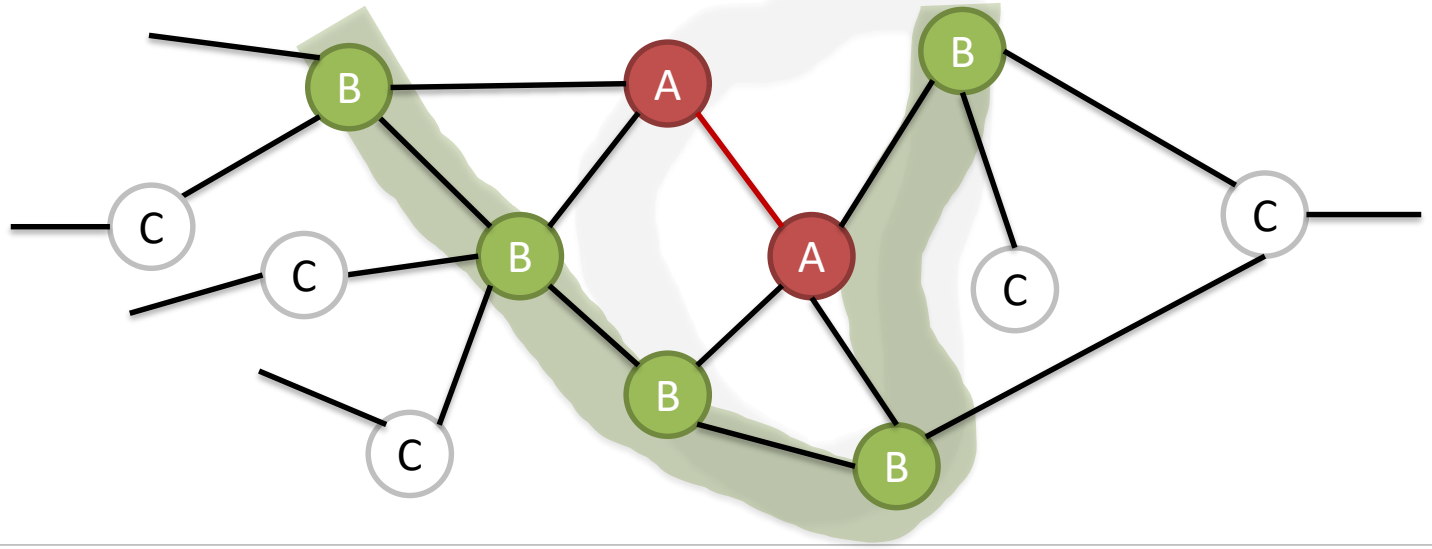
Méthode générale de parcours de graphe

- Méthode générale de parcours de graphe
  - Généralise le DFS aussi bien que le BFS.
- Cette méthode consiste à classer les sommets en 3 catégories:
  - La catégorie A des sommets déjà visités (ceux qui apparaissent dans l'arbre de parcours)
  - La catégorie B des sommets adjacents à ceux de la catégorie A mais pas encore visités (sommets qui peuvent être atteints)
  - La catégorie C des sommets invisibles qui n'ont pas encore été rencontrés du tout (qui ne peuvent pas être atteints depuis un sommet déjà visité)

Parcours de graphe



Les nœuds A et B forment toujours une composante connexe



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

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

Recherche

Explicitation de la solution

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

Algorithme general de recherche

- Catégorie A: Nœuds déjà visités
  - sortis de la liste
  - «ensemble connexe dominant la frange»
- Catégorie B: Nœuds pas encore visités avec voisins visités (en A)
  - Nœuds dans la liste, «frange»
- Catégorie C: nœuds pas encore visités (ni en B)
  - Nœuds jamais passés dans la liste

Un état fera le trajet C-B-A

Noeud et expansion

Un nœud *v* de l'arbre contient:

- L'état *s* correspondant (nœud du graphe)
  → state(*v*)=*s*
- Le parent dans l'arbre (backtracking)
- La profondeur dans l'arbre
- Eventuellement:
  - les enfants
  - le coût du chemin jusqu'à lui (*g*(*v*))

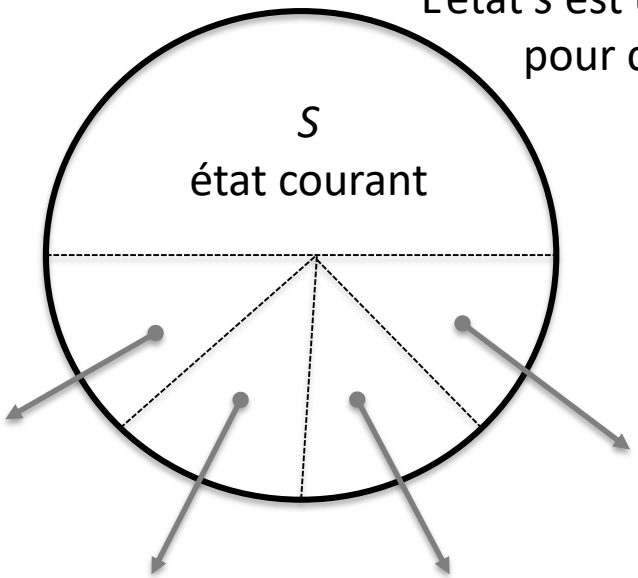
Expansion d'un nœud *v* :

- C'est l'application de la fonction $\Gamma(s)$ sur l'état *s*=state(*v*) correspondant à *v* (modulo la gestion de cycles)

Nœuds du graphe / nœuds de l'arbre

Graphe: nœud *s*

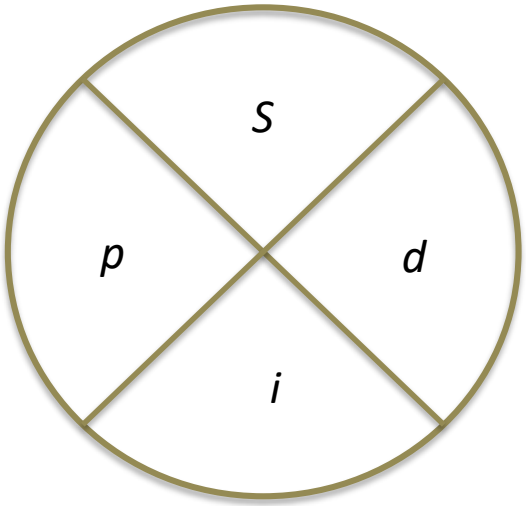
L'état *s* est unique pour chaque nœud



Pointeurs vers les nœuds contenant un état *s'* vers lequel la transition est possible

→ Le graphe peut être représenté par sa matrice de transition ou par Γ

Arbre: nœud *v*



s = state(*v*): état courant (éventuellement pointeur vers le nœud du graphe).
p: parent du nœud dans l arbre
d: profondeur du nœud dans l'arbre
i: informations complémentaires

- Coût jusqu'à *v* = *g*(*v*)
- Enfants de *v* dans l'arbre

Propriétés d'un algorithme de recherche

Complétude:

- Un algorithme est complet si il trouve la solution du problème (en temps fini) si elle existe

Contre-exemple: reste pris dans un cycle

Optimalité (de la solution):

- Un algorithme est optimal si il trouve la meilleure solution (cout minimal) quand il y en a plusieurs

Exemple: plus court chemin

Indicateurs de complexité

- Nombre d'états (taille du graphe)
- Degré moyen du graphe
- Distribution des états solution
- Espace mémoire / état
- Temps d'exécution de la fonction Γ

On verra d'autres indicateurs particuliers à chaque technique de recherché

Mise en pratique

- Recherche aveugle
  - *Uninformed search*

- Recherche heuristique
  - *Informed search*
  - *Heuristic-based search*

Résumé

Certains problèmes peuvent se formaliser sous forme de recherche de chemin dans un graphe d'état

- Notion d'état + action / transition = graphe d'état
- Chemin = solution
- Le chemin est trouvé par un algorithme de recherche

(!) Pas tous les problèmes de recherche se résolvent de cette façon