Résolution d'un problème grace à la recherche dans un graphe

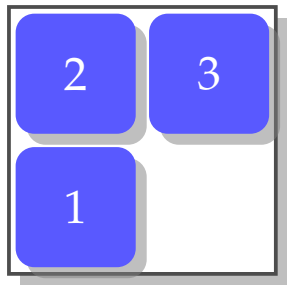
M1 Miage 2017–2018 *Intelligence Artificielle*

Stéphane Airiau

LAMSADE

DAUPHINE
UNIVERSITÉ PARIS

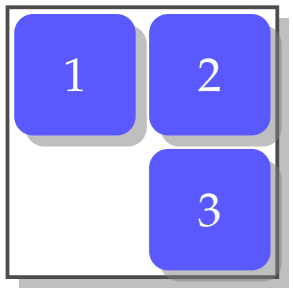
Exemple d'un jeu à résoudre : le taquin



taquin 2×2

Un taquin $n \times n$ est constitué de $n^2 - 1$ jetons carrés à l'intérieur d'un carré pouvant contenir n^2 jetons : on a donc une **case vide**.
Les coups permis sont ceux qui font glisser dans la case vide un des 2, 3 ou 4 jetons contigus.

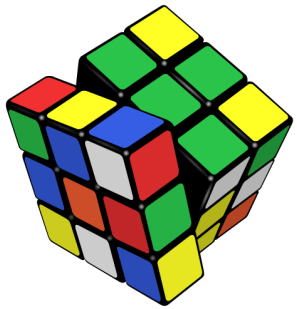
But : partir de l'état du taquin ci-dessus pour arriver à l'état du taquin ci-dessous.

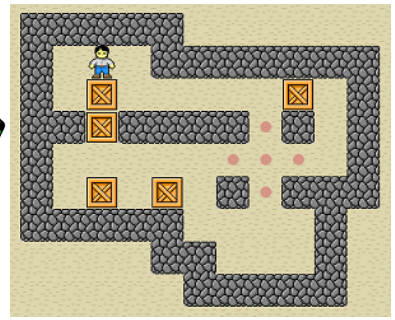


Un programme peut il résoudre ce type de problèmes pour n=2,3,4,5… ?

Jeux dans le même esprit

- poser n reines sur un échiquier de taille $n \times n$ sans qu'aucune reine n'attaque une autre reine.
- résoudre un "rubik's cube"





- résoudre un niveau de "Sokoban"
- jouer "aux chiffres" de l'emission des chiffres et des lettres.
- trouver un itinéraire d'un point A à un point B à l'aide d'une carte
- trouver un itinéraire qui passe exactement une fois par chaque ville

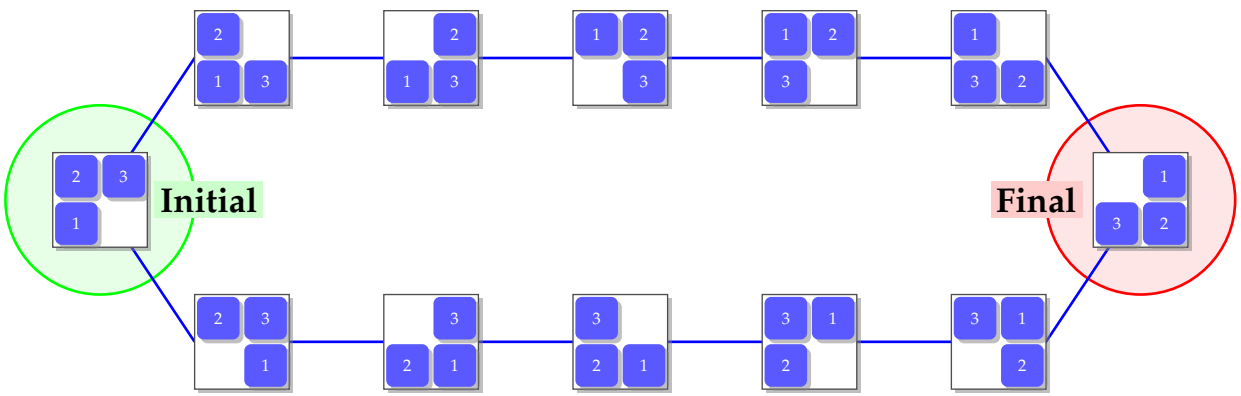
On peut aussi se poser des question **d'optimisation** :

- résoudre le taquin en un minimum de coups
- trouver un itinéraire qui minimise le temps de parcours
- …

Définition du problème : la modélisation

- 1ière étape : décider ce que sont les états et les actions.
- 2nde étape : bien définir le problème avec
  - un état de **départ**.
  - les **actions** possibles pour chaque état
  - le modèle de **transition** : c'est une fonction qui donne l'état qui résulte d'avoir effectué une action depuis un état.

  ⇨ on peut représenter l'état de départ, les actions et le modèle de transition par un **graphe**.
  - une fonction qui teste si le but est **atteint**.
  - éventuellement une fonction de coût pour chaque action dans chaque état.



Graphe pour le taquin 2×2

Définition du problème : la modélisation

- un état de **départ**.
- les **actions** possibles pour chaque état
- le modèle de **transition**
- une fonction qui teste si le but est **atteint**.
- éventuellement une fonction de coût pour chaque action dans chaque état.

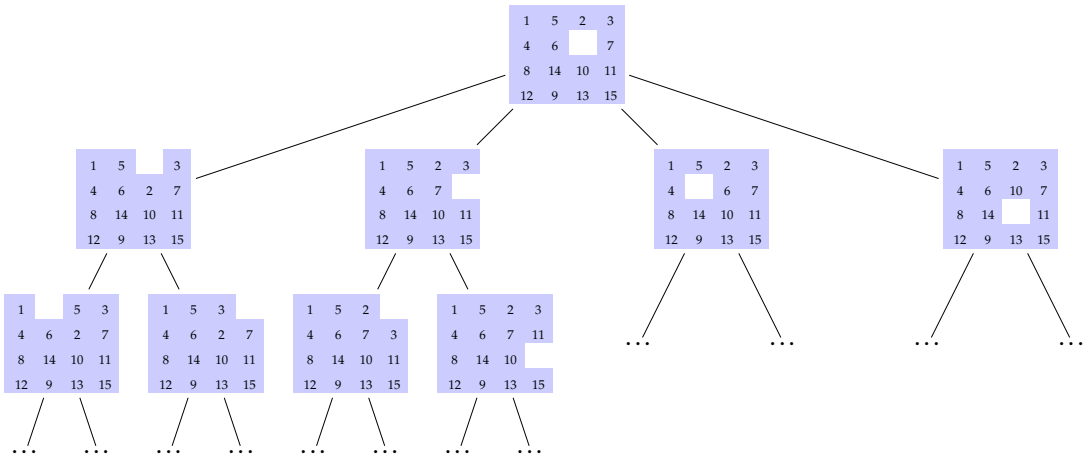
Une fois que le problème est modélisé
⇨ on peut utiliser des algorithmes génériques / solveurs indépendants

⇨ force brute de l'ordinateur
mais une certaine intelligence : avoir une méthode/algorithme qui résoud plein de problèmes
(quelques efforts faits par un humain pour bien modéliser le problème)

Résolution

La séquence d'actions possibles à partir de l'état initial forme un **graphe de recherche** :

- les noeuds représentent les états
- les branches représentent les différentes actions
- la racine est l'état initial

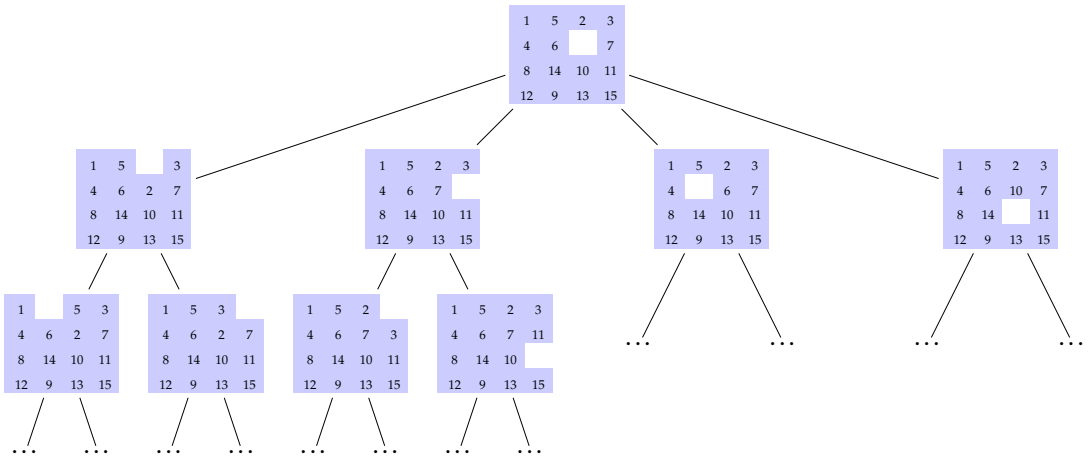


- Pour résoudre : on choisit un noeud à développer et on teste si les fils sont des états finaux.
- Quelle politique pour choisir le noeud à développer ?
- Attention : on peut répéter des noeuds ⇨on peut tourner en boucle (sauf si on se rappelle des états visités) !

Résolution

La séquence d'actions possibles à partir de l'état initial forme un **graphe de recherche** :

- les noeuds représentent les états
- les branches représentent les différentes actions
- la racine est l'état initial



Attention : pour expliquer un algorithme, on dessine/parcourt un graphe **mais** lors de l'exécution de l'algorithme, tout le graphe n'est pas forcément stocké en mémoire.

Taquin $n \times n$

- résoudre un problème de taquin 2×2 a l'air facile !
- Pensons au taquin 4×4
  - Combien y a-t-il de sommets exactement ?
  - ?
  - ⇨ la réponse est un entier proche de 2.10^{13}
  - Certains problèmes sont impossibles (il n'existe pas de chemin de l'état initial à l'état final).
  - en fait pour le 4×4, le graphe possède deux composantes connexes
  - ⇨ si on tire au sort l'état initial et l'état final
    on a 50% de chance qu'une solution existe.
  - il existe un test simple pour le savoir.
- Peut-on utiliser des algorithmes de la théorie des graphes ?

⇨ suppose que le graphe puisse être stocké en mémoire
peut être possible pour 4×4, pas sûr pour 5×5 ou 6×6 !

- on peut utiliser des algorithmes enseignés en cours d'algorithmique.

Résoudre avec des algorithmes de base

- expansion d'un graphe en largeur d'abord
  (**"breadth-first search" (BFS)**)
- expansion d'un graphe en profondeur d'abord
  (**"depth-first search" (DFS)**)
- expansion d'un graphe avec côut uniforme (**"Uniform-cost search"**) :
  développer le noeud qui a le coût le plus bas.
- recherche en profondeur limitée (**"depth-limited search"**) :
  utilise DFS jusqu'à une profondeur l donnée
  cela évite le problème du chemin infini, mais pause problème si la solution est plus profonde que l.
- recherche en profondeur itérative (**"iterative deepening DFS"**) :
  combine DFS et BFS : itérativement utilise DFS jusqu'à une profondeur de 1, 2, … jusqu'à trouver le but.
  les états sont générés de nombreuses fois
- recherche bidirectionnelle (**"Bidirectional search"**) : effectue deux recherche : une de l'état initial vers l'état final, l'autre dans le sens inverse (donc depuis l'état final).

Répétition des états

On ajoutera à ces stratégies la faculté de se souvenir ou non des noeuds visités

- sans mémoire : **“tree-search”** risque que les états se répètent
  ⇨risque de boucle
- avec mémoire : **“graph-search”** nécessite des ressources de stockage
  évite les boucles !

⇨ on ajoute une liste d’états visités (parfois appelée “explored set” ou “closed list”)

Critères pour comparer ces algorithmes

- **Complétude** : a-t-on une garantie de trouver une solution quand elle existe ?
- **Complexité du temps de calcul** : combien de temps a-t-on besoin (dans le pire des cas)
- **Complexité de l'espace mémoire** : combien d'espace mémoire a-t-on besoin (dans le pire des cas)
- **Optimalité** : est-ce-que la solution trouvée est optimale ?

Comparaison

	BFS	Uniform-Cost	DFS	Depth-limited	Iterative-deepening	Bidirectional
complet ?	✔ ? ✘	✔ ? ✘	✔ ? ✘	✔ ? ✘	✔ ? ✘	✔ ? ✘
complexité temps	$\mathcal{O}(?)$	$\mathcal{O}(?)$	$\mathcal{O}(?)$	$\mathcal{O}(?)$	$\mathcal{O}(?)$	$\mathcal{O}(?)$
complexité mémoire	$\mathcal{O}(?)$	$\mathcal{O}(?)$	$\mathcal{O}(?)$	$\mathcal{O}(?)$	$\mathcal{O}(?)$	$\mathcal{O}(?)$
optimal ?	✔ ? ✘	✔ ? ✘	✔ ? ✘	✔ ? ✘	✔ ? ✘	✔ ? ✘

- b sera le facteur de branchement de l'arbre
- d est la profondeur de la solution la moins profonde
- m est la profondeur maximale de l'arbre de recherche
- l est la limite de profondeur