TD1 – Résolution de problèmes à l'aide de graphes d'états

Nathalie Guin & Marie Lefevre

Nous avons vu en cours que pour résoudre un problème, nous pouvons le modéliser de manière à construire un graphe d'états et utiliser des algorithmes permettant de rechercher une ou plusieurs solutions dans ce graphe. Une solution est donc un chemin dans le graphe, constitué de la liste des états par lesquels passer, associée aux opérateurs permettant de passer d'un état à un autre.

Nous avons également vu qu'il existe plusieurs classes de problèmes : les problèmes de satisfaction de contraintes où nous cherchons un état but respectant les contraintes, peu importe le chemin parcouru ; et les problèmes de planification où nous cherchons les étapes à effectuer pour arriver sur un état but connu.

Dans ce premier TD, nous allons donc modéliser un problème de planification et un problème de satisfaction de contraintes de façon à pouvoir construire un graphe d'états et à parcourir ce graphe avec l'algorithme de recherche informée A*.

PARTIE 1 - MODELISATION D'UN PROBLEME DE PLANIFICATION

TAQUIN

Le taquin est une sorte de puzzle où il faut remettre les nombres dans l'ordre. Il est formé d'une grille de N × N cases où sont placées N^2-1 tuiles étiquetées par les nombres 1 à N^2-1, une des cases restant vide. Une des tuiles situées à côté de la case vide peut être déplacée vers cette case. Les déplacements se font donc verticalement ou horizontalement, mais pas en diagonale.

Le taquin est souvent utilisé pour tester les algorithmes de recherche. En augmentant la taille de la grille, les problèmes deviennent de plus en plus complexes. Les algorithmes d'aujourd'hui arrivent à résoudre les taquins 3 × 3 et 4 × 4 (qui ont des espaces d'états respectivement de taille 181 440 et d'environ 1,3 milliards), mais les instances du taquin 5 × 5 (avec un espace d'états de taille 10^{25}) restent difficiles...

Nous souhaitons résoudre un taquin de taille 3 × 3, où les tuiles sont numérotées de 1 à 8. Les états initial et final sont donnés sur les figures ci-dessous.

2	8	3
1	6	4
7		5

État initial

1	2	3
8		4
7	6	5

État final

Q1 : Modélisez ce problème selon le canevas présenté en cours :

Pour rappel, le canevas de modélisation d'un problème est :

{ États / État initial / Opérateurs (action, condition d'application et fonction de successeur) / Test de but / Fonction de coût (simple ou avec heuristique) }

Q2 : Dessinez les 3 premiers niveaux du graphe d'états.

Q3 : Proposez une heuristique permettant de guider la recherche.

========== Indices de correction ==========

Modélisation n°1 de ce problème :

États : Les états sont des configurations des huit tuiles numérotées de 1 à 8 et de la tuile vide dans les neuf cases de la grille 3x3.

État initial : N'importe quel état pourrait être choisi comme l'état initial, on prend celui de l'énoncé.

Opérateur : Il y a 4 actions possibles correspondant aux quatre façons de **changer la position du carré vide** : haut, bas, gauche, droite. Dans certaines configurations, il n'y aura que 2 ou 3 actions possibles (définir les conditions d'application correspondantes).

Fonction de successeur : Cette fonction spécifie les états résultants des différentes actions. Par exemple, la fonction va nous dire que l'exécution de l'action droite dans l'état de la figure gauche ci-dessous produira l'état de la figure droite ci-dessous.

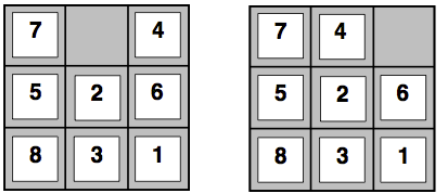


FIG. 1 – Deux états du jeu de taquin.

Test de but : L'état but est unique et fixé au début du jeu (n'importe quel état peut être choisi comme état but, même si en pratique il s'agit de remettre les nombres dans l'ordre).

Coût des actions : Chaque déplacement d'une tuile a coût de 1 afin de trouver une solution avec le moins de déplacements.

Cette première modélisation n'est pas assez formalisée pour pouvoir être implémentée sans ambiguïté… il faut donc reprendre tout cela et être plus formel et plus précis comme dans la modélisation n°2.

Modélisation n°2 de ce problème :

États : 8 variables (une par élément du taquin) avec pour valeur le couple (i,j) de ses coordonnées sur le tableau et une variable pour le trou (VE), où i ∈[1 ;3] et j ∈[1 ;3]

	1	2	3
1	2	8	3
2	1	6	4
3	7		5

Sur le cas présenté dans l'énoncé :

État initial : V1=(1,2), V2=(1,1), V3=(3,1), V4=(3,2), V5=(3,3), V6=(2,2), V7=(1,3), V8=(2,1),**VE=(2,3)**

État but : V1=(1,1), V2=(2,1), V3=(3,1), V4=(3,2), V5=(3,3), V6=(2,3), V7=(1,3), V8=(2,3), **VE=(2,2)**

Opérateurs : (il existe d'autres possibilités, l'important est que les étudiants aient compris la notion d'opérateurs)

Les opérateurs les plus simples consistent à **déplacer le « trou »**

H : déplacer vers le haut le trou. Condition : le trou ne doit pas être sur la ligne 1. Dans l'état, c'est la variable représentant la case juste au-dessus qui change de valeur. Plus formellement :

Condition : VE=(C,L) avec L >1 (C = colonne, L = ligne)

Modification de l'état :

- VE [qui était égal à (C,L)] <- VE=(C,L-1)[le « vide » monte d'une ligne, ne change pas de colonne] ;

- Vk [qui est la pièce qui descend, avait pour valeur (C,L-1)] <- Vk=(C,L [elle descend d'une ligne, ne change pas de colonne])

(les autres variables ne changent pas)

B : déplacer vers le bas le trou : ….

D : déplacer vers la droite le trou : ….

G : déplacer vers la gauche le trou : …..

Sans heuristique, il faut explorer tout l'espace des états

A chaque état, on peut (ou non) appliquer un opérateur et on recommence avec l'état obtenu. Si l'état final est obtenu, on a satisfait le but et la solution est le chemin constitué par la succession des opérateurs à partir de l'état initial. Si on veut trouver la solution optimale, il faut continuer à explorer l'espace des états et c'est le chemin le plus court qui est optimal.

Différentes heuristiques :

Heuristique W : nombre de cases non encore à la bonne place

Cette heuristique est minorante et monotone. Elle consiste à prendre h(e) le nombre de cases qui ne sont pas, dans l'état e, à la place exigée par l'état but b. Cette heuristique est simple mais elle ne tient pas compte du nombre de coups pour amener une case pleine à sa destination dans l'état but b selon qu'elle en est plus ou moins éloignée.

Heuristique P : somme des distances « en pâté de maison »

On utilise donc la distance « pâté de maison » (en anglais « city-block » ou « Manhattan » distance). Cette heuristique est également minorante et monotone ; elle n'est pas formellement mieux informée que W mais l'expérience confirme qu'elle est beaucoup plus efficace. Si Le(c) et Ce(c) (resp. Lb(c) et Cb(c) désignent les numéros de la ligne et de la colonne de la case c dans l'état e (resp. dans l'état but), on définit : $d(c, n, b) = | Le(c) – Lb(c) | + | Ce(c) – Cb(c) |$

Alors h(n) est égal à la somme des distance d(x, e, b) des cases pleines. P est plus fine que W, mais ne tient pas compte de la difficulté à inverser deux cases voisines.

Heuristique P + 3S

Contrairement à W et P, cette heuristique ne peut s'utiliser que dans le cas du taquin 3x3. Pour un état e, on définit d'abord S de la façon suivante :

- On choisit un sens de parcours des cases non centrales autour de la case centrale. On ne considère que les cases non centrales différentes de la case vide. A chacune d'entre elles, on attribue :
  - Un poids de 0 si elle est aussi non centrale dans l'état but b et si dans les deux états (e et b) la case suivante (dans le sens de parcours choisi) est la même.
  - 2 sinon.
- A la case centrale, si elle est pleine dans l'état e, on attribue :
  - 0 si elle est bien placée par rapport à b ; 1 sinon.

On a donc S(e) = somme des poids des cases pleines dans l'état e et h(e) = P(e) + 3 * S(e).

==========

PARTIE 2 – RECHERCHE HEURISTIQUE DANS UN GRAPHE D'ETATS

Nous avons vu en cours qu'il existe plusieurs algorithmes pour rechercher une solution dans un graphe d'états. Dans ce TD, nous allons travailler sur l'algorithme A* qui est un algorithme de recherche informée.

L'ALGORITHME A*

Algorithme A*

1. Initialisation : OUVERTS ← u_0 ; FERMES ← Ø ; g(u_0) ← 0 ; u ← u_0

```
2.     Itérer tant que [OUVERTS ≠ Ø et u non terminal]
2.1       Supprimer u de OUVERTS et le mettre dans FERMES
2.2       Itérer sur les nœuds v successeurs de u
        Si [v ∉ (OUVERTS ∪ FERMES) ou g(v) > g(u) + coût(u, v)] Alors faire :
           g(v) ← g(u) + coût(u, v)
           f(v) ← g(v) + h(v)
           père(v) ← u
           Ranger v dans OUVERTS, dans l'ordre f croissant, puis g décroissant
     Fin Itération 2.2
2.3       Si OUVERTS ≠ Ø Alors u ← tête(OUVERTS)
          Fin Itération 2
3.     Si OUVERTS = Ø   Alors le problème n'admet pas de solution
                        Sinon fournir la solution chemin(u)
```

Q4 : Appliquez cet algorithme au problème du taquin tel que vous l'avez modélisé dans la partie 1, avec au moins une heuristique. Il s'agit de faire « tourner à la main » l'algorithme en traçant les différentes structures et variables utilisées.

PARTIE 3 - MODELISATION D'UN PROBLEME DE SATISFACTION DE CONTRAINTES

THEOREME DES QUATRE COULEURS

Le théorème des quatre couleurs indique qu'il est possible, en n'utilisant que quatre couleurs différentes, de colorer n'importe quelle carte découpée en régions connexes, de sorte que deux régions limitrophes, c'est-à-dire ayant une frontière (et non simplement un point) en commun, reçoivent toujours deux couleurs distinctes.

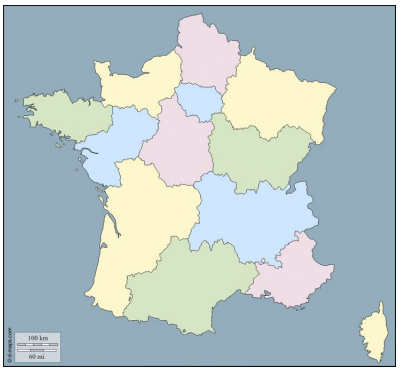
Nous souhaitons donc colorier une carte des 13 régions métropolitaines françaises en utilisant uniquement quatre couleurs.

Q5 : Modélisez ce problème selon le canevas présenté en cours.

Q6 : Dessinez les 3 premiers niveaux du graphe d'états.

Q7 : Proposez une heuristique permettant de guider la recherche.

Q8 : Appliquez l'algorithme A* à ce problème.



========== Indices de correction ==========

Modélisation 1 :

États : 13 variables correspondants aux 13 régions, soit vide, soit avec une des 4 couleurs affectées

État initial : 13 variables vides

Opérateurs (action, condition et fonction de successeur) : affecter une couleur à une variable vide, sans aucun test de conformité

Test de but : les régions sont toutes coloriées et les adjacentes ont toutes des couleurs différentes

Fonction de coût (simple ou avec heuristique) :

- Simple : 1 par affectation
- **Heuristique du degré de saturation** :
  - DSAT(v) = nombre de couleurs différentes utilisées pour colorier les régions adjacentes.
  - Prendre comme heuristique 1/DSAT(v) ou [NombreDeCouleurs (ici 4) – DSAT(v)] afin de choisir les régions de degré maximum à colorer en 1er en prenant l'heuristique la plus faible

Modélisation 2 (graphes plus petit) :

États : 13 variables correspondants aux 13 régions, soit vide, soit avec une des 4 couleurs affectées

État initial : 13 variables vides

Opérateurs : affecter une couleur à une variable vide, avec comme condition de validité de l'opérateur : pas de couleur identique dans les cases adjacentes

Test de but : les régions sont coloriées (et c'est tout, le reste a été vérifié avant)

Fonction de coût (simple ou avec heuristique) :

- Simple : 1 par affectation
- Heuristique du degré de saturation : idem à la première modélisation

⇨ **Mettre en avant les avantages de la 2nde modélisation sur la 1ère.**
⇨ D'autres modélisations sont possibles.

⇨ **METTRE EN AVANT QUE A* N'EST PAS ADAPTE POUR CE TYPE DE PROBLEME car le nombre de successeurs à chaque étape est beaucoup trop important**
⇨ **Nous retraiterons ce problème sous forme de CSP**

==========