Course Name — TDs

Ivan Lejeune

15 décembre 2025

Table des matières

TD1 — Résolution de problèmes — Modélisation et recherche aveugle

Exercice 1.1 Inspiré d'un exercice du AIMA. Le problème des missionnaires et des cannibales est le suivant : 3 missionnaires et 3 cannibales sont d'un côté d'une rivière, avec une barque qui peut transporter 1 ou 2 personnes à la fois. Trouver une façon de faire passer tout ce monde de l'autre côté de la rivière, sans jamais laisser en un lieu des missionnaires en minorité par rapport aux cannibales (auquel cas il ne resterait plus rien des missionnaires).

1. Formaliser le problème sous la forme de parcours d'un problème d'exploration d'un espace d'états :
   - Comment représenter un état de problème ? Combien d'états « potentiels » votre représentation permet-elle de définir ?
   - Tous les états potentiels sont-ils des états « valides » du problème, c'est-à-dire correspondent à une situation où les missionnaires ne sont pas en minorité ?
   - Quel est l'état initial du problème ?
   - Quelles actions sont applicables à chaque état (attention une action ne doit conduire qu'à un état valide du problème) ?
   - Tous les états valides sont-ils accessibles de l'état initial ?
   - Qu'est-ce qu'un état but ?
   - Quelle pourrait être une fonction de coût pour ce problème ?
2. Dessiner l'espace d'états de ce problème. Finalement, quelle est la taille de cet espace d'états ? On rappelle que les états d'un problème sont définis comme l'ensemble des états atteignables depuis l'état initial.
3. Donner une solution optimale (i.e. de coût minimal) en terme de nombre d'actions.
4. Quel sera le facteur de branchement d'une stratégie de recherche sur ce problème ?
5. Une recherche en largeur sur ce problème permet-elle de trouver une solution ? Si oui, à quelle profondeur et sera-t-elle une solution optimale. Précisez si vos réponses dépendent de l'ordre d'exploration des successeurs d'un noeud.
6. Une recherche en profondeur sur ce problème permet-elle de trouver une solution ? Si oui, à quelle profondeur et sera-t-elle une solution optimale. Précisez si vos réponses dépendent de l'ordre d'exploration des successeurs d'un noeud.

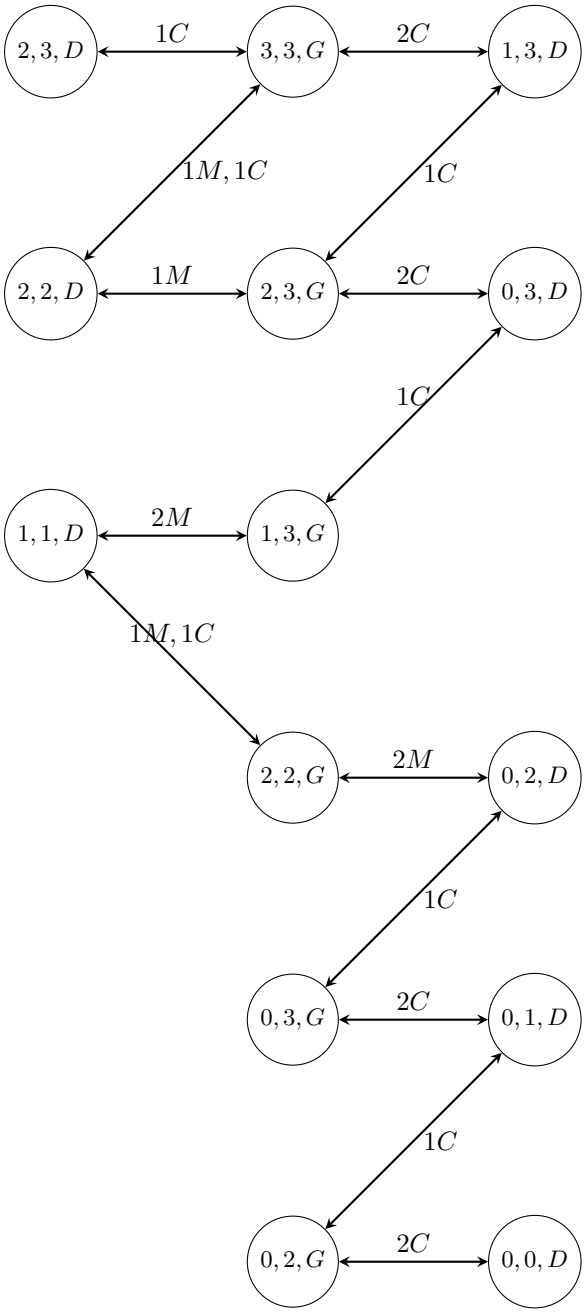
Solution. On commence par formaliser nos etats. Voici le tableau représentatif des etats : Le nombre d'etats potentiels est donc de $4 \times 4 \times 2 = 32$.
Les contraintes de validité sont les suivantes :

- $CG \leq MG$ (i.e. il y a moins de cannibales que de missionnaires à gauche),
- $CD \leq MD$ (avec $CD = 3 - CG$ et $MD = 3 - MG$).

L'état initial est $(3, 3, G)$ et l'état but est $(0, 0, D)$.
Les actions possibles sont :

- traverser avec $2C$ (si possible),
- traverser avec $1C$ (si possible),
- traverser avec $2M$ (si possible),
- traverser avec $1M$ (si possible),
- traverser avec $(1M, 1C)$ (si possible).

Une fonction de coût possible est le nombre de traversées effectuées.
Le graphe des états est le suivant :



Exercice 1.2. On dispose de 3 cubes A, B et C sur une table. Un cube peut être soit directement sur la table, soit sur un autre cube. On est dans la situation où les cubes A et B sont à même la table et le cube C est posé sur le cube A. On cherche à obtenir la situation où les 3 cubes sont empilés de la manière suivante :

- le cube A est sur le cube C,
- le cube C est sur le cube B,
- le cube B est sur la table.

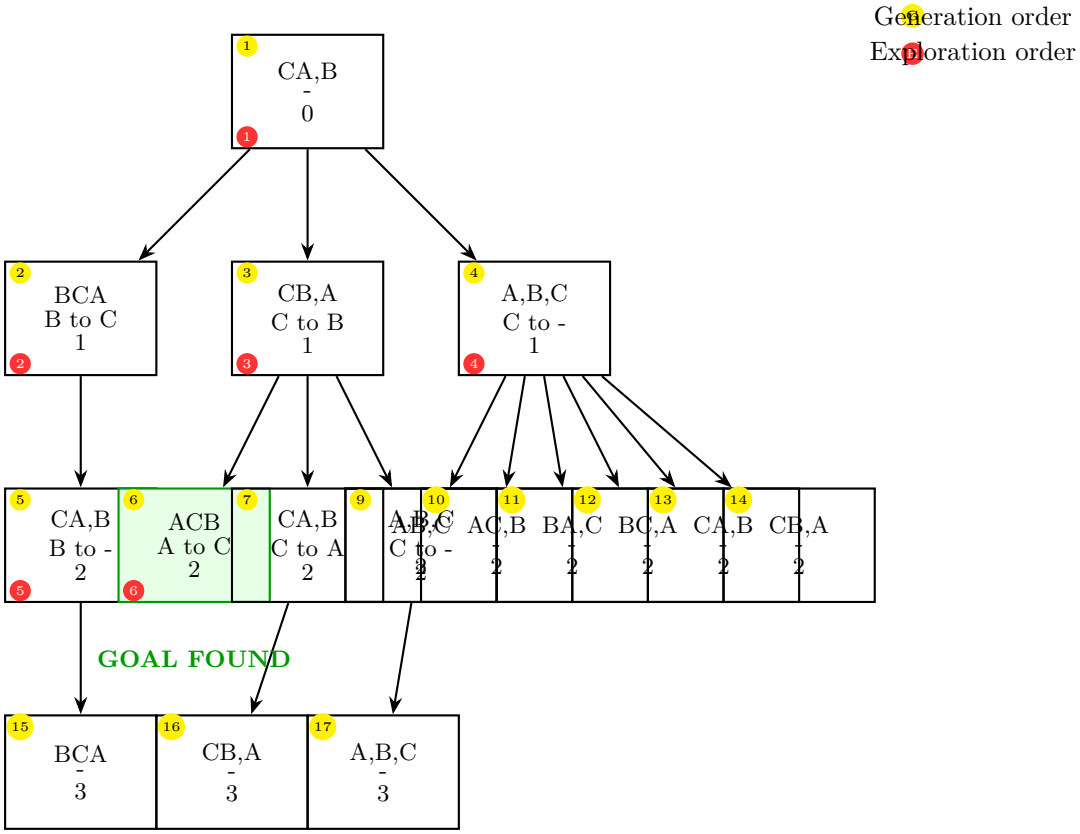
La seule action possible est de déplacer un cube en haut de pile soit sur la table soit sur une autre pile.

1. Formaliser le problème en termes d'états, d'actions, de but et de coût.
2. Dessiner le graphe des états (représentant l'espace des états).

3. Quelle est la taille de l'espace des états ?
4. Quel sera le facteur de branchement d'une stratégie de recherche sur ce problème ?
5. Que faut-il préciser dans l'algorithme général pour mettre en place une stratégie en largeur ? Appliquer l'algorithme de recherche en largeur (pour l'ordre des actions issues d'un état (cf. liste retournée par XXINSERTCODEHEREXX), on considèrera en priorité les actions qui créent des empilemments plus hauts puis l'ordre alphabétique en cas d'égalité). Préciser l'ordre de génération et d'exploration de chaque noed. Combien de noeuds sont générés et explorés ? Combien de fois retrouve-t-on l'état initial dans un noeud généré ? Dans un noeud exploré ?
6. Mêmes questions pour la recherche en profondeur. Quel problème cela pose-t-il ?
7. On décide maintenant d'ajouter à l'algorithme général de recherche un test évitant de réexplorer un état déjà exploré. Modifier l'algorithme en conséquence. Discuter de l'influence de votre modification sur les stratégies en largeur et en profondeur : taille de l'arbre de recherche, de la frontière, ordre d'exploration, complétude de la stratégie, optimalité de la solution trouvée.
8. Combien de noeuds sont alors générés et explorés par cet algorithme optimisé de recherche en largeur.
9. Même question pour la recherche en profondeur.
10. Proposer une version récursive de l'algorithme (non optimisé) de recherche en profondeur.
11. Combien de noeuds sont alors générés et explorés par cette version ?
12. On dit que la recherche en profondeur a une complexité spatiale en $\mathcal{O}(b, d_m)$ où b est le facteur de branchement et d_m la profondeur maximale à laquelle une solution est cherchée. Si le langage d'implémentation de l'algorithme ne dispose pas d'un système de gestion dynamique de la mémoire par « ramasse-miettes », quelle instruction de suppression faut-il ajouter à l'algorithme récursif pour assurer cette complexité spatiale ?
13. On considère à nouveau l'algorithme de recherche en profondeur récursif. Adapter cet algorithme pour restaurer la complétude de la recherche sans mémoriser d'états précédemment explorés. On distinguera le cas où l'on peut borner la taille de l'espace d'états du cas où l'espace d'états est infini.

⋆ Pour s'entrainer on pourra reprendre intégralement cet exercice en choisissant comme but la situation où le cube A est sur le cube B, le cube C étant seul à coté.

Solution. Notre tableau des etats :
L'état initial est (T, T, A) et l'état but est (C, T, B).
Le nombre d'états est 13, le facteur de branchement est 6.
Le graphe peut aussi ressembler à :



On marquera en jaune les étants de génération et en rouge ceux d'exploration.
On a une fonction qui explore un problème en profondeur et rend un noeux. Voici son implementation :

Fonction ExplorerProfondeur(p : Problème) : Noeud
racine ← NouveauNoeud(p.etatInitial, $null$, $null$, 0)
retourner ExplorerRec(racine);

Fonction ExplorerRec(n : Noeud) : Noeud
si p.but?(n.etat) alors
    retourner n;
sinon
    pour chaque a dans p.actions(n.etat) faire
        res ← ExplorerRec(NouveauNoeud(a.resultat(n.etat), n, a, n.cout + a.cout));
        si res ≠ null alors retourner res;
    fin pour
    retourner null;

Une fois de plus, pour éviter de ré-explorer des états, on suppose disposer d'une fonction qui teste l'appartenance d'un état à ses parents.
On modifie donc l'algorithme précédent en rajoutant la ligne suivante au début de la boucle :

$e' \leftarrow a$.resultat(n.etat); si non (n.AppartientEtatAncetres(e', n)) alors ajouter (e', n) à la liste des états explo

On aurait aussi pu la rajouter avant l'appel récursif avec la modification suivante :

si AppartientEtatAncetres(n.etat, n.parent) alors retourner $null$;

Supposons maintenant qu'on utilise cette version optimisée (la première), alors on trouve la solution après seulement 4 noeuds générés et explorés.

Exercice 1.3 Taquin. Modéliser le jeu du Taquin comme un problème d'exploration d'espace d'états.

Solution.

TD2 — Recherche de solution optimale

Soient les structures et fonctions générales de recherche données en cours permettant de parcourir les états d'un espace d'états par priorité croissante. On précise que :

- Explorer peut ré-explorer plusieurs fois un même état,
- ExplorerOptimise évite de ré-explorer un état déjà exploré et ne conserve dans la frontière qu'un noeud par état.

Exercice 2.1. Préciser pour chacune des stratégies suivantes comment doivent être choisis les coûts et priorités c_0, c_{sn}, p_0 et p_{sn} des deux fonctions Explorer et ExplorerOptimise.

- Recherche par coût minimal (le coût étant défini comme la somme du coût des actions ayant conduit de l'état du noeud racine à l'état du noeud courant).
- La recherche gloutonne.
- La recherche avec l'algorithme A^*.

Solution. On construit le tableau suivant :

	c_0	p_0	c_{sn}	p_{sn}
coût min	0	k quelconque	$c(n) + \text{cout}(a)$	$c(n) + \text{cout}(a)$
glouton	0	k' quelconque	$c(n) + \text{cout}(a)$	$h(a.\text{res}(x.\text{etat}))(x)$
A^*	0	$h(e_0)$	$c(n) + \text{cout}(a)$	$h(se) + c_{sn}$

Maintenant pour la recherche gloutonne :
Le cout de cette solution est de 450 alors que la solution optimale est de 411 (à vérifier).
Et enfin pour A^* :

Exercice 2.2. Soit le problème de calcul d'une route de *Arad* à *Bucharest* (cf. document « Problème de recherche de route » sur Moodle). Dessiner les arbres de recherche correspondant à l'exécution de ces 3 stratégies en distinguant le cas où la fonction Explorer est utilisée de celui où l'on utilise la fonction ExplorerOptimise.

Solution. A faire

Exercice 2.3. Même question de *Iasi* à *Fagaras*. On pourra prendre les distances à vol d'oiseau suivantes pour Fagaras :

Solution. Pour le glouton non optimisé on a : Pour le glouton optimisé on a :

Exercice 2.4. Dire pour chacune des deux versions de l'algorithme d'exploration, si les 3 stratégies précédentes sont complètes. Préciser éventuellement sous quelles conditions elles le sont.

Solution. Synthèse sur la complétude : où on suppose que :

- le cout de toute action est positif,
- $\forall n, g(n) = \sum_{i \in \text{branche}(n)} \text{cout}(i)$,
- l'heuristique de tout etat est positif,
- l'heuristique de l'etat but est nulle.

Synthèse sur l'optimalité où on suppose qu'on a déjà la complétude

Exercice 2.5. Une heuristique est dite **admissible** si pour tout état e elle ne sur-estime jamais le cout du chemin optimal de e à l'état but :

$$\forall e, h(e) \leq g*(e)$$

où $g*(e)$ dénote le cout d'un des chemins optimaux de e à l'état but le plus proche.
On considère la stratégie A^* avec l'algorithme Explorer.

1. Exhiber un exemple montrant que si l'heuristique utilisée n'est pas admissible alors la solution trouvée peut ne pas être optimale.
2. Montrer que si l'heuristique est admissible alors la solution trouvée est optimale.
3. Est-ce vrai pour l'algorithme ExplorerOptimise ?

Solution.

1. Il suffit de prendre le graphe suivant : Clairement le chemin optimal est (I, B, D) mais A^* fait (I, C, D).
2. Montrons par l'absurde que l'état but trouvé par A^* est une solution optimale, quand h est admissible.
   Supposons que le noeud but trouvé par A^* est non optimal. Alors,
   $$g(b) > g^*$$
   (le coût de b est supérieur au cout optimal). Soit n un noeud non exploré (mais généré et donc une feuille de l'arbre de recherche) sur le chemin conduisant à une solution optimale. Alors, $g(n) \leq g^*$ car les couts sont croissant le long d'une branche. Donc $g(n) < g(b)$ par hypothèse. De plus, on a h admissible, donc
   $$\forall e, h(e) \leq g^*(e)$$
   On note e_n l'état du noeud n. Alors
   $$h(e_n) \leq g^*(e_n)$$
   Comme n est sur un chemin optimal vers un but, on a
   $$g(n) + g^*(e_n) = g^*$$
   et donc
   $$f(n) = g(n) + h(n) \leq g(n) + g^*(e_n) = g^*$$
   Or, comme b a été trouvé par $A^\star$, on sait que
   $$f(b) \leq f(n) \leq g^*$$
   mais $f(b) > g^*$. Contradiction, donc b est optimal.

Exercice 2.6. A faire

Solution. A faire

Exercice 2.7. A faire

Solution.

1. L'espace d'états est le suivant :
   - Les états sont des cases,
   - l'état initial est $(A, 2)$ et
   - le but est $(D, 4)$.
2. Voici le tableau des heuristiques : Ces deux heuristiques sont bien admissibles et monotones.
3. Clairement ici $h_{\text{manhattan}}$ domine h_{vol}

Exercice 2.8. A faire

Solution. A faire

TD3 — Satisfaction de contraintes

Exercice 3.1 Le puzzle du zèbre. Le puzzle du zèbre est un jeu logique bien connu, atribué à Albert Einstein ou à Lewis Caroll, sans certitude que l'inventeur soit l'un des deux. Il existe plusieurs variantes de ce jeu, voici l'énoncé d'origine.
Il faut aussi ajouter que les maisons sont supposées être sur une ligne. La question « qui boit de l'eau » doit être comprise comme « sachant que quelqu'un boit de leau, qui est-ce ? » (sinon on peut trouver une solution où personne ne boit de l'eau). De même la question « qui possède le zèbre » doit être comprise comme « sachant que quelqu'un possède le zèbre, qui est-ce ? ». Si on sait que quelqu'un boit de l'eau et que quelqu'un possède un zèbre, on peut en fait déterminer qui vit où, la couleur de sa maison, sa nationalité, ce qu'il boit et fume et son animal de compagnie.
Modéliser ce problème comme un problème de satisfaction de contraintes.
Quelle est la taille de l'espace de recherche ?
Peut-on reformuler certaines contraintes pour diminuer l'espace de recherche ?

Solution. On a 25 variables représentant les caractéristiques de chaque maison. Plus précisément, on aura 5 variables représentant chacune la

1. nationalité,
2. la couleur,
3. la boisson,
4. ce qui est fumé,
5. l'animal de compagnie.

des 5 maisons. Pour différencier ces maisons, on leur fixe une position prédéfinie de 1 à 5 (ie. on attribue un ordre aux maisons dont on pourra se servir plus tard).
Pour ce qui est des domaines des variables, ce sont celles énoncées dans les contraintes. 5 valeurs par domaine, 5 domaines, tous différent.
Une contrainte est représentée comme suit :
C_2 : l'anglais habite dans la maison rouge :

$$C = \{c_1\langle N_1, C_1\rangle = \{(E, R), (J, \neg R), (N, \neg R), \cdots\}\}$$

donc 17 tuples de contraintes :D. Pour la maison 1 et encore une fois pour chaque autre maison, donc 85 en tout, par contrainte.
Une autre manière de les représenter serait la suivante :
5 contraintes pour chaque i de la forme

$$N_i = E \iff C_i = R$$

On peut modeliser la majorité des contraintes de la même manière. Certaines sont différentes, comme celle-ci :

$$C_i = I \iff C_{i-1} = G$$

pour $i \in \{2, 3, 4, 5\}$.
Pour modéliser la différence entre toutes les variables, on ajoute

$$\begin{aligned}
&\text{Alldiff}(C_1, C_2, C_3, C_4, C_5)\\
&\text{Alldiff}(N_1, N_2, N_3, N_4, N_5)\\
&\text{Alldiff}(B_1, B_2, B_3, B_4, B_5)\\
&\text{Alldiff}(S_1, S_2, S_3, S_4, S_5)\\
&\text{Alldiff}(P_1, P_2, P_3, P_4, P_5)
\end{aligned}$$

qui correspondent à 10 contraintes binaires chacune.

Cette implémentation et description des contraintes est très lourde et longue à écrire. Voici une autre approche possible où on inverse le rôle des valeurs et des variables.

$$X = \{E, S, U, N, J, \quad I, B, G, R, Y, \quad \cdots\}\,.$$

Ensuite, pour décrire l'unicité de chaque valeur, on utilise encore une contrainte Alldiff :

$$\text{Alldiff}(E, S, U, N, J), \quad \text{Alldiff}(I, B, G, R, Y), \quad \cdots$$

On peut alors décrire des contraintes comme suit : « Verte à droite de l'ivoire » :

$$C_6 = \{(I, G) \mid G = I + 1\} = \{(1, 2), (2, 3), (3, 4), (4, 5)\}$$

ou encore : « Chesterfield à coté de la maison du renard » :

$$C_{11} = \{(C, R) \mid |C - R| = 1\} = \{(1, 2), (2, 1), (2, 3), (3, 2), (3, 4), (4, 3), (4, 5), (5, 4)\}$$