Correction TD IA - 2026

Équipe Pédagogique IA

1 TD3 : Recherche Heuristique

Exercice 3 : Le jeu du Taquin et A*

Rappel des notations :

$h_1(n)$: Nombre de tuiles mal placées (l'espace vide n'est pas compté).

$h_2(n)$: Distance de Manhattan ($\sum |x_{actuel} - x_{but}| + |y_{actuel} - y_{but}|$).

$c(n)$: Coût réel du chemin depuis la racine (ici, la profondeur dans l'arbre).

$f(n)$: Fonction d'évaluation globale pour A*, définie par $f(n) = c(n) + h(n)$.

Proposition de Corrigé

1. Déroulement sur l'arbre de recherche

L'image ci-dessous détaille le développement des premiers niveaux de l'arbre. Pour chaque état, nous indiquons le coût c (profondeur) et les valeurs des deux heuristiques.

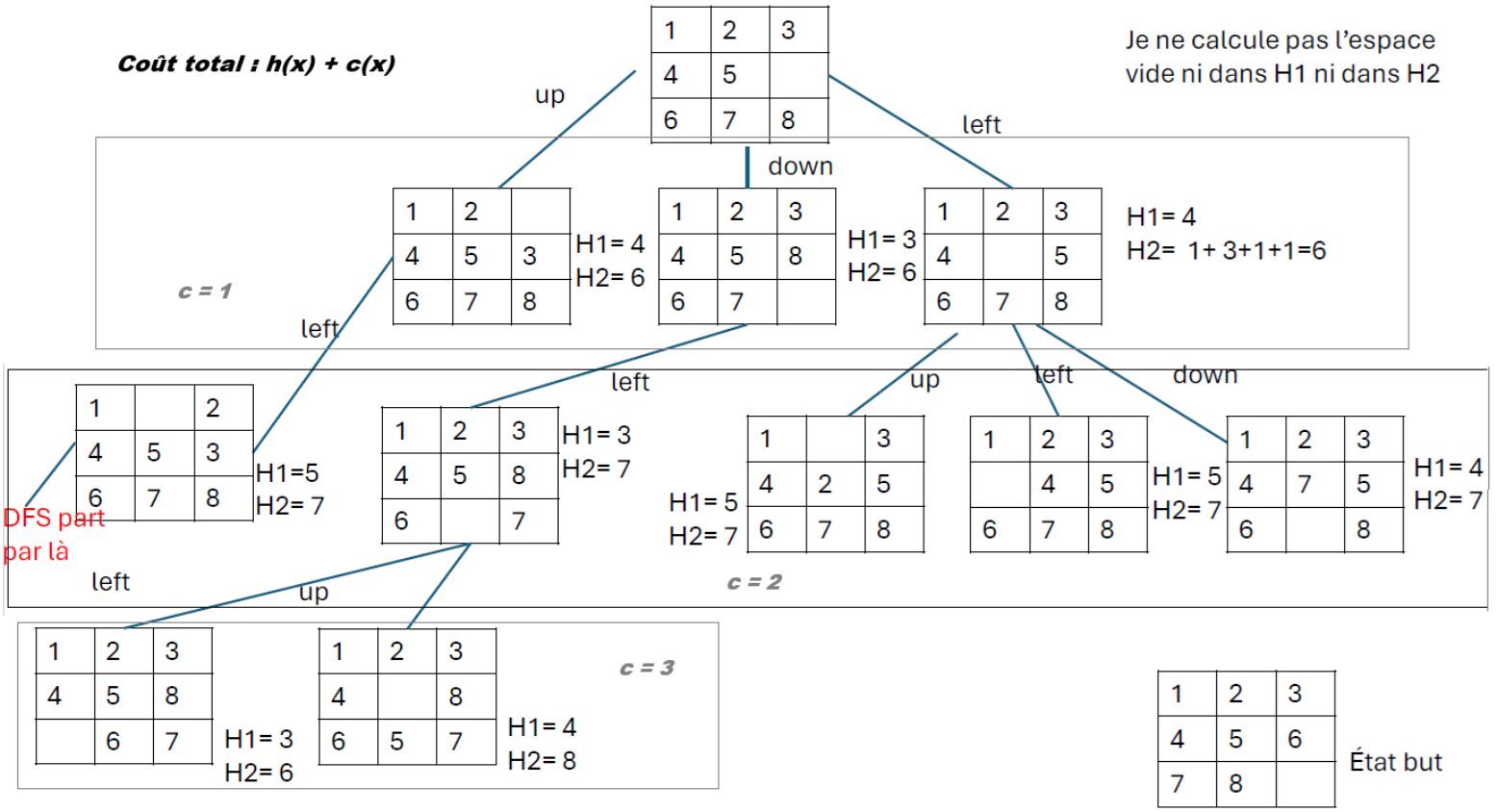

FIGURE 1 – Arbre de recherche du Taquin : comparaison des coûts (c) et heuristiques (h_1 vs h_2).

2. Analyse et comparaison des stratégies

Rappel théorique : Le rôle clé de l'heuristique

L'heuristique $h(n)$ est une estimation du coût restant pour atteindre le but. C'est elle qui permet de passer d'une exploration aveugle à une exploration intelligente.

— **Le but (Pruning) :** L'heuristique sert à anticiper le coût d'un chemin pour **élaguer** (pruner) les branches qui mènent à des solutions manifestement trop coûteuses.
Paradoxe : Si on connaissait la vraie distance exacte, le problème serait résolu. L'heuristique est donc utile précisément quand on ne connaît pas la solution, mais qu'on sait estimer "à peu près" une distance minimale approximative => Je sais pas combien de coups il me faudra encore faire, mais je sais qu'il m'en faudra au moins X (plus les c coups déjà effectués dans le passé : eux, je connais la vraie valeur donc je l'utilise).

— **La règle d'or (Optimisme/Admissibilité) :** Pour garantir que l'algorithme ne jette pas par erreur la meilleure solution, l'estimation doit être **optimiste** (ou admissible). Elle ne doit **jamais surestimer** le coût réel ($h(n) \leq h^*(n)$).
Risque : Si l'heuristique est pessimiste (trop élevée), l'algorithme va croire à tort que le chemin optimal est trop long et risque de le couper (supprimant ainsi la vraie solution de l'arbre).

— **L'efficacité (Tightness) :** Pour être efficace, l'heuristique doit être aussi "serrée" (*tight*) que possible, c'est-à-dire aussi proche du coût réel que possible sans jamais le dépasser.
→ Si h est trop basse (ex : 0), on n'élague rien (retour à BFS).
→ Si h est proche du réel, on élague massivement et on trouve la solution très vite.
→ Si h est supérieur au réel, on va élaguer la vraie solution et on ne convergera jamais. INTERDIT

En analysant l'arbre de la figure 1, voici comment chaque algorithme se comporte :

— **DFS (Depth-First Search) :** Comme illustré par l'annotation "DFS part par là", cet algorithme fonctionne de manière "impulsive". Il choisit le premier opérateur disponible (ex : *Left*) et explore cette branche en profondeur.
→ *Inconvénient :* Il ignore si l'état est prometteur. Il n'est ni complet (peut boucler), ni optimal.

— **BFS (Breadth-First Search) :** Il explorerait l'arbre niveau par niveau (tous les nœuds à $c = 1$, puis tous ceux à $c = 2$, etc.).
→ *Inconvénient :* Bien qu'optimal, l'exploration "en cercles concentriques" est très coûteuse en mémoire et en temps.

— **A* (A-Star) avec h_1 (Tuiles mal placées) :** Il utilise l'estimation $f(n) = c(n) + h_1(n)$.
Détail sur l'exemple :

1. **Racine (S_0)** : $c = 0, h_1 = 3 \Rightarrow \mathbf{f = 3}$.
2. **Niveau 1** : Parmi les fils, certains ont $h_1 = 4$ (donc $f = 1 + 4 = 5$) et un autre a $h_1 = 3$ (donc $f = 1 + 3 = 4$).
3. **Décision** : A* choisit le nœud avec $f = 4$.

→ *Limite :* Les valeurs de h_1 sont petites et peu variées (beaucoup d'égalités), ce qui guide moins efficacement la recherche que h_2.

— **A* (A-Star) avec h_2 (Manhattan) :** Il utilise l'estimation $f(n) = c(n) + h_2(n)$ pour choisir le nœud le plus prometteur.
Détail sur l'exemple :

1. **Racine (S_0)** : $c = 0, h_2 = 7 \Rightarrow \mathbf{f = 7}$.
2. **Niveau 1** : On génère les 3 fils possibles.
   — Fils 1 (Up) : $c = 1, h_2 = 6 \Rightarrow \mathbf{f = 7}$.
   — Fils 2 (Down) : $c = 1, h_2 = 6 \Rightarrow \mathbf{f = 7}$.
   — Fils 3 (Left) : $c = 1, h_2 = 7 \Rightarrow \mathbf{f = 8}$.

3. **Décision** : A* laisse de côté le nœud à $f = 8$ et choisit d'explorer en priorité l'un des nœuds à $f = 7$.

→ *Conclusion :* A* reste focalisé sur les chemins qui minimisent le coût total estimé, évitant l'exploration inutile des branches coûteuses.

Exercice 4 : Implémentation Machine

Voir le notebook Jupyter de correction disponible sur Moodle.

Point clé à retenir du TP : L'heuristique h_2 (Manhattan) est dite plus *informée* que h_1 (car $h_2(n) \geq h_1(n)$ pour tout n). Cela permet à A* avec h_2 de converger vers la solution optimale en développant beaucoup moins de nœuds qu'avec h_1.