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.

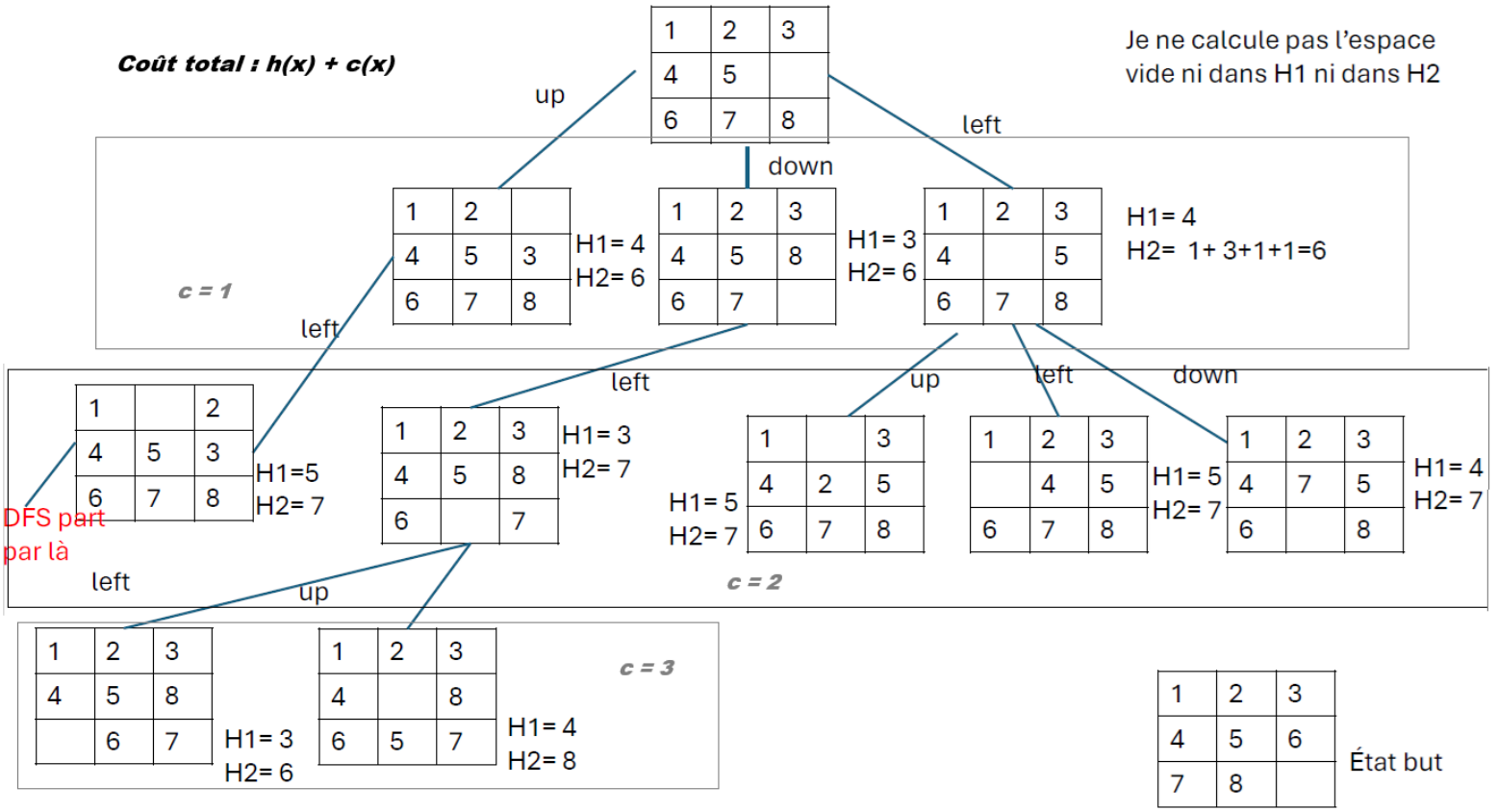

FIGURE 1 – Arbre de recherche du Taquin : comparaison des coûts (c) et heuristiques (h_1 vs h_2).

TD IA-2026修正案

IA教学团队

1　TD3：启发式搜索

练习3：塔昆游戏与A*

符号说明：

$h1(n)$:错位瓦片数量（空格不计入）。$h2(n)$:曼哈顿距离（P |*当前值* − *期望值*|+|*当前值* − *期望值*|）。

$c(n)$:根节点到路径的实际成本（此处为树的深度）。$f(n)$:A* 的全局评估函数，定义为$f(n) = c(n) + h(n)$ 。

修正提案

1. 在搜索树上的遍历

下图详细展示了树结构的初始层级发展过程。针对每个状态，我们标注了成本c（深度）以及两种启发式算法的数值。

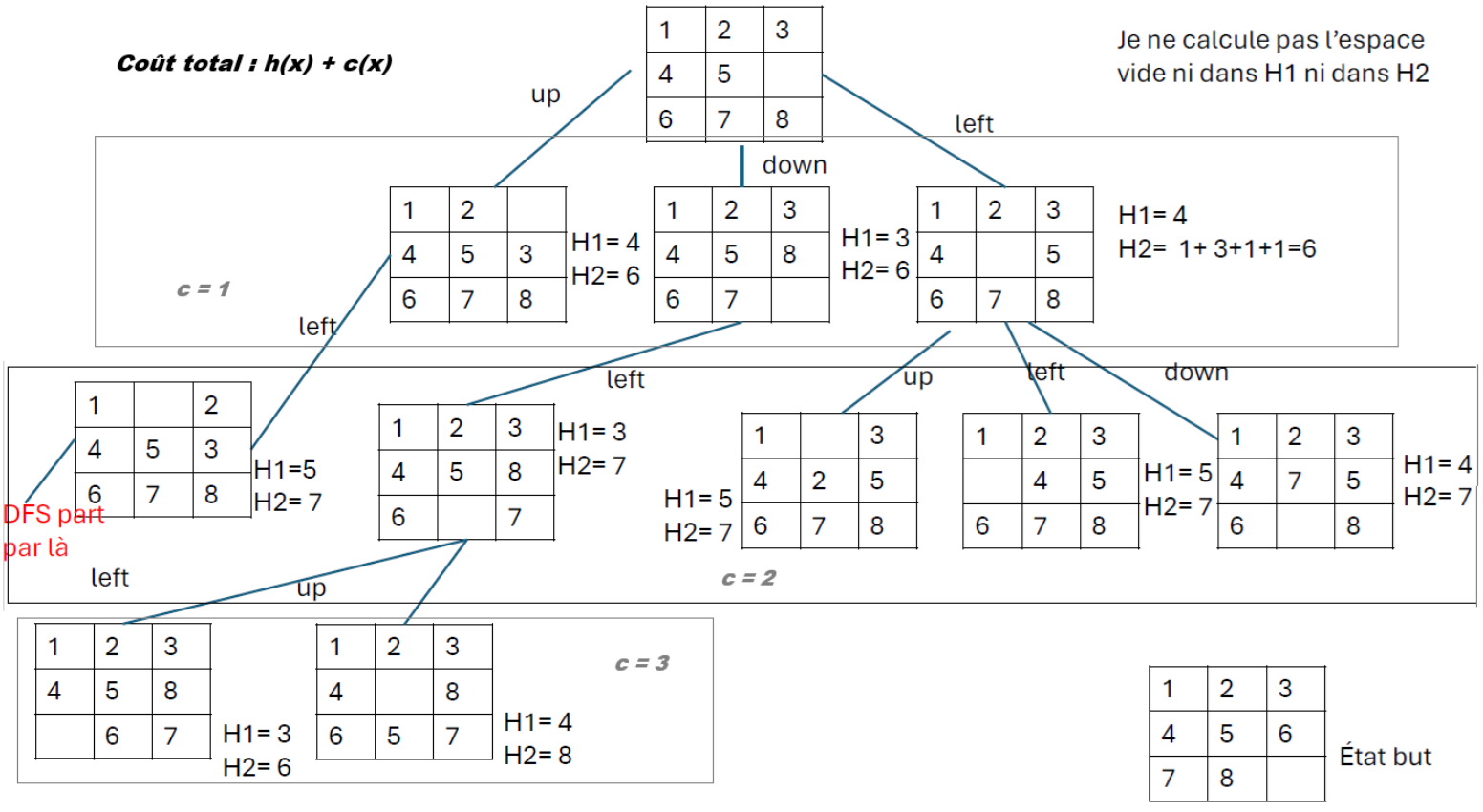

图1 – Taquin搜索树：成本（c）与启发式（$h1$ vs $h2$）的比较。

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) : $c = 1, h_2 = 7 \Rightarrow \mathbf{f = 8}$.

2. 战略分析与比较

理论回顾：启发式方法的核心作用

启发式算法$h(n)$是用于估算达成目标所需剩余成本的评估方法。正是它实现了从盲目探索到智能探索的转变。

—目的（剪枝）：这种启发式方法旨在预估路径成本，通过**剪除**导向明显过高成本解决方案的分支。

*矛盾之处在于：*若能知晓确切距离，问题便迎刃而解。因此启发式方法的妙处恰恰在于：当无法获得精确解法时，我们仍能估算出“大致”的最小距离——比如“我还不知道需要再走多少步，但至少得走X步（加上之前已走过的c步：这些步数我已知其真实值，所以直接代入计算）”。

—金规则（乐观性/可接受性）：为确保算法不会错误地舍弃最优解，估计必须是**乐观的**（或可接受的）。它**绝不能高估**实际成本（$h(n) \leq h^*(n)$）。

*风险：*若启发式算法过于悲观（阈值过高），算法会错误地认为最优路径过长，从而可能将其截断（即删除树结构中的真实解）。

—效率（紧致度）：为了有效，启发式算法必须尽可能“紧致”（*tight*），即在不超出实际成本的前提下，尽可能接近真实成本。

→若h值过低（例如0），则无需修剪（返回BFS）。

→当h接近实数时，可进行大规模筛选，从而快速找到解。

→当h大于实数时，我们将逐步剔除真实解，导致算法永远无法收敛。严禁！通过分析图1的树状结构，以下是各算法的具体表现：

—深度优先搜索（DFS）：如“DFS从这里开始”的注释所示，该算法采用“冲动式”运作模式。它会先选择第一个可用操作符（例如*左*），然后深入探索该分支。

→ *缺点：*无法判断状态是否理想。既不完整（可能无法闭合），也不达到最优状态。

—BFS（广度优先搜索）：它会逐层遍历树结构（所有节点的$c=1$，然后所有节点的$c=2$，依此类推）。

→ *缺点：*虽然采用“同心圆”探索法是最佳方案，但该方法在内存和时间成本上都极为高昂。

—A*（A-Star）与*h1*（错位瓦片）：它使用估计$f(n)=c(n)+h1(n)$.*示例说明：*

1. **根（*S0*）：**$c=0$，$h1=3$ 🡐 **f = 3。**
2. **第一级：**在这些线中，有些线的$h1=4$（因此$f=1+4=5$），另一条线的$h1=3$（因此$f=1+3=4$）。
3. **决策：**A* 选择具有$f=4$ 的节点。

→ *极限：h1*的值较小且变化不大（存在大量等式），其引导搜索的效果不如*h2*。

—A*（A-Star）与*h2*（曼哈顿）：它使用估计值$f(n)=c(n)+h2(n)$ 来选择最有前景的节点。

示例详情：

1. **根（*S0*）：**$c=0$，$h2=7$ 🡐 **f = 7。**
2. **第一级：**生成三种可能的线程。
   - 儿子1（上）：$c=1$，$h2=6$ 🡐 **f=7**.
   - 儿子2（唐氏）：$c=1$，$h2=6$ 🡐 **f=7**.
   - 儿子3（左）：$c=1$，$h2=7$ 🡐 **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.

3. **决定**：A* 会跳过 $f = 8$ 的节点，优先选择 $f = 7$ 的节点进行探索。

→ *结论：*A*算法始终聚焦于能最小化总估计成本的路径，避免对高成本分支进行无谓探索。

练习4：机器实现

请查看Moodle平台上的Jupyter代码校对笔记本。

TP的关键要点：启发式算法*h2*（曼哈顿）被认为比*h1*（更*有信息量*，因为对于所有n，*h2*（n）≥*h1*（n）。这使得采用*h2*的A*算法能够收敛到最优解，且比使用*h1*时生成的节点数量少得多。