École nationale
Supérieure
d'Informatique

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Rapport de Travaux Pratiques

Module : Théorie de Programmation et Recherche Opérationnelle (TPRO)

TP4 : Optimisation et Recherche Heuristique

Le Problème du Taquin 15

Étude comparative des algorithmes A et Weighted A**

Réalisé par le binôme :
ATTIA Oussama Abderraouf
SRAICH Imene

Encadré par :
M. HADIM Boukhalfa

Année Universitaire : 2025 - 2026

Table des matières

1 Introduction Générale

Le problème du Taquin (15-puzzle) représente l'un des puzzles les plus célèbres en informatique, particulièrement dans le domaine de l'intelligence artificielle et de la recherche opérationnelle. Inventé à la fin du XIXe siècle, il consiste en une grille de 4×4 contenant 15 jetons numérotés et un espace vide, permettant le glissement des pièces adjacentes. L'objectif est d'atteindre une configuration cible, généralement ordonnée, à partir d'un mélange initial.

D'un point de vue mathématique, ce problème est fascinant car il illustre parfaitement l'explosion combinatoire. Avec $16!/2$ états possibles (soit environ $1,05 \times 10^{13}$), une exploration exhaustive est impossible. Ce TP explore comment les heuristiques et les structures de données avancées permettent de naviguer dans cet océan de possibilités pour trouver une solution de manière quasi-instantanée.

2 Description du Problème et État de Départ

2.1 Objectifs du Travail Pratique

Le Travail Pratique n°4 nous a été confié avec des consignes claires visant à transformer un prototype basique en un solveur de niveau professionnel :

- **Transition structurelle :** Passer d'une file de priorité naïve gérée par un tableau ($O(N)$) à une structure de **Tas Binaire** (Min-Heap) offrant des performances en $O(\log N)$.
- **Amélioration heuristique :** Implémenter et comparer la **Distance de Manhattan** aux jetons mal placés.
- **Généralisation algorithmique :** Mettre en œuvre le **Weighted A* (WA*)** pour explorer le compromis optimalité/vitesse.
- **Analyse expérimentale :** Réaliser des benchmarks sur plusieurs métriques (nœuds, temps, profondeur).

2.2 Analyse de la Version de Référence

La version initiale fournie par le professeur présentait une implémentation fonctionnelle de l'algorithme A* mais limitée par ses choix techniques. La recherche du nœud de coût minimal s'effectuait par un balayage complet de la liste des nœuds ouverts. Sur des configurations demandant des milliers d'étapes, cette opération devenait le goulot d'étranglement principal, limitant la profondeur de résolution accessible en un temps raisonnable. De plus, l'heuristique des "jetons mal placés", bien qu'admissible, s'avère peu discriminante pour guider efficacement la recherche dans un espace de 4×4.

3 Optimisations et Choix de Conception

3.1 Structure de Données : Le Tas Binaire (Min-Heap)

Pour répondre aux exigences de performance, nous avons implémenté un Tas Binaire. Cette structure d'arbre binaire complet garantit que la racine contient toujours l'élément de coût minimal.

Contrairement à une file de priorité classique par tableau où l'extraction est en $O(N)$, le tas binaire permet :

— **Insertion (HeapPush) :** Le nœud est ajouté à la fin, puis "remonté" (*bubbleUp*) pour restaurer la propriété du tas en $O(\log N)$.

— **Extraction du Min (HeapPop) :** La racine est remplacée par le dernier élément, puis "redescendue" (*bubbleDown*) en $O(\log N)$.

3.2 Les Heuristiques de Recherche

3.2.1 Distance de Manhattan : Une estimation puissante

La **Distance de Manhattan** (ou distance L_1) est la somme des déplacements horizontaux et verticaux nécessaires pour ramener chaque jeton à sa case cible. Elle est nettement supérieure à l'heuristique des jetons mal placés car elle prend en compte la géométrie de la grille. Elle reste **admissible** (ne surestime jamais le coût réel), assurant ainsi l'optimalité de l'algorithme A* standard.

3.2.2 Weighted A* : Le compromis optimalité-efficacité

Nous avons implémenté la variante pondérée où la fonction d'évaluation devient $f(x) = g(x) + p \cdot h(x)$ avec $p \geq 1$. Ce coefficient p permet de "pousser" l'algorithme à privilégier davantage l'heuristique. Dans un contexte réel, cela permet de trouver des solutions à des puzzles extrêmement complexes que A* classique ne pourrait résoudre par manque de mémoire, au prix d'une solution légèrement plus longue.

4 Implémentation Algorithmique

4.1 Cœur de l'Algorithme A*

L'implémentation repose sur une boucle robuste gérant l'expansion des nœuds et la gestion des états visités. Voici le pseudo-code structuré de notre moteur de recherche :

```
Fonction A_Star(Configuration_Initiale, But):
    Frontiere = Initialiser_Tas_Min()
    Visites = Initialiser_Hachage_Ou_Liste()

    Initial.f = g(Initial) + p * h(Initial)
    Frontiere.Push(Initial) // Insertion initiale

    Tant que Frontiere est non vide:
        n = Frontiere.Pop() // Extraction du meilleur noeud en
           O(log N)

        Si n == But:
            Retourner Chemin_Solution(n)

        Ajouter n a Visites

        Pour chaque mouvement possible (Haut, Bas, Gauche, Droite):
            m = Generer_Voisin(n)
```

```
            Si m est deja dans Visites: Continuer

            m.g = n.g + 1
            m.f = m.g + p * h(m)
            m.pere = n

            Frontiere.Push(m) // Insertion en O(log N)
```

Listing 1 – Pseudo-code de l'algorithme principal

4.2 Architecture et Modularité

Le code a été scindé en plusieurs fonctions clés pour garantir une lisibilité maximale. Les fonctions de manipulation de la grille, le calcul de l'inversion count pour la solvabilité, et le moteur de recherche WA* sont distincts. Cette modularité a permis de créer facilement deux versions de l'exécutable : l'une avec une interface riche (`conio2.h`) et l'autre purement standard pour une portabilité totale.

5 Benchmarking et Analyse des Performances

Les expérimentations ont été menées sur une configuration initiale nécessitant 18 mouvements optimaux. Les résultats suivants mettent en lumière l'efficacité des optimisations.

Algo (p)	Heuristique	Temps (s)	Nœuds	Frontière	Depth
A*	Tiles	0.0050	1891	2108	18
A*	Manhattan	0.0000	191	219	18
WA*(1.5)	Tiles	0.0020	1652	1872	18
WA*(1.5)	Manhattan	0.0000	105	127	18
WA*(3.0)	Manhattan	0.0010	311	362	18
WA*(5.0)	Manhattan	0.0020	813	934	28

TABLE 1 – Comparaison des algorithmes sur une instance de profondeur 18

5.1 Analyse Critique des Résultats

1. **Efficacité Exceptionnelle de Manhattan :** On observe une réduction de plus de **90%** du nombre de nœuds explorés par rapport aux jetons mal placés. Manhattan est extrêmement directionnelle, ce qui limite les explorations inutiles.
2. **Le "Sweet Spot" de WA* :** Pour $p = 1.5$, nous parvenons à réduire encore le nombre de nœuds (105 contre 191) tout en conservant la solution optimale de 18 pas. C'est l'équilibre parfait pour ce type d'instance.
3. **Dérive vers la Sub-optimalité :** Pour $p = 5.0$, bien que l'algorithme "fonce" vers le but, il néglige tellement le coût réel accumulé qu'il finit par trouver un chemin de 28 pas. Cela illustre le risque de sur-pondération de l'heuristique.

6 Conclusion

Ce TP nous a permis de mettre en application des concepts fondamentaux de la recherche opérationnelle et de l'algorithmique avancée. L'implémentation du Tas Binaire a démontré l'importance du choix des structures de données dans la résolution de problèmes à grande échelle. L'étude de la distance de Manhattan et de la pondération WA* a souligné l'importance de "l'intelligence" intégrée dans les fonctions heuristiques.

Au final, nous avons abouti à un solveur robuste, performant et documenté, capable de répondre aux exigences de rapidité et d'efficacité posées par le module TPRO.