Université de Genève

-

Sciences Informatiques

Algorithmique - TP 05

Noah Munz (19-815-489)

Dec 2022

Contents

TP 05

1 Taquin / Fifteen Puzzle (5 Points)

1.1 Fonction de coût

(a) *Quel est le rôle de $h(x)$ dans la fonction de coût $\hat{c}(x) = g(x) + h(x)$?*

(b) *A quel type de recherche correspond la fonction de coût*

1) $\hat{c}(x) = g(x) + h(x)$ *?*
2) $\hat{c}(x) = h(x)$ *?*

(c) *En se basant sur les critères discutés en cours concernant les propriétés de $\hat{c}$ vis-à-vis d'une fonction de coût idéale $c(x)$, montrez que si l'on fait une recherche Branch-and-Bound avec la fonction de coût définie au point précédent et que l'on trouve une solution, cette solution est optimale.*

(a) Avant de parler du rôle de la fonction $h(x)$, il faut d'abord expliquer pourquoi il est avantageux (dans ce contexte) de garder notre $\hat{c}(x_j)$ le plus constant possible pour tout j.

Soit T, l'arbre des possibilités de racine x_0 (parfois appelé *root*) du jeu (où une solution de notre problème est simplement une suite de edge de T) et soit x_j un noeud de T, i.e. x_j est un état de notre problème.

On veut garder $\hat{c}(x) \leq \hat{c}(y)$ pour tout enfants y de x, car pour une recherche directe qui minimise le coût $\hat{c}(x)$, le relation (3.3) du cours donne que $c(x) \leq c(y)$.

Or $\forall$ enfant y de x : $c(x) \leq c(y)$ implique que $\hat{c}(x^\star) = c(x^\star) \geq c(x_0) = c(\textit{optimum})$. ($x^\star$ est un noeud solution)
Donc vu que le coût des noeuds dans la descendance d'un x_0 et jusqu'à $x^\star$ ne peut pas diminuer, et que $x^\star$ est garantie d'être une solution optimale, le mieux qu'on puisse faire (pour minimiser le coût du chemin total $x_0 \dots x_i \dots x^\star$) est de ne pas augmenter le coût des $x_{i>0}$ i.e. le garder constant.

En résumé, on veut garder les $\hat{c}(x_{i+k})$ le plus proche de $c(x_0)$ et donc des autres $\hat{c}(x_i)$ possible.
Ce qui ce traduit par "garder le coût" constant car la relation (3.3) nous garanti que $c(x_0)$ est le plus petit coût possible. (i.e. le coût de la solution qui minimise le plus le coût total)
(Ce qui est particulièrement dur quand $\hat{c}$ est une mauvaise approximation i.e. les $\hat{c}(x)$ sont loin des $c(x)$)

$c(x_0) \approx \hat{c}(x_0) \approx \hat{c}(x_i) \approx \hat{c}(x_{i+k}) \approx \hat{c}(x^\star)$ serait une recherche parfaite (pour un $\hat{c}(x_0)$ proche de $c(x_0)$, $k, i \in \mathbb{N}$)

On peut donc maintenant constater que le rôle de h est d'imposer que le coût des x_{i+k} ne peut rester constant que si les $g(x_{i+k})$ diminuent. C'est à dire que si g est une "mauvaise" borne/fonction (g est une borne inférieur du nombre de coup qui reste à faire), cela va directement se voir sur le coût total $\hat{c}$ et les $\hat{c}(x_{i+k})$ (pour les plus hautes valeurs de k). En effet on a que:

$$\begin{aligned}
\hat{c}(x_{i+k}) \leq \hat{c}(x_i) &\iff g(x_{i+k}) + h(x_{i+k}) \;\leq\; g(x_i) + h(x_i) \\
&\iff g(x_{i+k}) + h(x_i) + k \;\leq\; g(x_i) + h(x_i) \\
&\iff g(x_{i+k}) + k \;\leq\; g(x_i) \\
&\iff k \;\leq\; g(x_i) - g(x_{i+k})
\end{aligned}$$

Ce qui veut dire que pour garder le coût constant, il faut que entre $g(x_i)$ et $g(x_{i+k})$ on se soit rapproché de la solution optimale $x^\star$ de k coups.

Or comme on fait seulement un mouvement par étape, (i.e. $\max(g(x_i) - g(x_{i+1})) = 1$)
cela implique qu'à chaque étape on se rapproche de $x^\star$ autant que possible, i.e. qu'on prenne le meilleur chemin / E-Node à chaque étape.

Ce qui nous donne la relation "on est sur le bon chemin" ($\hat{c}(x_{i+k})$ constants) si et seulement si on le nombre de case mal placé diminue (i.e. $g(x_{i+k}) < g(x_i)$).

(b) *A quel type de recherche correspond la fonction de coût*

1) $\hat{c}(x) = g(x) + h(x)$ *?*
C'est une recherche "least cost" (comme défini en (3.4) dans le cours), on est dans le cas d'un jeu où l'espace à explorer est un graph dirigé qui décrit les différents mouvements possibles à partir d'une configuration x_0.
On cherche donc une configuration gagnante $x^\star$ telle que la longueur du chemin *path(x_0, $x^\star$)* i.e. la somme des coûts des noeuds du chemin $(x_0 \dots x_i \dots x^\star)$ soit minimale.

Pour ce faire on définit donc $h(x)$ comme étant la profondeur de la configuration x dans l'arbre de recherche, (*height*) (i.e. le nombre de coups déjà fait / coût accumulé jusqu'à x) et $g(x)$ comme une lower bound du nombre de coups restant à faire pour atteindre $x^\star$.
Comme ça, on a une estimation du nombre de coups minimum qui compose *path(x_0, $x^\star$)*. (un coup == un edge de l'arbre)

2) $\hat{c}(x) = h(x)$ *?*
D'après la relation (3.4) du cours, encore une fois, on aurait à faire à une recherche à *coût uniforme*.

> "*On peut bien sûr choisir $g(x) = 0$ pour tout x. On parle de recherche à coût uniforme. Le Branch & Bound trouvera l'optimum recherché, mais l'efficacité de l'exploration sera sans doute faible.*"

On remarque que dans ce cas la recherche "least cost" nous donne en fait un BFS (breadth first search).
car seul la profondeur détermine le coût donc on va explorer les noeuds sur la même largeur de x en priorité. Elle est donc très peu efficace, niveau complexité en temps car il n'y aucun "guide" sur le chemin à prendre.

3) Les critères et propriétés de $\hat{c}(x)$ ayant été longuement abordés au point 1) on se contentera de rappeler le théorème 2 du cours:

> "*Soit une recherche Branche & Bound à coût minimum (least cost) et soit une fonction $\hat{c}$ qui vérifie que $\hat{c}(x) \leq c(x)$ pour tous les noeuds x, et $\hat{c}(x^\star) = c(x^\star)$ pour les noeuds réponse $x^\star$.*"

Donc en se basant sur le fait que l'équation 1 de l'énoncé $\hat{c}(x) = g(x) + h(x)$ vérifie bien les critères du théorème 2, il ne nous reste plus qu'à prouver que $h(x) \leq g(x) + h(x)$ (car $g(x) + h(x) \leq c(x)$) et que $h(x^\star) = h(x^\star) + g(x^\star)$ (car $g(x^\star) + h(x^\star) = c(x^\star)$):

$$\begin{aligned} & h(x) \leq g(x) + h(x) \\ \iff & h(x) - h(x) \leq g(x) \\ \iff & 0 \leq g(x) \end{aligned}$$

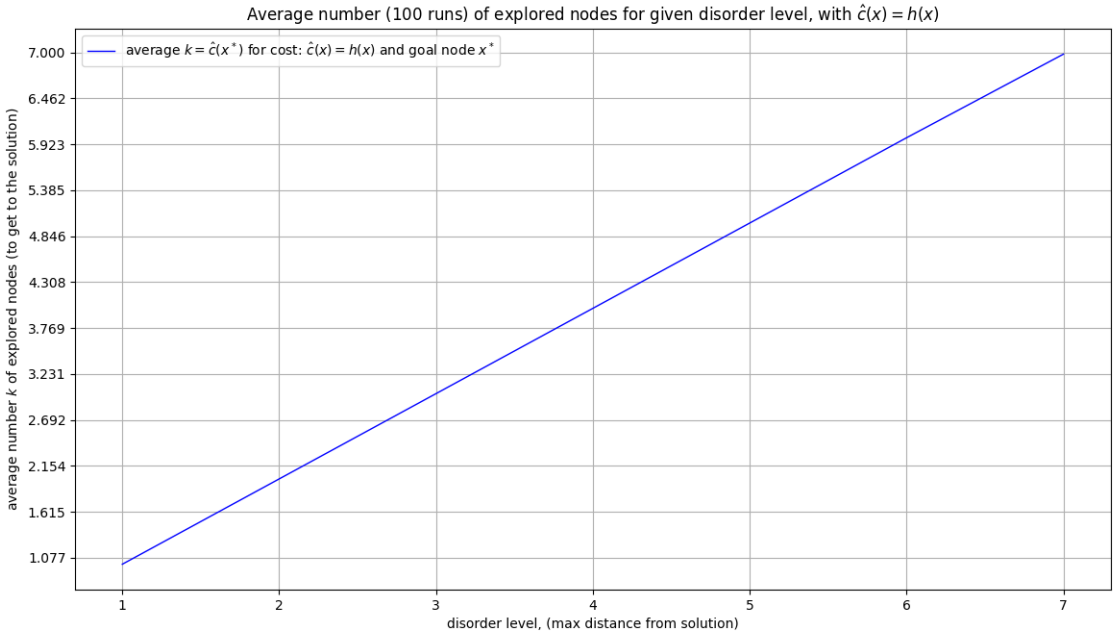
Or $g(x)$ est une lower bound du nombre de coups restant à faire pour atteindre $x^\star$ elle donc évidemment toujours plus grande à 0. Sauf pour $g(x^\star)$ qui vaut 0.

$$\begin{aligned} & h(x^\star) = h(x^\star) + g(x^\star) \\ \iff & h(x^\star) = h(x^\star) + 0 \\ \iff & h(x^\star) = h(x^\star) \end{aligned}$$

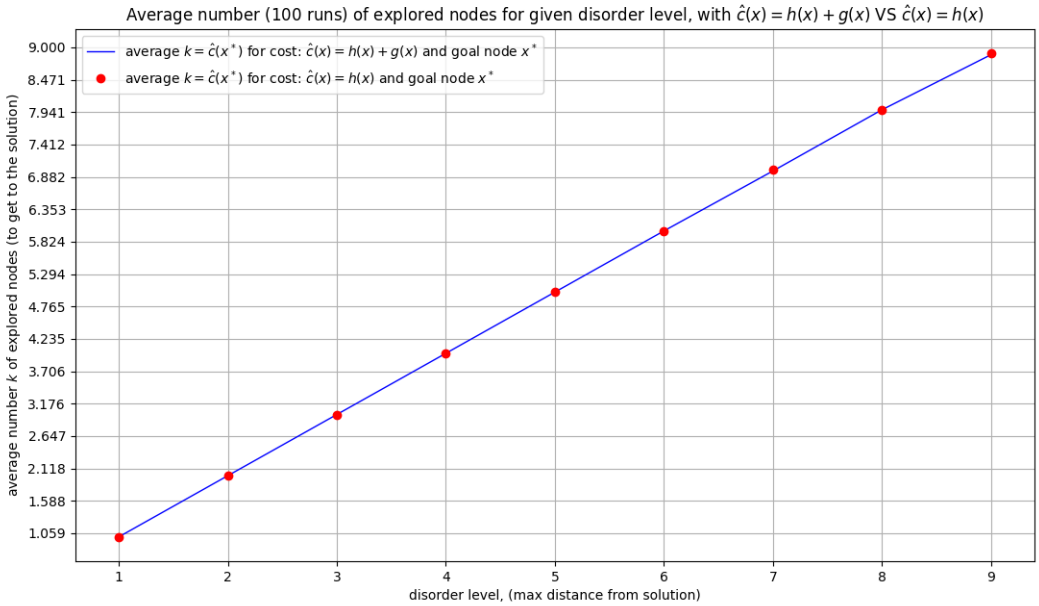
Donc par le théorème 2, si on trouve une solution avec cette fonction de coût ($\hat{c}(x) = h(x)$) elle est optimale.

1.3 Efficacité de l'algorithme

1. Optimalité de la solution.



. *Figure 1. (légende sur graph)*

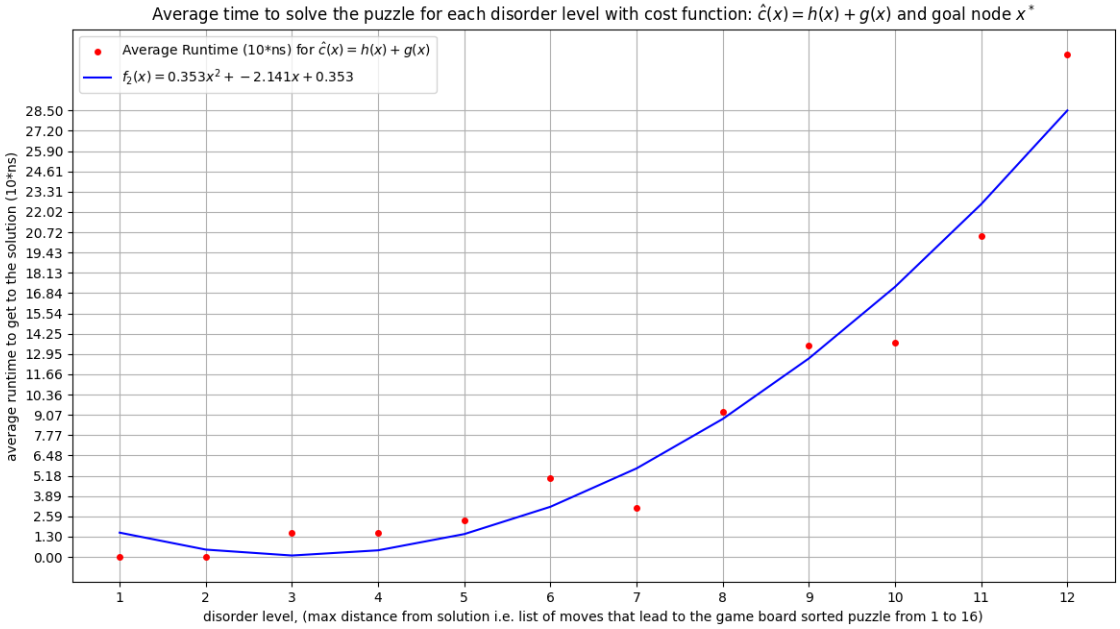


. *Figure 2. (légende sur graph)*

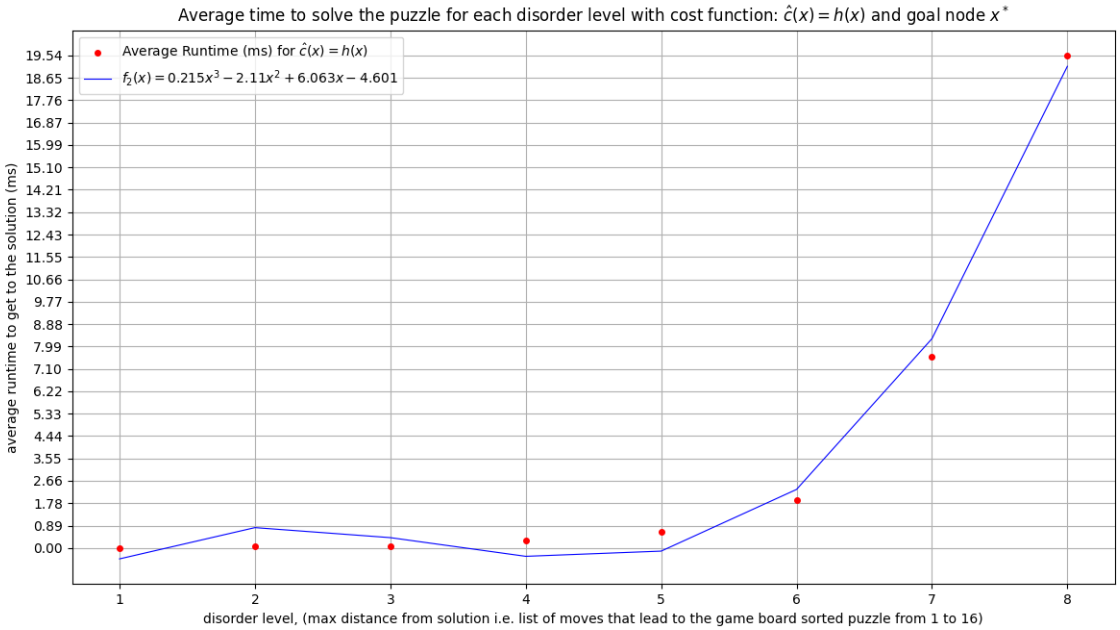
Soit n le nombre de coup optimal i.e. $g(x_0)$ et m la taille de la matrice du board. Ici m est fixé cependant on pourrait très bien utiliser la même implémention pour le problème avec un board de taille $m \times m$. pour m variable (*la version $m \times m$, est testée dans le fichier exo1_3.py*).
(On a bien que l'axe x sur les 2 graphs représente le n passé à la fonction `gen_disorder()` .)

Comme on le voit sur la figure 2, l'efficacité en terme de "quelle solution est la plus optimale?" est relativement la même pour les 2 fonctions de coûts. En effet, on voit bien que le nombre de coups à faire grandit parfaitment linéairement avec le désordre (i.e. distance à la solution qui est juste le board trié de 1 à 16). Les 2 courbes sont parfaitement superposées. Pour un move de plus on a bien au max 1 noeud en plus à parcourir pour les 2.
Par contre au niveau de la complexité en temps, les résultats sont nettements différents pour les 2 comme on va le voir au point précédent.

2. Complexité en temps.



. *Figure 3*



. *Figure 4*

Comme on peut le voir, la complexité en temps de la fonction de coût de la figure 3 ($h(x) + g(x)$) est en $O(\hat{c}(x_0)^2) = O(\hat{c}(x^\star)^2) = O(n^2)$

Car on fait en général un boucle avec n itérations et sur chaque itération on effectue au maximum n "moves" sur le board. (voir fonction `update_misplaced_compute_cost` de l'exo1.) et c'est le "bout de code" le plus cher niveau complexité qu'on fait, i.e. la complexité du tout serait juste $O(n^2) + O(n^2) + \cdots + O(n) + \cdots + O(1) = O(n^2)$

On voit que le runtime de l'autre fonction de coût (Figure 4) fit bien mieux à une courbe d'ordre $O(n^3)$ (le polynôme que l'on voit en légende à été calculé avec la méthode des moindres carrées, i.e. ce polynôme est celui qui minimise les carrées des distances entre les points et la courbe. Montrant qu'on a bien du $O(n^3)$).

Il est important de noter que le 1er graph mesure le temps en **10·ns** (i.e. $10^{-8}s.$) et le 2e en **ms** (i.e. $10^{-3}s.$)
L'écart entre les 2 deux est donc *massivement* plus grand que ce qu'on pourrait penser au 1er coup d'oeil, on a bien un facteur de $\times 10'000$ (i.e. $10^5 \cdot 10^{-8} = 10^{-3}$ s).

3 Plus court chemin (Extra 1 Point)

3.1 Fonction de coût

La fonction de coût qui remplie les critères nécessaires pour garantir l'optimisation d'une solution est tout simplement la même que celle utilisée pour le problème du taquin.
En effet le théorème 2 du cours et le "guide" sur comment choisir sa fonction de coût sont tellements généraux qu'on pourrait les appliquer à beaucoup de problèmes. La seule chose qui va changer ici, fondamentalement, c'est la fonction $g(x)$ qui a été choisie pour être simplement la distance définie par la norme L_1 (i.e. $\|\cdot\|_1$) entre la position de x et la destination b.