Recherche du plus court chemin par l'utilisation d'heuristiques

Pierre Vigier

15 février 2017

Introduction

Les problèmes de graphes se divisent en deux grandes catégories. D'abord, il y a les problèmes d'optimisation où l'ont part d'un état initial et l'on recherche un état final optimal. Puis il y a les problèmes où sont donnés un état initial et un état final, l'objectif est alors de déterminer un chemin voire le plus court chemin permettant de relier ces deux états dans le graphe. Cette deuxième catégorie de problèmes va nous intéresser dans la suite.

La première partie, va montrer que les algorithmes spécifiés dans le programme d'option informatique de parcours de graphe sont trop lents pour traiter des problèmes réels de recherche du plus court chemin. L'algorithme A* et les heuristiques vont alors être présentés pour répondre à ce problème de performance. Dans un second temps, le taquin sera étudié pour ensuite générer des heuristiques adaptées à la résolution du puzzle et effectuer des simulations avec celles-ci. Finalement, la dernière partie traitera de la définition et de la comparaison de mesures de la qualité des heuristiques.

1 Algorithme A*

1.1 Algorithmes de recherche best-first

1.1.1 Algorithme général

La plupart des algorithmes de parcours de graphe se présentent de la même manière. Ils peuvent être décrit avec un même pseudo-code (figure 1). Les différences entre les algorithmes résident alors dans la manière de sélectionner les nœuds dans la frontière et de les stocker. La sélection des nœuds dans la frontière peut être modélisée par l'utilisation d'une fonction f des nœuds du graphe dans $\mathbb{R}$, l'algorithme choisit alors à chaque itération le nœud qui a la plus petite valeur par f. Dans le cas général, on peut utiliser comme structure de données pour la frontière une file de priorité implémentée par un tas mais parfois d'autres structures de données seront plus performantes. Le parcours en profondeur, le

```
fonction explorer(état initial)
  initialiser la frontière avec l'état initial
  tant que la frontière n'est pas vide
    choisir un noeud dans la frontière et l'enlever
    si le noeud est l'état final alors
      retourner la solution correspondante
    pour tous noeud accessible depuis ce noeud
      si il n'est ni dans la frontière ni dans les noeuds déjà
          explorés alors
        on l'ajoute à la frontière
      si il est dans la frontière
        on met à jour sa position dans la frontière
```

FIGURE 1 – Pseudo-code de l'algorithme best-first

Type de parcours	$f(n)$	Structure de données adaptée
Parcours en largeur	$depth(n)$	File
Parcours en profondeur	$-depth(n)$	Pile
Algorithme de Dijkstra / Parcours à coût uniforme	$g(n)$	File de priorité

FIGURE 2 – Description d'algorithmes de type best-first

parcours en largeur et l'algorithme de Dijkstra peuvent s'écrire de cette manière (figure 2).

1.1.2 Comparaison du comportement

L'algorithme de Dijkstra renvoie le plus court chemin entre un état initial et un état final mais il parcourt trop de nœuds dans le cas général. En effet, regardons la carte des autoroutes françaises simplifiée (figure 3). Imaginons qu'on doive établir le plus court chemin entre Paris et Perpignan. Immédiatement, l'intuition en considérant les géodésiques va conseiller le chemin Paris→Orléans→Clermont-Ferrand→Perpignan et il s'agit bien du plus court chemin. Au contraire l'algorithme de Dijkstra afin de déterminer ce chemin va visiter les villes les unes après les autres dans l'ordre croissant de leur distance par rapport à Paris jusqu'à arriver à Perpignan. Il va ainsi visiter Lille, Rouen, Strasbourg, ... ce qui peut paraitre stupide puisqu'elles sont plus loin de Perpignan que Paris. L'algorithme que nous allons présenter, A*, va en quelque sorte reproduire l'intuition humaine en utilisant des heuristiques.

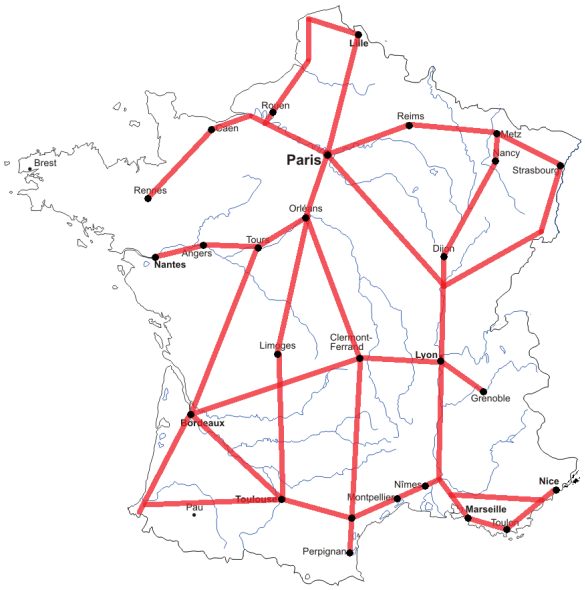

FIGURE 3 – Autoroutes françaises

1.2 Présentation de l'algorithme A*

1.2.1 Définitions

Définissons tout d'abord les heuristiques, ce sont elles qui vont jouer le rôle de l'intuition pour l'algorithme A*.

Definition 1. *Une heuristique h est une fonction des nœuds d'un graphe dans $\mathbb{R}$ estimant le coût minimal pour aller du nœud passé en argument à l'état final.*

Posons g la fonction qui associe à un nœud sa distance à l'état initial. Nous pouvons maintenant décrire l'algorithme A*. L'algorithme A* est une recherche best-first avec comme fonction de sélection $f = g + h$. $f(n)$ représente ainsi le coût estimé de la solution optimale passant par le nœud n. La figure 4 illustre g, h et k où $k(n)$ est le coût minimal pour aller du nœud n à l'état final.

1.2.2 Preuve de l'optimalité

Afin d'avoir un nombre de nœuds limité à explorer, nous voulons, comme pour l'algorithme de Dijkstra, que le premier chemin trouvé menant à l'état final soit une solution optimale au problème. Dans le cas général, ce n'est pas vrai, il est donc nécessaire de poser des conditions sur l'heuristique afin que le résultat soit vrai. La condition suivante est suffisante.

Definition 2. *h est consistante si pour chaque nœud n et chacun de ses successeurs n' et pour toute arête a permettant d'aller de n à n' de coût $c(n, a, n')$,*

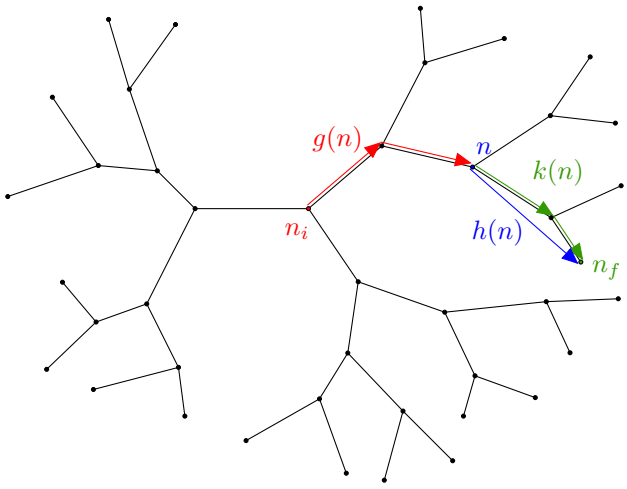

FIGURE 4 – Illustration de g, h et k

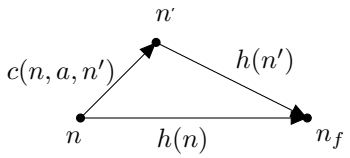

FIGURE 5 – Inégalité triangulaire

on a $h(n) \leq c(n, a, n') + h(n')$.

Le fait que h soit consistante peut s'interpréter comme une inégalité triangulaire (figure 5). De plus la consistance de h implique que h est toujours inférieure au coût réel du chemin optimal k.

On a alors la proposition suivante.

Proposition 1. *A* est optimal si h est consistante.*

Preuve : Soit h consistante.

— Soit n un nœud différent de l'état initial et n' un de ses voisins étant sur un chemin optimal de l'état initial à n. On a donc a une arête permettant d'aller de n' à n telle que $g(n) = g(n') + c(n', a, n)$. Ainsi $f(n') = g(n') + h(n') \leq g(n') + c(n', a, n) + h(n') = g(n) + h(n) = f(n)$ d'où f est croissante sur un chemin optimal.

— Supposons qu'il existe un nœud n développé par A* tel que le chemin optimal vers celui-ci n'a pas été trouvé. Il existe alors n' appartenant à la frontière qui est sur un chemin optimal entre l'état initial et n. D'où

3	7	5
6		2
4	8	1

3	2	1
4	5	6
	8	7

FIGURE 6 – Exemples de configuration initiale et finale au taquin

3	2	1
4	5	6
	8	7

FIGURE 7 – Ordre de numérotation

d'après le point précédent $f(n') \leq f(n)$. D'où n' aurait été développé avant n. Contradiction.

Par conséquent, le premier nœud atteignant l'état final a été atteint via un chemin optimal. Donc la première solution trouvée est optimale.

On supposera dans la suite que toutes les heuristiques sont consistantes.

2 Exemple du taquin

2.1 Définition

Definition 3. *Le taquin à $np-1$ pièces se compose d'un plateau de $n \times p$ cases dont $np-1$ sont occupées par des pièces numérotées et une est vide. Les pièces adjacentes à la case vide peuvent échanger leur place avec celle-ci. Le but du jeu est de passer d'une configuration initiale donnée à une configuration finale.*

Dans toute la suite, le taquin à 8 pièces sera utilisé dans les exemples et simulations car comme nous le verrons, le graphe des états est de taille raisonnable.

2.2 États atteignables à partir d'un état initial

Dans les simulations, un état initial et un état final seront générés puis le chemin optimal de l'un vers l'autre sera recherché encore faut il qu'il en existe un. Nous verrons dans cette sous-partie à quelle condition il est possible de passer d'une configuration à une autre au taquin.

Pour ce faire associons à chaque configuration une permutation de S_{np-1} pour cela, numérotons les cases non vides dans l'ordre décrit sur la figure 7.

Par exemple à la configuration initiale de la figure 6 est associée la permutation $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 7 & 3 & 6 & 2 & 1 & 8 & 4 \end{pmatrix}$.

On remarque que deux configurations ont la même permutation si et seulement si on peut passer de l'une à l'autre en déplaçant la case vide le long du parcours décrit sur la figure 7. On a alors le résultat suivant.

Proposition 2. *Il est possible de passer d'une configuration à une autre si et seulement si les permutations associées ont des signatures égales.*

Preuve : ($\Rightarrow$) Supposons qu'il existe une suite de mouvements allant d'une configuration à une autre. Remarquons que les mouvements ne changent pas la signature :

- les mouvements horizontaux ne changent pas la permutation donc a fortiori pas sa signature ;
- notons (i, j) les coordonnées de la case vide et regardons comment change la permutation lors des deux mouvement verticaux possibles. On compose la permutation par un cycle de longueur $2(j-1)+1$ ou $2(p-j-1)+1$ selon le sens du parcours. D'où la signature est invariante.

Donc la suite de mouvements laisse la signature invariante d'où les deux configurations ont la même signature.

($\Leftarrow$) Remarquons qu'on peut obtenir les permutations $\begin{pmatrix} p-1 & p & p+1 \end{pmatrix}$ en déplaçant la pièce en $(2,2)$ sur le vide placé en $(1,2)$, et $\begin{pmatrix} 1 & 2 & \dots & 2p-1 \end{pmatrix}$ en déplaçant la pièce en $(2,p)$ sur le vide placé en $(1,p)$. Or en composant $\begin{pmatrix} p-1 & p & p+1 \end{pmatrix}$ par des puissances de $\begin{pmatrix} 1 & 2 & \dots & 2p-1 \end{pmatrix}$, on obtient $\begin{pmatrix} i & i+1 & i+2 \end{pmatrix}$ pour tout $i \in [\![0; 2p-3]\!]$. Et en réopérant cette opération à chaque coude, on obtient les 3-cycles $\begin{pmatrix} i & i+1 & i+2 \end{pmatrix}$ pour tout $i \in [\![0; np-3]\!]$. Et la proposition suivante permet de conclure.

Proposition 3. *Les 3-cycles* $\begin{pmatrix} i & i+1 & i+2 \end{pmatrix}$ *pour tout* $i \in [\![0; n-2]\!]$ *génèrent* A_n *pour tout* $n \geq 3$.

On en conclut que dans les simulations les état initial et final devront avoir même signature pour que l'algorithme de recherche de solutions termine. De plus le résultat montre que le graphe des états accessibles à partir de l'état initial contiendra $np\frac{(np-1)!}{2}$ nœuds.

3 Génération et comparaison d'heuristiques

Nous avons donc décrit l'algorithme A* et le taquin mais nous n'avons pas encore d'heuristique permettant de résoudre le taquin avec A*. Il sera donc vu dans cette partie comment générer des heuristiques puis comment les comparer.

3.1 Problèmes relaxés

3.1.1 Définition et résultats

Definition 4. *Un problème dont les restrictions sur les actions sont moindres est un problème relaxé.*

Afin de comprendre l'intérêt des problèmes relaxés, il faut remarquer les points suivants.

- Le graphe des états possibles d'un problème est un sous graphe des états possibles de ses problèmes relaxés. En particulier, les solutions optimales

```
tant que la position finale n'est pas atteinte
  si la case vide n'est pas à sa position finale
    on l'échange avec la pièce qui devrait être à sa place
  sinon
    on l'échange avec une pièce mal placée
```

FIGURE 8 – L'algorithme permettant de calculer h_3

d'un problème sont aussi des solutions du problème relaxé car elles sont réalisables mais il peut exister de meilleurs solutions.

— La fonction k qui renvoie le coût de la solution optimale d'un problème à partir d'un nœud est consistante. On en déduit que la fonction qui renvoie le coût de la solution optimale d'un problème relaxé est consistante sur le graphe du problème relaxé donc aussi sur ses sous graphes. Elle est donc consistante pour le problème.

On en conclut donc qu'elle fait une bonne heuristique. Les problèmes relaxés qui ont de telles fonctions simples à calculer vont donc nous intéresser.

3.1.2 Exemple sur le taquin

Rappelons les règles du taquin.

Definition 5. *Une pièce peut se déplacer de la case A vers la case B si A et B sont adjacentes et B est vide.*

On en déduit trois problèmes relaxés :

1. Une pièce peut se déplacer de la case A vers la case B.
2. Une pièce peut se déplacer de la case A vers la case B si A et B sont adjacentes.
3. Une pièce peut se déplacer de la case A vers la case B si B est vide.

On peut alors associer une heuristique à chacun de ses problèmes relaxés :

1. h_1 est le nombre de différences entre l'état initial et l'état final ;
2. h_2 est la somme des distances de Manhattan que doivent parcourir chaque pièce pour aller de leur position initiale à leur position finale ;
3. pour calculer h_3, on exécute un algorithme simple (figure 8) qui donne la solution optimale puis on renvoie sa longueur.

3.2 Performance d'A*

Avant de comparer des heuristiques, nous devons d'abord voir comment elles influent sur la performance d'A* que nous discuteront en terme de nombre de nœuds développés. On pourrait vouloir mesurer la performance d'A* via la complexité temporelle mais il faudrait prendre en compte le coût temporelle induit par les différentes heuristiques qui est inconnu dans le cas général. De

plus, ce n'est pas forcément intéressant car comme nous le verrons, il vaut mieux une heuristique coûteuse en temps qu'une exploration moins bien informée.

Si C est le coût des solutions optimales, on a la proposition suivante.

Proposition 4. *L'algorithme A* explore au moins tous les nœuds n du graphe tel que $f(n) < C$ et au plus tous les nœuds n tels que $f(n) \leq C$.*

Et comme $f = g + h$, ceci se réécrit.

Proposition 5. *L'algorithme A* explore au moins tous les nœuds n du graphe tel que $g(n) < C - h(n)$ et au plus tous les nœuds n tels que $g(n) \leq C - h(n)$.*

On en déduit alors un moyen de comparer des heuristiques.

Proposition 6. *Soient h_1 et h_2 des heuristiques, si $h_1 \leq h_2$ alors dans le meilleur des cas et dans le pire des cas, on explorera moins de nœuds avec h_2 qu'avec h_1.*

Donc plus une heuristique est proche du coût réel de la solution optimale à partir d'un nœud k, plus l'algorithme A* est efficace.

Grâce à ce résultat, on peut prouver que l'on a $h_2 \geq h_1$ et $h_3 \geq h_2$ et donc que h_2 et h_3 sont meilleurs que h_1 mais il est plus difficile de comparer h_2 et h_3. De plus dans le cas générale, une analyse mathématique n'est pas toujours possible, on préférera donc mesurer la performance d'une heuristique sur des exemples et lui associer une valeur numérique.

Nous allons dans la suite discuter des différents choix de mesure possible.

3.3 Comparaison d'heuristiques

3.3.1 Facteur d'embranchement effectif

Une première idée serait de définir une mesure dépendant du nombre de nœuds explorés. Cependant, il n'est pas intéressant de définir le nombre de nœuds moyens explorés car il dépend trop fortement de la profondeur de la solution. Il faut donc la définir à une profondeur donnée.

Ainsi, une mesure courante dans la littérature pour évaluer une heuristique est le facteur de branchement effectif. Si on considère un problème, une solution de profondeur d et N le nombre de nœuds développés par A* pour la trouver. Le facteur de branchement effectif est le facteur de branchement d'un arbre de profondeur d contenant $N + 1$ nœuds. On peut donc définir le facteur d'embranchement effectif comme suit.

Definition 6. *Le facteur d'embranchement b^* est l'unique solution de $1 + b + ... + b^d = N + 1$.*

Si b^* converge quand d tend vers l'infini, on a alors que A* est en $O((b^*)^d)$.

Sur l'exemple du taquin, le nombre de nœuds explorés par A* sur 100 problèmes a été évalué pour chaque heuristique pour des profondeurs entre 2 et 24 (figure 9). On en déduit ensuite pour chacune de ces profondeurs le facteur

d	A*(h_1)	A*(h_2)	A*(h_3)
2	6	6	6
4	11	10	10
6	17	14	16
8	33	20	30
10	69	32	62
12	168	57	143
14	395	105	326
16	946	210	749
18	2205	380	1815
20	5224	659	3991
22	11821	1194	9119
24	26118	2232	19471

FIGURE 9 – Nombre nœuds moyens visités par différentes heuristiques

d	A*(h_1)	A*(h_2)	A*(h_3)
2	1.73	1.73	1.73
4	1.36	1.32	1.34
6	1.29	1.22	1.27
8	1.30	1.19	1.28
10	1.33	1.20	1.31
12	1.38	1.22	1.35
14	1.40	1.24	1.38
16	1.42	1.27	1.40
18	1.44	1.28	1.42
20	1.45	1.28	1.42
22	1.45	1.29	1.43
24	1.46	1.30	1.44

FIGURE 10 – Facteur d'embranchement effectif pour différentes heuristiques

d'embranchement effectif en résolvant l'équation par dichotomie (figure 10). On remarque alors bien que h_2 et h_3 sont meilleurs que h_1 mais aussi que h_2 est en fait clairement meilleur que h_3.

3.3.2 Variation de la performance avec l'importance de l'heuristique

Une autre manière de mesurer la qualité d'une heuristique est de mesurer les améliorations de la performance d'A* en fonction de l'importance qu'on donne à l'heuristique. On considère $g + th$ pour $0 \leq t \leq 1$. La recherche est toujours optimale puisque l'heuristique $h' = th$ est aussi consistante. Afin de percevoir l'évolution de la performance, le nombre de nœuds explorés en moyenne par A* a été mesuré pour une profondeur de solution donnée (figure 11).

Les trois courbes ont une allure exponentielle, ce qui se vérifie en mettant une échelle logarithmique en ordonnée (figure 11). On approche chacune des courbes par une fonction de la forme $\phi : t \mapsto \alpha \exp(-\beta t)$ en se servant des points d'abscisse 0.1 et 1 pour calculer α et β. On se place pour le premier point en 0.1 à cause d'un cas limite pour $t < 0.1$. Le coefficient β est caractéristique de l'évolution, c'est donc lui qui va être intéressant. Plus il est grand, plus le nombre de nœuds décroit rapidement avec l'importance de l'heuristique et donc plus l'heuristique est importante pour les performances de l'algorithme.

Les calculs de β pour les heuristiques du taquin (figure 12) classent les heuristiques dans le même ordre que le facteur d'embranchement effectif : h_2 , h_3, h_1 de la meilleure à la moins bonne.

3.3.3 Erreur relative d'une heuristique

La proposition 6 montre que plus une heuristique est proche du coût réel de la solution optimale k, plus A* est performant et donc plus l'heuristique est de qualité. Il parait donc logique de définir une mesure de qualité de l'heuristique par son écart avec k.

Definition 7. *L'erreur relative d'une heuristique h est $\epsilon = E(\frac{k(N)-h(N)}{k(N)})$ où N est une variable aléatoire suivant une loi uniforme sur une partie finie des nœuds du graphe.*

Plus ϵ est proche de 0, meilleur l'heuristique est et plus ϵ est proche de 1 moins l'heuristique est de qualité. A priori, l'inconvénient de cette mesure est de ne pas donner directement d'estimation des performances d'A*.

Afin de pouvoir effectuer l'estimation, on se place dans un cas simple. On se donne un arbre de facteur de branchement fixé à b et avec un seul état final n_f à la profondeur d_f. Pour simplifier le calcul, on se donne aussi une heuristique h telle que $h = (1 - \epsilon)k$ avec $\epsilon \in [0; 1]$. On détermine alors le nombre de nœuds N que l'algorithme parcours avant d'arriver à n_f.

Dans le meilleur des cas, le premier nœud visité à la profondeur d_f est n_f et on explore donc tous les nœuds n tels que $g(n) < d_f - h(n)$ et n_f. Dans le pire des cas, n_f est le dernier nœud visité à la profondeur d_f, on explore donc tous les

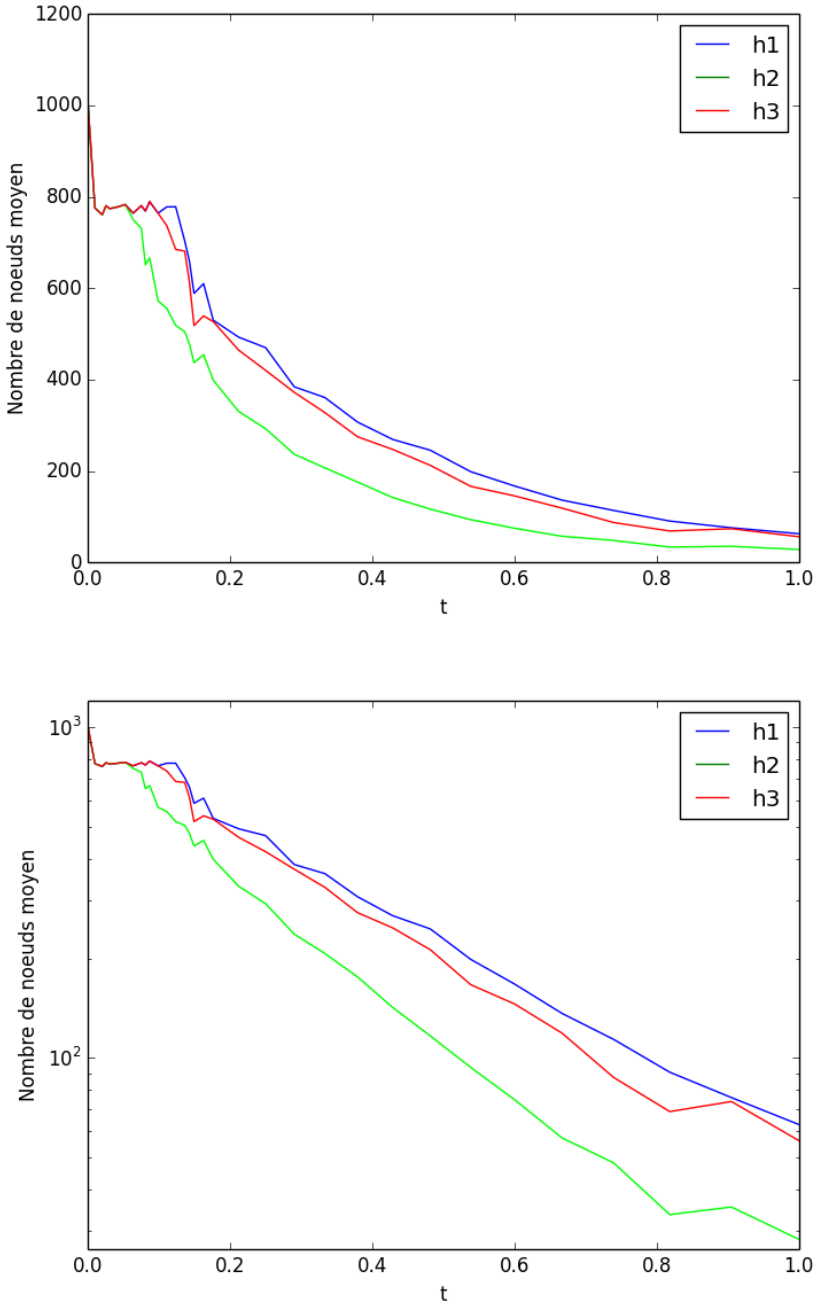

FIGURE 11 – Évolution du nombre de nœuds explorés moyen à une profondeur $d = 10$ en fonction de l'importance de l'heuristique en échelle linéaire et logarithmique

heuristique	h_1	h_2	h_3
β	3.13	4.09	3.28

FIGURE 12 – Valeurs de β pour les heuristiques du taquin

nœuds tels que $g(n) \leq d_f - h(n)$. On a donc $m = |\{n, g(n) < d_f - h(n)\}| + 1 \leq N \leq |\{n, g(n) \leq d_f - h(n)\}| = M$.

Or pour un nœud n à la profondeur d, on a $g(n) = d$ et $k(n) = (d_f - d_a) + (d - d_a)$ où d_a est la profondeur de l'ancêtre commun à n et n_f. On dénombre donc les nœuds selon la profondeur de leur ancêtre commun avec n_f.

Calculons d'abord m. Soit $d_a \in \{0, ..., d_f - 1\}$, les nœuds qui ont un ancêtre commun avec n_f à d_a et qui vont être explorés sont ceux de profondeur d telle que $d < d_f - (1-\epsilon)((d_f - d_a) + (d - d_a))$ i.e. $d < \frac{\epsilon d_f + 2(1-\epsilon)d_a}{(2-\epsilon)}$ soit $d \leq \left\lceil \frac{\epsilon d_f + 2(1-\epsilon)d_a}{(2-\epsilon)} - 1 \right\rceil = d_l$. Il y en a donc $(b-1)\sum_{d=d_a+1}^{d_l} b^{d-(d_a+1)} + 1 = b^{d_l - d_a}$. Puis en sommant sur d_a, on a $m = \sum_{d_a=0}^{d_f-1} b^{\left\lceil \frac{\epsilon d_f + 2(1-\epsilon)d_a}{(2-\epsilon)} - 1 \right\rceil - d_a} + 1$.

On procède de même pour M. Soit $d_a \in \{0, ..., d_f\}$, les nœuds qui ont un ancêtre commun avec n_f à d_a et qui vont être explorés sont ceux de profondeur d telle que $d \leq d_f - (1-\epsilon)((d_f - d_a) + (d - d_a))$ i.e. $d \leq \left\lfloor \frac{\epsilon d_f + 2(1-\epsilon)d_a}{(2-\epsilon)} \right\rfloor = d_l$. Il y en a donc $(b-1)\sum_{d=d_a+1}^{d_l} b^{d-(d_a+1)} + 1 = b^{d_l - d_a}$. Puis en sommant sur d_a, on a $M = \sum_{d_a=0}^{d_f} b^{\left\lfloor \frac{\epsilon d_f + 2(1-\epsilon)d_a}{(2-\epsilon)} \right\rfloor - d_a}$.

On obtient finalement le résultat suivant.

Proposition 7.

$$\sum_{d_a=0}^{d_f-1} b^{\left\lceil \frac{\epsilon d_f + 2(1-\epsilon)d_a}{(2-\epsilon)} - 1 \right\rceil - d_a} + 1 \leq N \leq \sum_{d_a=0}^{d_f} b^{\left\lfloor \frac{\epsilon d_f + 2(1-\epsilon)d_a}{(2-\epsilon)} \right\rfloor - d_a}$$

On en utilisant que $x \leq \lceil x \rceil$ et $\lfloor x \rfloor \leq x$, on obtient un premier corollaire.

Corollaire 1.

$$\frac{1}{b} \frac{b^{\frac{\epsilon}{2-\epsilon}(d_f+1)} - b^{\frac{\epsilon}{2-\epsilon}}}{b^{\frac{\epsilon}{2-\epsilon}} - 1} \leq N \leq \frac{b^{\frac{\epsilon}{2-\epsilon}(d_f+1)} - 1}{b^{\frac{\epsilon}{2-\epsilon}} - 1}$$

On en déduit alors deux corollaires immédiats.

Corollaire 2. *Le nombre de nœuds explorés par A* est en* $O(b^{\frac{\epsilon}{2-\epsilon}d})$

Corollaire 3.

$$b^* \leq b^{\frac{\epsilon}{2-\epsilon}}$$

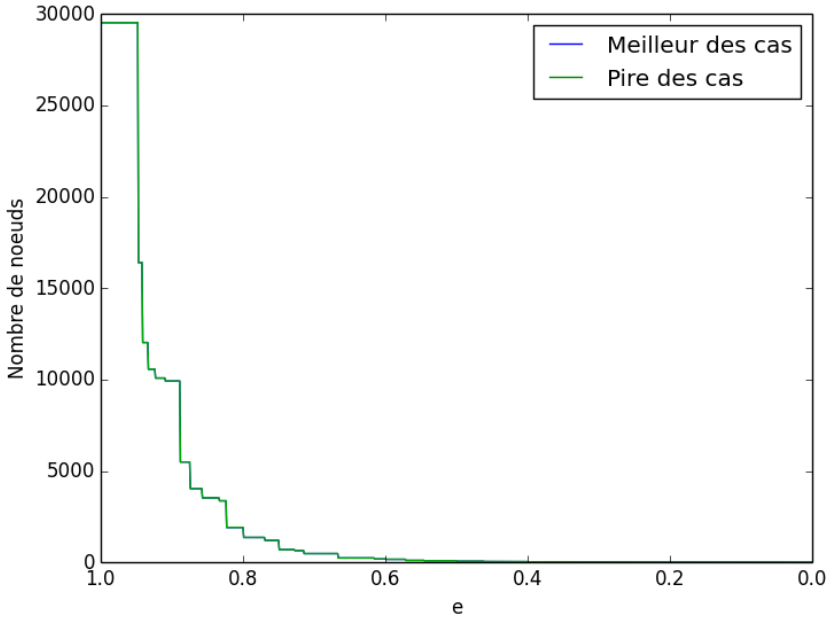

FIGURE 13 – Évolution du nombre de nœuds explorés en fonction de l'erreur relative pour $b = 3$ et $d = 10$

heuristique	h_1	h_2	h_3
ϵ	0.59	0.34	0.56

FIGURE 14 – Valeurs de l'erreur relative pour les heuristiques du taquin

Avec ce modèle, il est clair qu'avoir une bonne heuristique améliore considérablement les performances d'A* (figure 13). De plus, vu la dépendance du nombre de nœuds explorés de ϵ, il vaut mieux avoir une heuristique coûteuse en temps bien informée i.e. avec un ϵ plus faible qu'une moins coûteuse mais moins bien informée.

Il est maintenant intéressant de savoir si ces résultats restent d'une certaine manière valables dans le cas général. C'est-à-dire, savoir si le nombre de nœuds développés par A* avec une heuristique h quelconque d'erreur relative ϵ est en $O(b_{moy}^{\frac{\epsilon}{2-\epsilon}d})$ avec b_{moy} le facteur de branchement moyen du graphe d'états.

Pour le taquin, on calcule ϵ pour chacune des trois heuristiques (figure 14). On remarque que comme pour les mesures précédentes, on a que h_2 est meilleure que h_3 qui est meilleure que h_1.

Calculons maintenant le facteur de branchement moyen du graphe des états

heuristique	h_1	h_2	h_3
$b_{moy}^{\frac{\epsilon}{2-\epsilon}}$	1.51	1.22	1.46

FIGURE 15 – Majorant du facteur de branchement effectif du modèle simplifié appliqué aux heuristiques du taquin

du taquin. On a :

— Les états où le vide est dans un coin de la grille ont 2 voisins, il y en a $4(np-1)!$;

— Les états où le vide est sur un bord mais pas dans un coin ont 3 voisins, il y en a $(2(n+p)-8)(np-1)!$;

— Les autres états ont 4 voisins, il y en a $(np-(2(n+p)-4))(np-1)!$.

On a donc $b_{moy} = \frac{2\times 4+3(2(n+p)-8)+4(np-(2(n+p)-4))}{np}$ ce qui pour le taquin à 8 pièces fait $b_{moy} = \frac{8}{3}$.

Les majorants des facteurs de branchement effectif sont calculés via la formule obtenue dans la proposition 3 (figure 15) Pour h_1 et h_3, la valeur obtenue est en accord avec les valeurs expérimentales de la figure 10. Cependant la valeur obtenue pour h_2 est bien plus faible. La variance d'une variable aléatoire définie comme dans la définition 7 est d'environ 0.01 pour h_1 et h_3 mais de 0.02 pour h_2. Les valeurs sont plus dispersées loin de l'espérance pour h_2 ce qui rend le modèle simplifié adopté moins valable et explique l'écart trouvé.

Conclusion

En conclusion, on peut tout d'abord noter l'importance d'A* en intelligence artificielle, en particulier, il est utilisé dans les jeux-vidéos et par les récepteurs GPS.

Ensuite, souvent ce qui empêchera la terminaison sur des ordinateurs d'A* n'est pas son coût temporel trop important mais le coût spatial induit par la frontière et la structure de données contenant les nœuds déjà parcourus. Il existe des variations d'A* comme SMA* qui en dépit d'une perte de performance utilise une quantité de mémoire bornée.

Finalement, il demeure plusieurs questions ouvertes :

— A quelle condition sur l'heuristique le facteur d'embranchement converge-t'il quand d tend vers l'infini ?

— A-t'on toujours une exponentielle lorsqu'on fait varier l'importance d'une heuristique et peut-on l'expliquer ?

— Pour le taquin les trois mesures classent dans le même ordre les heuristiques testées. Peut-on alors montrer une équivalence dans l'ordre défini par les trois mesures ?