

# Comment optimiser A\* adaptatif

## Adaptive A\* : how to best exploit past problem-solving episodes

Lou Fedon<sup>1,2</sup>

Antoine Cornuéjols<sup>2</sup>

<sup>1</sup> Équipe Inférence et Apprentissage,  
Laboratoire de Recherche en Informatique (LRI)  
Bâtiment 490, Université Paris-Sud, 91405 - Orsay Cedex (France)

[lou.fedon@lri.fr](mailto:lou.fedon@lri.fr)

<sup>2</sup> UMR AgroParisTech/INRA 518  
16, rue Claude Bernard, F-75231 Paris Cedex 05 (France)

[antoine.cornuejols@agroparistech.fr](mailto:antoine.cornuejols@agroparistech.fr)

<http://www.lri.fr/~antoine>

### Résumé

La recherche efficace d'un chemin (quasi) optimal dans un graphe reste une tâche fondamentale en Intelligence Artificielle. Des travaux récents [7, 4, 5, 8] ont contribué à renouveler l'approche de ce problème en proposant une technique de mise au point d'heuristique par apprentissage et non par une abstraction statique de la description du problème. En nous référant à ces travaux, nous montrons qu'il est possible d'exploiter beaucoup plus précisément l'information apportée par des épisodes de résolution passés. Les expériences confirment la réduction de l'espace de recherche associé.

Par ailleurs, nous généralisons ces techniques d'adaptation d'heuristiques au cas de buts changeants. Les expériences et leur analyse montrent que, pour ces techniques, le changement de but doit obéir à des conditions strictes pour que le calcul adaptatif d'heuristique soit rentable.

### Mots Clefs

Planification, Apprentissage, Recherche heuristique, Graphes, A\* adaptatif.

### Abstract

Efficient search of (quasi-)optimal paths in graphs remains a fundamental task in Artificial Intelligence. Recent works [7, 4, 5, 8] have contributed to a new point of view on this problem whereby heuristics are learned from past solving experiences rather than derived through a static abstraction of the description of the problem.

In this paper, we show how to improve this work by better exploiting information from past solving episodes. The experiments reported here confirm the significant reduction

in search space achieved by our algorithm.

In a second part, we show how to generalize these learning techniques to the case of changing goal states. Extensive experiments and their analysis show that the variations of the goal states must obey strict laws in order for these adaptive A\* algorithms to be advantageous.

### Keywords

Heuristic search, Graphs, Adaptive A\*, Learning.

## 1 Introduction

Il existe plusieurs algorithmes de recherche dans les arbres et les graphes, tels que la programmation dynamique ou l'algorithme de Dijkstra [2]. Ce dernier est utilisé pour chercher un chemin de coût minimal entre un nœud de départ  $s$  et un nœud d'arrivée  $\bar{s}$ . L'algorithme A\* est une généralisation de ces algorithmes utilisant une fonction heuristique explicite [11]. Le coût d'un chemin est supposé égal à la somme des coûts des arêtes traversées (coûts tous supposés  $> 0$ ). L'algorithme A\* maintient à chaque instant un ensemble de nœuds frontière (noté OUVERT), chacun d'eux étant associé, de manière unique, à un chemin potentiel passant par ce nœud. Évaluant un nœud, l'algorithme A\* évalue ainsi de fait la valeur d'un chemin potentiel en calculant une fonction  $f(n) = g(s, n) + h(n, \bar{s})$  où  $g(s, n)$  est le coût du meilleur chemin connu jusque là entre  $s$  et  $n$ , et où  $h(n, \bar{s})$  est une estimation du coût restant entre  $n$  et  $\bar{s}$ . De nouveaux chemins sont considérés par l'algorithme en développant les nœuds frontière les plus prometteurs selon la fonction d'évaluation  $f$ .

Il est prouvé que A\* trouve un chemin optimal s'il en existe un entre  $s$  et  $\bar{s}$  et que la fonction heuristique uti-

lisée  $h(\cdot, \bar{s})$  est une sous-estimation du coût restant pour tous les nœuds. On parle alors de fonction heuristique *admissible*. Par ailleurs, étant données deux fonctions heuristiques  $h_1$  et  $h_2$  telles que  $\forall n \in \mathcal{S}$ , (l'ensemble des états)  $h_1(n, \bar{s}) < h_2(n, \bar{s})$ , l'ensemble des nœuds développés par l'algorithme heuristique utilisant  $h_2$  est un sous-ensemble des nœuds développés par le même algorithme utilisant  $h_1$  (on dit que  $h_2$  est *mieux informée* que  $h_1$ ).

La précision de la fonction heuristique est déterminante pour distinguer les problèmes résolubles en un temps CPU raisonnable. Ainsi par exemple, le taquin à 15 tuiles nécessitait en moyenne 6 jours de calcul en 2000 pour être résolu sans usage de fonction heuristique, alors que l'utilisation de l'heuristique comptant le nombre de tuiles mal placées ramenait ce temps à 50s, et celle sommant les distances de Manhattan à une fraction de seconde<sup>1</sup> ! ([9]).

C'est pourquoi de nombreux travaux de recherche ont porté sur la détermination de fonctions heuristiques aussi précises que possible. On peut distinguer dans ce cadre trois types d'approches :

1. par *abstraction*. Une fonction heuristique est obtenue par calcul d'une solution d'un problème simplifié obtenue de manière automatique par examen de la spécification du problème (voir notamment [12]). Lorsque le problème ne peut facilement être complètement décrit de manière formelle ou que le problème peut changer avec le temps, des approches par apprentissage sont préférables.
2. par mémorisation d'étapes de recherche intermédiaires dans les approches de *cognition située*. Les algorithmes proposés (par exemple [7, 1, 3, 6, 14, 15]) ne modifient pas explicitement la fonction heuristique utilisée pour se guider dans l'espace de recherche, en revanche, ils mémorisent des dépendances entre les états visités de l'environnement pour accélérer les recherches ultérieures.
3. par *apprentissage* à partir d'épisodes passés de résolution de problèmes similaires.

Des recherches récentes proposent des algorithmes de **modification explicite de la fonction heuristique** en fonction de l'expérience passée pour des problèmes similaires. Il s'agit en particulier des travaux de Koenig et de ses collaborateurs ([4, 5]). Le principe est très proche de ce que l'on appelle *apprentissage analytique* [10]. Il s'agit d'analyser la (les) trace(s) de résolution de problème(s) passé(s) afin de ne pas répéter les recherches inutiles et d'accélérer l'exploration des directions prometteuses, par exemple en apprenant des macro-opérateurs. Dans le cadre de la recherche A\*, l'idée des méthodes d'apprentissage est de ré-évaluer la fonction heuristique sur les états développés dans les explorations précédentes de telle manière que ces états ne soient pas développés inutilement dans une nou-

<sup>1</sup> Dans le cas du taquin à 24 tuiles, l'utilisation de  $h_2$  ramènerait le temps moyen de résolution à 65 000 ans au lieu de 12 milliards d'années sans emploi de fonction heuristique.

velle résolution du même problème. Koenig *et al.* ont montré que cette approche est avantageuse dans le cas où le but reste le même pour les différentes explorations, mais où la source peut changer et où l'environnement peut éventuellement subir certaines modifications (augmentation du coût de certains arcs).

Dans cet article, notre contribution est double. D'une part, nous montrons qu'*il est possible de mieux tirer parti des expériences de résolution passées* que ne le font Koenig *et al.* pour le même type de changements de tâches et d'environnements (section 2). D'autre part, nous *généralisons leur méthode au cas de source et de but changeants*<sup>2</sup>, et montrons que dans ce cas les conditions pour obtenir un gain en recherche sont très restrictives (section 3). Les perspectives ouvertes par ce travail sont discutées dans la conclusion.

## 2 Vers un meilleur transfert de connaissances

Les algorithmes que nous présentons ici traitent deux passes d'exploration. Toutes les grandeurs seront indicées par un paramètre  $i \in \{1, 2\}$  précisant à quelle passe elles font référence. Nous utiliserons les notations suivantes :

- $n$  représente un état (aussi appelé nœud) appartenant à l'espace  $\mathcal{S}$  (dans nos expériences, l'espace des états correspond à une grille comportant des obstacles (voir la figure 1))
- $g_i(s_i, n)$ , la distance minimale estimée (ou coût courant) entre l'état départ (ou source)  $s_i$  et l'état  $n$  à un moment donné de l'exploration.
- $h_i(n, \bar{s}_i)$ , la distance estimée entre  $n$  et l'état but  $\bar{s}_i$  retournée par la fonction heuristique  $h_i$ .
- $d_i(n, m)$ , la distance réelle entre deux nœuds  $n$  et  $m$  dans l'environnement associé au  $i^{\text{ème}}$  épisode.
- $f_i(n) = g_i(s_i, n) + h_i(n, \bar{s}_i)$  le coût estimé du chemin le plus court entre  $s_i$  et  $\bar{s}_i$ , passant par  $n$ .
- $\Gamma_i^*$ , le chemin optimal de  $s_i$  à  $\bar{s}_i$  dans le  $i^{\text{ème}}$  environnement.
- $C_i^* = d_i(s_i, \bar{s}_i)$ , le coût de  $\Gamma_i^*$ .

### 2.1 L'algorithme AdaptiveA\*

L'algorithme présenté par Koenig *et al.* ([4, 5]), utilise une première exploration d'une grille par un algorithme A\* entre une source  $s_1$  et un but  $\bar{s}$  pour améliorer, en terme de nombres de nœuds développés et de temps CPU, une recherche « similaire » entre une seconde source  $s_2$  et le but supposé identique  $\bar{s}$ .

Les auteurs considèrent des tâches avec deux sources proches, une heuristique initiale consistante<sup>3</sup>, et un environnement changeant « peu » entre les deux explorations. Plus précisément, le seul type de modification de l'envi-

<sup>2</sup> Depuis la rédaction de cet article, Koenig *et al.* [8] ont publié une version de leur algorithme pour le cas particulier de buts se déplaçant dans un labyrinthe. Notre étude généralise ce cas.

<sup>3</sup> On dit qu'une fonction heuristique est *consistante* si l'équation  $h(n, \bar{s}) \leq d(n, n') + h(n', \bar{s})$  est vérifiée pour toute paire de nœuds  $n$  et  $n'$ ,  $d(n, n')$  étant le coût d'un chemin optimal entre  $n$  et  $n'$ .

ronnement admis est l'ajout de contraintes, ce qui se traduit dans le cas d'un graphe par l'augmentation du coût des arcs, et dans une grille par l'ajout d'obstacles.

La mise à jour de l'heuristique se fait pour les nœuds développés par l'algorithme lors de la première étape et qui constituent l'ensemble  $\text{FERMÉ}_1$ . On obtient ainsi  $h_2$  :

$$h_2(n, \bar{s}) = \begin{cases} C_1^* - g_1(s_1, n) & \forall n \in \text{FERMÉ}_1 \\ h_1(n, \bar{s}) & \text{sinon} \end{cases} \quad (1)$$

Cette mise à jour cherche à la fois à exploiter au mieux l'information obtenue lors de la première recherche (les valeurs de  $g$  pour les nœuds de  $\text{FERMÉ}_1$ ) en vue d'une seconde tâche similaire tout en préservant l'admissibilité de  $h_2$  (voir section 2.2).

Cependant, s'il est indéniable que l'heuristique  $h_2$  sera mieux informée que  $h_1$ , le choix de développer un nœud ou pas se fait ultimement sur son estimation  $f$ . Or il est très frappant de remarquer que, lorsque confronté à la même recherche entre  $s_1$  et  $\bar{s}$ , mais en étant guidé par  $h_2$ , tous les nœuds de  $\text{FERMÉ}_1$  seraient évalués à :

$$f_2(n) = g_1(s_1, n) + h_2(n, \bar{s}) = g_1(s_1, n) + C_1^* - g_1(s_1, n) = C_1^*$$

La recherche serait confinée à  $\text{FERMÉ}_1$  et guidée par le hasard puisque tous les nœuds apparaîtraient comme également attractifs.

## 2.2 Quelques propriétés remarquables de AdaptiveA\*

$h_2$  possède trois propriétés particulièrement intéressantes :

1.  $h_2$  est *admissible*<sup>4</sup> : le coût du chemin optimal entre  $s_1$  et  $\bar{s}$  est nécessairement inférieur au coût du chemin le plus court entre ces deux points contraint à passer par un point  $n$  quelconque, ce que l'on écrit :

$$\begin{aligned} C_1^* &\leq g_1(n, s_1) + d_1(n, \bar{s}) \\ \Leftrightarrow C_1^* - g_1(n, s_1) &\leq d_1(n, \bar{s}) \end{aligned}$$

En considérant  $h_2$  et l'ensemble de ses valeurs, on a :

$$h_2(n, \bar{s}) = \begin{cases} C_1^* - g_1(n, s_1) & \forall n \in \text{FERMÉ}_1 \\ h_1(n, \bar{s}) & \forall n \notin \text{FERMÉ}_1 \end{cases} \leq d_1(n, \bar{s})$$

$h_2$  est donc garantie de sous-estimer la distance au but.

2.  $h_2$  est *mieux informée* que  $h_1$ . En effet,  $\text{FERMÉ}_1$  constitue l'ensemble des nœuds développés lors de la première exploration parce qu'ils semblaient plus prometteurs en terme de  $f$  que le nœud but, dont la valeur

$f$  est nécessairement évalué à  $C_1^*$  lorsqu'on utilise une heuristique consistante. Soit :

$$\begin{aligned} \forall n \in \text{FERMÉ}_1, & \quad f_1(n) \leq C_1^* \\ \Leftrightarrow & \quad g_1(s_1, n) + h_1(n, \bar{s}) \leq C_1^* \\ \Leftrightarrow & \quad h_1(n, \bar{s}) \leq C_1^* - g_1(s_1, n) = h_2(n, \bar{s}) \\ \forall n \notin \text{FERMÉ}_1, & \quad h_1(n, \bar{s}) = h_2(n, \bar{s}) \end{aligned}$$

On en conclut :  $\forall n, h_1(n, \bar{s}) \leq h_2(n, \bar{s})$  : l'heuristique  $h_2$  est au moins aussi bien informée que  $h_1$ .

### 3. $h_2$ reste consistante<sup>5</sup>.

Une heuristique admissible  $h_2$  développera moins de nœuds qu'une heuristique  $h_1$  si elle est *strictement mieux informée* sur  $\mathcal{S}$  :  $\forall n, h^*(n) \geq h_2(n) > h_1(n)$  avec  $h^*$  une heuristique parfaitement informée.

Sans cette relation d'ordre strict, on ne peut pas conclure sur le gain en nœuds développés de  $h_2$  par rapport à  $h_1$ . C'est pourquoi [4] présente des expérimentations pour confirmer le gain en terme de nœuds développés et de temps CPU obtenu avec leur approche.

On peut cependant discuter cette technique sur deux points. D'une part, même s'il est démontré que l'heuristique apprise est mieux informée que l'heuristique initiale, la qualité de l'information obtenue lors de la mise à jour est très variable selon les nœuds. Pour tous les points à une même distance  $g_1$  de la première source, on affecte la même estimation  $h_2 = C_1^* - g_1$ . L'implication immédiate est que les courbes de niveaux de  $h_2$  (dans la région  $\text{FERMÉ}_1$ ) sont centrées autour de  $s_1$ , bien que cette grandeur soit supposée estimer la distance à  $\bar{s}$ .

D'autre part, nous allons montrer que, dans le cas où le but  $\bar{s}$  reste le même, l'algorithme AdaptiveA\* n'exploite pas toute l'information qui peut être tirée d'une première exploration par l'algorithme A\*.

### 2.3 L'algorithme ReverseAdaptiveA\*

L'algorithme ReverseAdaptiveA\*, que nous présentons ici, se place dans les mêmes conditions que AdaptiveA\* : on réalise deux recherches successives partant de la source  $s_1$  puis de la source  $s_2$  vers un même but  $\bar{s}$ . L'information obtenue lors de la première exploration est utilisée pour créer une nouvelle heuristique  $h_2$  hybride : les évaluation des nœuds sur l'espace  $\text{FERMÉ}_1$  exploré lors de la première recherche sont modifiées alors que l'évaluation de l'heuristique  $h_1$  sur le reste de l'espace est conservée. L'environnement peut devenir plus contraint entre les deux tâches de recherche de chemin.

L'observation essentielle dont découle notre approche est que, dans la  $i^{\text{ème}}$  exploration par A\*, les grandeurs  $g_i(s_i, n)$  et  $h_i(n, \bar{s})$  associées à un nœud  $n$  jouent des rôles symétriques. Elles estiment respectivement les distances

<sup>4</sup>i.e,  $\forall n \in \mathcal{S}, h_2(n, \bar{s}) \leq d(n, \bar{s})$ .

<sup>5</sup>La démonstration est disponible dans l'article [5].

$d_i(s_i, n)$  et  $d_i(n, \bar{s})$ . De plus, lorsque l'heuristique utilisée est consistante, on peut montrer (voir [11], théorème 10, p.83) que les évaluations  $g$  des nœuds développés sont exactes ( $g_i(s_i, n) = d_i(s_i, n)$ ). Si comme Koenig et al. on suppose  $h_1$  consistante, on a accès pour tous les nœuds de  $\text{FERMÉ}_1$  à leur distance exacte à la source. Notre approche consiste à réutiliser cette information dans une heuristique :

$$h_2(n, \bar{s}) = \begin{cases} g_1(\bar{s}, n) & \forall n \in \text{FERMÉ}_1 \\ h_1(n, \bar{s}) & \text{sinon} \end{cases} \quad (2)$$

Concrètement cela signifie qu'au lieu d'effectuer une exploration entre  $s_1$  et  $\bar{s}$  avant de chercher un chemin entre  $s_2$  et  $\bar{s}$ , le sens de la première recherche peut être inversé avec profit. En effet, la première recherche  $A^*$  effectuée, avec  $\bar{s}$  comme source et  $s_1$  comme but, permet d'obtenir les distances exactes des nœuds développés à  $\bar{s}$  grâce aux valeurs de  $g_1(\bar{s}, n)$  calculées. Cette technique ne peut s'employer que pour des environnements où les arcs ont des coûts symétriques, ce qui est un cas fréquent.

---

**Algorithme 1** ReverseAdaptiveA\*( $s_1, s_2, \bar{s}, h_1$ , environnement)

---

```

source =  $\bar{s}$ 
but =  $s_1$ 
FERMÉ1 = A*(source, but,  $h_1$ , environnement)
 $h_2 = h_1$ 
environnement = environnement + obstacles
pour  $n \in \text{FERMÉ}_1$  faire
     $h_2(n, \bar{s}) = g_1(\bar{s}, n)$ 
fin pour
source =  $s_2$ 
but =  $\bar{s}$ 
FERMÉ2 = A*(source, but,  $h_2$ , environnement)

```

---

On peut rapprocher cette manière d'informer l'heuristique en partant de l'état but des techniques de recherche bi-directionnelle. Cependant, même s'il existe un travail sur ce sujet, la recherche bi-directionnelle guidée par heuristique est très délicate à mettre en œuvre. Une référence plus pertinente pourrait être l'approche "pattern database" développée par Korf en particulier [13], où on effectue une recherche limitée en  $A^*$  à partir du but afin de renseigner l'heuristique en préalable de l'exploration que l'on veut réellement réaliser. Notre méthode se distingue de ces deux approches, d'une part parce qu'elle profite des expériences passées, et, d'autre part, parce qu'elle ne nécessite pas d'analyse préalable du problème et s'adapte aux mondes changeants.

## 2.4 Quelques propriétés

La nouvelle heuristique générée dans ReverseAdaptiveA\* est parfaitement informée sur  $\text{FERMÉ}_1$  et identique à  $h_1$  sur le reste du monde. De fait,  $h_2$  est *admissible* et est *mieux informée* que  $h_1$

**Théorème 1** (Consistance de  $h_2$ ). *L'heuristique  $h_2$  apprise par ReverseAdaptiveA\* est consistante.*

*Démonstration.* Pour prouver la consistance de l'heuristique apprise, il est suffisant de prouver que :

$\forall n, n'$  tels que  $n'$  est un descendant de  $n$ ,

$$h_2(n, \bar{s}) \leq h_2(n', \bar{s}) + d(n, n')$$

Si  $n, n' \in \text{FERMÉ}_1$ ,

$$\forall n, h_2(n, \bar{s}) = g(\bar{s}, n) = h^*(n, \bar{s}) = d(n, \bar{s}).$$

Lorsque  $n, n' \notin \text{FERMÉ}_1$ , on a  $h_2(n, \bar{s}) = h_1(n, \bar{s})$  et  $h_1$  est supposé être consistant pour n'importe quel état but.

Il reste alors le cas où  $n \notin \text{FERMÉ}_1$  et  $n' \in \text{FERMÉ}_1$ .

Prouvons le théorème par l'absurde.

Supposons que  $h_2$  n'est pas consistante, c'est-à-dire que

$$h_2(n, \bar{s}) > h_2(n', \bar{s}) + d(n, n')$$

Puisque  $n \notin \text{FERMÉ}_1$  et  $n' \in \text{FERMÉ}_1$ , on a  $h_2(n, \bar{s}) = h_1(n, \bar{s})$  et  $h_2(n', \bar{s}) = g(\bar{s}, n')$ , d'où :

$$h_1(n, \bar{s}) > g(\bar{s}, n') + d(n, n')$$

$g(\bar{s}, n') + d(n, n')$  représente le coût du chemin optimal entre  $\bar{s}$  et  $n$  passant par  $n'$ , on peut donc le minorer par  $C^*(n, \bar{s})$ . Ce qui entraîne :  $h_1(n, \bar{s}) > C^*(n, \bar{s})$ .

Ceci contredit l'hypothèse initiale sur l'admissibilité de  $h_1$ .  $\square$

L'heuristique générée par notre méthode est mieux informée que celle calculée par AdaptiveA\*. Elle conserve néanmoins toutes les propriétés importantes des bonnes heuristiques, et le coût en terme de mémoire est identique : les deux algorithmes doivent conserver une table des valeurs des nœuds de  $\text{FERMÉ}_1$ . Finalement, l'information gagnée lors de la mise à jour est équilibrée sur l'ensemble de  $\text{FERMÉ}_1$  : dans le cas de AdaptiveA\*, l'estimation pour un nœud  $n$  étant surtout précise pour les nœuds proches de  $\Gamma_1^*$ , alors qu'elle est parfaite pour tous les nœuds de  $\text{FERMÉ}_1$  pour ReverseAdaptiveA\*.

Pour toutes ces raisons, ReverseAdaptiveA\* doit développer moins de nœuds pour la deuxième tâche qu'en utilisant AdaptiveA\*, et cela sans coût calculatoire (le temps CPU moyen de développement d'un nœud est très lié au temps d'accès des informations stockées) ou coût mémoire supérieur.

## 2.5 Expériences et résultats

Nous avons comparé les deux algorithmes sur des grilles (voir figure 1) de taille  $1000 \times 1000$ , pour différentes densités d'obstacles. Ceux-ci ont été générés de manière aléatoire suivant un tirage uniforme sur la grille. Les positions des sources et du but ont également été choisies de manière aléatoire dans un carré centré par rapport à la grille et de dimension  $600 \times 600$ . Cette précaution permet de s'affranchir de possibles artefacts induits par une trop grande proximité entre les sources et/ou les cibles et le

bord de la grille. La position de la source  $s_2$  a été choisie dans une boule (au sens de la distance  $L_1$ ) centrée sur  $s_1$ . Le rayon de cette boule est également un paramètre des expérimentations. Tous les résultats obtenus utilisent la distance de Manhattan comme heuristique  $h_1$ .

Dans cet article, nous nous sommes concentrés sur un monde de grilles connectées à ses quatre voisins cardinaux. Cette modélisation est souvent considérée comme trop limitée car le facteur de branchement est faible et constant et les coûts des arcs sont uniformes. C'est pourquoi nous avons introduit dans notre approche des éléments permettant de rendre ce type de monde assez générique, malgré tout. Ainsi, faire varier le taux d'obstacles revient à modifier le facteur de branchement, non seulement directement (par exemple, quand le taux d'obstacles est de 25%, le facteur de branchement moyen tombe à 3), mais aussi plus globalement car le nombre de chemins entre deux points peut varier considérablement avec le taux d'obstacles. Par ailleurs, selon la granularité de l'observation, le monde des grilles peut être vu comme relativement uniforme (même facteur de branchement, même coût des arcs, vue « de près »), ou au contraire comme très varié (vue « de loin »). Nous nous sommes limités à un taux d'obstacles maximal de 24%, car à partir de cette valeur et pour le type de grilles sur lequel nous avons conduit nos expériences, il devient rare qu'il existe un chemin entre les sources et le but. Il faut noter que les mesures obtenues sont dépendantes de la gestion des priorités entre deux nœuds dont la valeur  $f$  est également attractive. Dans cette étude, nous donnons la priorité aux nœuds dont le  $g$  est maximal.

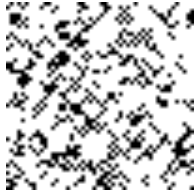


FIG. 1 – Exemple de grille de taille  $50 \times 50$  avec obstacles engendrés aléatoirement ( $\tau = 24\%$ ).

Le gain (voir formule 3) représente l'économie en terme de nœuds développés pour la deuxième exploration par rapport à une exploration avec un A\* classique. Il est ici fonction de l'ensemble  $\text{FERMÉ}'_2$  des nœuds développés pour la recherche d'un chemin entre la source  $s_2$  et la cible  $\bar{s}$  sans apprentissage et de l'ensemble  $\text{FERMÉ}'_2$  des nœuds développés avec apprentissage

$$\text{gain} = \frac{|\text{FERMÉ}'_2 \setminus \text{FERMÉ}_2|}{|\text{FERMÉ}'_2 \setminus \Gamma^*|} \in [0, 1] \quad (3)$$

On remarquera que le gain est normalisé et indépendant des tailles de l'espace de recherche. Il est ainsi maximal (égal à 1) dans le cas où  $\text{FERMÉ}'_2 = \Gamma^*$ , c'est-à-dire dans le cas où, après apprentissage, seuls les nœuds du

$\tau$	$\mu^{\text{AdaptiveA}^*}$	$\sigma^{\text{AdaptiveA}^*}$	$\mu^{\text{ReverseAdaptiveA}^*}$	$\sigma^{\text{ReverseAdaptiveA}^*}$
5.	<b>0.21</b>	0.32	0.17	0.29
8.	0.32	0.33	0.31	0.34
10.	0.40	0.31	0.41	0.34
12.	0.46	0.28	<b>0.50</b>	0.32
15.	0.49	0.24	<b>0.57</b>	0.29
18.	0.50	0.23	<b>0.62</b>	0.27
20.	0.51	0.23	<b>0.64</b>	0.28
22.	0.49	0.27	<b>0.57</b>	0.31
24.	0.44	0.29	<b>0.47</b>	0.34

TAB. 1 – Gain en fonction de la densité d'obstacles.

chemin optimal sont développés. Il est minimal (égal à 0) dans le cas où  $\text{FERMÉ}'_2 = \text{FERMÉ}_2$ , c'est-à-dire quand l'apprentissage n'a pas modifié l'espace de recherche.

Dans la **première expérience**, nous avons fait varier la source entre deux explorations. La densité d'obstacles a été prise dans l'ensemble  $[5\%, 24\%]$  et a été choisie comme principal paramètre pour comparer le gain moyen des deux algorithmes. Les expériences ont été conduites 8000 fois sur deux sortes de systèmes : deux processeurs bicore AMD Opteron 265 1.8 GHz avec 2 Go de mémoire vive et deux processeurs dualcore Intel Xeon 5130 2 GHz avec 4 Go de mémoire vive.

Les résultats sont présentés dans la table 1 et la figure 2. La première colonne précise le *taux d'obstacles* présents dans la grille. Les deuxième et troisième colonnes fournissent respectivement le *gain moyen* et la *déviations standard* pour AdaptiveA\*. Les quatrième et cinquième colonnes présentent les mêmes mesures pour ReverseAdaptiveA\*.

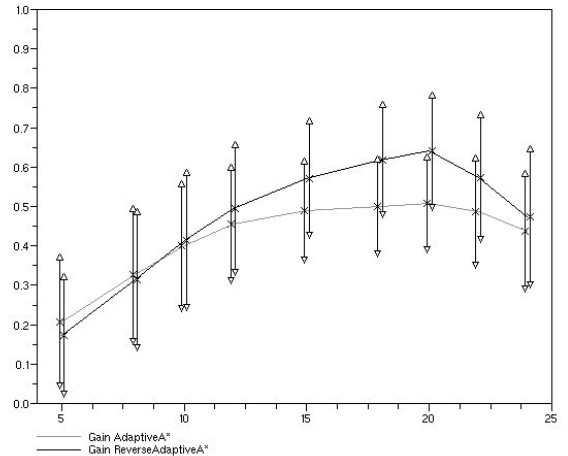


FIG. 2 – Gain moyen pour différentes valeurs de la densité d'obstacles avec déviation standard.

Ces résultats montrent que ReverseAdaptiveA\* tire un meilleur avantage de l'information calculée lors de la première exploration : son gain moyen est presque toujours supérieur à celui de AdaptiveA\*. Pour une densité

d'obstacles de 20%, le gain est de 25% par rapport à AdaptiveA\*. Ceci représente une amélioration significative par rapport à  $h_{\text{AdaptiveA}^*}$  : en moyenne,  $h_{\text{ReverseAdaptiveA}^*}$  explore seulement 36% (= 1-0.64) des nœuds explorés sans apprentissage, alors que  $h_{\text{AdaptiveA}^*}$  en explore encore 49%. Or il est à noter, à nouveau, que cette amélioration est obtenue pour un coût computationnel (calculatoire et en espace mémoire) supplémentaire nul. Dans cette expérience ainsi que dans le reste de l'article, nous ne comparons les différents algorithmes qu'en terme de nœuds développés. La précision de nos expériences sur les mesures temporelles (durées de l'ordre du dixième de seconde) étant trop faible pour fournir des comparaisons significatives.

La **seconde expérience** reprend le même protocole expérimental, à la différence près qu'on rajoute des obstacles entre les deux explorations. On compare les gains de AdaptiveA\* et ReverseAdaptiveA\* pour une densité initiale allant de 5% à 24% et une augmentation du nombre d'obstacles allant de 0% (première expérience) à 20%.

Les résultats présentés dans la table 2, se lisent comme suit : les lignes indiquent la densité d'obstacles initiale et les colonnes la proportion additionnelle ajoutée entre les deux explorations. Dans chaque cellule, la valeur de gauche représente le gain moyen pour AdaptiveA\* et celle de droite celui de ReverseAdaptiveA\*. Les expériences ont été conduites 8000 fois sur les mêmes systèmes que lors de l'expérience précédente.

Ces résultats sont comparables au cas où l'on fait uniquement varier la source. Cela peut paraître surprenant car la caractéristique qui faisait la force de l'approche ReverseAdaptiveA\* n'est pas conservée : l'information incorporée lors de la mise à jour n'est plus optimale. Lorsque des obstacles apparaissent entre les deux explorations, les distances entre les nœuds augmentent et on ne peut plus garantir que  $h_2$  estimera exactement les distances sur  $\text{FERMÉ}_1$ . Elle est néanmoins toujours mieux informée que si l'on avait appliqué AdaptiveA\*, ce qui conduit à une économie du nombre de nœuds développés.

Au vu des résultats (table 2), le taux d'obstacles a un impact significatif sur les gains réalisés par les deux algorithmes : plus il y a d'obstacles qui apparaissent entre deux tâches successives, plus les gains réalisés par les deux algorithmes seront réduits.

Enfin, des expériences similaires ont été réalisées en utilisant comme heuristique de base la distance  $L_2$ . Les résultats confirment ceux obtenus en utilisant  $L_1$  : notre approche permet, dans ce cas également, une meilleure utilisation de l'information, ce qui rend l'algorithme beaucoup plus rapide lors de la deuxième exploration

Le figure 3 illustre le fait que l'ensemble des nœuds explorés peut être extrêmement réduit lorsqu'on utilise l'algorithme AdaptiveA\* et encore plus avec

ReverseAdaptiveA\*. Les lignes représentent les états frontière de l'exploration (également connus comme l'ensemble OUVERT) et les losanges les nœuds développés (l'ensemble FERMÉ).

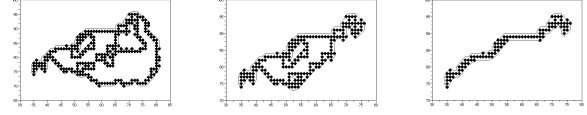


FIG. 3 – Contour de  $\text{FERMÉ}_2$  pour une exploration par un A\* classique (à gauche), pour AdaptiveA\* (centre), pour ReverseAdaptiveA\* (à droite).

### 3 Quand le but change

L'algorithme décrit par Koenig et al. [4] tire profit de l'épisode de recherche précédent en examinant l'espace de recherche développé par l'algorithme A\*. Il est alors naturel de se demander si cette approche peut être utilisée lorsque le but change également.

Des adaptations sont nécessaires. L'information mémorisée entre les deux étapes de l'apprentissage est en effet liée au but  $\bar{s}$ . D'autre part, il semble problématique de conserver les propriétés d'admissibilité et de consistance pour une heuristique apprise par rapport à un but et utilisée pour guider la recherche vers un but différent.

#### 3.1 Adaptation directe de AdaptiveA\*

En effet, si le nouveau but  $\bar{s}_2$  est plus proche de la source que le précédent but  $\bar{s}_1$ , l'admissibilité de la fonction heuristique n'est plus garantie. Si l'on tient à conserver l'assurance que le chemin retourné est optimal, il devient nécessaire de corriger la fonction heuristique pour garantir son caractère admissible quelle que soit la position (plus proche ou plus éloignée) du nouveau but.

Pour cela, il faut corriger la fonction heuristique apprise en la diminuant de la distance entre  $\bar{s}_1$  et  $\bar{s}_2$  :  $d(\bar{s}_1, \bar{s}_2)$ , soit :  $\forall n \in \text{FERMÉ}_1$ ,

$$h'_2(n, \bar{s}_2) = \begin{cases} h_2(n, \bar{s}_1) - d(\bar{s}_1, \bar{s}_2) & \text{si } h_2(n, \bar{s}_1) > d(\bar{s}_1, \bar{s}_2) \\ h_1(n, \bar{s}_1) & \text{sinon} \end{cases} \quad (4)$$

Cela nécessite bien entendu que la distance  $d(\bar{s}_1, \bar{s}_2)$  soit déterminée, ce qui peut s'opérer par une recherche en A\* utilisant la fonction heuristique initiale (supposée admissible)  $h_1$ . On obtient ainsi la nouvelle fonction d'évaluation :

$\forall n \in \text{FERMÉ}_1$ ,

$$f_2(n) = g(s_2, n) + \text{Max}[(C_1^* - d(\bar{s}_1, \bar{s}_2) - g(s_1, n)), h_1(n, \bar{s}_2)] \quad (5)$$

Le premier terme dans la fonction *Max* correspond à l'heuristique apprise grâce à l'épisode d'exploration précédent, tandis que le second terme est lié à la fonction heuristique précédente. L'utilisation de la fonction *Max* garantit que

	0		4		8		12		16		20	
5	0.21	0.17	0.32	0.36	0.29	0.35	0.27	0.34	0.26	0.32	0.24	0.31
8	0.32	0.31	0.36	0.44	0.32	0.42	0.29	0.40	0.27	0.38	0.25	0.36
10	0.40	0.41	0.38	0.41	0.36	0.39	0.34	0.38	0.33	0.38	0.31	0.37
12	0.46	0.50	0.41	0.48	0.39	0.47	0.36	0.45	0.34	0.44	0.32	0.42
15	0.49	0.57	0.43	0.55	0.39	0.52	0.34	0.50	0.32	0.48	0.29	0.45
18	0.50	0.62	0.40	0.57	0.33	0.53	0.29	0.49	0.25	0.46	0.22	0.43
20	0.51	0.64	0.39	0.59	0.32	0.54	0.28	0.50	0.25	0.47	0.22	0.44
22	0.49	0.57	0.39	0.52	0.34	0.48	0.30	0.46	0.27	0.41	0.24	0.38
24	0.44	0.47	0.37	0.43	0.32	0.40	0.29	0.36	0.26	0.34	0.24	0.31

TAB. 2 – Gains réalisés dans le cas d’ajouts d’obstacles.

la fonction heuristique résultante est à la fois admissible (puisque les deux termes le sont) et au moins aussi bien informée que chacune des deux fonctions heuristiques.

Il est clair que plus la distance entre les cibles  $\bar{s}_1$  et  $\bar{s}_2$  est grande, plus le terme issu de la première exploration va être mécaniquement diminué et l’effet de l’apprentissage réduit.

### 3.2 Expériences

Afin de déterminer les conditions dans lesquelles l’algorithme décrit ci-dessus permet d’obtenir un gain en terme de taille de l’espace de recherche, nous avons réalisé une série d’expériences de recherche de chemins minimaux dans une grille en faisant varier la densité des obstacles et les positions des nœuds source et cible.

Le gain est calculé selon la même formule que dans la section 2.5 à ceci près que désormais les ensembles FERMÉ<sub>2</sub> et FERMÉ’<sub>2</sub> représentent les nœuds développés, pour la recherche d’un chemin entre la source  $s_2$  et la cible  $\bar{s}_2$ .

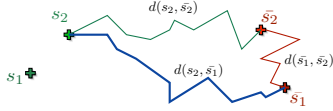


FIG. 4 – Critère déterminant pour le gain de l’algorithme d’apprentissage :  $\frac{d(s_2, \bar{s}_1) - d(s_2, \bar{s}_2) - d(\bar{s}_1, \bar{s}_2)}{d(s_2, \bar{s}_2)}$ .

Les expériences rapportées ici ont été réalisées sur une grille  $1000 \times 1000$ , avec un taux d’obstacles de 24% et des variations de source et de cible contraintes à rester dans un carré de côté 20 (ce qui n’implique pas que  $d(s_1, s_2) \leq 40$  ou  $d(\bar{s}_1, \bar{s}_2) \leq 40$ ).

Lorsque l’on trace le gain en fonction du critère normalisé  $(d(s_2, \bar{s}_1) - d(s_2, \bar{s}_2) - d(\bar{s}_1, \bar{s}_2)) / d(s_2, \bar{s}_2)$ , on obtient la surprenante figure 5. Obtenue pour les plus de 4000 expériences réalisées, celle-ci indique qu’en dehors de conditions très restrictives sur les distances entre  $s_2$ ,  $\bar{s}_1$  et  $\bar{s}_2$ , l’apprentissage n’apporte aucun gain. En revanche pour des environnements tels que :  $d(s_2, \bar{s}_1) \approx d(s_2, \bar{s}_2) + d(\bar{s}_2, \bar{s}_1)$ , le gain varie entre 0 (valeur minimale possible) et 1 (valeur maximale possible).

Cela signifie que pour que le transfert d’information puisse être profitable, il est nécessaire que la nouvelle cible  $\bar{s}_2$  soit sur un chemin optimal (ou quasi optimal) entre la nouvelle

source  $s_2$  et l’ancienne cible  $\bar{s}_1$ . Comment expliquer une telle condition ?

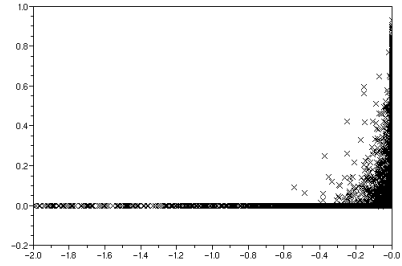


FIG. 5 – Gain obtenu par l’adaptation d’AdaptiveA\* en fonction de  $\frac{d(s_2, \bar{s}_1) - d(s_2, \bar{s}_2) - d(\bar{s}_1, \bar{s}_2)}{d(s_2, \bar{s}_2)}$ .

### 3.3 Analyse

Supposons que la nouvelle cible soit à une distance de  $\bar{\epsilon}$  de la cible  $\bar{s}_1$  :  $d(\bar{s}_1, \bar{s}_2) = \bar{\epsilon}$ . Parmi toutes les positions possibles de la nouvelle cible sur le « cercle » centré en  $\bar{s}_1$ , on cherche alors une position optimale, que nous noterons  $\bar{s}_2^*$ , tel que le gain en nombre de nœuds explorés soit maximal lorsque l’on utilise l’heuristique  $h_2$ .

La taille de l’espace de recherche en utilisant l’heuristique  $h_1$  est le cardinal de l’ensemble des états  $n$  tels que  $f(n) = g(s_2, n) + h_1(n, \bar{s}_2) < d(s_2, \bar{s}_2)$ , c’est-à-dire de l’ensemble des états développés en utilisant  $h_1$ .

De même, la taille de l’espace de recherche en utilisant  $h_2$  est le cardinal de l’ensemble des états  $n$  vérifiant :

$$f_2(n) = g(s_2, n) + \text{Max}[C_1^* - \bar{\epsilon} - g_1(s_1, n), h_1(n, \bar{s}_2)] < d(s_2, \bar{s}_2)$$

Ces deux ensembles ne peuvent différer que sur les états  $n$  pour lesquels les valeurs  $h_1(n, \bar{s}_2)$  et  $h_2(n, \bar{s}_2)$  sont différentes. Cela arrive quand  $g(s_1, n) \leq C_1^* - \bar{\epsilon} - h_1(n, \bar{s}_2)$ . L’ensemble des états développés avec  $h_1$ , mais pas avec  $h_2$ , noté ici  $D(h_1, -h_2)(\bar{s}_2)$ , inclut les états  $n$  tels que :

$$D(h_1, -h_2)(\bar{s}_2) \supseteq \begin{cases} g(s_2, n) + h_1(n, \bar{s}_2) < d(s_2, \bar{s}_2) \\ g(s_2, n) - g(s_1, n) + C_1^* - \bar{\epsilon} > d(s_2, \bar{s}_2) \\ g(s_1, n) < C_1^* - \bar{\epsilon} - h_1(n, \bar{s}_2) \end{cases}$$

c'est-à-dire les états développés en utilisant  $h_1$  (ligne 1), mais pas par  $h'_2$  (ligne 2) et pour lesquels les valeurs des deux heuristiques diffèrent (ligne 3).

Cet ensemble, et donc sa taille, dépendent de la position de la nouvelle cible  $\bar{s}_2$ . Afin de simplifier l'analyse, nous faisons l'hypothèse qu'au premier ordre, lorsque le but  $\bar{s}_2$  se déplace, il y a approximativement autant d'états  $n$  tels que  $h_1(n, \bar{s}_2)$  augmente, que d'états  $n$  pour lesquels  $h_1(n, \bar{s}_2)$  décroît. Dans ce cas,  $|D(h_1, -h'_2)(\bar{s}_2)|$  est extrémal quand la distance  $d(s_2, \bar{s}_2)$  ne varie pas en fonction de  $\bar{s}_2$ , c'est-à-dire quand  $\bar{s}_2$  est situé sur un « cercle » centré en  $s_2$ .

Mais nous avons aussi supposé que  $d(\bar{s}_1, \bar{s}_2) = \bar{\epsilon}$ . Par conséquent, le gain en taille de l'espace de recherche  $|D(h_1, -h'_2)(\bar{s}_2)|$  est extrémal pour des états buts  $\bar{s}_2$  rendant sa dérivée presque nulle, donc appartenant à un « cercle » centré en  $s_2$ , et, par hypothèse de départ, positionné sur un « cercle » centré en  $\bar{s}_1$ . Dans le cas de la distance euclidienne, seuls deux points satisfont strictement cette condition (quelle que soit la dimension de l'espace) (voir figure 6). L'un est tel que  $d(s_2, \bar{s}_1) = d(s_2, \bar{s}_2) + d(\bar{s}_2, \bar{s}_1)$ , et l'autre tel que  $d(s_2, \bar{s}_2) = d(s_2, \bar{s}_1) + d(\bar{s}_1, \bar{s}_2)$ .

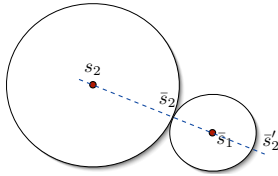


FIG. 6 – Positions rendant  $|D(h_1, -h'_2)(\bar{s}_2)|$  extrémal.

Dans le cas d'une distance définie sur une grille avec des obstacles, il peut y avoir plus que deux points satisfaisant cette contrainte. Néanmoins, c'est seulement lorsque  $d(s_2, \bar{s}_1) = d(s_2, \bar{s}_2) + d(\bar{s}_2, \bar{s}_1)$  que  $|D(h_1, -h'_2)(\bar{s}_2)|$  sera extrémal. C'est en effet seulement pour les états développés lors de la première exploration (entre  $s_1$  et  $\bar{s}_1$ ) que l'heuristique apprise est mieux informée que  $h_1$ . Si le nouveau but se situe en dehors de cette région ( $\bar{s}'_2$  sur la figure), le gain sera moindre.

Cette analyse est très qualitative. Elle prédit cependant que le gain en terme de taille d'espace de recherche par utilisation de  $h_2$  sera maximal quand le nouvel état but  $\bar{s}_2$  est dans une position vérifiant  $d(s_2, \bar{s}_1) = d(s_2, \bar{s}_2) + d(\bar{s}_2, \bar{s}_1)$ . Or les expériences réalisées s'accordent parfaitement avec cette prédiction.

## 4 Conclusion

Les travaux récents en intelligence artificielle et en apprentissage automatique reflètent l'intérêt croissant pour l'étude de tâches évoluant avec le temps et plus particulièrement pour l'adaptation incrémentale de fonctions heuristiques. Dans ce cadre, deux questions centrales se posent. D'abord, *comment exploiter au mieux les expériences passées pour aider à résoudre la tâche courante ou les tâches à attendre ?* Ensuite, *comment les connaissances acquises*

*sur une tâche se transfèrent à d'autres tâches ?* Les travaux existants sur l'apprentissage de fonctions heuristiques ne répondent que partiellement à ces questions, restreignant en particulier le type de transfert de tâche (même but).

Parmi ces travaux, les plus emblématiques sont ceux de Koenig et *al.* portant en particulier sur l'algorithme AdaptiveA\* [4, 5, 8]. Dans ce travail, nous avons montré, d'une part, qu'il est possible de mieux exploiter les expériences passées lorsque le but reste le même, et, d'autre part, que les conditions favorables de transfert d'une tâche à l'autre pour l'algorithme AdaptiveA\* sont très restrictives.

## Références

- [1] S. Aine, P. P. Chakrabarti, and R. Kumar. AWA\*, a window constrained anytime heuristic search algorithm. In *Int. Joint Conf. on Artificial Intelligence (IJCAI-07)*, pages 2250–2255, Hyderabad, India, 2007.
- [2] D. Bertsekas. *Dynamic Programming and Optimal Control (3rd Ed.)*, volume 1. Athena Scientific, 2005.
- [3] C. Hernandez and P. Meseguer. Improving lrtA\*(k). In *IJCAI-07*, Hyderabad, India, 2007.
- [4] S. Koenig and M. Likhachev. Adaptive A\*. In *AAMAS-05*, 2005.
- [5] S. Koenig and M. Likhachev. A new principle for incremental heuristic search : Theoretical results. In *ICAPS-06*, 2006.
- [6] S. Koenig and M. Likhachev. Real-time adaptive A\*. In *AAMAS-06*, 2006.
- [7] S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning A\*. *Artificial Intelligence journal*, 155 :93–146, 2004.
- [8] S. Koenig, M. Likhachev, and X. Sun. Speeding up moving-target search. In *AAMAS-07*, 2007.
- [9] R. Korf. Recent progress in the design and analysis of admissible heuristic functions. In *SARA 2000, Texas, USA.*, volume 1864/2000. Springer Verlag, 2000.
- [10] S. Minton. *Learning search control knowledge : An explanation-based approach*. Kluwer Academic Publishers, 1988.
- [11] Judea Pearl. *Heuristics. Intelligent search strategies for computer problem solving*. Addison Wesley, 1984.
- [12] Armand Prieditis. Machine discovery of effective admissible heuristics. *Machine Learning*, 12 :117–141, 1993.
- [13] A. Felner R. Korf. Disjoint pattern database heuristics. *Artificial Intelligence journal*, 134 :9–22, 2002.
- [14] A. Stentz. The focussed d\* algorithm for real-time replanning. In *IJCAI-95*, pages 1652–1659, 1995.
- [15] X. Sun and S. Koenig. The fringe-saving A\* search algorithm - a feasibility study. In *IJCAI-07*, Hyderabad, India, 2007.