Recherche heuristique basée sur le calcul de moyenne pour la planification temps réel

Damien Pellier, Bruno Bouzy, Marc Métivier

Laboratoire d'Informatique de Paris Descartes
Université Paris Descartes
45, rue des Saints Pères, 75006 Paris
{damien.pellier, bruno.bouzy, marc.metivier}@parisdescartes.fr

Résumé : Dans cet article, nous présentons un nouvel algorithme de recherche heuristique basé sur le calcul de moyennes pour la planification temps réel, appelé MHSP (Mean-Based Heuristic Search Planning). Il associe les principes d'UCT (Upper Confidence for Tree), un algorithme de type bandit ayant donné de très bons résultats dans le domaine des jeux, et plus particulièrement dans le cadre du jeu de Go, et une recherche heuristique en vue d'obtenir un planificateur temps réel dans le contexte de la planification d'actions. MHSP est évalué sur différents problèmes de planification et comparé aux algorithmes de recherche en ligne et d'apprentissage existants. Nos résultats mettent en évidence la capacité de MHSP à retourner en temps réel des plans qui tendent vers un plan optimal au cours du temps. Ils montrent de plus que MHSP est plus rapide et retourne des plans de meilleure qualité que les algorithmes existants dans la littérature.

1 Introduction

Le point de départ de ce travail est d'appliquer UCT (Kocsis & Szepesvari, 2006), un algorithme efficace bien connu par les communautés de l'apprentissage automatique et des jeux, sur des problèmes de planification d'actions. UCT, conçu pour les PDM (Processus Décisionnels de Markov), est basé sur la prise de décision de type bandit (Auer *et al.*, 2002). Dans le contexte de cet article, l'aspect intéressant d'UCT est sa propriété *anytime* au sens strict. À tout moment, UCT est en mesure de retourner la première action d'un plan, un plan partiel, un plan de solution, ou un plan optimal en fonction du temps qui lui est donné. UCT a obtenu de très bons résultats dans le domaine des jeux, et plus particulièrement au jeu de Go avec le programme Mogo (Gelly *et al.*, 2006). En effet, dans le cadre de ce jeu, UCT est très efficace car sa complexité est élevée et parce que ses parties sont jouées en temps réel, ce qui répond à la propriété *anytime* d'UCT. Quelques travaux ont regroupé planification d'actions et méthodes de type Monte Carlo, tels que UCT. Ainsi, en 2006, Chaslot et al. (Chaslot *et al.*, 2006) ont utilisé un algorithme proche de UCT pour résoudre des problèmes de gestion de production. En 2009, Bjarnason et al. (Bjarnason *et al.*, 2009) ont présenté une adaptation efficace d'UCT pour répondre au problème de planification probabiliste représenté par le jeu du Solitaire Klondike. En planification d'actions déterministe, Nakhost et Müller (Nakhost & Müller, 2009) ont présenté le planificateur ARVAND utilisant des simulations aléatoires Monte-Carlo. Il ne s'agit cependant pas d'une adaptation d'UCT, les valeurs des actions n'étant pas basées sur un calcul de moyennes.

L'état de l'art de la planification est vaste (Roberts & Howe, 2009) et nous le divisons en deux catégories : la planification hors ligne et la planification en ligne. Dans la planification hors ligne, les planificateurs développent un plan solution, puis exécutent les actions du plan. Un élément important est l'existence de très bonnes fonctions heuristiques qui conduisent la recherche vers le but de manière efficace. Les planificateurs de pointe actuels, tels que FF (Hoffmann & Nebel, 2001) ou LAMA (Richter & Westphal, 2008), peuvent trouver un plan solution très rapidement. Toutefois, bien qu'ils trouvent des plans solutions très rapidement sur des problèmes suffisamment simples, ces planificateurs ne sont pas des planificateurs temps réel, et ils échouent s'ils ne disposent pas d'assez de temps pour trouver un plan solution.

À l'inverse, dans le contexte de la planification en ligne, les planificateurs prennent les decisions en temps limité, et exécutent ensuite l'action correspondante, ou la séquence d'actions, dans l'environnement. La littérature distingue deux approches, l'une basée sur les PDM appliquée aux problèmes non-déterministes,

par exemple, (Barto *et al.*, 1995; Hansen & Zilberstein, 2001), et l'autre basée sur la recherche temps réel (RTR), par exemple, (Korf, 1990). Si la première approche a été récemment appliquée à la planification (Fabiani *et al.*, 2007), la seconde approche RTR est restée fortement liée, depuis les travaux précurseurs de Korf sur les puzzles, au développement de jeux vidéo dans lequel ils est nécessaire de mettre en œuvre des algorithmes de recherche de chemin s'exécutant en temps réel. Cette dernière approche n'a pas été élargie aux problèmes de planification d'actions. Classiquement, il existe plusieurs méthodes de recherches temps réel, par exemple, RTA* (Korf, 1990), γ-Trap (Bulitko, 2004), LRTS (Bulitko & Lee, 2006) ou même A* (Hart *et al.*, 1968). Ces algorithmes effectuent leur sélection d'actions en utilisant une recherche heuristique. Puisque le temps de sélection d'action est limité, ces algorithmes explorent une petite partie de l'espace d'états autour de l'état actuel. Par conséqent, les plans exécutés ne sont pas optimaux. Toutefois, la plupart des algorithmes de type RTR convergent vers des plans solutions optimaux lorsque la tâche est répétée de façon itérative. Cela ouvre ainsi la littérature RTR à l'apprentissage automatique. Considérant alors l'aspect apprentissage comme son objectif principal, et focalisant sur les preuves de convergence, la littérature a réduit la phase de sélection d'action à une recherche gloutonne de profondeur un.

L'objectif de cet article est de se concentrer sur l'étape de sélection d'action de la RTR, afin de présenter MHSP, un algorithme de recherche heuristique basé sur le calcul de moyennes, i.e. une adaptation d'UCT à la recherche temps réel dans le contexte de la planification d'actions. Nous montrons que MHSP obtient de meilleurs résultats dans la sélection d'action temps réel que les sélectionneurs d'actions temps réel actuels, avec ou sans apprentissage.

Le plan de cet article est le suivant. Dans une première partie, nous décrivons le domaine de la recherche temps réel (RTR). Puis, nous présentons l'algorithme UCT ainsi que notre nouvel algorithme MHSP, une adaptation d'UCT à la RTR. La partie suivante décrit quelques expériences exécutées afin de prouver la pertinence de notre approche. Enfin, nous discutons nos résultats et proposons des directions de recherches de futures.

2 La recherche temps réel

Alors que les méthodes classiques de recherche hors ligne trouvent d'abord un plan solution, puis l'exécute, la recherche temps réel entrelace planification et exécution. À chaque étape, une recherche locale est effectuée à partir et autour de l'état courant et la meilleure action trouvée est alors choisie pour être exécutée tant que le but n'est pas atteint. Les états rencontrés pendant la phase de sélection de l'action sont appelés les états explorés. La caractéristique de la recherche temps réel est d'effectuer cette recherche en temps limitée. Lorsque le but est atteint, l'algorithme essaie de résoudre le problème une nouvelle fois, on parle alors d'épisode. La RTR peut être considérée avec ou sans apprentissage. Sans apprentissage, l'efficacité de la recherche dépend de sa capacité à sélectionner une action pertinente pour le but. Avec l'apprentissage, les valeurs heuristiques des états rencontrés sont mises à jour et si certaines conditions sont satisfaites, la recherche converge au fur et à mesure des épisodes vers le plan solution optimal. Dans la suite de l'article, nous qualifieront les états résultant de l'exécution d'une action comme les états visités.

L'article fondateur de la RTR est Real-Time Heuristic Search (Korf, 1990). Il présente Real-Time A* (RTA*), un algorithme utilisant une fonction heuristique qui calcule un arbre comme le fait A*, mais en un temps limité, et sa version avec apprentissage appelée LRTA*. L'article prouve que si LRTA* est exécuté répétitivement sur un même problème, il converge vers le plan optimal. SLA* (Search and Learn A*) (Shue & Zamani, 1993) est présenté comme une amélioration de LRTA* qui inclut un mécanisme de backtracking. FALCONS (Furcy & Koenig, 2000) est un algorithme d'apprentissage qui converge rapidement sous certaines hypothèses. Dans (Shimbo & Ishida, 2003), Shimbo montre comment weighted-A* et la recherche basée sur la borne supérieure méritent d'être examinés dans la recherche temps réel. Comme weighted-A*, γ-Trap (Bulitko, 2004) pondère la fonction heuristique. Il intègre de plus de la recherche en avant et exécute une séquence d'actions au lieu de la première action du plan sélectionné. Enfin, LRTS (Bulitko & Lee, 2006) est un cadre unificateur pour l'apprentissage dans la recherche temps réel, incluant LRTA*, SLA* et γ-Trap.

La principale limitation des algorithmes de type RTR est de se focaliser sur la partie apprentissage et moins sur la partie recherche de la sélection d'action. (Korf, 1990) et les travaux suivants démontrent la convergence de leur algorithme d'apprentissage temps réel à condition que la sélection d'action soit faite avec une recherche gloutonne de profondeur un. Toutefois, les travaux de Korf montrent l'importance d'une recherche efficace pour la sélection de l'action. L'exception à cette limitation est γ-Trap et LRTS dans

lesquels une recherche en avant est utilisée pour la sélection de l'action. La recherche en avant de γ-Trap est une sorte de recherche en largeur d'abord. Par conséquent, il est intéressant de voir si une adaptation d'UCT peut être utilisée efficacement pour la sélection de l'action dans le cadre des algorithmes de type RTR. Cette adaptation sera ensuite utilisée avec ou sans apprentissage.

3 UCT

UCT a obtenu de bons résultats dans les programmes jouant au jeu de Go et a été utilisé sous de nombreuses versions, définissant le cadre de la recherche Monte-Carlo dans les arbres (MTCS : Monte-Carlo Tree Search) (Chaslot *et al.*, 2008). Tant qu'il reste du temps, un algorithme MCTS développe itérativement un arbre dans la mémoire de l'ordinateur en suivant les étapes suivantes :

1. A partir de la racine, parcourir l'arbre jusqu'à atteindre une feuille en utilisant la règle de sélection UCB (voir plus loin) ;
2. Choisir une action et développer le nœud fils de la feuille ;
3. Exécuter une simulation aléatoire à partir de ce nœud fils jusqu'à la fin du jeu et obtenir le retour, i.e., le résultat du jeu ;
4. Mettre à jour la valeur moyenne des nœuds parcourus avec le retour obtenu.

Avec un temps infini, la valeur du nœud racine converge vers la valeur mini-max de l'arbre de jeu (Kocsis & Szepesvari, 2006). La règle de sélection UCB (Upper Confidence Bound) (1) répond à la nécesité d'être optimiste lorsqu'une décision doit être prise dans l'uncertain (Auer *et al.*, 2002) :

$$N_{select} = \arg\max_{n \in N} \{m + C\sqrt{\frac{\log p}{s}}\} \quad (1)$$

où N_{select} est le nœud sélectionné, N est l'ensemble des nœuds fils, m est la valeur moyenne du nœud n, s est le nombre d'itérations passées par le nœud n, p est le nombre d'itérations passées par le nœud parent de n, et C est une valeur constante définie expérimentalement. L'équation (1) est la somme de deux termes : la valeur moyenne m et la valeur de biais UCB qui garantit l'exploration. Le respect de la règle de sélection UCB garantit l'exhaustivité et l'exactitude de l'algorithme.

4 MHSP

MHSP (Mean-based Heuristic Search for real-time Planning) est un nouvel algorithme basé sur UCT Kocsis & Szepesvari (2006). Comme cadre formel, nous considérons un espace de recherche défini par le tuple (S, O, s_0, g, h) où S est un ensemble fini d'états, O est un ensemble fini d'opérateur STRIPS de la planification classique, s_0 est l'état initial, g est le but à atteindre, c'est à dire un ensemble de propositions, et h est l'heuristique initiale. Nous avons fait deux choix importants dans la conception de MHSP. Ces choix sont décrits dans les prochaines sections suivies du pseudo-code de MHSP.

4.1 Remplacement des simulations par l'appel à une fonction heuritisque

Dans le cadre de la planification, les simulations aléatoires ne sont pas appropriées. En effet, parcourir de manière aléatoire l'espace d'états ne permet pas à l'algorithme d'atteindre des états buts assez souvent pour collecter des informations pertinentes sur l'espace de recherche. Beaucoup d'exécutions se terminent sans avoir atteint un état but. Par conséquent, remplacer les simulations par un appel à l'heuristique devient nécessaire. Dans le jeu de Go, les simulations aléatoires étaient appropriées principalement parce qu'elles se terminent toujours après un nombre limité de coups, et les valeurs de retour (gagné ou perdu) étaient à peu près également réparties sur la plupart des positions du jeu. En outre, les deux valeurs de retour correspondent aux valeurs actuelles d'un jeu terminé. En planification, un retour signifie qu'une solution a été trouvée (épisode terminé), et l'autre non. Cette différence est fondamentale entre les problèmes de planification et les problèmes de jeux à deux joueurs.

En planification, les fonctions heuristiques sont très informatives et peuvent apporter des connaissances presque équivalentes aux simulations. Dans le jeu de Go, remplacer les simulations par des appels à des fonctions d'évaluation est interdit par cinquante années d'histoire du jeu de Go sur ordinateur qui a connue

le chemin inverse : les fonctions d'évaluation complexes ont été remplacées par des simulations pseudo-aléatoires. Ainsi, dans MHSP, nous remplaçons l'étape (3) de MTCS ci-dessus par un appel à une fonction heuristique h.

4.2 Valeurs moyennes initiales optimistes

La pratique des programmes de jeux montre que le biais UCB de l'équation (1) peut être supprimé, à condition que les valeurs moyennes des nœuds soient initialisées avec des valeurs suffisamment optimistes. Cette simplification supprime le problème de la détermination de la valeur du paramètre C. En règle générale, pour estimer un nœud donné, les heuristiques de la planification donnent une estimation de la longueur du chemin. La garantie de la convergence vers le meilleur plan est fournie par les heuristiques admissibles, i.e., des heuristiques optimistes. Ainsi, la valeur retournée par les heuristiques de planification sur un nœud peut être utilisée pour initialiser la valeur moyenne initiale de ce nœud.

Dans MHSP, les retours sont négatifs ou nuls et doivent être l'opposé de la distance de s à g. Par conséquent, nous initialisons la valeur moyenne d'un nœud avec $h(s)$, qui est l'opposé de l'estimation de la distance pour atteindre g à partir de s. Avec cette méthode d'initialisation, le meilleur nœud selon l'heuristique sera exploré en premier. Sa valeur sera réduite après quelques itérations quelle que soit son efficacité, et les autres nœuds seront alors examinés dans l'ordre donné par l'heuristique.

Dans cet article, nous avons utilisé l'heuristique de FF (Hoffmann & Nebel, 2001) comme heuristique initiale dans toutes les expériences. Notez que l'heuristique de FF n'est pas admissible. Toutefois, en pratique, elle est informative et peut être calculée très rapidement. Une étude de différentes heuristiques dans MHSP se trouve dans (Pellier *et al.*, 2010).

4.3 L'algorithme

L'algorithme 1 décrit la procédure MHSP de recherche dans un arbre, et utilise les notations suivantes : O est l'ensemble des opérateurs, s_0 l'état initial, g l'état but, h la fonction heuristique, $C[s]$ l'ensemble des fils de l'état s, $R[s]$ le retour cumulé de l'état s, $V[s]$ le nombre de visites de l'état s et $P[s]$ le parent de s.

Le $while$ extérieur (ligne 2) assure la propriété temps réel. Le premier $while$ intérieur (ligne 4) correspond à l'étape (1) du cadre MCTS. La récompense est pessimiste par défaut : $(R[s_0]/V[s_0]) + 1$ est le seuil de pessimisme actuel. Les deux premiers if testent si le $while$ intérieur s'est terminé avec un objectif atteint (ligne 7) ou avec une feuille (ligne 9). Si l'objectif n'est pas atteint, la feuille est étendue, étape (2) du cadre MCTS. Le second if correspond aussi à l'étape (2). L'étape (3) est appliquée par l'emploi de $h(s')$ comme retour. Le second $while$ intérieur (ligne 22) correspond à l'étape (4).

La fonction *reconstruct_solution_plan*() parcourt l'arbre en sélectionnant le nœud fils ayant la moyenne la plus élevée, ce qui produit le plan de solution. La fonction *reconstruct_best_plan*() parcourt l'arbre en sélectionnant le nœud enfant ayant le plus grand nombre de visites, produisant un plan considéré comme le meilleur. La reconstruction du meilleur plan est effectuée quand le temps de décision se termine avant qu'un plan solution n'ait été trouvé. Dans ce cas, il est important de reconstituer un plan robuste, qui ne sera peut-être pas le meilleur en termes de valeurs moyennes. En se basant sur les enfants ayant la plus haute moyenne, un plan avec des nœuds nouvellement créés pourraient être choisis et le plan ne serait pas solide. A l'inverse, la sélection des enfants ayant le plus grand nombre de visites garantit que le plan a été essayé à plusieurs reprises et devrait être par conséquent robuste.

Enfin, nous devons discuter de la façon dont MHSP gère les puits de l'espace de recherche. Normalement, la valeur heuristique calculée par le planificateur FF pour les états puit est infinie. La valeur infinie doit alors servir à mettre à jour la valeur moyenne de tous les nœuds parcourus. Dans ce cas, cela pourrait être critique pour les étapes 3 et 4 de MHSP. Pour résoudre ce problème, MSHP rétro-propage une valeur calculée à partir de la valeur du nœud racine et un cœfficient ajustable afin de sanctionner le nœud impasse mais pas tous les nœuds parcourus.

5 Expériences

Le but de ces expériences est de comparer MHSP avec les algorithmes corrects et complets de sélection d'actions existants dans le contexte de la RTR, utilisés avec ou sans apprentissage, pour différents temps de décision et sur un ensemble de problèmes de planification.

Algorithm 1: MHSP(O, s_0, g, h)

```
1  C[s_0] ← ∅ ; R[s_0] ← h(s_0) ; V[s_0] ← 1 ; π ← nil
2  while has_time do
3      s ← s_0
4      while g ⊈ s and V[s] ≠ 1 do
5          s ← argmax_{s'∈C[s]}(R[s']/V[s'])
6      reward ← (R[s_0]/V[s_0]) + 1
7      if g ⊆ s then
8          reward ← 0
9      else if V[s] = 1 then
10         A ← {a | a instance d'un operateur dans O et precond(a) ⊆ s}
11         foreach a ∈ A do
12             s' ← (s ∪ effects⁺(a)) − effects⁻(a)
13             C[s] ← C[s] ∪ {s'}
14             R[s'] ← h(s')
15             P[s'] ← s
16             V[s'] ← 1
17         if C[s] ≠ ∅ then
18             s ← argmax_{s'∈C[s]}(R[s'])
19             reward ← R[s]
20     l ← s
21     i ← 0
22     while s ≠ s_0 do
23         s ← P[s]
24         R[s] ← R[s] + (reward − i)
25         V[s] ← V[s] + 1
26         i ← i + 1
27     if g ⊆ l then
28         π' ← reconstruct_solution_plan()
29         if length(π) > length(π') then π ← π'
30 if π = nil then return reconstruct_best_plan()
31 else return π
```

5.1 Planificateurs et domaines

MHSP est comparé à deux algorithmes : A* et la recherche en largeur d'abord (BFS : Best First Search). Nous avons choisi A* car celui-ci est l'algorithme de référence en planification (LRTA*). Il s'agit d'un algorithme *meilleur d'abord* qui vise à minimiser la fonction heuristique classique f de A*. A* étend les nœuds de la liste ouverte avec des valeurs f diminuant avec le temps d'exécution, la valeur de f atteignant zéro avec un temps suffisant. De ce fait, A* peut être interrompu à tout moment. Lorsque A* est arrêté, le chemin du dernier nœud élargi au nœud racine donne la "meilleure" action choisie par A*. En parallèle, nous avons choisi BFS. BFS est une simple généralisation des sélecteurs d'action actuels, comme LRTS ou γ-Trap utilisés dans la RTR pour la recherche de chemin, à d'autres domaines de la planification. Nous n'avons pas choisi d'algorithmes en profondeur d'abord, tels que Mini-min, car ils s'adaptent difficilement à la contrainte de temps réel. Finalement, nos trois algorithmes sont MHSP, A*, et BFS.

Comme mentionné dans l'introduction, les algorithmes de la RTR sont principalement appliqués à la recherche de chemin pour les jeux vidéo. Afin d'étendre le domaine de test existant, nous avons choisi d'autres domaines et problèmes à partir de la Compétition Internationale de Planification qui illustrent l'efficacité des techniques mises en oeuvre dans MHSP. Les domaines sont ferry, gripper et satellite. Pour chaque domaine, nous avons sélectionné 20 problèmes classés par ordre de complexité en termes de nombre de faits et nombre d'opérateurs. Dans le reste de l'article, nous présentons des résultats issus de cet ensemble de problèmes afin d'illustrer la puissance de MHSP.

5.2 Présentation des tests

Trois tests ont été mis en œuvre afin de souligner l'efficacité de MHSP. Le test 1 est global et ne focalise pas particulièrement sur la sélection de l'action : il donne la longueur moyenne des plans solutions trouvés par les trois algorithmes pour différents temps de décision et problèmes représentatifs. Il entend montrer que MHSP est globalement meilleur que A* et BFS en termes de longueur de plans solutions. Le test 1 ne comprend pas d'apprentissage.

Le test 2 ré-exécute le test 1 avec apprentissage en appliquant la règle de mise à jour (2) sur les nœuds s visités au cours des épisodes. La règle (2) n'est pas appliquée sur les nœuds explorés pendant le temps de sélection de l'action. Le test 2 entend montrer l'efficacité des trois sélecteurs d'action sur la convergence des plans solutions vers des plans optimaux lorsque le nombre d'épisodes augmente.

$$H(s) = max\{H(s), min_{s' \in C[s]}\{1 + H(s')\}\} \quad (2)$$

Le test 3 est le test le plus important pour souligner la capacité des algorithmes à effectuer en temps réel la sélection d'une action, ou d'un plan partiel, efficace. Ce test fait varier le temps de décision et évalue la qualité des plans partiels obtenus. La qualité des plans partiels est estimée grâce à deux distances : la distance à l'objectif et la distance à l'optimum.

La distance à l'objectif d'un plan partiel est la longueur du plan optimal reliant l'état final de ce plan partiel à l'état but. Lorsque la distance à l'objectif diminue, le plan partiel a été construit dans la direction appropriée. Lorsque la distance de l'objectif est nulle, le plan partiel est un plan solution.

La distance à l'optimum d'un plan partiel est la longueur du plan partiel, plus la distance à l'objectif du plan partiel, moins la longueur du plan optimal. Quand la distance à l'optimum d'un plan partiel est nulle, le plan partiel est le début d'un plan optimal. La distance à l'optimum d'un plan solution est la différence entre sa longueur et la longueur optimale. La distance à l'optimum du plan vide est zéro. La distance à l'objectif et la distance à l'optimum d'un plan optimal est nulle. Réciproquement, si la distance à l'objectif et la distance à l'optimum d'un plan partiel sont nuls, alors ce plan partiel est un plan optimal. Pour chaque problème, les résultats sont présentés avec des graphiques montrant la distance à l'objectif et la distance à l'optimum du plan partiel en fonction du temps d'exécution. Ces distances sont calculées en appelant un planificateur optimal (A*).

Enfin, tous les tests ont été effectués sur un ordinateur équipé d'un processeur Intel Core 2 Quad 6600 (2,4 GHz) avec 4 Go de RAM. L'implémentation de MHSP utilisées pour ces expériences est écrite en Java et basée sur la bibliothèque PDDL4J [1].

5.3 Test 1

Le tableau 1 a les entrées suivantes : le nom du domaine (ferry, gripper ou satellite), le numéro de problème, l'algorithme utilisé pour la sélection de l'action, le temps de décision. Les sorties sont : la longueur du plan optimal, le temps moyen des épisodes, la longueur moyenne, la longueur maximale et la longueur minimale des plans solutions trouvés par l'algorithme, et le pourcentage d'échecs.

La longueur du plan optimal, calculée hors ligne, est utilisée comme une référence. Sur ferry-05 avec un temps de décision de 40ms, MHSP exécute les plans solutions qui sont presque optimaux (18,02 contre 18) tandis que A* et BFS sont loin d'être optimaux (26,26 et 27,76). La longueur maximale des plans est très élevée pour A* et BFS (277 et 567) et presque optimale pour MHSP (19). La longueur minimale des plans est optimale pour BFS et MHSP. Afin de limiter le temps des expériences, elles sont limitées à 50 épisodes. Un algorithme qui n'atteint pas l'objectif au cours d'un épisode a échoué. Ici, BFS a 16% de taux d'échecs sur les 50 épisodes. Par ailleurs, l'algorithme MHSP est le plus rapide. Au début d'un épisode, tous les algorithmes utilisent la totalité du temps décision. Toutefois, lorsque l'épisode touche à sa fin, la sélection de l'action est plus facile qu'auparavant, et certains algorithmes n'utilisent pas tout le temps disponible, et sont plus rapides que les autres. MHSP est clairement le plus rapide parce qu'il trouve l'objectif plus facilement que A* ou BFS lorsque l'objectif n'est pas loin.

Sur ferry-10 avec un temps de décision de 200ms, MHSP exécute des plans solutions qui sont presque optimaux (35,66 contre 35), tandis que A* et BFS sont de nouveau loin d'être optimaux (181.94 et 41.34). La longueur maximale des plans est très élevée pour A* et BFS (807 et 109) et presque optimale pour

[1] `http://sourceforge.net/projects/pdd4j/`

problème	algo.	décision	temps moy.	long. moy.	long. opt.	long. max.	long. min.	% échecs
ferry-05	A*	40	1.09	26.26	18	277	19	0
ferry-05	BFS	40	8.91	27.76	18	567	18	16
ferry-05	MHSP	40	0.59	18.02	18	19	18	0
ferry-10	A*	200	97.9	184.94	35	807	42	32
ferry-10	BFS	200	9.05	41.35	35	109	35	0
ferry-10	MHSP	200	6.26	35.46	35	36	35	0
ferry-15	A*	2000	157.85	31.95	51	88	58	55
ferry-15	BFS	2000	103.75	51.45	51	52	51	0
ferry-15	MHSP	2000	86.45	52.45	51	53	51	0
ferry-20	A*	4000	–	–	73	–	–	100
ferry-20	BFS	4000	–	–	73	–	–	100
ferry-20	MHSP	4000	260.49	74.87	73	78	73	0
gripper-05	A*	50	0.56	15.04	15	17	15	0
gripper-05	BFS	50	0.71	15	15	15	15	0
gripper-05	MHSP	50	0.49	15	15	15	15	0
gripper-10	A*	165	92.28	140.04	29	651	33	36
gripper-10	BFS	165	7.86	37	29	37	37	0
gripper-10	MHSP	165	5.08	29	29	29	29	0
gripper-15	A*	450	160.1	47.54	45	229	77	70
gripper-15	BFS	450	31.42	46.52	45	47	45	0
gripper-15	MHSP	450	38.53	54.88	45	55	53	0
gripper-20	A*	1100	–	–	59	–	–	100
gripper-20	BFS	1100	102.76	61	59	61	61	0
gripper-20	MHSP	1100	134.86	73.92	59	75	73	0
satellite-05	A*	300	3.49	15.08	15	18	15	0
satellite-05	BFS	300	20.28	63.72	15	522	19	0
satellite-05	MHSP	300	3.01	15	15	15	15	0
satellite-10	A*	2000	–	–	29	–	–	100
satellite-10	BFS	2000	–	–	29	–	–	100
satellite-10	MHSP	2000	67.912	31.0	29	31	31	0

TAB. 1 – Résultats du test 1 sur les domaines ferry, gripper et satellite, sans apprentissage

MHSP (36). La longueur minimale des plans est optimale pour BFS et MHSP. Ici, BFS a 32% de taux d'échecs.

Sur gripper-05 avec un temps de décision de 50ms, MHSP, A* et BFS sont presque optimaux. Sur gripper-10 avec un temps de décision de 165ms, MHSP exécute des plans optimaux (29) tandis que A* et BFS sont encore loin d'être optimaux (140,04 et 37). Ici, A* a 36% de taux d'échecs.

Enfin, sur satellite-05 avec un temps de décision de 300ms, MHSP, A* et BFS sont presque optimaux avec une légère préférence pour MHSP. Mais sur satellite-10 avec un temps de décision de 2000ms, BFS et A* ne trouvent aucune solution à la différence de MHSP. La principale raison de ce résultat est le facteur de branchement des problèmes qui est plus élevée que sur les autres problèmes étudiés (par exemple, 66 pour satellite-10, 13 pour le gripper-20 et 16 pour ferry-20). Ainsi, ce facteur de branchement élevé pénalise fortement les stratégies de recherche exhaustives de A* et BFS.

Pour résumer le premier test, MHSP trouve des plans plus courts que A* ou BFS, et MHSP est plus rapide que A* et BFS.

5.4 Test 2

Le tableau 2 montre les performances des trois algorithmes lorsque l'apprentissage est appliqué. Par rapport aux résultats de la table 1, nous pouvons observer que l'apprentissage améliore le plus souvent la qualité des meilleurs plans. En effet, à l'exception de BFS dans gripper-20, la durée minimale des plans est toujours plus petite avec apprentissage que sans, si elle n'est pas déjà optimale dans le test 1.

problème	algo.	decision	temps moy.	long. moy.	long. opt.	long. max.	long. min.	% échecs
ferry-05	A*	40	0.68	19.22	18	31	18	0
ferry-05	BFS	40	12.03	133.18	18	532	18	14
ferry-05	MHSP	40	0.48	18.02	18	19	18	0
ferry-10	A*	200	9.36	48.12	35	67	37	0
ferry-10	BFS	200	17.22	78.5	35	892	35	0
ferry-10	MHSP	200	6.24	35	35	35	35	0
ferry-15	A*	2000	112.03	62.36	51	81	57	55
ferry-15	BFS	2000	104.31	51.75	51	53	51	0
ferry-15	MHSP	2000	87.39	52.8	51	53	51	0
ferry-20	A*	4000	247.32	121.1	73	146	107	0
ferry-20	BFS	4000	284.44	73.8	73	75	75	35
ferry-20	MHSP	4000	255.31	73.6	73	73	73	15
gripper-05	A*	50	0.45	15	15	15	15	0
gripper-05	BFS	50	35.72	34.06	15	615	15	68
gripper-05	MHSP	50	0.48	15	15	15	15	0
gripper-10	A*	165	76.48	35.04	29	43	29	0
gripper-10	BFS	165	9.96	32.22	29	37	29	0
gripper-10	MHSP	165	4.8	29	29	29	29	0
gripper-15	A*	450	56.32	54.32	45	57	49	14
gripper-15	BFS	450	36.92	54.8	45	318	45	4
gripper-15	MHSP	450	36.52	46.76	45	55	45	0
gripper-20	A*	1100	–	–	59	–	–	100
gripper-20	BFS	1100	132.44	74.12	59	75	73	2
gripper-20	MHSP	1100	99.47	59.06	59	63	73	0
satellite-05	A*	300	3.54	15.62	15	18	15	0
satellite-05	BFS	300	6.1	18.98	15	41	19	0
satellite-05	MHSP	300	3.17	15	15	15	15	0
satellite-10	A*	2000	–	–	29	–	–	100
satellite-10	BFS	2000	–	–	29	–	–	100
satellite-10	MHSP	2000	65.612	31.2	29	32	30	0

TAB. 2 – Résultats du test 1 sur les domaines ferry, gripper et satellite, avec apprentissage

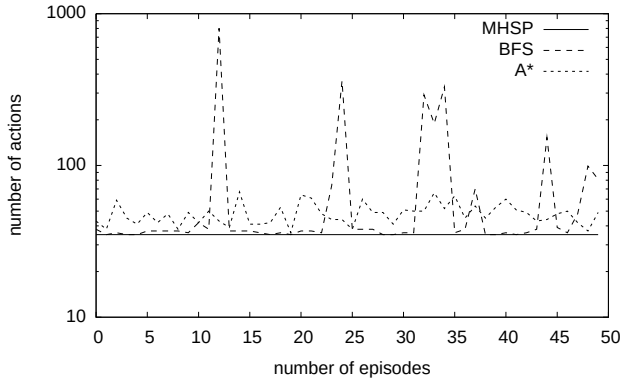
De plus, l'apprentissage a permis à BFS et A* de trouver des plans solutions dans ferry-15. Aucun d'entre eux n'est parvenu à obtenir un plan optimal, mais le meilleur plan de BFS est à deux actions de l'optimal. Il a également permis à MHSP de devenir optimal dans satellite-05.

Afin d'illustrer les comportements des algorithmes au cours du temps, la figure 1 montre l'évolution de la longueur des plans selon les épisodes, dans trois problèmes : ferry-10, gripper-10 et satellites-5. Comme on peut le voir, sur ces problèmes, MHSP atteint un plan optimal très rapidement et la longueur de plan est presque constante et stable. En revanche, A* et BFS ne sont pas optimaux dans ferry-10 et gripper-10, et sont très instables. Les pics de BFS correspondent à la longueur maximale de plan autorisée.

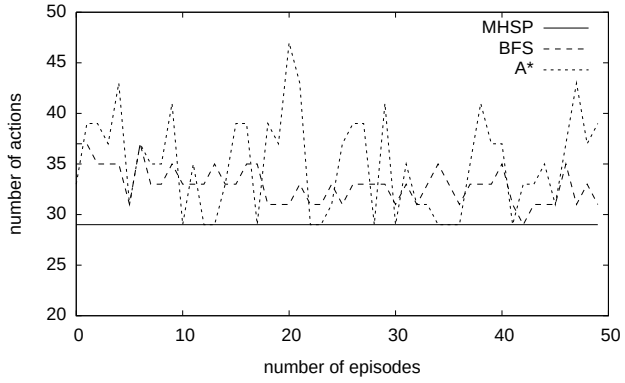
Toutefois, sur ces graphiques, l'apprentissage n'est pas observée par une claire décroissance de la longueur des plans comme attendu. Il y a deux raisons. Tout d'abord, la règle de mise à jour est appliquée dans les nœuds visités, et non dans les nœuds explorés. Par conséquent, la stratégie de sélection de l'action n'a pas vraiment d'impact sur la longueur des plans quand le nombre d'épisodes augmente. Deuxièmement, les algorithmes utilisent des valeurs heuristiques calculée hors ligne par des techniques de graphes de planification, qui sont presque optimales sur les problèmes d'une complexité assez faible. Par conséquent, la règle de mise à jour n'est pas efficace très souvent.

5.5 Test 3

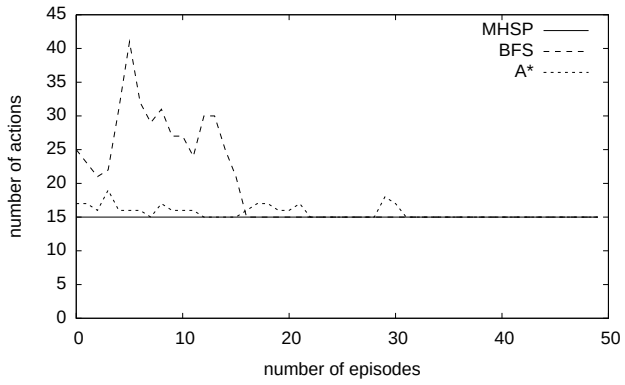
Le test 3 évalue la qualité des plans partiels disponibles à la fin de la phase de sélection d'action en fonction du temps donné à la décision. Les figures 2b, c et d montrent la distance à l'objectif et la distance



(a) ferry-10



(b) gripper-10



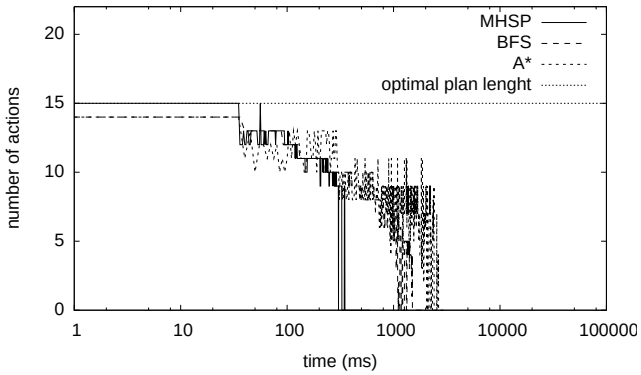
(c) satellite-05

FIG. 1 – Test 2 – Convergence des algorithmes de planificaion temps réel avec apprentissage

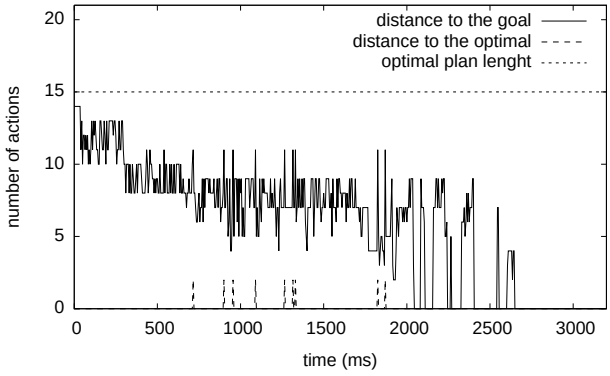
à l'optimum pour chaque algorithme en fonction du temps de décision sur le problème gripper-05. La figure 2a donne un aperçu de la distance à l'objectif des trois algorithmes en fonction du temps de décision sur une échelle logarithmique.

Dans ces résultats, nous observons que A* a besoin d'un temps de décision d'au moins 2650ms pour trouver systématiquement des plans optimaux, tandis que BFS a besoin d'au moins 1504ms et MHSP de seulement 349ms. Quel que soit le temps décision, A* fournit des plans partiels proches de l'optimal. BFS ne fournit que des plans partiels des plans optimaux. Enfin, lors de sa recherche, MHSP fournit des plans partiels ayant une distance à l'optimum élevée. Cela peut s'expliquer par la fin des plans partiels qui sont instables. Sélectionner des plans partiels plus courts aboutirait à une meilleure distance à l'optimum.

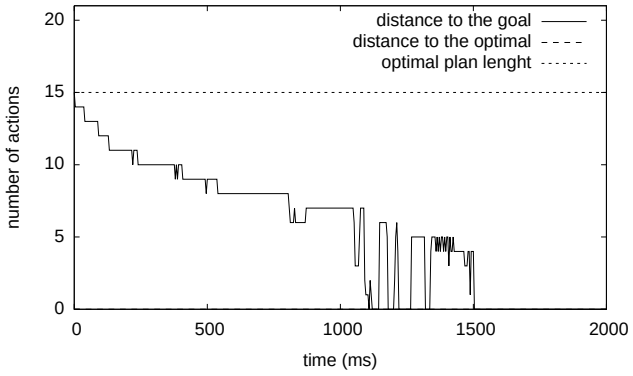
Le tableau 3 résume ces résultats et ajoute les résultats sur ferry-05 et satellite-05. Comme dans gripper, on peut voir que MHSP est le plus rapide des algorithmes pour trouver des plans optimaux.



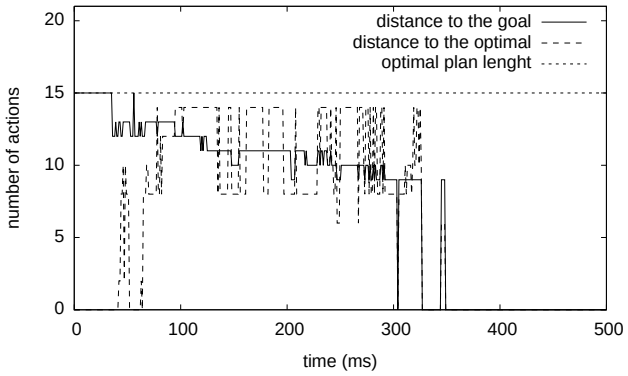
(a) gripper-05 – overview



(b) gripper-05 – A*



(c) gripper-05 – BFS



(d) gripper-05 – MHSP

FIG. 2 – Test 3 – Qualité de la sélection d'action

algorithme	ferry-05	gripper-05	satellite-05
A*	808	2650	8180
BFS	-	1504	-
MHSP	288	349	1710

TAB. 3 – Temps (ms) pour trouver des plans optimaux dans les domaines ferry, gripper et satellite

6 Discussion et travaux futurs

Dans cet article, nous avons présenté et étudié un nouvel algorithme de recherche heuristique basé sur le calcul de valeurs moyennes pour la recherche temps réel (RTR), appelé MHSP, adapté dans le contexte de la planification classique. Cet algorithme combine une recherche heuristique et les principes de l'algorithme UCT, i.e., les valeurs des états reposant sur les retours moyens et l'optimisme face à l'incertain.

MHSP calcule des valeurs moyennes pour la décision et non des valeurs directes. Cela signifie que la valeur d'une action dépend de tous les nœuds explorés au-delà de cette action, et pas seulement sur le meilleur nœud trouvé. Ce fait peut avoir un fort impact sur la façon dont le système explore les nœuds parce que le système focalise sur les actions permettant d'atteindre des nœuds globalement bons, et non sur l'action permettant d'atteindre le nœud ayant la meilleure valeur heuristique. Dans un contexte de temps limité, se concentrer sur l'action qui mène à des nœuds globalement bons au lieu d'un seul nœud en particulier peut limiter les effets de valeurs heuristiques fortement erronées. Elle permet d'explorer subtilement l'arbre. Plus le problème est complexe, plus cet effet devrait être visible.

Trois tests ont été conçus afin de comparer MHSP, A * et BFS dans la RTR. Le premier a donné un aperçu de l'efficacité globale de chaque algorithme à trouver de bons plans dans différents problèmes des domaines ferry, gripper et satellite. Elle a montré que MHSP est globalement meilleur que A* et BFS en termes de longueur de plan solution. Le deuxième essai avait pour but d'observer la convergence des performances lorsque l'apprentissage est appliqué dans les trois algorithmes, i.e., lorsque les valeurs heuristiques des nœuds visités peuvent être mises à jour en fonction de l'exploration. Les premiers résultats montrent que l'apprentissage améliore la qualité des meilleurs plans obtenus avec les trois algorithmes. Ils ont par ailleurs montré que MHSP tend à converger très rapidement vers un plan optimal, tandis que A* et BFS peuvent rester sous-optimaux et instables. Enfin, le troisième essai a été conçu afin d'évaluer la capacité des algorithmes à effectuer en temps réel la sélection d'une action, ou d'un plan partiel, efficace. Ce test fait varier le temps de décision et évalue la qualité des plans partiels obtenus au moyen de deux distances : la distance à l'objectif et la distance à l'optimum. Les résultats ont mis en évidence que, le temps de décision augmentant, MHSP est beaucoup plus rapide à fournir des plans optimaux que les deux autres algorithmes.

À l'avenir, nous pouvons étudier plusieurs aspects du travail présenté :

– Tout d'abord, nous souhaitons étudier la possibilité d'exécuter des séquences d'actions au lieu d'une seule action, comme le font des algorithmes tels que γ-Trap ou LRTS. En effet, au lieu de faire une action unique entre les étapes de recherche en avant, ils appliquent d actions afin d'amortir le coût de la planification. Cela permet d'accélérer la recherche d'un plan solution lorsque la fonction heuristique est informative.
– Récemment, Nakhost et Müller (Nakhost & Müller, 2009) ont présenté le planificateur ARVAND appliquant des simulations aléatoires Monte-Carlo pour la planification classique déterministe. Une comparaison entre cette approche et MHSP présente un grand intérêt.
– Puisque MHSP utilise des valeurs moyennes, nous voulons aussi appliquer MHSP sur des problèmes non déterministes et le comparer aux algorithmes en ligne basés sur les processus de décision markoviens.
– Les expériences montrent que la première action choisie à un impact significatif sur la qualité du plan solution trouvé en termes de longueur. Par exemple, dans les domaines blocksworld, choisir d'abord un mauvais bloc à déplacer implique d'ajouter de nombreuses actions pour réparer ce mauvais choix. Par conséquent, l'idée est d'allouer plus de temps à la première étape de raisonnement.
– Dans nos expériences, pour chaque étape de la sélection d'action, l'arbre est calculé à partir de zéro. Ré-utiliser l'arbre calculé au cours des étapes précédentes de sélection de l'action est une amélioration intéressante des travaux. Il permettra aux algorithmes temps réel de résoudre des problèmes plus difficiles.
– Notre travail se fait en arrière-plan de la recherche en temps réel et de la sélection de plans partiels.

Supprimer la contrainte de temps réel et tester MHSP sur des problèmes dans lesquels des plans solutions complets sont nécessaires est une voie de recherche intéressante. Toutefois, les tests préliminaires montrent que, avec des heuristiques presque exactes, MHSP est difficilement comparable aux planificateurs efficaces et généraux utilisant Enforced-Hill Climbing pour trouver des plans solutions complets sur un large éventail de problèmes.

– Enfin, apprendre les heuristiques utiles à la planification en utilisant MHSP, ou un autre algorithme en temps réel, et les comparer avec les heuristiques obtenues avec les graphes de planification est une perspective de recherche intéressante reliant les deux domaines de l'apprentissage et de la planification.

7 Remerciements

Ce travail a été effectué dans le cadre du programme COSINUS (projet EXPLO-RA n°ANR-08-COSI-004) de l'ANR.

Références

AUER P., CESA-BIANCHI N. & FISHER P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, **47**(2–3), 235–256.

BARTO A., BRADTKE S. & SINGH S. (1995). Learning to Act Using Real-Time Dynamic Programming. *Artificial Intelligence*, **72**(1-2), 81–138.

BJARNASON R., FERN A. & TADEPALLI P. (2009). Lower bounding klondike solitaire with monte-carlo planning. In *Proc. ICAPS*.

BULITKO V. (2004). *Learning for Adaptive Real-Time Search*. Rapport interne, Department of Computer Science, University of Alberta, http ://arxiv.org/abs/cs.AI/0407016.

BULITKO V. & LEE G. (2006). Learning in Real-Time Search : A Unifying Framework. *Journal of Artificial Intelligence Research*, **25**, 119–157.

CHASLOT G., DE JONG S., SAITO J. & UITERWIJK J. (2006). Monte-Carlo Tree Search in Production Management Problems. In P.-Y. SCHOBBENS, W. VANHOOF & G. SCHWANEN, Eds., *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence*, p. 91–98.

CHASLOT G., WINANDS M., VAN DEN HERIK H., UITERWIJK J. & BOUZY B. (2008). Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, **4**(3), 343–357.

FABIANI P., FUERTES V., BESNERAIS G., MAMPEY R., PIQUEREAU A. & TEICHTEIL F. (2007). The ReSSAC Autonomous Helicopter : Flying in a Non-Cooperative Uncertain World with embedded Vision and Decision Making. In *A.H.S Forum*.

FURCY D. & KOENIG S. (2000). Speeding up the Convergence of Real-Time Search. In *Proceedings of the National Conference on Artificial Intelligence*, p. 891–897.

GELLY S., WANG Y., MUNOS R. & TEYTAUD O. (2006). *Modification of UCT with Patterns in Monte-Carlo Go*. Rapport interne RR-6062, INRIA.

HANSEN E. & ZILBERSTEIN S. (2001). LAO* : A Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence*, **129**(1-2), 35–62.

HART P., NILSSON N. & RAPHAEL B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics SSC4*, **2**, 100—107.

HOFFMANN J. & NEBEL B. (2001). The FF Planning System : Fast Plan Generation Through Heuristic Search. *JAIR*, **14**(1), 253–302.

KOCSIS L. & SZEPESVARI C. (2006). Bandit-based Monte-Carlo Planning. In *Proc. ECML*, p. 282–293.

KORF R. (1990). Real-Time Heuristic Search. *Artificial Intelligence*, **42**(2-3), 189–211.

NAKHOST H. & MÜLLER M. (2009). Monte-carlo exploration for deterministic planning. In *Proc. IJCAI*.

PELLIER D., BOUZY B. & MÉTIVIER M. (2010). An UCT Approach for Anytime Agent-based Planning. In *Proc. PAAMS*.

RICHTER S. & WESTPHAL M. (2008). The lama planner using landmark counting in heuristic search. In *Proceedings of the International Conference on Planning and Scheduling*.

ROBERTS M. & HOWE A. (2009). Learning from Planner Performance. *Artificial Intelligence*, **173**, 536–561.

SHIMBO M. & ISHIDA T. (2003). Controlling the Learning Process of Real-Time Heuristic Search. *Artificial Intelligence*, **146**(1), 1–41.

SHUE L. & ZAMANI R. (1993). An Admissible Heuristic Search Algorithm. In *Methodologies for Intelligent Systems*, number 689 in LNAI. Springer.