Planifier lorsque le but change. Une approche inspirée de la recherche de cible mouvante

Damien Pellier, Humbert Fiorino, Marc Métivier

Planifier lorsque le but change : une approche inspirée de la recherche de cible mouvante

D. Pellier* — **H. Fiorino**** — **M. Métivier***

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

*** Laboratoire d'Informatique de Grenoble – Université Joseph Fourier*
110 avenue de la Chimie 38400 Saint-Martin-d'Hères
humbert.fiorino@imag.fr

RÉSUMÉ. *Dans cet article, nous proposons un nouvel algorithme de planification temps réel appelé MGP (Moving Goal Planning) capable de s'adapter lorsque le but évolue dynamiquement au cours du temps. Cet algorithme s'inspire des algorithmes de type Moving Target Search (MTS). Afin de réduire le nombre de recherches effectuées et améliorer ses performances, MGP retarde autant que possible le déclenchement de nouvelles recherches lorsque que le but change. Pour cela, MGP s'appuie sur deux stratégies : Open Check (OC) qui vérifie si le nouveau but est présent dans l'arbre de recherche déjà construit lors d'une précédente recherche et Plan Follow (PF) qui estime s'il est préférable d'exécuter les actions du plan courant pour se rapprocher du nouveau but plutôt que de relancer une nouvelle recherche. En outre, MGP utilise une stratégie "conservatrice" de mise à jour incrémentale de l'arbre de recherche lui permettant de réduire le nombre d'appels à la fonction heuristique et ainsi d'accélérer la recherche d'un plan solution. Finalement, nous présentons des résultats expérimentaux qui montrent l'efficacité de notre approche.*

ABSTRACT. *In this paper, we propose a novel planner, called Moving Goal Planner (MGP) in order to adapt plans when the goal changes over time. This planner draws inspiration from Moving Target Search (MTS) algorithms. In order to limit the number of search iterations and to improve its efficiency, MGP delays as much as possible starting new searches when the goal changes. To this purpose, MGP uses two strategies: Open Check (OC) that checks if the new goal is still in the current search tree and Plan Follow (PF) that estimates whether executing the actions of the current plan brings MGP closer to the new goal. Moreover, MGP uses a parsimonious strategy to adapt incrementally the search tree at each new search that reduces the number of calls to the heuristic function and speeds up the search. Finally, we show evaluation results that demonstrate the effectiveness of our approach.*

MOTS-CLÉS : *Planification de tâches, recherche de cible mouvante*

KEYWORDS: *Task Planning, Moving Target Search*

1. Introduction

À l'heure actuelle, l'immense majorité des robots en service est conçue pour des applications industrielles ou militaires[1]. Mais, des robots personnels à usage domestique réalisant des tâches simples commencent à rentrer dans nos vies quotidiennes (par exemple, le robot-aspirateur Roomba d'iRobot [SUN 09b, FOR 06] capable de slalomer entre les meubles et de revenir tout seul à sa base pour se recharger). Bien qu'un grand nombre d'applications de la robotique personnelle soient au stade d'identification ou d'expérimentation (robots compagnons pour personnes âgées [FOR 04], robots de rééducation [TAP 08], robots pour l'enseignement [KAN 07, TAN 06] etc.), pour beaucoup de personnes la robotique personnelle sera la prochaine révolution industrielle et provoquera la même rupture technologique et sociétale que l'électricité ou Internet [SAB 10]. Les espoirs placés dans la robotique personnelle sont donc importants mais les verrous scientifiques et technologiques à surmonter sont encore très nombreux.

L'un des freins sérieux au développement de la robotique personnelle est sans doute son déficit de maturité technologique et, par voie de conséquence, d'utilisabilité des robots. Nul doute que des avancées décisives doivent être faites en matière de logiciel système, navigation, perception et capteurs, gestion d'énergie, mécatronique etc. Mais quelles que soient les avancées importantes obtenues dans ces domaines, la clé pour la robotique personnelle se trouve, de notre point de vue, dans les Interactions Homme-Robot et la capacité des robots à adapter en permanence leurs décisions dans un environnement dynamique du fait des activités humaines.

Dans de tels environnements, les robots doivent constamment faire face aux événements qui viennent perturber leurs plans en entrelaçant planification et exécution. On parle alors de planification en boucle fermée [FOX 06]. Il n'est pas rare que les activités humaines conduisent à une modification de la tâche assignée au robot.

Dans cet article, nous proposons un nouvel algorithme de planification temps réel appelé MGP (*Moving Goal Planning*) pour la planification en boucle fermée où la recherche d'un plan est vue comme une recherche heuristique de type MTS[2] entrelaçant planification et exécution où seul le but but confié au système évolue au cours du temps. Les algorithmes MTS sont des algorithmes de recherche temps réel de cibles mouvantes (MTS) : un agent « chasseur » poursuit une cible mouvante aussi appelée « proie » et les algorithmes MTS entrelacent recherche d'un chemin vers la proie (planification de trajectoire) et déplacement du chasseur. Ces algorithmes heuristiques (fondées sur le calcul de distances) ont été très peu utilisés dans le cadre de la planification de tâches. Ceci s'explique en partie par le fait qu'il a longtemps été considéré difficile de trouver des fonctions heuristiques à la fois informatives et facilement calculables dans le cadre de la planification de tâches. Ce n'est que vers la fin des années

1. Étude prospective (Pôle interministériel de prospective et d'anticipation des mutations économiques - Pipame), "Le développement industriel futur de la robotique personnelle et de service en France", http ://www.industrie.gouv.fr/p3e/etudes-prospectives/robotique/robotique.pdf
2. *Moving Target Search*

90 que des planificateurs tels que HSP et HSP-r [BON 99, BON 01] ont renversé la tendance en fournissant des méthodes génériques et efficaces pour calculer de telles fonctions heuristiques en temps polynomial. Ceci a eu pour conséquence de faire évoluer de manière quasiment indépendante ces deux domaines de recherche (MTS et planification de tâches). Il nous apparaît dorénavant important de capitaliser sur les récentes avancées dans ces deux domaines pour proposer de nouveaux algorithmes de planification temps réel en boucle fermée.

Le reste de l'article est organisé selon le plan suivant : la partie 2 donne l'état de l'art en matière de planification en boucle fermée ; la partie 3 présente formellement le problème considéré et décrit l'algorithme proposé pour le résoudre ; la partie 4 présente les résultats expérimentaux obtenus qui montrent l'efficacité de notre approche ; finalement, la partie 5 présente nos conclusions sur le travail réalisé.

2. Etat de l'art

La littérature distingue principalement deux manières d'aborder le problème de la planification en boucle fermée [NEB 95, KAM 92, COX 98, FOX 06, KRO 05] : planifier un nouveau plan en partant de zéro ou réparer le plan courant pour prendre en compte le nouveau contexte. Bien qu'en théorie ces deux approches soient équivalentes en terme de coût dans le pire des cas [NEB 95], les résultats expérimentaux montrent que l'approche qui consiste à réparer un plan existant est plus efficace que celle qui consiste à replanifier sans réutiliser les informations provenant des précédentes recherches [KRO 05]. De plus, faire évoluer aussi peu que possible les plans (stabilité des plans) est un autre argument en faveur de la seconde approche [FOX 06].

Le problème de la replanification dans des environements dynamiques a également été abordé de manière partielle dans le domaine des jeux vidéo pour résoudre le problème de la capture en temps réel de cibles mouvantes (MTS). Ce problème reste un problème ouvert pour de grands espaces de recherche. Par essence, un algorithme de type MTS entrelace recherche d'un chemin et exécution : un agent « chasseur » poursuit une cible mouvante aussi appelée « proie » dans une grille à deux dimensions ou une carte. Depuis les travaux pionniers d'Ishida [ISH 91, ISH 95, ISH 98], les approches de type MTS peuvent se partager principalement en deux catégories selon la stratégie sur laquelle elles s'appuient pour réutiliser l'information des précédentes recherches.

La première stratégie consiste à utiliser une fonction heuristique pour guider la recherche et apprendre le chemin le plus court entre des couples de points sur une carte. À chaque nouvelle recherche, la fonction heuristique est plus informative et la recherche devient plus performante car elle s'appuie de plus en plus sur des valeurs apprises et observées de l'environement en lieu et place des valeurs estimées par la fonction heuristique. Dans sa version originale, l'algorithme MTS est une adaptation de l'algorithme LRTA* (*Learning Real-Time A**) [KOR 90]. Cette approche est complète lorsque les déplacements de la cible sont plus lents que celui de l'agent. Toutefois, elle

n'est pas sans inconvénient à cause des dépressions de la fonction heuristique[3] et de la perte d'information induite par les déplacements de la cible [MEL 93].

Actuellement, l'état de l'art des algorithmes fondés sur cette première stratégie sont des variantes de AA* [KOE 05], MTAA* [KOE 07] et GAA* [SUN 08]. MTAA* adapte et apprend la fonction heuristique à partir des déplacements de la cible pour améliorer la convergence. GAA* généralise MTAA* avec la prise en compte d'actions auxquelles on a associé des coûts qui augmentent et diminuent au cours du temps. MTAA* et GAA* nécessitent tous deux l'utilisation de fonctions heuristiques admissibles pour garantir leur correction et leur complétude.

La seconde stratégie consiste à utiliser incrémentalement l'arbre de recherche entre deux recherches successives. Le premier algorithme fondé sur cette stratégie est D* [STE 95]. Ses deux principaux successeurs sont décrits dans [KOE 02, SUN 10b]. Ces algorithmes ont été conçus et développés pour calculer rapidement des trajectoires dans des environnements inconnus et dynamiques. Ils s'appuient tous sur une recherche en chaînage arrière. Ils obtiennent de bons résultats lorsque l'environnement évolue peu au cours du temps. En revanche, ils sont surclassés par l'approche naïve qui consiste à appeler successivement l'algorithme A* [HAR 68] à chaque fois que la cible se déplace [SUN 08] lorsque les seules modifications portent sur le but. Récemment, un autre algorithme s'appuyant sur une recherche en chaînage avant appelé FRA* a été proposé [SUN 09a]. Cet algorithme n'est pas capable de prendre en compte les changements de l'environnement, mais obtient de très bons résultats quand il s'agit de prendre en compte les déplacements d'une cible au cours du temps. Chaque fois que la cible se déplace, FRA* adapte rapidement l'arbre de recherche précédemment construit à la nouvelle position de la cible et rappelle la fonction de recherche A* sur le nouvel arbre de recherche. FRA* est actuellement l'algorithme MTS le plus efficace. Toutefois, l'adaptation de l'arbre est largement dépendante de la modélisation de l'environnement. FRA* considère en effet que l'environnement est modélisé sous la forme d'une grille ce qui en limite les applications dans le cas général. Afin de pallier cet inconvénient, une variante de cet algorithme appelée GFRA* [SUN 10a] a été proposée pour fonctionner dans des environnements modélisés par des graphes quelconques, y compris des treillis d'états utilisés pour la navigation des véhicules terrestres autonomes.

En conclusion, les algorithmes de la première stratégie permettent de prendre en compte plus facilement les éventuels évolutions de l'environnement mais souffrent de la perte d'information induite par les déplacements de la cible, ce qui les rend peu efficaces. Par opposition les algorithmes de la seconde stratégie sont plus efficaces, car plus spécialisés, surtout pour prendre en compte les évolutions du but au cours du temps. Parmi ces derniers, GFRA* semble être le plus intéressant à appliquer dans le cadre de la planification. En effet, il s'appuie sur une recherche en chaînage avant dans

3. On appelle *dépression heuristique* un ensemble d'états connectés, ayant les valeurs heuristiques égales ou plus petites que les valeurs des états qui les entourent.

des graphes comme les plus performants planificateurs actuels et n'impose pas que la fonction heuristique soit admissible, ce qui est souvent le cas en planification.

3. Planifier avec des buts dynamiques

Dans cette partie, nous présentons tout d'abord formellement le problème traité. Dans un deuxième temps, nous présentons l'algorithme MGP initié par [VAN 11] et détaillons sa stratégie de recherche. Finalement, nous introduisons deux stratégies qui permettent de retarder le déclenchement d'une nouvelle recherche lorsque le but évolue dynamiquement : *Open Check* et *Plan Follow*, et terminons en présentant la stratégie incrémentale de mise à jour de l'arbre de recherche. L'arbre de recherche est un arbre n-aire composé d'un nœud racine et d'un ensemble pouvant être vide de nœuds fils qui définissent récursivement des sous-arbres n-aires. Les nœuds représentent les états du monde atteignables et les arcs les actions à exécuter pour les atteindre.

3.1. *Modélisation du problème*

Nous nous intéressons aux problèmes de planification séquentielle de type STRIPS [FIN 71]. Tous les ensembles considérés sont finis. Un *état* s est un ensemble de propositions logiques. Une *action* a est un tuple $a = (pre(a), add(a), del(a))$ où $pre(a)$ définit les *préconditions* de l'action a et $add(a)$ et $del(a)$ représentent respectivement ses *effets* positifs et négatifs. Un état s' est atteignable à partir d'un état s en appliquant à l'action a la fonction de transition γ suivante :

$$s' = \gamma(s, a) = \begin{cases} (s - del(a)) \cup add(a) & \text{si } pre(a) \subseteq s \\ \text{indéfini} & \text{sinon} \end{cases} \quad (1)$$

L'exécution d'une séquence d'actions $\pi = \langle a_1, \ldots, a_n \rangle$ dans un état s est défini récursivement par :

$$\gamma(s, \langle a_1, \ldots, a_n \rangle) = \gamma(\gamma(s, a_1), \langle a_2, \ldots, a_n \rangle) \quad (2)$$

Un problème de planification en boucle fermée avec but dynamique est un tuple (A, s_t, g_t) : à un temps donné t un agent est dans un état s_t ; g_t est son but courant (s_t et g_t sont des ensembles de propositions) et A représente l'ensemble des actions qu'il peut exécuter. L'agent exécute des actions appartenant à A pour atteindre son but mais celui-ci peut évoluer dynamiquement à tout moment. L'agent ne possède aucune information sur la manière dont le but évolue au cours du temps. Cependant, nous supposons qu'à tout instant, le but g_t est atteignable : un *plan* est une séquence d'actions $\pi_t = \langle a_1, \ldots, a_n \rangle$ avec $a_i \in A$ telle que $g_t \subseteq \gamma(s_t, \pi)$. g_t est *atteignable* si un tel plan existe. Un *état but* est un état s tel que $g_t \subseteq s$. À un temps donné t, un problème de planification avec but dynamique est résolu si $g_t \subseteq s_t$, autrement dit l'agent a atteint son but.

3.2. *Principe de l'algorithme*

Le pseudo-code de MGP est donné par l'algorithme 1. MGP prend en entrée un problème de planification (A, s_0, g_0). Les variables g et s représentent respectivement le but courant et l'état courant initialement égaux à g_0 et s_0. i est un compteur qui permet de comptabiliser le nombre de recherches effectuées.

MGP appelle itérativement une procédure de recherche (ligne 3) tant que le but courant n'a pas été atteint. La procédure de recherche Search est détaillée dans le paragraphe §3.3.1. Cette procédure construit un arbre de recherche en chaînage avant dans un espace d'états. Si le but courant n'est pas atteignable, la recherche échoue impliquant l'échec de MGP également (ligne 3). Sinon, MGP extrait un plan à partir de l'arbre de recherche construit au cours de la recherche (lignes 4-5) et essaie de retarder autant que possible un nouvel appel à la procédure de recherche, i.e., une nouvelle expansion de l'arbre de recherche (boucle While et procédure CanDelayNewSearch ligne 4). Cette boucle prend fin lorsque le but évolue et n'est plus présent dans l'arbre de recherche précédemment construit. La procédure CanDelayNewSearch est détaillée dans le paragraphe §3.3.2. La procédure CanDelayNewSearch s'appuie sur deux critères pour déterminer si une recherche peut être retardée : (1) elle vérifie que le but n'est pas déjà atteignable, i.e., n'est pas présent dans l'espace de recherche (procédure OpenCheck) ; (2) elle estime l'éloignement du nouveau but par rapport à l'ancien et renvoie vrai si cette distance est inférieure à une borne donnée (procédure PlanFollow). Tant que le but est accessible à partir du plan précédemment extrait ou que le but n'a pas évolué de manière significative (procédure PlanFollow également présentée dans le §3.3.2), MGP exécute alors les actions de ce plan (lignes 6). Dans le cas où MGP atteint le but courant, MGP se termine avec succès (ligne 12). Dans le cas contraire le but a évolué (ligne 11) et MGP doit élaguer son arbre de recherche. La fonction UpdateGoal simule l'évolution du but et retourne le nouveau but s'il a évolué. L'arbre de recherche est alors le sous-arbre dont la racine est représentée par son état courant s (DeleteStatesOutOfTree, ligne 13). Si le nouveau but est présent dans ce sous-arbre, il n'est pas nécessaire de lancer une nouvelle recherche. MGP extrait directement un nouveau plan à partir de celui-ci et exécute ses actions pour atteindre le nouveau but. Sinon, MGP met à jour les valeurs heuristiques des nœuds de l'arbre de recherche en estimant la distance pour atteindre le nouveau but (ligne 14) et finalement étend l'arbre de recherche ainsi modifié (ligne 3). La procédure de mise à jour de l'arbre de recherche UpdateSearchTree est détaillée dans le paragraphe §3.3.3.

3.3. *Implémentation de l'algorithme*

Dans cette partie, nous présentons les détails relatifs à l'implémentation de MGP. L'arbre de recherche est représenté par deux listes notées OPEN et CLOSED : la liste OPEN contient les états feuilles de l'arbre de recherche restant à explorer et la liste CLOSED, la liste des états déjà explorés. Dans un premier temps, nous présentons la stratégie de recherche utilisée Search (cf. §3.3.1). Puis, nous présentons la procédure

Algorithm 1: MGP(A, s_0, g_0)

```
1  s ← s_0, g ← g_0, i ← 1
2  while g ⊈ s do
3      if Search(A, s, g, i) = FAILURE then return FAILURE
4      while CanDelayNewSearch(s, g) do
5          Extract plan π = ⟨a_1, ..., a_n⟩ from the search tree
6          while (g ⊈ s and g ⊆ γ(s, π)) or (PlanFollow(s, g) and π ≠ ∅) do
7              a ← extract the first action of π
8              Executes a
9              s ← γ(s, a)
10             π ← extract the trail of π (π without its first action)
11             g ← UpdateGoal(g)
12         if g ⊆ s then return SUCCESS
13         DeleteStatesOutOfTree(s)
14     UpdateSearchTree(s, g)
15     i ← i + 1
```

CanDelayNewSearch utilisée pour retarder les appels coûteux à la procédure de recherche. Finalement, nous présentons la procédure UpdateSearchTree qui met à jour l'arbre de recherche entre deux recherches successives (cf. §3.3.3).

3.3.1. *A* pondéré comme stratégie de recherche*

Contrairement à GFRA* qui s'appuie sur une simple recherche de type A*, MGP utilise sa version pondérée. Cette variante de A* surestime volontairement la distance au but en multipliant la valeur retournée par la fonction heuristique d'un coefficient w. La fonction d'évaluation $f(s)$ pour un état s est alors $f(s) = g(s) + w \times h(s)$ où $g(s)$ est le coût pour atteindre s à partir de l'état initial s_0 et $h(s)$ le coût estimé pour atteindre g depuis s. Plus w est élevé, plus la fonction heuristique est prépondérante dans l'exploration de l'espace de recherche. Expérimentalement, il a été démontré que cette stratégie couplée avec une fonction heuristique informative et admissible améliore les performances en temps mais, en contrepartie, ne permet plus d'obtenir des solutions optimales [POH 70]. Toutefois, cette approche est pertinente pour le problème considéré car il est plus important de trouver rapidement une solution que de trouver un plan solution optimal. Par conséquent, utiliser une recherche pondérée de type A* peut améliorer de manière significative les performances de MGP (cf. §4). En outre, il est important de noter que l'utilisation de cette procédure de recherche n'affecte pas la correction et la complétude de MGP.

La version de A* pondéré utilisée dans notre approche (cf. Algo. 2) est une version légèrement modifiée de l'algorithme classique proposé par [POH 70]. Il prend en entrée un problème de planification (A, s_0, g), un coefficient w et le compteur i qui comptabilise le nombre de recherches effectuées. À chaque état s, il associe quatre va-

leurs : sa valeur $g(s)$ et sa valeur $h(s)$, un pointeur vers son état père, $parent(s)$, dans l'arbre de recherche et l'itération à laquelle il a été créé, $iteration(s)$. $iteration(s)$ permet à l'algorithme de ne mettre à jour que les états nouvellement crées (cf. § 3.3.3). Au premier appel de la procédure, les listes OPEN et CLOSED contiennent l'état initial s_0 du problème de planification tel que $g(s_0) = 0$ and $h(s_0) = H(s_0, g)$ où H est la fonction heuristique qui estime le coût de s_0 au but g.

La procédure de recherche A* pondérée (cf. Algo. 2) consiste à répéter itérativement les étapes suivantes : (1) sélectionner l'état s de la liste OPEN qui possède la plus petite valeur de $f(s)$ (ligne 2), (2) supprimer l'état s de la liste OPEN (ligne 4), et (3) développer les états fils de s en utilisant la fonction de transition γ (cf. eq. 1). Pour chaque état fils s' obtenu avec la fonction de transition γ, si s' n'est pas dans une des deux listes OPEN et CLOSED, alors A* pondéré calcule $g(s') = g(s) + c(s, s')$ où $c(s, s')$ représente le coût pour atteindre s' depuis s. Si s' n'a pas encore été exploré (lignes 8-12), s' est ajouté à la liste OPEN. Sinon, A* pondéré met à jour les informations relatives à s'. De plus, si A* trouve un plan plus court pour atteindre s' (lignes 14-17), il met alors à jour s' : $g(s') = g(s) + c(s, s')$ et $parent(s') = s$. Finalement, si la valeur heuristique est obsolète (lignes 18-20) parce que le but a changé, A* pondéré met à jour la valeur heuristique de s'.

A* pondéré possède deux conditions d'arrêt : (1) lorsque la liste OPEN est vide (aucun état contenant le but n'a pu être trouvé) et (2) lorsqu'un état contenant le but est trouvé (ligne 3). Dans ce cas, un plan solution peut être extrait de l'arbre de recherche construit par la procédure de recherche (cf. Algo. 1, ligne 5).

3.3.2. *Stratégies de retardement*

Afin de limiter le nombre de recherches effectuées et améliorer ses performances, MGP retarde autant que possible le déclenchement d'une nouvelle recherche lorsque le but évolue. Pour cela, MGP utilise deux stratégies originales mises en œuvre dans la procédure CanDelayNewSearch (cf. Algo. 1 ligne 4) :

Open Check (OC)

MGP vérifie si le nouveau but est encore présent dans l'arbre de recherche, i.e., dans la liste OPEN ou dans la liste CLOSED. Dans ce cas, un nouveau plan peut être extrait directement à partir de l'arbre de recherche du nouvel état but à l'état courant. Contrairement à GFRA* qui ne teste que la liste CLOSED contenant les états explorés, MGP effectue également le test des états feuilles de l'arbre de recherche contenus dans la liste OPEN avant de débuter une nouvelle recherche. Cette vérification limite le nombre de recherches inutiles et les réajustements coûteux de l'arbre de recherche.

Plan Follow (PF)

MGP effectue une estimation pour savoir si l'exécution de son plan courant le rapproche du but, même si celui-ci a changé. Chaque fois que le but évolue et avant de lancer une nouvelle recherche, MGP évalue si le nouveau but est proche du but

Algorithm 2: Search(A, s, g, i)

```
1  while open ≠ ∅ do
2      s ← argmin_{s'∈open}(g(s') + w × h(s'))
3      if g ⊆ s then return SUCCESS
4      open ← open \ {s}
5      closed ← closed ∪ {s}
6      foreach action a ∈ {a' ∈ A | pre(a') ⊆ s} do
7          s' ← γ(s, a)
8          cost ← g(s) + c(s, s')
9          if s' ∉ open ∪ closed then
10             open ← open ∪ {s'}
11             g(s') ← cost
12             h(s') ← H(s', g)
13             parent(s') ← s
14         else
15             if cost < g(s') then
16                 open ← open ∪ {s'}
17                 g(s') ← cost
18                 parent(s') ← s
19             if iteration(s') < i then
20                 h(s') ← H(s', g)
21                 iteration(s') ← i
22 return FAILURE
```

précédent et détermine si le plan courant peut encore être utilisé. Cette évaluation s'appuie sur l'estimation de la fonction heuristique et le calcul d'une comparaison entre l'état courant s, le but précédent p et le nouveau but g :

$$H(s, g) \times c > H(s, p) + H(p, g) \tag{3}$$

c est appelé le coefficient de retardement. MGP suit le plan courant tant que l'inégalité 3 est vérifiée, i.e., tant qu'il estime préférable d'exécuter le plan courant pour atteindre l'ancien but puis se "diriger" vers le nouveau but plutôt que d'exécuter un plan allant directement de l'état courant s au nouveau but g. Les valeurs de $c > 1$ permettent d'ajuster le délai avant une nouvelle recherche. Étant donné que les recherches sont très coûteuses, les retarder améliore les performances de MGP mais altère la qualité des plans (cf. §4).

3.3.3. *Mise à jour incrémentale de l'arbre de recherche*

MGP adapte incrémentalement l'arbre de recherche à chaque nouvelle recherche (cf. UpdateTreeSearch Algo. 1 ligne 13). Contrairement à GFRA* qui met à jour la

valeur heuristique de tous les états de l'arbre de recherche en estimant la distance au nouveau but, MGP s'appuie sur une stratégie moins agressive de mise à jour afin de réduire le nombre d'appels à la fonction heuristique et ainsi améliorer ses performances globales.

Dans cet objectif, MGP copie les états de la liste OPEN contenant les états feuilles de l'arbre de recherche dans la liste CLOSED, puis vide la liste OPEN et y ajoute l'état courant en prenant soin de mettre à jour sa valeur heuristique (appel de la fonction heuristique pour estimer la distance de l'état courant au nouveau but). Pour indiquer que la valeur heuristique est à jour, MGP marque l'état avec le compteur incrémenté à chaque nouvelle recherche (cf. Algo. 2, ligne 20). Chaque fois qu'un état dans la liste CLOSED est rencontré avec une valeur d'itération inférieure à la valeur du compteur d'itération courant, l'état est ajouté dans la liste OPEN et sa valeur heuristique est mise à jour. Cette stratégie possède deux avantages : (1) elle réduit le nombre d'états créés en réutilisant les états précédemment construits et (2) réduit de manière significative le temps nécessaire à la mise à jour des valeurs heuristiques des états à partir du nouveau but en se limitant aux états explorés au cours de la nouvelle recherche.

4. Expérimentation et évaluation

L'objectif de ces expériences est d'évaluer les performances de MGP en fonction des différentes stratégies de retardement proposées : *Open Check* (OC) et *Plan Follow* (PF) comparativement aux approches existantes. Nous avons choisi de comparer MGP à l'algorithme GFRA* qui est l'algorithme de référence dans le domaine des algorithmes de type MTS et à l'approche naïve SA* (*Successive A**) qui consiste à appeler la procédure de recherche A* à chaque fois que le but évolue. Les benchmarks utilisés pour l'évaluation sont issus des compétitions internationales de planification (IPC). Nous utilisons pour l'ensemble de nos tests la fonction heuristique non-admissible proposée par [HOF 01] dans le planificateur FF pour guider la recherche. Dans la suite, nous évaluons six algorithmes : Successive A* (SA*), Generalized Fringe-Retrieving A* (GFRA*), MGP sans stratégie de retardement (MGP), MGP avec Open Check (MGP+OC), MGP avec Plan Follow (MGP+PF) et finalement MGP avec les deux stratégies simultanément (MGP+OC+PF).

4.1. *Simulation et dynamique d'évolution du but*

Classiquement, les algorithmes MTS supposent que la cible mouvante – la « proie » – suit toujours le plus court chemin de sa position courante à sa nouvelle position choisie de manière aléatoire parmi les positions libres sur une grille ou une carte. Chaque fois que la proie a atteint cette position, une nouvelle position est alors sélectionnée de manière aléatoire et le processus se répète. Tous les n déplacements, la proie reste immobile laissant ainsi au « chasseur » une chance de l'attraper. Cette approche n'est malheureusement pas directement transposable à la planification de

tâches en boucle fermée. En effet, dans le cadre plus générale de la planification, le nouveau but n'est pas systématiquement atteignable depuis l'ancien.

Afin de l'adapter au mieux, nous avons décidé de faire évoluer le but à un pas de temps donné en lui appliquant de manière aléatoire une action parmi les actions définies dans le problème de planification. Nous faisons ici l'hypothèse que le but évolue en accord avec le modèle que l'agent a du monde. Le but que nous faisons évoluer est l'état solution obtenu par le premier appel à la procédure de recherche. En effet, dans le cas général, il peut y avoir plusieurs états but (le but d'un problème de planification de type STRIPS est un état partiel dont l'inclusion dans un état complet est requise). Le processus est répété plusieurs fois pour rendre le but plus difficile à atteindre. Notons que cette façon de faire évoluer le but garantit que le nouveau but est toujours atteignable depuis le but courant. Cependant, il n'est pas garanti que MGP soit capable d'atteindre le but puisque celui-ci peut évoluer bien plus rapidement que MGP.

En outre, il est important de mentionner que notre cadre expérimental est plus difficile que celui utilisé classiquement pour comparer les algorithmes de type MTS. En effet dans notre cas, le but n'évolue pas en fonction des actions exécutées mais en temps réel en fonction du temps de calcul nécessaire à la génération des plans solution. Autrement dit, plus un algorithme passe de temps à explorer l'espace de recherche du problème qui lui a été confié, plus le but évolue et plus le problème devient difficile à résoudre.

Pour paramétrer l'évolution du but en fonction du temps de recherche, un compteur t est incrémenté chaque fois qu'un état est exploré au cours d'une recherche avec A* pondéré et chaque fois que la fonction heuristique est appelée pour estimer la distance au but. En effet, ces deux opérations sont de loin les plus coûteuses de MGP. À partir de ce compteur, le nombre n d'actions appliquées au but est calculé comme suit :

$$n = (t - t_p)/g_r \tag{4}$$

où t_p représente la précédente valeur du compteur t et g_r le coefficient d'évolution du but. g_r nous permet d'ajuster la vitesse d'évolution du but et la difficulté de l'atteindre indépendamment du temps ou des caractéristiques techniques de la machine physique sur laquelle sont réalisées les expérimentations.

4.2. *Cadre expérimental*

Étant donné l'évolution aléatoire du but, chaque expérience a été réalisée 100 fois pour un problème et un coefficient d'évolution de but donné pour avoir des données statistiquement représentatives. Les résultats présentés par la suite sont donc des moyennes. Toutes les expériences ont été menées sur une machine Linux Fedora 16 équipée d'un processeur Intel Xeon 4 Core (2.0Ghz) avec un maximum de 4Go de mémoire vive et 60 secondes de temps CPU allouées pour chaque recherche.

Dans un premier temps, les différents algorithmes sont testés sur le domaine Blockworld extrait de IPC-2 afin de mesurer leurs performances respectives en fonction du coefficient d'évolution du but. Dans un deuxième temps, nous présentons quelques résultats montrant l'impact du coefficient de retardement ainsi que de la pondération de l'heuristique (nous présentons ici seulement les résultats obtenus par les meilleurs algorithmes de la première évaluation). Pour terminer, dans un troisième temps, nous proposons une vue d'ensemble des performances des différents algorithmes sur différents domaines et problèmes.

Les performances des algorithmes sont mesurées par : (1) le pourcentage de succès, i.e., le nombre de fois où un algorithme réussit à atteindre le but, (2) le temps mis pour atteindre le but, (3) la longueur des plans solution trouvés.

4.3. *Comparaison des algorithmes sur Blockworld*

Dans cette partie, nous présentons une comparaison des six algorithmes précédemment introduits – SA*, GFRA*, MGP, MGP+OC, MGP+PF et MGP+OC+PF – sur le problème 20 du domaine Blockworld de la compétition IPC-2 en fonction du coefficient d'évolution du but ainsi qu'une comparaison des mêmes algorithmes sur tout le domaine Blockworld afin d'en donner une comparaison globale. Le coefficient de retardement pour MGP+PF et MGP+OC+PF est fixé arbitrairement à 1.6 et le coefficient de pondération de l'heuristique est de 1 pour tous les algorithmes. L'étude de l'impact de ces paramètres est disponible au paragraphe 4.4.

4.3.1. *Le problème 20 du domaine Blockworld*

Les figures 1(a), 1(b) et 1(c) présentent respectivement les résultats obtenus en termes de pourcentages de succès, temps de recherche et longueur des plans trouvés sur le problème 20 extrait du domaine Blockworld. Les coefficients de variation moyens, minimums et maximums[4] des résulats présentés sont donnés à titre indicatif par le tableau 1.

En ce qui concerne le critère de succès, le meilleur algorithme est MGP+OC+PF. Même avec un coefficient d'évolution du but extrême $g_r = 1$, MGP+OC+PF parvient à trouver une solution dans 95% des expériences réalisées. Pour les autres stratégies de retardement, les résultats sont légèrement moins bons mais restent très satisfaisants puisqu'ils restent au dessus de 80% de succès. L'approche naïve SA* nécessite un coefficient d'évolution du but 5 fois plus grand pour obtenir le même pourcentage de succès. GFRA* lui n'atteint pas ce pourcentage de succès même avec un coefficient

4. En théorie des probabilités et statistiques, le coefficient de variation, également appelé écart relatif, est une mesure de dispersion relative : il se calcule comme le rapport entre l'écart type et la moyenne. Ce nombre est sans unité ; c'est une des raisons pour lesquelles il est préféré à la variance pour traiter des grandeurs physiques. Il nous permet ici de caractériser la qualité des résultats obtenus étant donné le caractère aléatoire de l'évolution du but.

	Temps			Longeurs		
	Moy.	Min.	Max.	Moy.	Min.	Max
SA*	2,49	0,35	5,63	0,20	0,11	0,46
GFRA*	1,64	0,01	5,61	0,13	0,01	0,33
MGP	2,11	0,52	3,37	0,14	0,09	0,25
MGP+OC	0,76	0,30	1,23	0,01	0,00	0,02
MGP+PF	2,38	0,87	5,29	0,20	0,11	0,71
MGP+OC+PF	3,02	1,10	5,95	0,19	0,10	0,59

Tableau 1. *Coefficients de variation moyens, minimums et maximums calculés sur le problème Blockworld 20.*

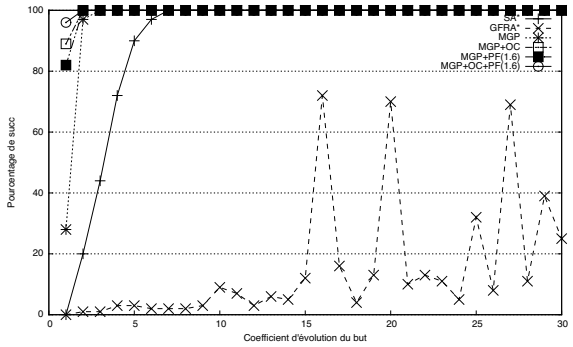
d'évolution du but 30 fois supérieur. Finalement, GFRA* est largement surclassé par les autres algorithmes.

En terme de temps de recherche, le meilleur algorithme est MGP+OC. Puis viennent ensuite MGP+OC+PF, MGP+PF et MGP. Finalement, nous avons SA* et très loin dernière GFRA*. Il semble que dans le cadre du problème étudié la stratégie OC soit extrêment efficace, le nouveau but étant souvent présent dans la liste des nœuds pendants de A*. En outre, le couplage des deux stratégies de retardement améliore de manière significative la version naïve de MGP. On constate également que les variantes de MGP avec l'option *Plan Follow* ont tendance à avoir un coefficient de variation plus grand. Ceci s'explique en partie par le comportement optimiste de ces variantes et le coût plus important à payer lorsque celles-ci effectuent un mauvais choix.

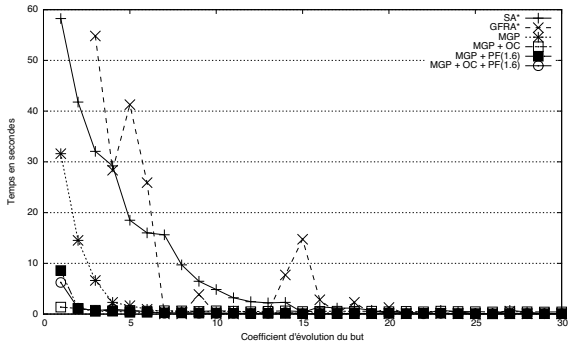
S'agissant de la longueur des plans trouvés, MGP et ses différentes variantes produisent des plans plus longs que SA* et GFRA* (presque une fois et demi plus longs dans le cas extrême où $g_r = 1$ dans le cas de MGP+OC). Deux raisons expliquent cette différence. Tout d'abord, rappelons que la stratégie de retardement OC vérifie si le nouveau but est déjà présent dans l'arbre de recherche. Si c'est le cas, MGP extrait alors directement un nouveau plan à partir de l'arbre de recherche. Toutefois, si ce mécanisme limite le nombre d'appel à la procédure de recherche, il ne garantit pas que le plan extrait soit optimal. En effet, même si l'heuristique est admissible, l'arbre de recherche n'a pas été construit pour le nouveau but. Deuxièmement, la stratégie de retardement PF (cf. figure 1(c)) tend à augmenter la longueur des plans trouvés. La différence de qualité des plans trouvés compense en partie des temps de recherche et des pourcentages de succès bien meilleurs. Cela dépend bien évidemment du critère à optimiser. En outre, les coefficients de variation observés sur la longueur des plans restent faibles quels que soient les algorithmes.

4.3.2. *Le domaine Blockworld*

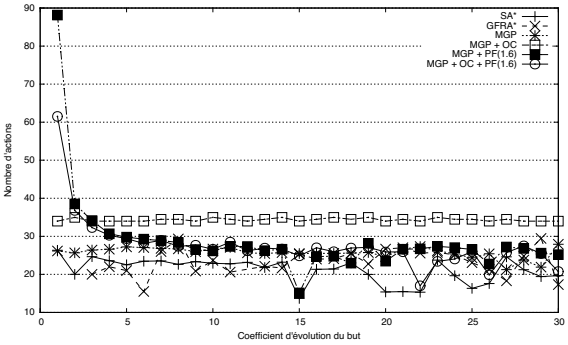
Les figures 2(a), 2(b) et 2(c) présentent respectivement les résultats obtenus en termes de pourcentages de succès, temps de recherche et longueurs des plans trou-



(a) Pourcentage de succès en fonction du coefficient d'évolution du but.



(b) Temps de recherche en fonction du coefficient d'évolution du but.



(c) Longueurs des plans en fonction du coefficient d'évolution du but.

Figure 1. *Analyse comparée des différentes approches sur le problème 20 de Blockworld.*

	Temps			Longeurs		
	Moy.	Min.	Max.	Moy.	Min.	Max
SA*	2,52	1,00	5,44	0,30	0,00	0,18
GFRA*	2,07	0,00	5,67	0,33	0,00	0,69
MGP	1,56	0,88	3,10	0,21	0,08	0,56
MGP+OC	1,23	0,23	3,08	0,02	0,00	0,09
MGP+PF	1,86	0,00	3,45	0,22	0,00	0,50
MGP+OC+PF	1,92	0,00	4,10	0,24	0,00	0,61

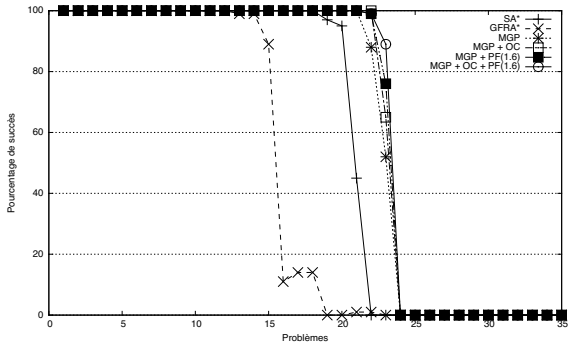
Tableau 2. *Coefficients de variation moyens, minimums et maximums calculés sur le domaine Blockworld.*

vés sur tous les problèmes du domaine Blockworld. Les problèmes sont classés par ordre de difficulté en fonction du nombre d'états qu'ils peuvent générer. Pour cette série d'expériences, le coefficient d'évolution du but a été fixé à 5. Les autres paramètres restent identiques : chaque expérience est répétée 100 fois et 60 secondes sont allouées pour chaque expérience. Les coefficients de variation moyens, minimums et maximums des résulats présentés sont donnés à titre indicatif par le tableau 2.

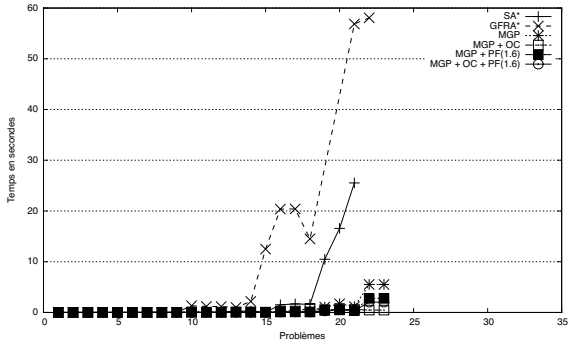
En ce qui concerne le pourcentage de succès, nous retrouvons les résultats obtenus sur le problème 20. GFRA* est largement distancé. Son poucentage de succès chute de manière drastique dès le problème 16 à 20%. Puis vient ensuite SA* qui atteint un peu plus des 40% au problème 21. Pour finir, viennent les différentes versions de MGP dont les pourcentages de succès commencent à chuter à partir des problèmes 22 et 23. On peut toutefois noter un léger avantage à MGP+OC+PF qui obtient près de 90% de succès sur le problème 24 contre un peu moins de 80% pour MGP+PF, un peu moins de 70% pour MGP+OC et 50% pour MGP. Étant donné que le temps de recherche est fixe, il arrive un moment où il n'est plus possible de résoudre les problèmes dans le temps imparti. Dans le cadre de notre expérimentation, cette limite se situe au niveau de la plage de problèmes de 22 à 24.

Ces résultats se retrouvent également sur les temps de recherche. Les différentes variantes de MGP surclassent SA* et GFRA* avec des temps de recherche inférieurs à 5 secondes. On retrouve ici, comme pour le problème 20, un léger avantage pour MGP+OC.

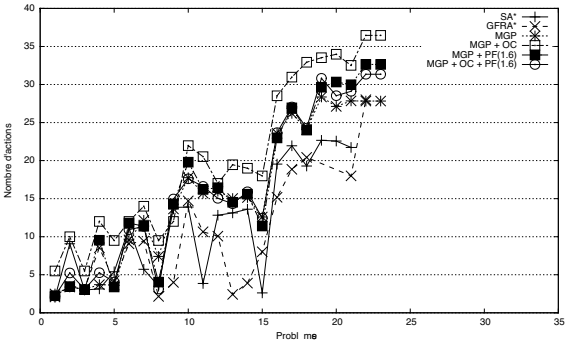
Pour terminer, en termes de taille de plans, les longueurs des plans trouvés sont assez semblables. Toutefois, les résultats obtenus sur l'ensemble du domaine confirment une nouvelle fois les résultats obtenus sur le problème 20. SA* et GFRA* trouvent des plans plus courts. Viennent ensuite les variantes de MGP avec un avantage pour MGP, et MGP+OC+PF.



(a) Pourcentage de succès en fonction de la difficulté des problèmes.



(b) Temps de recherche en fonction de la difficulté des problèmes.



(c) Longueurs des plans en fonction de la difficulté des problèmes.

Figure 2. *Analyse comparée des différentes approches sur le domaine Blockworld.*

4.4. *Impact du coefficient de retardement et de l'heuristique pondérée*

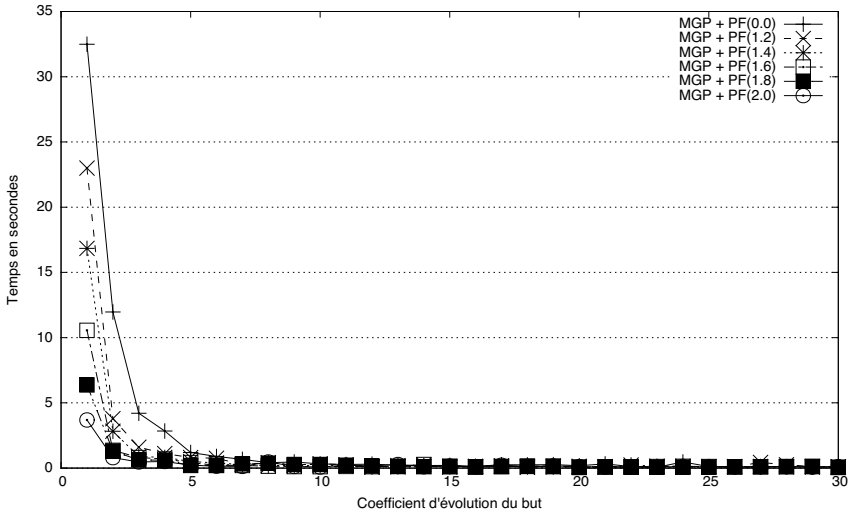
Dans cette partie, nous évaluons l'impact du coefficient de retardement ainsi que de la pondération de l'heuristique sur le meilleur algorithme obtenu précédemment, i.e., MGP+OC+PF. Les expériences suivantes ont été réalisées sur le même problème (Blockworld problème 20) pour permettre une comparaison des résultats. Étant donné que MGP+OC+PF a un pourcentage de succès proche de 100% nous ne présentons dans ce qui suit que les résultats en termes de temps de recherche et de longueur de plans.

4.4.1. *Impact du coefficient de retardement*

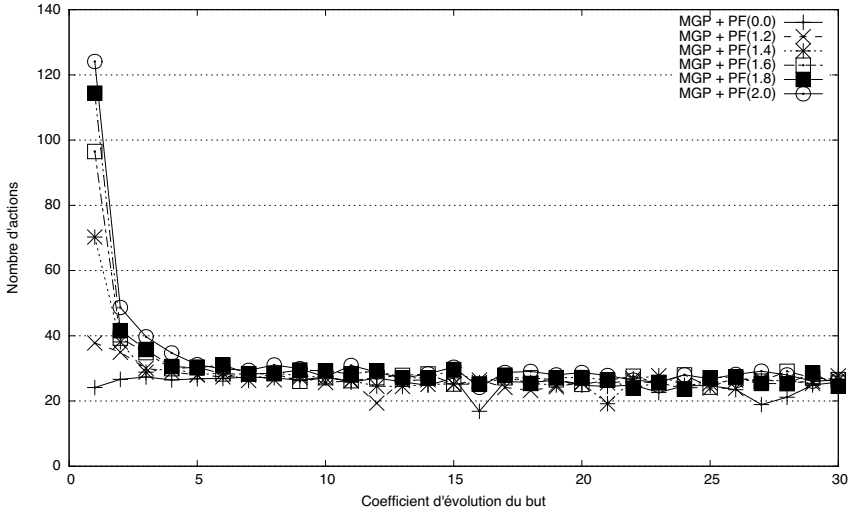
Les figures 3(a), 3(a), 4(a) et 4(b) présentent respectivement le temps de recherche et la longueur des plans obtenus en fonction du coefficient d'évolution du but pour les variantes de MGP, MGP+PF et MGP+OC+PF. Nous pouvons faire quatre observations. Premièrement, nous pouvons constater que le coefficient de retardement améliore de manière significative les performances de MGP. Par exemple, MGP+OC+PF (cf. figure 4(a)) avec un coefficient de retardement de 2 est 6 fois plus rapide que MGP avec un coefficient de retardement de 0 lorsque l'évolution du but est très rapide ($g_r = 1$). Deuxièmement, on peut constater que le temps de recherche ainsi que la longueur des plans convergent rapidement vers une même valeur et ce quelle que soit par la suite la valeur de g_r. Par exemple, avec un coefficient $g_r \geq 4$, le coefficient de retardement n'a quasiment plus d'impact sur le temps de recherche et sur la longueur des plans. Troisièmement, on constate qu'une augmentation du coefficient de retardement implique également une augmentation de la longueur des plans trouvés mais réduit le temps de recherche. Par conséquent, le coefficient de retardement doit être choisi de manière à être un compromis entre la longueur des plans trouvés et temps de recherche. Quatrièment, on retrouve dans ces courbes les meilleures performances de MGP+OC+PF relativement à MGP+PF en terme de temps de recherche. Toutefois, cet avantage a tendance à s'atténuer avec l'augmentation du coefficient de retardement.

4.4.2. *Impact de l'heuristique pondérée*

La figure 5 présente respectivement le temps de recherche et la longueur des plans obtenus en fonction de la pondération de l'heuristique des différentes variantes de MGP. On constate que la pondération de l'heuristique w améliore, comme le coefficient de retardement, de manière significative les performances de MGP (MGP est 4 fois plus rapide avec $w = 2.0$ qu'avec $w = 1.0$ pour un coefficient d'évolution du but $g_r = 1$). L'augmentation est aussi sensible pour des plus grandes valeurs de g_r même si nous ne le voyons pas au travers des expériences présentées ici. En outre, l'impact de w sur la longueur des plans obtenus n'est pas significatif : quelle que soit la pondération de l'heuristique, la longueur des plans est comparable pour une valeur de coefficient d'évolution de but donné. En d'autres termes, la pondération de l'heuristique augmente les performances de MGP comme on pouvait s'y attendre, mais ne détériore pas de manière importante la qualité des plans trouvés. En effet, habituellement, l'utilisation d'une heuristique pondérée détériore bien plus la qualité des plans

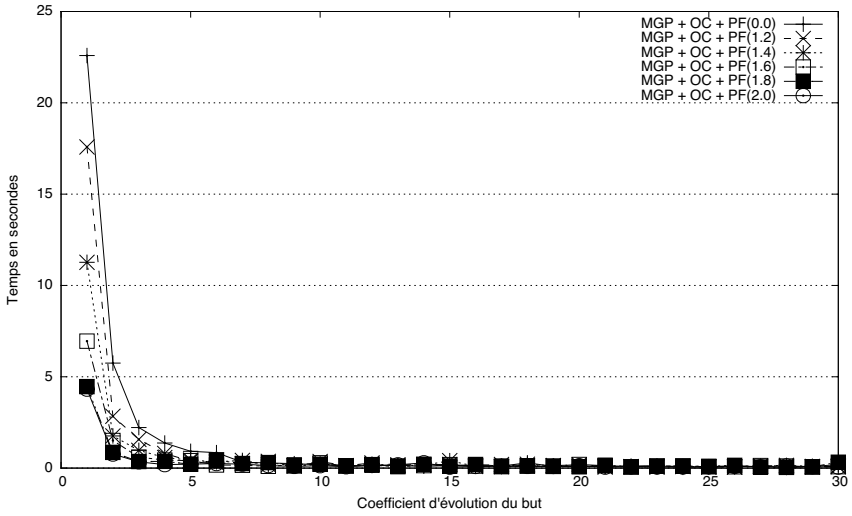


(a) Temps de recherche en fonction du coefficient de retardement et du coefficient d'évolution du but.

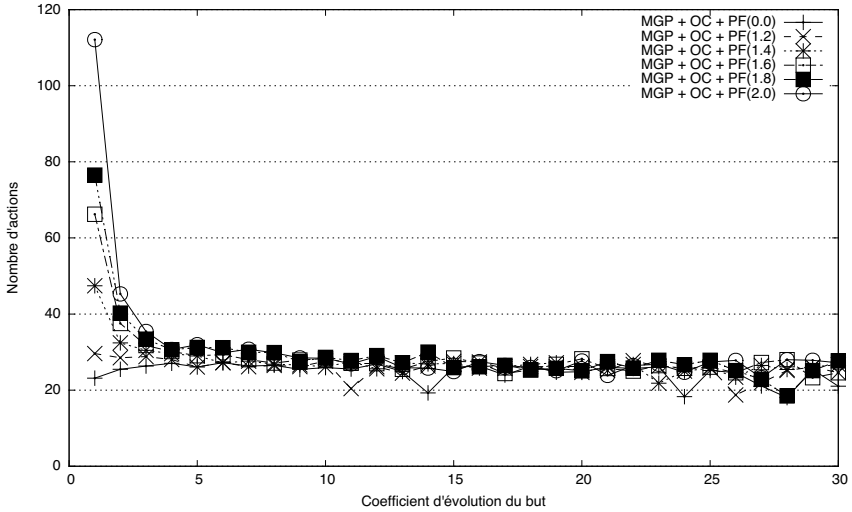


(b) Longueur des plans en fonction du coefficient de retardement et du coefficient d'évolution du but.

Figure 3. *Impact du coefficient de retardement sur le problème 20 de Blockworld de MGP+PF.*



(a) Temps de recherche en fonction du coefficient de retardement et du coefficient d'évolution du but.



(b) Longueur des plans en fonction du coefficient de retardement et du coefficient d'évolution du but.

Figure 4. *Impact du coefficient de retardement sur le problème 20 de Blockworld de MGP+OC+PF.*

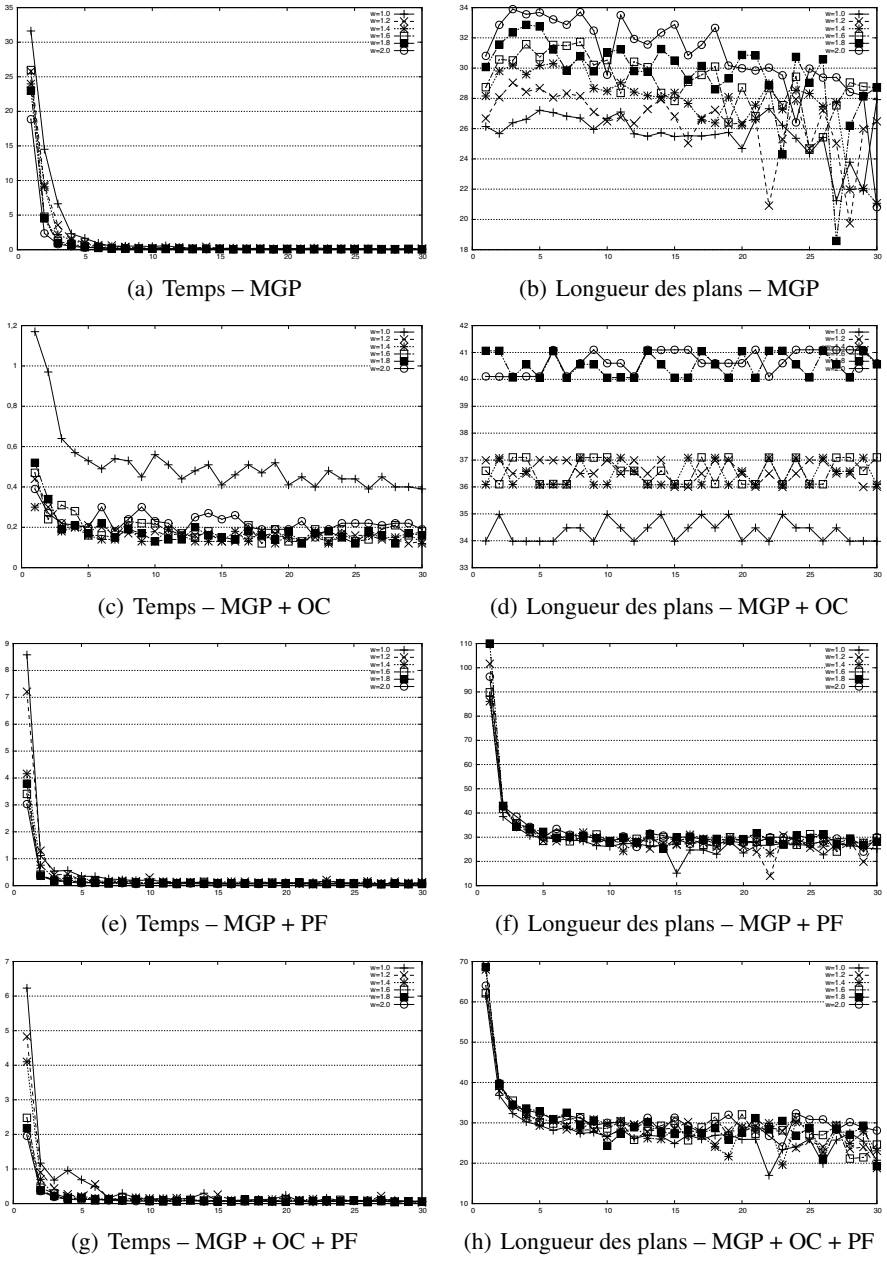
obtenus. Ceci peut s'expliquer par le fait que dans le cadre de notre approche, il est plus important de trouver un plan solution pour donner la direction à suivre pour atteindre le but qui évolue au cours du temps, que de trouver systématiquement un plan solution optimal.

4.5. *Vue d'ensemble des performances sur différents problèmes*

Dans cette partie, nous présentons une vue d'ensemble des performances des différents algorithmes sur plusieurs problèmes issus des compétitions IPC : le tableau 3 donne les temps de recherche obtenus, le tableau 4 les pourcentages de succès et le tableau 5 les longueurs des plans trouvés. Comme précédemment, chaque expérience a été répétée 100 fois pour obtenir des résultats statistiquement représentatifs. À chaque expérience, un temps CPU de 60 secondes et un maximum de 4Go de mémoire vive ont été alloués. Le coefficient de retardement a été fixé à 1.6. Pour tous les tests, nous avons également fixé le coefficient de changement de but à 100. Contrairement au domaine simple Blockworld, nous avons choisi ici un coefficient de changement de but plus grand pour aborder des problèmes plus complexes. Nous avons lancé l'expérimentation sur tous les problèmes des domaines décrits dans les tableaux en suivant la démarche expérimentale présentée en détail sur le domaine Blockworld (cf. § 4.3.2). Toutefois, nous ne montrons ici que les résultats obtenus sur les problèmes pour lesquels on observe une inflexion des performances.

En terme de temps de recherche, on retrouve la plupart des résultats obtenus sur le problème Blockworld 20 et le domaine Blockworld : MGP+OC+PF est largement plus rapide que les autres algorithmes. Toutefois, on constate que les bonnes performances observées par MGP+OC dans le domaine blockworld ne se vérifient pas sur les autres domaines testés. De plus, MGP+OC+PF surclasse les variantes de MGP avec une seule stratégie de retardement. De même, MGP+OC+PF surclasse les autres algorithmes en terme de pourcentage de succès. Cependant, on peut constater que MGP sans stratégie de retardement échoue sur certains problèmes (Elevator P35 and Freecell P26) que SA* résout, même si d'une manière générale, la version naïve de MGP reste meilleure que SA* et GFRA*. Finalement, en terme de longueur de plans, toutes les approches trouvent des plans de longueurs comparables ce qui ne constitue par conséquent pas un critère discriminant entre les différentes approches.

Pour synthétiser ces résultats, OC et PF améliorent individuellement les performances de MGP qui surclasse SA* et GFRA*. Les deux stratégies de retardement OC+PF couplées donnent de meilleurs résultats que lorsqu'elles sont utilisées séparément. Les résultats obtenus sur différents domaines et problèmes confirment que MGP+OC+PF est le meilleur algorithme et que GFRA* est largement dépassé (GFRA* met à jour la fonction heuristique de tous les états de l'arbre de recherche avant chaque nouvelle recherche ce qui le pénalise comparativement aux autres algorithmes).



(a) Temps – MGP (b) Longueur des plans – MGP

(c) Temps – MGP + OC (d) Longueur des plans – MGP + OC

(e) Temps – MGP + PF (f) Longueur des plans – MGP + PF

(g) Temps – MGP + OC + PF (h) Longueur des plans – MGP + OC + PF

Figure 5. *Impact de la pondération de l'heuristique sur le problème 20 de Blockworld.*

Problème	SA*	GFRA*	MGP	OC	PF	OC+PF
airport p16	3,38	0,91	0,20	0,18	0,17	0,12
airport p19	-	13,48	0,40	0,36	0,37	0,21
depot p03	8,12	-	2,40	1,28	1,67	0,78
depot p07	-	-	7,41	6,42	1,73	1,13
driverlog p03	0,03	0,02	0,33	0,22	0,18	0,17
driverlog p06	13,57	-	4,32	5,50	5,53	4,35
elevator p30	24,85	23,69	29,32	3,23	2,22	1,46
elevator p35	23,04	-	-	44,60	35,31	20,80
freecell p20	10,94	-	-	4,72	6,20	3,65
freecell p26	48,76	-	56,61	26,70	32,12	24,20
openstack p06	55,58	55,80	55,20	54,03	48,69	36,30
openstack p07	54,55	57,51	57,35	50,33	46,13	43,32
pipeworld p04	0,50	17,73	1,68	0,49	0,55	0,48
pipeworld p08	-	-	18,02	13,89	13,70	12,51
pathway p02	15,60	9,52	6,92	3,53	2,83	1,74
pathway p04	-	-	-	-	8,05	4,39
rover p03	23,35	14,17	3,72	2,85	2,15	1,87
rover p07	-	-	-	23,51	-	22,54
satellite p03	-	17,26	9,36	3,67	4,56	3,18
satellite p06	-	-	-	-	-	8,97

Tableau 3. *Comparaison du temps de recherche.*

5. Conclusion

Comme nous l'avons souligné en introduction, une des clés de la robotique personnelle se trouve, de notre point de vue, dans les Interactions Homme-Robot et la capacité des robots à adapter en permanence leurs décisions dans des environnements rendus dynamiques par les activités humaines : faisant constamment face aux événements qui perturbent leurs plans, les robots doivent entrelacer planification et exécution.

Afin d'aborder ce problème, nous avons proposé dans cet article une approche pour la planification de tâches en boucle fermée, appelée MGP, qui adapte continuellement son plan aux évolutions successives du but. MGP s'appuie sur une recherche heuristique pondérée incrémentale de type A* et permet d'entrelacer planification et exécution. Afin de limiter le nombre de recherches effectuées pour prendre en compte les changements dynamiques du but au cours du temps, MGP retarde autant que possible le déclenchement de nouvelles recherches. Pour cela, MGP utilise deux stratégies de retardement : *Open Check* (OC) qui vérifie si le but est encore présent dans l'arbre de recherche et *Plan Follow* (PF) qui estime s'il est préférable d'exécuter les actions du plan courant pour se rapprocher du nouveau but plutôt que de relancer une nouvelle recherche. De plus, MGP utilise une stratégie conservatrice et efficace de mise à jour

Problème	SA*	GFRA*	MGP	OC	PF	OC+PF
airport p16	97	100	100	100	100	100
airport p19	-	72	81	100	100	100
depot p03	10	-	99	30	60	100
depot p07	-	-	33	12	14	88
driverlog p03	100	100	100	100	100	100
driverlog p06	1	-	1	56	88	98
elevator p30	69	99	91	100	100	100
elevator p35	1	-	-	19	5	55
freecell p20	39	-	-	99	100	100
freecell p26	56	-	1	100	100	100
openstack p06	77	48	62	70	96	99
openstack p07	44	15	21	100	99	99
pipeworld p04	99	7	99	99	98	100
pipeworld p08	-	-	48	70	60	76
pathway p02	100	100	100	100	100	100
pathway p04	-	-	-	-	22	48
rover p03	48	8	99	99	94	99
rover p07	-	-	-	32	-	52
satellite p03	-	1	4	16	15	52
satellite p06	-	-	-	-	-	28

Tableau 4. *Comparaison du pourcentage de succès.*

incrémentale de l'arbre de recherche lui permettant de réduire le nombre d'appels à la fonction heuristique et d'accélérer la recherche d'un plan solution.

Nous avons montré expérimentalement que MGP surclasse l'approche naïve SA* et l'algorithme de référence GFRA*. Nous avons également montré que : (1) la combinaison des deux stratégies de retardement OC+PF donne de meilleurs résultats que chacune considérée individuellement ; (2) le coefficient de retardement qui permet de paramétrer la stratégie PF doit être un compromis entre la longueur des plans trouvés et le temps de recherche et (3) la pondération de l'heuristique améliore de manière significative les performances de MGP sans trop détériorer la qualité des plans trouvés.

Ce travail ouvre la voie à plusieurs pistes de recherche. Dans un premier temps, il serait intéressant de complexifier notre cadre expérimental. Nous supposons en effet que le but évolue de manière aléatoire en fonction du temps qui s'écoule. Même si cette hypothèse de travail est plus réaliste que celle utilisée classiquement pour comparer des algorithmes de type MTS, elle reste restrictive puisque le but évolue de proche en proche sans stratégie particulière. Pour relaxer cette hypothèse, nous envisageons de guider l'évolution du but en utilisant une fonction heuristique. Cette approche nous permettrait alors de caractériser plus précisément la difficulté d'atteindre un but et donc les problèmes à résoudre. De plus, un prolongement naturel de ce tra-

Problème	SA*	GFRA*	MGP	OC	PF	OC+PF
airport p16	80,98	80,98	79,15	81,25	80,81	81,12
airport p19	-	91,98	90,12	91,52	90,62	91,14
depot p03	20,80	-	21,73	21,67	25,18	22,83
depot p07	-	-	25,24	23,08	26,00	24,74
driverlog p03	7,30	4,14	8,42	11,57	12,57	9,57
driverlog p06	12,00	-	14,00	15,00	11,23	16,91
elevator p30	29,20	27,88	27,33	28,42	27,74	28,80
elevator p35	34,00	-	-	33,05	32,20	32,18
freecell p20	29,97	-	-	29,99	29,99	29,99
freecell p26	37,02	-	38,00	37,01	37,01	37,01
openstack p06	50,41	51,13	51,28	50,68	50,06	50,54
openstack p07	51,12	52,14	50,76	51,25	50,72	51,02
pipeworld p04	3,21	11,86	7,61	9,19	10,20	8,12
pipeworld p08	-	-	18,21	21,09	19,76	21,02
pathway p02	27,18	26,39	26,27	26,93	37,02	40,76
pathway p04	-	-	-	-	34,92	35,12
rover p03	44,53	44,73	44,40	44,54	44,66	44,24
rover p07	-	-	-	43,20	-	43,50
satellite p03	-	42,00	26,00	24,69	32,12	25,80
satellite p06	-	-	-	-	-	26,00

Tableau 5. *Comparaison de la longueur des plans obtenus.*

vail consiste à l'étendre dans un cadre d'une planification en temps borné. Finalement, nous pensons qu'il serait également intéressant de tester notre approche sur des problèmes de type recherche de chemins pour montrer que le gain d'expressivité induit par le passage à une representation STRIPS ne nuit pas aux performances.

6. Bibliographie

[BON 99] BONET B., GEFFNER H., « Planning as Heuristic Search : New Results », *Proceedings of the European Conference on Planning*, 1999, p. 359–371.

[BON 01] BONET B., GEFFNER H., « Planning as heuristic search », *Artificial Intelligence*, vol. 129, n° 1–2, 2001, p. 5–33.

[COX 98] COX M. T., VELOSO M. M., « Goal Transformations in Continuous Planning », *Proceedings of the AAAI Fall Symposium on Distributed Continual Planning*, 1998.

[FIN 71] FINKE R., NILSSON N., « STRIPS : A new approach to the application of theorem proving to problem solving », *Artificial Intelligence*, vol. 3-4, n° 2, 1971, p. 189–208.

[FOR 04] FORLIZZI J., DISALVO C., GEMPERLE F., « Assistive Robotics and an Ecology of Elders Living Independently in Their Homes », *Human-Computer Interaction*, vol. 19, n° 1-2, 2004, p. 25-59.

[FOR 06] FORLIZZI J., « Service Robots in the Domestic Environment : A Study of the Roomba Vacuum in the Home », *In Proc. HRI '06*, Press, 2006, p. 258–265.

[FOX 06] FOX M., GEREVINI A., LONG D., SERINA I., « Plan stability : Replanning versus plan repair », *Proceedings of the International Conference on Planning and Scheduling*, 2006, p. 212–221.

[HAR 68] HART P., NILSSON N., RAPHAEL B., « A formal basis for the heuristic determination of minimum cost paths », *IEEE Transactions on Systems, Man and Cybernetics*, vol. 2, 1968, p. 100—107.

[HOF 01] HOFFMANN J., NEBEL B., « The FF Planning System : Fast Plan Generation Through Heuristic Search », *Journal of Artificial Intelligence Research*, vol. 14, n° 1, 2001, p. 253–302.

[ISH 91] ISHIDA T., KORF R., « Moving target search », *In Proceedings of the International Joint Conference on Artificial Intelligence*, 1991, p. 204–210.

[ISH 95] ISHIDA T., KORF R., « A realtime search for changing goals », *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, n° 17, 1995, p. 609–619.

[ISH 98] ISHIDA T., « Real-time search for autonomous agents and multiagent system », *Autonomous Agents and Multi-Agent Systems*, vol. 1, n° 2, 1998, p. 139–167.

[KAM 92] KAMBHAMPATI S., HENDLER J. A., « A validation-structure-based theory of plan modification and reuse », *Artificial Intelligence*, vol. 55, 1992, p. 193–258.

[KAN 07] KANDA T., SATO R., SAIWAKI N., ISHIGURO H., « A Two-Month Field Trial in an Elementary School for Long-Term Human-Robot Interaction », *Robotics, IEEE Transactions on*, vol. 23, n° 5, 2007, p. 962 -971.

[KOE 02] KOENIG S., LIKHACHEV M., « D* Lite », *Proceedings of the Association for the Advancement of Artificial Intelligence*, 2002, p. 476–483.

[KOE 05] KOENIG S., LIKHACHEV M., « Adaptive A* », *Proceedings of the International Conference on Autonomous Agent and Muti-Agent Systems*, 2005, p. 1311–1312.

[KOE 07] KOENIG S., LIKHACHEV M., SUN X., « Speeding up moving target search », *Proceedings of the International Conference on Autonomous Agent and Muti-Agent Systems*, 2007.

[KOR 90] KORF R., « Real-Time Heuristic Search », *Artificial Intelligence*, vol. 42, n° 2-3, 1990, p. 189–211.

[KRO 05] VAN DER KROGT R., DE WEERDT M., « Plan Repair as an Extension of Planning », *Proceedings of the International Conference on Planning and Scheduling*, 2005, p. 161–170.

[MEL 93] MELAX S., « New approaches to moving target search », *Proceedings of AAAI Falls Symp. Game Planning Learn*, 1993, p. 30–38.

[NEB 95] NEBEL B., KOEHLER J., « Plan Reuse versus Plan Generation : A Theoretical and Empirical Analysis », *Artificial Intelligence*, vol. 76, 1995, p. 427–454.

[POH 70] POHL I., « Heuristic search viewed as path finding in a graph », *Artificial Intelligence*, , 1970, p. 193—204.

[SAB 10] SABANOVIC S., « Robots in Society, Society in Robots - Mutual Shaping of Society and Technology as a Framework for Social Robot Design », *International Journal of Social Robotics*, vol. 2, n° 4, 2010, p. 439-450.

[STE 95] STENZ A., « The focused D* algorithm for real-time replanning », *In Proceedings of the International Joint Conference on Artificial Intelligence*, 1995, p. 1642–1659.

[SUN 08] SUN X., KOENIG S., YEOH W., « Generalized Adaptive A* », *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2008, p. 469–476.

[SUN 09a] SUN X., YEOH W., KOENIG S., « Efficient Incremental Search for Moving Target Search », *Proceedings of the International Joint Conference on Artificial Intelligence*, 2009, p. 615–620.

[SUN 09b] SUNG J., GRINTER R. E., CHRISTENSEN H. I., « Pimp My Roomba : designing for personalization », *in CHI'09. ACM*, 2009, p. 193–196.

[SUN 10a] SUN X., YEOH W., KOENIG S., « Generalized Fringe-Retrieving A* : Faster Moving Target Search on State Lattices », *Proceedings of the International Joint Conference on Autonous Agents and MultiAgent Systems*, 2010, p. 1081–1088.

[SUN 10b] SUN X., YEOH W., KOENIG S., « Moving Target D* Lite », *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, 2010, p. 67–74.

[TAN 06] TANAKA F., MOVELLAN J. R., FORTENBERRY B., AISAKA K., « Daily HRI evaluation at a classroom environment : reports from dance interaction experiments », *HRI*, 2006, p. 3-9.

[TAP 08] TAPUS A., MATARIC M., « User Personality Matching with a Hands-Off Robot for Post-stroke Rehabilitation Therapy », KHATIB O., KUMAR V., RUS D., Eds., *Experimental Robotics*, vol. 39 de *Springer Tracts in Advanced Robotics*, p. 165-175, Springer Berlin Heidelberg, 2008.

[VAN 11] VANNEUFVILLE M., « Planification de tâches adaptative », Master's thesis, Université Joseph Fourier, 2011.