Planning When Goals Change: A Moving Target Search Approach

Damien Pellier[1], Humbert Fiorino[1], and Marc Métivier[2]

[1] Laboratoire d'Informatique de Grenoble
110 avenue de la Chimie BP 38053 Grenoble
[2] Université Paris Descartes
45, rue des St Pères, 75006 Paris

Abstract. Devising intelligent robots or agents that interact with humans is a major challenge for artificial intelligence. In such contexts, agents must constantly adapt their decisions according to human activities and modify their goals. In this paper, we tackle this problem by introducing a novel planning approach, called Moving Goal Planning (MGP), to adapt plans to goal evolutions. This planning algorithm 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 triggering new searches when the goal changes over time. 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 actions of the current plan brings MGP closer to the new goal. Moreover, MGP uses a parsimonious strategy to update 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.

1 Introduction

Service robots performing simple domestic tasks begin to enter our daily lives. Still, many breakthroughs must be made in navigation, perception and sensors, energy management, mechatronics etc. But whatever the progresses made in these areas, one prominent issue is robot usability and their ability to adapt their decisions according to human activities. In such contexts, robots must constantly cope with events that modify their goals and disrupt their plans.

In order to tackle this problem, we propose in this paper a new planning algorithm that interleaves on-line planning and execution, called MGP (Moving Goal Planning) and built on the MTS (Moving Target Search) search strategy. MTS algorithms are search algorithms designed for path-finding and for real-time moving targets (an agent, "the hunter", follows a moving target, "the prey") interleaving path-finding toward the prey and hunter displacements. MTS algorithms are based on heuristic search (distance calculation) and, to our knowledge, have not been used for task planning. Thus, we propose to capitalize on

Y. Demazeau et al. (Eds.): PAAMS 2014, LNAI 8473, pp. 231–243, 2014.

recent advances in these two areas to devise a new and efficient planning algorithm able to adapt its plan when its goal changes over time as a new approach for continual planning [1].

The rest of the paper is organized as follows: Section 2 gives the state of the art; Section 3 formally introduces the moving goal planning problem and describes our algorithm; Section 4 presents the algorithms evaluation; Section 5 concludes and proposes possible avenues for future extensions.

2 Related Work

In task planning, the design of agents evolving in dynamic environments and able to adapt the execution of their current plan to goal changes is mainly studied in two different ways: rebuilding a plan from scratch or repairing it so that it can be executed in the new context. Although in theory both approaches are equally expensive in the worst case [2], experimental results show that plan repair is more efficient than replanning from scratch [3]. Preserving plan stability is another argument in favor of the plan repair strategy [4].

In path-finding, the agent's adaptation to dynamic environments is also a challenging issue especially in computer games to solve the Moving Target Search (MTS) with respect to real-time responses, large-scale search spaces and limited computation resources. In essence, a MTS algorithm interleaves path-finding and action execution for a "hunter" agent chasing a moving target – the "prey" – over a large map or grid. Since the pioneering works of Ishida [5], MTS approaches fall into two categories according to the strategy used to reuse the information collected in past searches.

The first strategy consists in using a heuristic to guide the search and learn shortest path distances between pairs of locations on a map. At each search, the heuristic is more informative and the search is sped up. The original MTS algorithm was an adaptation of the Learning Real-Time A* algorithm (LRTA*) [6] for a moving target. This approach was shown to be complete in turns based settings when the target periodically skips moves but it is subject to heuristic depressions and lost of information when the target moves [7]. Currently, the state-of-the-art algorithms with this first strategy are variants of the AA* algorithm [8]: MTAA* [9] and GAA* [10]. All these algorithms must use admissible heuristics to ensure their soundness and completeness.

The second strategy consists in reusing incrementally the search tree between two successive searches. The first algorithms based on this strategy are D* [11] and its successors. These algorithms were devised for replanning in unknown or changing environments and are both based on backward chaining. They perform correctly when the environment does not change much during the search. Otherwise, their performances are bypassed by simple successive calls to A* every time the target moves [10]. As for FRA* [12], changes in the environment are not taken into account but it performs properly when the target moves over time. FRA* is based on the A* forward search. Every time the target moves, FRA* adapts quickly the search tree and recalls A* on the new search tree. FRA* is currently the most efficient MTS algorithm. However, the adaption of the search

tree is widely dependent on the grid representation of the environment. In order to apply FRA* on more generic environments, a variant called GFRA* [13] (Generalized Fringe-Retrieving A*) has been recently proposed. Contrary to FRA*, GFRA* uses arbitrary graphs, including the state lattices used for Unmanned Ground Vehicles navigation. Finally, Sun at al. [14] proposes an algorithm called I-ARA*, which is the first incremental anytime search algorithm for moving target search. I-ARA* operates like repeated ARA* [15], except that it also uses incremental search as used in GFRA* to speed up the search by adapting the tree search and by reusing the information from the previous search.

To summarize, algorithms based on heuristic guidance (the first strategy) are more appropriate for environment changes rather than goal changes. On the other hand, algorithms based on incremental search (the second strategy) are efficient with goal changes but operate with less generic environments. Among them, GFRA* seems the more interesting to use for task planning mainly for two reasons: (1) it is based on a heuristic forward search as the most powerful state-of-the-art planners and (2) it can work with non-admissible heuristic function as it is often the case in task planning.

3 Moving Goal Planning

3.1 Problem Formulation

We address sequential planning in the propositional STRIPS framework [16]. All sets are finite. A *state* s is a set of logical propositions. An *action* a is a tuple $a = (pre(a), add(a), del(a))$ where $pre(a)$ are the action's *preconditions*, $add(a)$ and $del(a)$ are its positive and negative *effects*. The state s' is reached from s by applying an action a according to the transition function γ: $s' = \gamma(s, a) = (s - del(a)) \cup add(a)$ if $pre(a) \subseteq s$, undefined otherwise.

The application of a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ to a state s is recursively defined as $\gamma(s, \langle a_1, \ldots, a_n \rangle) = \gamma(\gamma(s, a_1), \langle a_2, \ldots, a_n \rangle)$. A Moving-Goal Pursuit problem is a tuple (A, s_t, g_t): at a given timestamp t, an agent is in a state s_t; g_t is its current goal (s_t and g_t are sets of propositions) and A is the set of actions that it can perform. It executes actions in order to reach its goal and the goal can change at any time. The agent has no information on how the goal changes over time. A *plan* is a sequence of actions $\pi_t = \langle a_1, \ldots, a_n \rangle$ ($a_i \in A$) such that the goal $g_t \subseteq \gamma(s_t, \pi_t)$ and g_t is *reachable* if such a plan exists. A goal state is a state s such that $g_t \subseteq s$. At a given time t, a Moving-Goal Pursuit problem is solved if $g_t \subseteq s_t$: the agent has reached its goal.

3.2 Algorithm

The MGP pseudocode is given in Algo. 1. MGP takes as input a Moving Goal Pursuit problem (A, s_0, g_0). The variables g and s denote respectively the current goal and the current state set initially to g_0 and s_0 (i is the search iteration counter).

MGP iterates a search procedure (line 2) as long as the current goal has not been reached. The Search procedure is detailed in section 3.3. This procedure

builds a search tree whose nodes are states and edges are actions. The search procedure fails if the current goal has not been reached and MGP fails (line 3). This is the case when the planning problem is unsolvable. Otherwise MGP postpones as much as possible triggering a new search and expansion of the search tree (while-loop line 4) and it extracts a plan from the search tree (lines 5). The while-loop ends when the goal evolves out of the search tree, i.e., none of the nodes is a goal state (procedure OpenCheck), or when the goal *significantly* changes (procedure PlanFollow). These two procedures are detailed in section 3.3.

As long as the goal is reachable with the extracted plan or does not significantly change, MGP executes the actions of this plan and update its current state (line 7). The goal changes are provided by the procedure UpdateGoal (line 8). Then, if MGP reaches its current goal, it returns success (line 9). Otherwise, MGP reduces its search tree to the subtree whose root is the current state s (DeleteStatesOutOfTree, line 10). If the new goal is in this subtree and a new search can be postponed, MGP extracts a new plan and executes its actions to reach the new goal. Otherwise, MGP updates the heuristic values of the search tree nodes according to the new goal (line 11) (UpdateSearchTree procedure detailed in the next section) and finally increments its search iteration counter (line 12) and expands its search tree (line 3).

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) fails then return Failure
4      while OpenCheck(g) and PlanFollow(s, g) do
5          Extract a solution plan π from the search tree
6          while (g ⊈ s and g ⊆ γ(s, π)) or (PlanFollow(s, g) and π ≠ ∅) do
7              a ← get and remove the first action of π, execute a, s ← γ(s, a)
8              g ← UpdateGoal(g) ;; simulate goals change
9          if g ⊆ s then return Success
10         DeleteStatesOutOfTree(s)
11     UpdateSearchTree(s, g, i)
12     i ← i + 1
```

3.3 Implementation

In this section, we describe the MGP implementation. The search tree is represented by two lists denoted OPEN and CLOSED: the OPEN list contains the pending states of the search and the CLOSED list contains the explored states.

Weighted A as Search Strategy.* Contrary to GFRA* that uses A* as basic search algorithm, MGP uses the Weighted-A* search strategy. This variant of A* overestimates the cost of the heuristic value according to a ratio w. The evaluation function $f(s)$ for a state s is $f(s) = g(s) + w \times h(s)$ where $g(s)$ is the

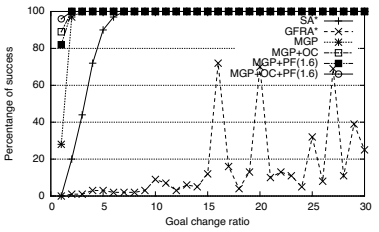
cost to reach s from the initial state s_0 and $h(s)$ the estimated cost from s to the goal g. The greater w, the greater is the weight of the heuristic in the guidance of the search. Usually, using Weighted A* with an informative and admissible heuristic speeds up the search but breaks up the optimality of the solution plans. This approach is relevant because it is more important to find quickly a good solution than to find an optimal solution that will become outdated after a goal change. We show that using Weighted-A* instead of A* significantly improves the MGP performances (see section 4). Moreover, Weighted-A* does not impair the soundness and the completeness of MGP. The Weighted-A* algorithm used in our approach is a modified version of the classical algorithm. It takes as input a search problem (A, s, g), a ratio w and the search iteration counter i. For each state, it maintains three values: the g-value and the h-value of the state, and the parent pointer $parent(s)$ that points to the parent state of s in the search tree. At the first procedure call, the OPEN and CLOSED lists hold the initial state of the search problem such as $g(s) = 0$ and $h(s) = H(s, g)$ where H is the heuristic function that estimates the cost from the initial state s to the goal g.

Search Delaying Strategies. In order to limit the number of search iterations and to speed up the algorithm performances, MGP delays as much as possible starting new searches when the goal changes. MGP uses two novel and different strategies.

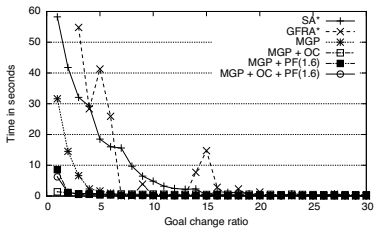
Open Check (OC). MGP checks if the new goal is still in the search tree. In that case, a new plan can be extracted in the current search tree. Contrary to GFRA* that only checks the states in the CLOSED list, MGP also checks the pending states in the OPEN list. This checkout avoids useless searches and readjustments of the search tree.

Plan Follow (PF). MGP estimates whether executing the actions of the current plan brings it closer to the new goal. Each time the goal changes and before starting a new search, MGP evaluates if the new goal is close to the previous one and determines if the current plan can still be used. This test is based on the heuristic function and the computation of an inequality between the current state s, the previous goal p and the new goal g: $H(s, g) \times c > H(s, p) + H(p, g)$ where c is called the delay ratio. MGP follows the current plan while the inequality is true, i.e., until it estimates that a straightforward plan from the current state s to the new goal g is better than achieving the previous goal and then finding a new plan from the previous goal to the new goal. Values of $c > 1$ allow us to adjust the delay before a new search. As searches are expensive, delaying them speeds up the algorithm but alters plan quality (see section 4).

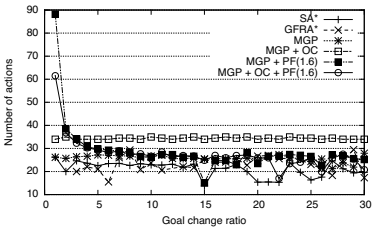
Incremental Updates of the Search Tree. MGP incrementally updates the search tree at each new search (see Algo. 1 line 14). Contrary to GFRA* that updates the heuristic value of all the states of the search tree with respect to the new goal, MGP uses a parsimonious strategy to reduce the number of calls to the heuristic function. To this purpose, MGP clears the OPEN list, adds the current state and updates its h-value by calling the heuristic function H. To indicate



(a) Percentage of success with respect to the goal change ratio.



(b) Search time with respect to the goal change ratio.



(c) Plan length with respect to the goal change ratio.

Fig. 1. Global analysis of Blockworld problem 20

that the heuristic value of the current state is up-to-date, MGP sets its iteration value to the MGP iteration counter. During the search, each time a state in the CLOSED list is encountered with an iteration value smaller than the iteration counter, it is added into the OPEN list and its h-value is updated. This strategy has two advantages: (1) it reduces the number of states generated by reusing the states in the CLOSED list during a search and (2) it reduces significantly the time needed to update the heuristic values by limiting the updates to the states explored during the new search.

4 Experiments and Evaluation

The objective of these experiments is to evaluate the performances of MGP with respect to the different search delaying strategies: Open Check (OC) and Plan Follow (PF). MGP is compared with the state-of-the-art algorithm GFRA* and the naive approach Successive A* (SA*) that consists in calling A* each time the goal changes. The benchmarks used for the evaluation are taken from the International Planning Competition (IPC). We use the non-admissible heuristic function of FF [17] to drive the search. We evaluate six algorithms: Successive A* (SA*), GFRA*, MGP without search delaying strategy (MGP), MGP with Open Check (MGP+OC), MGP with Plan Follow (MGP+PF) and MGP with both strategies (MGP+OC+PF).

4.1 Simulation of the Goal Changes

Classically, MTS algorithms assume that the moving target, the "prey", always follows the shortest path from its current position to a randomly selected and unblocked position. Each time the prey reaches this position, a new position is randomly selected and so on. Every n moves, the prey remains idle, allowing the "hunter" to catch it. As this approach is not transposable to task planning, we change the goal by randomly applying an action to the current goal state. This process is repeated many times to make the goal more difficult to reach. The new goal is always reachable from the current goal, but it is not guaranteed that MGP will reach it since it may evolve so quickly that MGP cannot reach it. This simulation of goal changes is more challenging than the one classically used to compare MTS algorithms: in our experiments, the goal does not change as a function of the executed actions but in a real time manner as a function of the time needed to find a solution plan. Thus, the more an algorithm takes time to find a solution, the more the goal evolves and becomes difficult to reach. To parametrize the goal evolution, a counter t is incremented every time a state is explored during the Weighted-A* search and every time the heuristic function is called to update a state h-value. These two procedures are the most time consuming. The number n of actions applied to the goal state is computed as follows: $n = (t - t_p)/g_r$ where t_p denotes the previous value of the counter t and g_r the goal change ratio. Hence, g_r allows us to adjust the swiftness of goal changes: small goal ratios mean fast goal changes and high goal ratios slow goal changes.

4.2 Experiment Framework

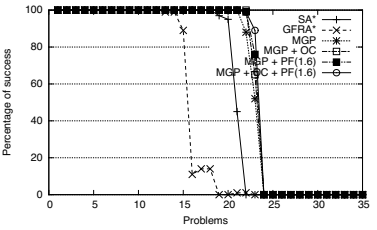
Each algorithm was tested 100 times on a planning problem with a given goal change ratio. Each test was conducted on an Intel Xeon 4 Core (2.0Ghz) with a maximum of 4 Gbytes of memory and was allocated a CPU time of 60 seconds.

In a first stage, the algorithms are tested with the IPC-2 Blockworld benchmark in order to measure their respective performances with respect to the goal change ratio. In a second stage, we have tested the impact of the delay ratio and of the heuristic weight in the performances of the best algorithm observed at the first stage. In a third stage, we have tested the algorithms on a large set of planning domains and problems.

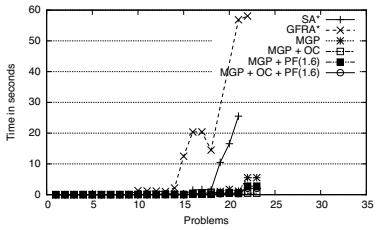
The performances presented are: (1) the success percentage, i.e., the number of times the algorithm succeed to reach the goal, (2) the search time and (3) the plan length.

4.3 Algorithms Comparisons on Blockworld

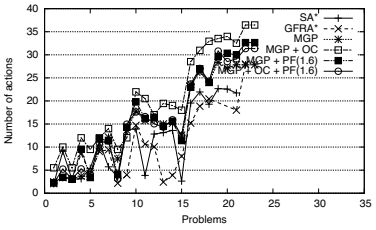
In this section, we present a comparison of the six algorithms SA*, GFRA*, MGP, MGP+OC, MGP+PF and MGP+OC+PF on the IPC-2 Blockworld P20 with respect to the goal change ratio and we give an overview of their respective performances on Blockworld domain. The delay ratio is arbitrary set to 1.6



(a) Percentage of success with respect to the problems.



(b) Search time with respect to the problems.



(c) Plan length with respect to the problems.

Fig. 2. Global analysis of Blockworld domain

for MGP+PF and MGP+OC+PF and the heuristic weight is set to 1 for all algorithms. Both parameteres are studied section 4.4.

Study in the Blockworld P20. Figures 1(a), 1(b), 1(c) show the results obtained in terms of percentage of success, search time and plan length. Regarding the percentage of success, the best algorithm is MGP+OC+PF. Even with a goal change ratio $g_r = 1$, MGP+OC+PF has a success rate above 95%. The other search delaying strategies are less efficient but obtain more than 80% of success. The naive approach SA* needs a goal change ratio 5 times bigger to obtain the same percentage of success. GFRA* does not reach this percentage of success even with a goal change ratio 30 times bigger. Finally, GFRA* is outclassed by the other algorithms. In terms of search time, the best algorithm is MGP+OC. Then, we have MGP+OC+PF, MGP+PF and MGP. Finally, we have SA* and far away GFRA*. MGP+OC is very efficient with Blockworld P20. Indeed, the new goal is often contained in the open list of A*. Moreover, OC+PF enhances significatively the naive version of MGP wrt search time but not plan length. This can be explained by the optimistic behaviour of the PF variants of MGP: when they make bad choices, the cost to pay is higher. In terms of plan length, MGP and its variants produce longer plans than SA* and GFRA*. Two reasons explain this difference. First, the OC strategy checks if the new goal was already explored and then extracts directly a new plan from the search tree. This cannot guarantee that the extracted plan is the shortest because the search tree was not built for this goal. Second, the PF strategy, as shown by the figure 2(c), tends to increase plan length. However, plan length narrows with the increase of the goal

change ratio and it is largely compensated by better search times and percentages of success.

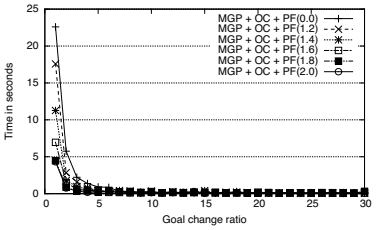
Overview of the Blockworld Domain. Figures 2(a), 2(b) and 2(c) show respectively the results obtained in terms of percentages of success, search time and plan length for all the problems of the blockworld benchmark. The problems are ordered with respect to their complexity. In these experiments, the goal change ratio is set to 5 to convert a large range of problems. The other parameters are unchanged: each experiment is repeated 100 times and 60 seconds is allocated for each experiment. In terms of percentage of success, the results are identical to Blockworld P20. GFRA* is widely outclassed. Its percentage of success decreases to 20% from P16. Then, we have SA* which reaches 40% of success at P21. Then, we have the variants of MGP. Their percentages of success starts decreasing between the P22 and P23. Even if the results are comparable, MGP+OC+PF performs better (90% of success) on P23 than MGP+PF (80% of success), MGP+OC (70% of success) and MGP (50% of success). These results are similar on search time. Different variants of MGP outperform SA* and GFRA* with search time less than 5 seconds. Here again, as for the problem P20, MGP+OC performs slightly better. Finally, in terms of plan length, the results on the other problems of Blockworld confirm the results on P20. SA* and GFRA* plans are shorter than for the MGP variants.

4.4 Impact of the Delay Ratio and the Heuristic

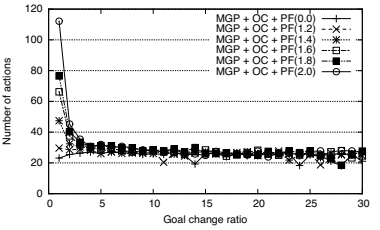
In this section, we evaluate the impact of the delay ratio and the heuristic weight on MGP+OC+PF which is the most efficient variant of MGP. On all tests, we use the same problem (blockworld P20). Since MGP+OC+PF have a success rate that is always close to 100%, we only present search time and plan length results in this section.

Impact of the Delay Ratio. Figure 3(a) and 3(b) show respectively the search time and the plan length wrt. the goal change ratio. We can make three observations. First, we see that the delay ratio significantly increases the performances. For instance, MGP+OC+PF with a delay ratio of 2 is 6 times quicker than with a delay ratio of 0 when the goal change ratio $g_r = 1$. Second, search time and plan length converge quickly on the same values with respect to the increase of the goal change ratio. With $g_r \geq 4$, the delay ratio has no impact on the search time and the plan length. Third, increasing the delay ratio augments the plan length and reduces the search time. Consequently, the delay ratio must be a tradeoff between the plan length and the search time.

Impact of the Heuristic Weight. Figures 4 shows respectively search time and plan length wrt. the heuristic weight of MGP+OC+PF. The heuristic weight w significantly increases the performances (MGP+OC+PF is 4 times quicker with $w = 2.0$ than with $w = 1.0$ for a goal change ratio $g_r = 1$). In addition, the impact of w on the plan length is not significant: whatever the heuristic weight, plan lengths are close for a given goal change ratio.

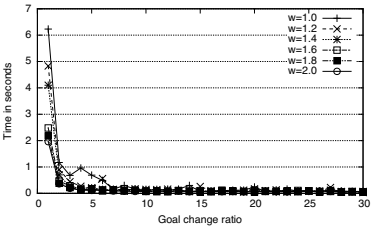


(a) Search time with respect to the delay & goal change ratios.

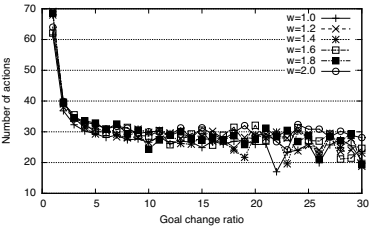


(b) Plan length with respect to the delay & goal change ratios.

Fig. 3. Delay ratio impact in Blockworld P20



(a) Time – MGP+OC+PF



(b) Plan length – MGP+OC+PF

Fig. 4. Heuristic weight impact in Blockworld P20

Table 1. Comparison of search time

Problem	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

4.5 Performance Overview

In this section, we give an overview of the algorithms' performances on different IPC domains: table 1 shows the search time, table 2 shows the percentage of success and table 3 presents the plan length. Each algorithm has been run 100 times to obtain statistically relevant results. Each experiment was allocated a CPU time of 60 seconds. The delay ratio was set to 1.6 and the goal change ratio to 100. The experimentation was carried out on *all problems of each domain* and tables 1, 2 and 3 are parts of our results. The problems were chosen to show the

Table 2. Comparison of percentage of success

Problem	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

Table 3. Comparison of plan length

Problem	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

performances' decrease of the algorithms as observed in Blockworld between the problems 15 and 24.

In terms of search time, MGP+OC+PF is broadly quicker than the other algorithms. Moreover, MGP+OC+PF outclasses MGP with one search delaying strategy (either OC or PF) as well as SA* and GFRA*. Likewise, MGP+OC+PF outclasses the other algorithms in terms of percentage of success. However, MGP sometimes fails on problems solved by SA* (Elevator P35 and Freecell P26) even if MGP is broadly better than SA* and GFRA*. Finally, in terms of plan length, all algorithms find plans with close lengths.

To summarize the evaluation, OC and PF increase the performances of MGP, which performs better than SA* or GFRA*. The combination of OC+PF strategies give better results than MGP with one search delaying strategy. The results obtained on different domains and problems confirm that MGP+OC+PF is the best algorithm and GFRA* is outdistanced because it updates the heuristic value of all states of the search tree at each incremental search.

5 Conclusion

In this paper, we have proposed a novel approach to planning, called MGP, which considers plan adaptation to constantly changing goals as a process pursuing "moving" goals. MGP is based on an incremental Weighted-A* search and interleaves planning and execution when the goal changes over time. 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 search delaying strategies: Open Check (OC) that checks if the new goal is still in the 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 based on an incremental update of the search tree at each new search in order to reduce the expensive calls to the heuristic function.

We have experimentally shown that MGP outperforms the naive approach SA* and the state-of-the-art approach GFRA*. We have shown that the combination of the search delaying strategies OC+PF gives better performances than one search delaying strategy. Moreover, the delay ratio of the Plan Follow (PF) strategy must be a tradeoff between plan length and search time while the heuristic weight in the WA* search enhances the search time.

We are currently pursuing two concurrent lines of work: (1) generating goal changes based on domain-dependent strategies and real-world applications and (2) extending our approach to real-time planning where search and execution are time bound.

References

1. Pellier, D., Fiorino, H., Métivier, M.: A new approach for continual planning. In: AAMAS, pp. 1115–1116 (2013)
2. Nebel, B., Koehler, J.: Plan reuse versus plan generation: A theoretical and empirical analysis. Artificial Intelligence 76, 427–454 (1995)
3. van der Krogt, R., de Weerdt, M.: Plan repair as an extension of planning. In: ICAPS, pp. 161–170 (2005)
4. Fox, M., Gerevini, A., Long, D., Serina, I.: Plan stability: Replanning versus plan repair. In: ICAPS, pp. 212–221 (2006)
5. Ishida, T., Korf, R.: Moving target search. In: IJCAI, pp. 204–210 (1991)
6. Korf, R.: Real-Time Heuristic Search. Artificial Intelligence 42(2-3), 189–211 (1990)
7. Melax, S.: New approaches to moving target search. In: AAAI Falls Symp. Game Planning Learn., pp. 30–38 (1993)
8. Koenig, S., Likhachev, M.: Adaptive A*. In: AAMAS, pp. 1311–1312 (2005)
9. Koenig, S., Likhachev, M., Sun, X.: Speeding up moving target search. In: AAMAS (2007)
10. Sun, X., Koenig, S., Yeoh, W.: Generalized adaptive A*. In: AAMAS, pp. 469–476 (2008)
11. Stenz, A.: The focused D* algorithm for real-time replanning. In: IJCAI, pp. 1642–1659 (1995)
12. Sun, X., Yeoh, W., Koenig, S.: Efficient incremental search for moving target search. In: IJCAI, pp. 615–620 (2009)

13. Sun, X., Yeoh, W., Koenig, S.: Generalized Fringe-Retrieving A*: Faster Moving Target Search on State Lattices. In: AAMAS, pp. 1081–1088 (2010)
14. Sun, X., Yeoh, W., Uras, T., Koenig, S.: Incremental ARA*: An Incremental Anytime Search Algorithm for Moving Target Search. In: ICAPS (2012)
15. Likhachev, M., Gordon, M., Thrun, S.: ARA*: Anytime a* with provable bounds on sub-optimality. In: NIPS (2003)
16. Finke, R., Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 3-4(2), 189–208 (1971)
17. Hoffmann, J., Nebel, B.: The FF Planning System: Fast Plan Generation Through Heuristic Search. JAIR 14(1), 253–302 (2001)