An UCT Approach for Anytime Agent-based Planning

Damien Pellier, Bruno Bouzy, and Marc Métivier

Abstract In this paper, we introduce a new heuristic search algorithm based on mean values for anytime planning, called MHSP. It consists in associating the principles of UCT, a bandit-based algorithm which gave very good results in computer games, and especially in Computer Go, with heuristic search in order to obtain an anytime planner that provides partial plans before finding a solution plan, and furthermore finding an optimal plan. The algorithm is evaluated in different classical planning problems and compared to some major planning algorithms. Finally, our results highlight the capacity of MHSP to return partial plans which tend to an optimal plan over the time.

1 Introduction

The starting point of this work was to apply Upper Confidence bounds for Trees (UCT) [13], an efficient algorithm well-known in the machine learning and computer games communities, and originally designed for planning, on planning problems. A weakness of classical planners is the all-or-nothing property. First, when the problem complexity is low enough, classical planners find the best plan very quickly. Second, when the problem complexity is medium, planners first try to find a solution plan (not the optimal one) and then pursue their search to extract a better solution [6, 4]. This technique is called anytime planning. Finally, when the problem complexity is too high, planners are not able to find any solution plan. In order to answer in part to this weakness, we introduce a new approach based on heuristic search and mean values for anytime planning able to provide partial plans before finding a first plan, and furthermore finding an optimal plan.

Anytime planning can be understood in two meanings. In the planning domain, anytime planning means finding a solution plan, and then refining it to find an optimal plan. There is a good chance that if you stop the planner before finding an optimal plan, the planner has already a solution plan to provide, and it looks like anytime. However, if you stop the planner before having a first solution plan, the planner is not anytime in a strict meaning. When stopped before having a first solution plan, an anytime planner should be able to give some relevant information, for example the beginning of a plan, a partial plan, or the first action. Until a solution plan is found, the longer the time the longer the partial plan is. In this work, the term anytime refers to the strict meaning. We are interested in partial plans.

Originally, UCT is a bandit-based planning algorithm designed for Markov Decision Process (MDP). UCT builds a tree whose root is the current state on which a decision must be taken. The principal variation of the tree is the current solution, and when a plan is found, the principal variation of the tree is the sequence of actions to perform to reach

Laboratoire d'Informatique de Paris Descartes 45, rue des Saints Pères, 75006 France

the goal. As time is going on, UCT builds up its tree adding nodes at each iteration. At any time, UCT has a principal variation which can be considered as a partial plan. However, [13] did not give known successful applications in the planning domain yet. Instead, UCT gave tremendous results in computer games, and specifically in computer go with the Go playing program Mogo [5]. In computer go, UCT is efficient for several reasons. The first reason is that the Go complexity is high, and games are played in limited time. Consequently Go playing programs must find moves that does not need to be optimal, but that need to be the less bad as possible given the limited time. The anytime property is crucial in computer games, and UCT has it. Consequently, studying UCT, anytime algorithm originally designed for planning problems successful in two-player games, was a good starting point to attempt removing the all-or-nothing weakness observed on classical planners. In this attempt, we reached an interesting point to contribute to the planning community. This paper presents the work consisting in associating UCT ideas with heuristics in state space search in order to obtain an anytime planner which provides partial plans before finding a first plan, and furthermore finding the best plan. The paper shows a new heuristic search algorithm based on mean values for anytime planning, called Mean-based Heuristic Search for anytime Planning (MHSP).

The outline of the paper is the following. Section 2 describes previous works. Section 3 presents MHSP. Section 4 shows experimental results. Finally, section 5 discusses this approach and concludes.

2 Previous works

UCT and Computer Go. UCT worked well in Go playing programs, and it was used under many versions leading to the Monte-Carlo Tree Search (MCTS) framework [3]. A MCTS algorithm starts with the root node as a tree, and while time remains, it iteratively grows up a tree in the computer memory by following the steps below: (a) starting from the root, browse the tree until reaching a leaf by using (1), (b) expand the leaf with its child nodes, (c) choose one child node, (d) perform a random simulation starting from this child node until the end of the game, and get the return, i.e. the game's outcome, and (e) update the mean value of the browsed nodes with this return.

With infinite time, the root value converges to the minimax value of the game tree [13]. The Upper Confidence Bound (UCB) selection rule (1) answers the requirement of being optimistic when a decision must be made facing uncertainty [1].

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

N_{select} is the selected node, N is the set of children, m is the mean value of node n, s is the number of iterations going through n, p is the number of iterations going through the parent of n, and C is a constant value setup experimentally. (1) uses the sum of two terms: the mean value m, and the UCB bias value which guarantees exploration.

Planning under time constraints. Besides, planning under time constraints is an active research domain that results in adaptive architectures [9], real-time control architectures [16], and real-time heuristic search [15].

3 MHSP

This section defines our algorithm MHSP. We made two important choices in designing MHSP after which we give the pseudo-code of MHSP.

Heuristic values replace simulation returns. On planning problems, random simulations are not appropriate. Browsing randomly the state space does not enable the algorithm to reach goal states sufficiently often. Many runs complete without reaching goal states. Replacing the simulations by a call to the heuristic is far better. Not only the algorithm finds the goal, but it may reach it very quickly: on certain classes of problems, MHSP is as fast as a classical planner to find the best solution. In Computer Go, the random simulations were adequate mainly because they always completed after a limited number of moves, and the return values (won or lost) were roughly equally distributed on most positions of a game. Furthermore, the two return values correspond to actual values of a completed game. In planning, one return means that a solution has been found (episode completed), and the other return means that the episode has not been completed. This simulation difference is fundamental between the planning problem, and the game playing problem. Furthermore, heuristic values bring domain-dependent knowledge into the returns. In Computer Go, replacing the simulations by evaluation function calls is forbidden by fifty years of computer Go history. However, in Computer Go, and in other domains, adding proper domain-dependent knowledge into the simulations improves the significance of the returns, henceforth the level of the playing program. Consequently, using heuristic values in our work should be positive bound to the condition that the heuristic value generator is good, which is the case in planning. In MHSP, we replace stage (d) of MCTS above by a call to a heuristic function.

Optimistic initial mean values. Computer games practice shows that the UCB bias of (1) can merely be removed provided the mean values of nodes are initialized with sufficiently optimistic values. This simplification removes the problem of tuning C, while respecting the optimism principle. Generally, to estimate a given node, the planning heuristics give a path length estimation. Convergence to the best plan is provided by admissible heuristics, i.e. heuristics ensuring the heuristic value is inferior to the actual distance from the node to the goal, i.e. optimistic heuristics. Consequently, the value returned by planning heuristics on a node can be used to initialize the mean value of this node. In MHSP, the returns are negative or zero, and they must be in the opposite of the distance from s to g. Thus, we initialize the mean value of a node with $\Delta(s,g)$ which is minus the distance estimation to reach g from s. With this initialization policy, the best node according to the heuristic value will be explored first. Its value will be lowered after some iterations whatever its goodness, and then the other nodes will be explored in the order given by the heuristic.

The algorithm. MHSP algorithm is shown in algo. 1 : O is the set of operators, s_0 the initial state, g the goal, $C[s]$ the set of children of state s, $R[s]$ the cumulative return of state s, $V[s]$ the number of visits of state s, and $P[s]$ the parent of s. The outer *while* (line 2) ensures the anytime property. The first inner *while* (line 4) corresponds to stage (a) in UCT. The default reward is pessimistic: $(R[s_0]/V[s_0])+1$ is the current pessimism threshold. The first two if test whether the inner *while* has ended up with a goal achieved

Algorithm 1: MHSP(O, s_0, g)

1 $C[s_0] \leftarrow \emptyset$; $R[s_0] \leftarrow \Delta(s_0, g)$; $V[s_0] \leftarrow 1$; $\pi \leftarrow nil$
2 **while** *has_time* **do**
3 $\quad s \leftarrow s_0$
4 $\quad$ **while** $g \not\subseteq s$ *and* $V[s] \neq 1$ **do** $s \leftarrow argmax_{s' \in C[s]}(R[s']/V[s'])$
5 $\quad reward \leftarrow (R[s_0]/V[s_0]) + 1$
6 $\quad$ **if** $g \subseteq s$ **then** $reward \leftarrow 0$
7 $\quad$ **else if** $V[s] = 1$ **then**
8 $\quad\quad A \leftarrow \{a \mid a$ ground instance of an operator in O and $precond(a) \subseteq s\}$
9 $\quad\quad$ **foreach** $a \in A$ **do**
10 $\quad\quad\quad s' \leftarrow (s \cup effects^+(a)) - effects^-(a)$
11 $\quad\quad\quad C[s'] \leftarrow C[s] \cup \{s'\}$; $R[s'] \leftarrow \Delta(s', g)$; $P[s'] \leftarrow s$; $V[s'] \leftarrow 1$
12 $\quad\quad$ **if** $C[s] \neq \emptyset$ **then** $s \leftarrow argmax_{s' \in C[s]}(R[s'])$; $reward \leftarrow R[s]$
13 $\quad i \leftarrow 0$
14 $\quad$ **while** $s \neq s_0$ **do** $s \leftarrow P[s]$; $R[s] \leftarrow R[s] + (reward - i)$; $V[s] \leftarrow V[s] + 1$; $i \leftarrow i + 1$
15 $\quad$ **if** $g \subseteq s$ **then**
16 $\quad\quad \pi' \leftarrow reconstruct_solution_plan()$
17 $\quad\quad$ **if** $length(\pi) > length(\pi')$ **then** $\pi \leftarrow \pi'$
18 **if** $\pi = nil$ **then** **return** $reconstruct_best_plan()$; **else** **return** π

(line 6) or with a leaf (line 7). If the goal is not reached, the leaf is expanded, stage (b) in MCTS. The second if corresponds to stage (c). Stage (d) is performed by writing $\Delta(s', g)$ into the return. The second inner *while* (line 14) corresponds to stage (e). Function *reconstruct_solution_plan*() browses the tree by selecting the child node with the best mean, which produces the solution plan. Function *reconstruct_best_plan*() browses the tree by selecting the child node with the best number of visits. The best plan reconstruction happens when the time is over before a solution plan has been found. In this case, it is important to reconstruct a robust plan, may be not the best one in terms of mean value. With the child with the best mean, a plan with newly created nodes could be selected, and the plan would not be robust. Conversely selecting the child with the best number of visits ensures that the plan has been tried many times, and should be robust to this extent.

4 Experimental Results

In this section, we present experimental results in two steps: a first experiment aiming at showing that MHSP can be compared to state-of-the-art planners, and a second experiment aiming at underlining the anytime feature of MHSP (anytime meaning building good partial plans when the running time is shorter than the time necessary to build the first solution plan). We present experimental results obtained in test domains and problems from International Planning Competition, which illustrates the effectiveness of our techniques implemented in MHSP. All the tests were conducted on an Intel Core 2 Quad 6600 (2.4Ghz) with 2 Gbytes of RAM. The implementation of MHSP used for experiments is written in Java based on the PDDL4J library.

First experiment. The experiments were designed in order to show that MHSP: (1) performs almost as well as classical planners on classical planning problems, and (2)

returns, given a fixed amount of time, the beginning of the optimal plan that classical planners cannot solve with the same amount of time.

Figures 1(a) and 1(b) show (on log scale) performance of MHSP-speed on two domains (blocksworld and ferry) according to the problem size. The planners used to the comparison were chosen for their planning techniques: IPP for planning graph [14], Satplan 2006 for planning by satisfiability [12], SGPlan-5 for subgoal decomposition planning [11] and FDP for constraint satisfaction techniques [7]. For both domains, MHSP was tested with three heuristics: Hs^{+}, Hs^{max} used by HSP planner [2] and FF-heuristic used by [10]. The CPU-time limit for each run was 10000 seconds, after which termination was forced. The results show that MHSP performs almost the most quickly (except SGPlan which is based on hill climbing and heuristic search and goal agenda techniques). However, the three heuristics do not perform as well. As expected, Hs^{max} is less informative than Hs^{+} and FF-heuristic. Thus, MHSP with Hs^{max} performs more slowly than with the other heuristics. Moreover, Hs^{+} is more efficient than FF-heuristic as displayed by Table 1. This difference can be explained by the fact that Hs^{+} returns values more dispatched than FF-heuristic which is more informative for MHSP. Finally, if we look at the number of actions of the first solution plan found by the different planners, we observe that MHSP finds solution plan of good quality. To conclude this first experimentation, let's consider the figure 1(c) that displays the behavior of MHSP on a specific blocksworld problem containing 17 blocks. This figure shows that the number of actions belongs to the optimal solution plan found by MHSP given a fixed amount of time. Notice that the results are statistically meaningful (MHSP was run 20 times each 10 ms time step using FF-heuristic). We observe that MHSP finds very quickly the first actions of the optimal solution plan. It meaningfully needs only 1500ms to find the first 10 actions of the optimal solution plan that has a length of 31 actions. Of course, MHSP performs only if the heuristic is informative and a complete study of the behavior of MHSP with all the heuristics available in the literature would be necessary.

Partial plan experiment. We present the results obtained by A*, Greedy Search (GS), MHSP-ff, and Enforced Hill-Climbing (EHC) [10] on four problems: Blocksworld problem 12, Ferry problem L6 C9, Gripper problem 6, and Hanoi problem 6. The aim of this second experiment is to see whether the partial plans built by the four algorithms are good or not when the running time is shorter that the time to build a first solution plan. To evaluate a partial plan, we define two distances: the distance to the goal and the distance to the optimum.

- *Distance to the goal.* The distance to the goal of a partial plan is the length of the optimal plan linking the end state of this partial plan to the goal state. When the distance to the goal diminishes, the partial plan has been built in the appropriate direction. When the distance to the goal is zero, the partial plan is a solution plan.
- *Distance to the optimum.* The distance to the optimum of a partial plan is the length of the partial plan, plus the distance to the goal of the partial plan, minus the length of the optimal plan. When the distance to the optimum of a partial plan is zero, the partial plan is the beginning of an optimal plan. The distance to the optimum of a solution plan is the diffence between its length and the optimal length. The distance to the optimum of the void plan is zero.

The distance to the goal and the optimal distance of an optimal plan is zero. Conversely, when the distance to the goal and the distance to the optimum of a partial plan are zero, the partial plan is an optimal plan. For each problem, the results are shown with figures giving the distance to the goal and the distance to the optimum of the partial plan in the running time. These distances are computed every ms by calling an optimal planner (i.e. A*).

Partial plans of the four algorithms. The partial plan given by A* at a given time is the path linking the root node to the current expanded leaf node. Given A* manages a list of whole leaf nodes, the partial plan provided by A* varies a lot from a given time to another. To get the partial plan provided by MHSP at a given time, we browse the MHSP tree from the root using the number of visits, and we stop when this number is below a threshold that equals the branching factor of the tree. This way, the partial plan is more stable, but shorter. GS browses and expands the tree starting from the root by selecting the child whose heuristic value is minimal. The weakness of GS is its non-optimality. EHC chooses the first node whose heuristic value is strictly inferior to the heuristic value of the current node, insuring a progress toward the goal is made when selecting this node. The weakness of EHC is its inability to go out of a deadend or a plateau. In such cases, our EHC returns a failure. With the state-of-the-art heuristics, when they find a solution, GS and EHC are actually very fast.

Blocksworld problem 12. In this problem, A* finds the optimal solution in 130ms (see figure 1(d)). When the running time is inferior to 50 ms, the distance to the goal remains at its initial value. Between 50 ms and 130 ms, the distance to the goal decreases but remains high before the optimal plan is found. Along the running time, the distance to the optimum is low but strictly greater than zero. Experimental results of GS and EHC are not shown. Both algorithms go to deep in the search space. Consequently the algorithm used to compute the distance to the optimum (in our experiments A*) falls to find the optimum plan in a reasonable time frame. These results highlight the weakness of both algorithms in that problem. MHSP optimally solves this problem in 230 ms (see figure 1(e)). Like A* does, when the running time is inferior to 50 ms, the distance to the goal remains at its initial value, and between 50 ms and 230 ms, the distance to the goal decreases but remains high before the optimal plan is found. MHSP explores along optimal partial plans for running times inferior to 200 ms. When looking at the distance to the goal, the relative comparison between A* and MSHP on this problem is in favour of A*, but the distance to the goal of MHSP decreases almost monotically while the distance to the goal of A* is decreasing with large oscillations. When looking at the distance to the optimum, the relative comparison between A* and MSHP on this problem is in favour of MHSP (except after 130 ms).

Hanoi problem 6. In this problem, A* finds the optimal solution in 1900ms (see figure 1(f)). On this problem, the distance to the goal decreases with oscillations. The distance to the optimum increases before the optimal plan is found. EHC falls in a deadend and cannot solve this problem. GS is very efficient on this problem, making good use of the heuristics (see figure 1(g)). It finds a solution plan in about 450 ms. MHSP solves this problem in 600 ms (see figure 1(h)). However, as time is running, the distance to the optimum of the partial plan of MHSP increases, and the solution found by MHSP on

this problem is not optimal at all. Finally, MHSP is slower than GS but faster than A* to find a plan on this problem.

Ferry problem L6 C9. On the Ferry problem L6 C9, MHSP finds the optimal solution in 1050 ms (see figure 2(a)), A* finds the optimal solution in 2100 ms (see figure 2(b)). MHSP is twice faster than A* on this problem. The distance to the goal of MHSP decreases more quickly than it does for A*. MHSP shows a better anytime ability than A* on this problem. However, EHC finds a solution in 230 ms, four times faster than MHSP (see figure 2(c)), but this solution is not optimal. Besides, GS finds a solution in 60 ms, four times faster than EHC (see figure 2(d)), but this solution is not optimal. EHC and GS are one order of magnitude faster than MHSP and A* and they find solutions not far from optimal. MHSP is the fastest algorithm to find an optimal plan on this problem.

Gripper problem 6. On the Gripper problem 6, the same remarks can be made. MHSP finds the optimal solution in 1400 ms (see figure 2(e)), A* finds the optimal solution in 8000 ms (see figure 2(f)), which is rather slow. MHSP is five times faster than A* on this problem. The distance to the goal of MHSP decreases more slowly than it does for A*. Furthermore MHSP partial plans are far from being optimal. EHC finds a solution in 100 ms, fourteen times faster than MHSP (see figure 2(g)), but this solution is not optimal. Besides, GS finds an optimal solution in 85 ms (see figure 2(h)). EHC and GS are one order of magnitude faster than MHSP and A*.

5 Conclusion

Anytime heuristic search has been studied already [8]. However, this work focused on finding a first plan, and refining it to find the best plan. Such a method cannot give any information before a first plan is found, especially a partial plan.

In this paper, we presented MHSP a new anytime planning algorithm which provides partial plans before finding a solution plan. This algorithm combines an heuristic search and the learning principles of UCT algorithm, i.e. states' values based on mean returns, and optimism in front of uncertainty. Of course, when given insufficient time, the partial plan is not garanteed to be a prefix of an optimal plan or of a solution plan. However, on average over our benchmark, the partial plans returned by MHSP are prefix of either solution plans or optimal plans. Evaluated in several classical problems, our first results showed that MHSP performs almost as well as classical planners on classical planning problems. However, MHSP is surpassed by SGPlan that uses a goal agenda (MHSP does not). We defined two distances to evaluate partial plans provided by a planner at a given time: the distance to the goal and the distance to the optimum. With such measures, we experimentally compared MHSP, A*, EHC and GS. Given a fixed amount of time, MHSP provides partial plans which tend to be the beginning of solution plans and then optimal plans when the running time increases. However, given the speed of EHC and GS when they find solution plans, anytime conclusions can hardly be drawn when considering absolute running times.

Averaging in MHSP may be discussed. A possibility is to replace averaging by backing-up the best child value. Then MHSP would look like Greedy Search which may fall into deadends, even with admissible heuristics. Therefore, a first reason for

averaging is to avoid deadends. With heuristics admissible or not, MHSP expands a different leaf than A* when, in the upper part of the tree, a mean value is high and leads to a bottom part of the tree where the heuristics values are high on average but inferior to the heuristic value of the node selected by A*. Our partial plan experiment shows that MHSP have both their pros and cons since MHSP is better than A* on Ferry and Gripper, but worse on Blocksworld and Hanoi.

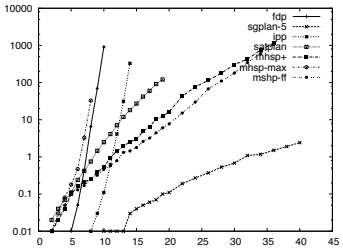
In the future, we want to apply MHSP on problems with non deterministic environments to take avantage of the averaging of MSHP. Furthermore, since MHSP is successfully validated on four benchmarks of classical planning, integrating MHSP into practical applications remains an enlightening future work. In the current study, we used three heuristic functions: seeing which function is best-suited to each problem is another interesting research direction.

References

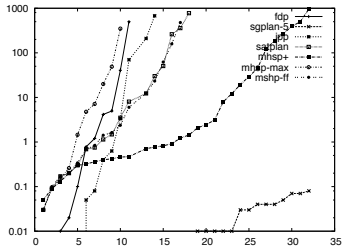
1. Auer, P., Cesa-Bianchi, N., Fisher, P.: Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning **47**(2–3), 235–256 (2002)
2. Bonet, B., Geffner, H.: Planning as Heuristic Search. Artificial Intelligence **129**(1–2), 5–33 (2001)
3. Chaslot, G., Winands, M., van den Herik, H., Uiterwijk, J., Bouzy, B.: Progressive Strategies for Monte-Carlo Tree Search. New Mathematics and Natural Computation **4**(3), 343–357 (2008)
4. Chen, Y., Huang, R., Zhang, W.: Fast Planning by Search in Domain Transition Graphs. In: Proc. AAAI, pp. 886–891 (2008)
5. Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with Patterns in Monte-Carlo Go. Tech. Rep. RR-6062, INRIA (2006)
6. Gerevini, A., Serina, I.: LPG: A Planner Based on Local Search for Planning Graphs with Action Costs. In: Proc. ICAPS, pp. 13–22 (2002)
7. Grandcolas, S., Pain-Barre, C.: Filtering, Decomposition and Search Space Reduction for Optimal Sequential Planning. In: Proc. AAAI (2007)
8. Hansen, E.A., Zhou, R.: Anytime Heuristic Search. JAIR **28**(1), 267–297 (2007)
9. Hayes-Roth, B.: An architecture for adaptive intelligent systems. Artificial Intelligence **72**, 329–365 (1995)
10. Hoffmann, J., Nebel, B.: The FF Planning System: Fast Plan Generation Through Heuristic Search. JAIR **14**(1), 253–302 (2001)
11. Hsu, C.W., Wah, B., Huang, R., Chen, Y.: Handling Soft Constraints and Goals Preferences in SGPlan. In: Proc. of the ICAPS Workshop on Preferences and Soft Constraints in Planning (2006)
12. Kautz, H.A., Selman, B.: Unifying SAT-based and Graph-based Planning. In: Proc. IJCAI, pp. 318–325 (1999)
13. Kocsis, L., Szepesvari, C.: Bandit-based Monte-Carlo Planning. In: Proc. ECML, pp. 282–293 (2006)
14. Koehler, J., Nebel, B., Hoffmann, J., Dimopoulos, Y.: Extending planning graphs to an ADL subset. In: Proc. ECP, pp. 273–285 (1997)
15. Korf, R.: Real-Time Heuristic Search. Artificial Intelligence **42**(2-3), 189–211 (1990)
16. Musliner, D., Goldman, R., Krebsbach, K.: Deliberation scheduling strategies for adaptive mission planning in real-time environments. In: Proceedings of the Third International Workshop on Self Adaptive Software (2003)

domains	**best**	**mhsp+**		**mhsp-ff**		**ipp**		**satplan**		**sgplan5**		**fdp**	
	plan	time	cost	time	cost	time	cost	time	cost	time	cost	time	cost
hanoi-4	15	0.32	15	0.31	15	0.00	15	0.53	15	0.00	15	0.05	15
hanoi-6	63	1.30	66	1.74	75	0.03	63	> 120	na	0.02	63	10.92	63
hanoi-7	127	4.56	145	5.19	147	0.11	127	> 120	na	0.08	127	> 120	na
gripper-7	21	4.55	23	6.66	21	0.16	21	20.57	21	0.00	21	1.42	15
gripper-8	21	11.91	27	22.51	23	0.43	23	25.38	23	0.01	23	4.10	23
gripper-9	27	28.88	29	85.57	27	1.61	27	> 120	na	0.01	27	15.39	27
satellite-2-4	20	0.71	20	27.87	20	41.36	23	0.45	25	0.00	24	> 120	na
satellite-2-5	29	7.76	29	> 120	na	> 120	na	11.40	43	0.04	35	> 120	na
satellite-2-6	43	28.79	43	> 120	na	> 120	na	> 120	na	0.11	71	> 120	na
zeno-2-6	15	9.71	16	118.79	15	0.03	17	0.18	19	0.01	24	51.61	15
zeno-3-6	11	24.48	11	> 120	na	0.04	18	0.39	18	0.01	15	> 120	na
zeno-3-8	23	24.28	23	> 120	na	77.23	29	0.92	27	0.01	29	> 120	na

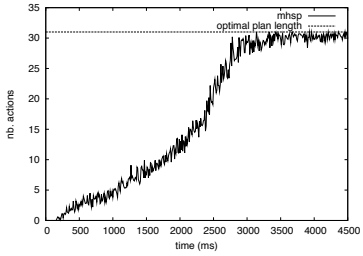
Table 1 Comparison of the time (sec.) and cost (number of actions) of the plan found by MHSP



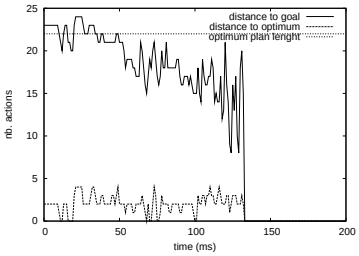
(a) Planning times (sec.) – blocksworld domain



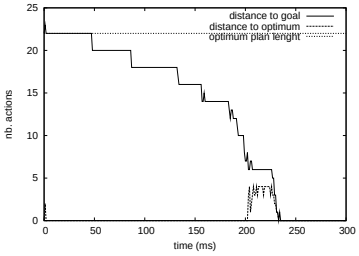
(b) Planning times (sec.) – ferry domain



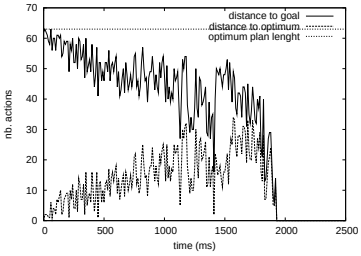
(c) The number of actions belongs to the optimal solution plan found by MHSP given a fixed amount of time on a specific blocksworld problem containing 17 blocks



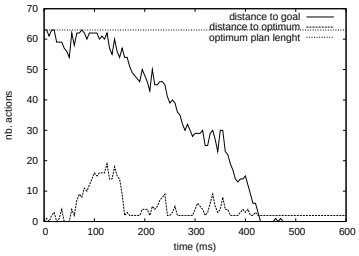
(d) A* Blocksworld problem 12



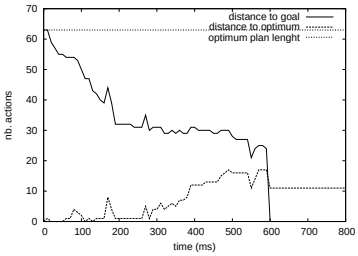
(e) MHSP Blocksworld problem 12



(f) A* Hanoi problem 6

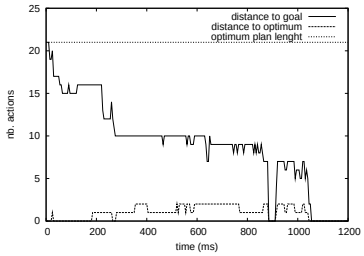


(g) Greedy Search Hanoi problem 6

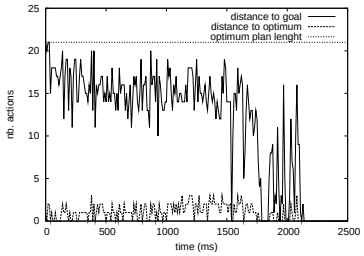


(h) MHSP Hanoi problem 6

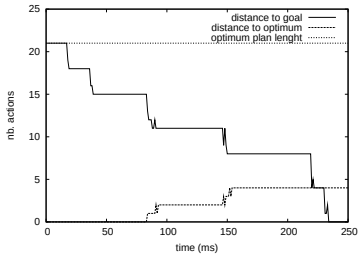
Fig. 1 First experimental results



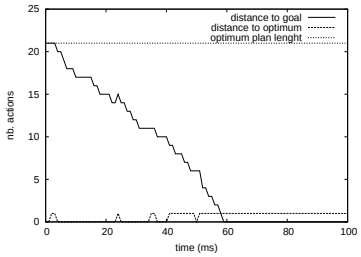
(a) MHSP Ferry problem L6 C9



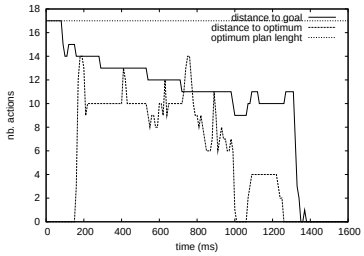
(b) A* Ferry problem L6 C9



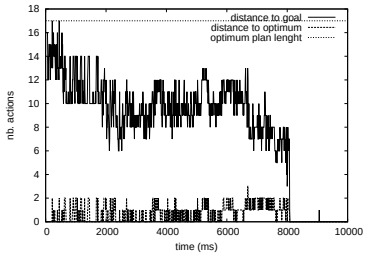
(c) EHC Ferry problem L6 C9



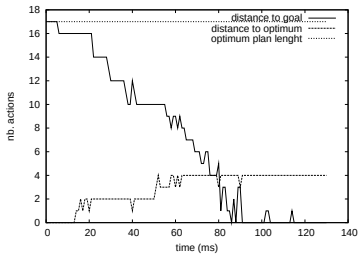
(d) Greedy Search Ferry problem L6 C9



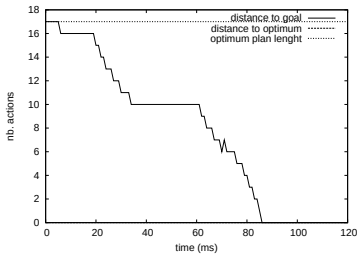
(e) MHSP Gripper problem 6



(f) A* Gripper problem 6



(g) EHC Gripper problem 6



(h) GS Gripper problem 6

Fig. 2 Second experimental results