HPP: A Heuristic Progressive Planner

M. D. R-Moreno[1], D. Camacho[2] and A. Moreno[1]

[1] Departamento de Automática. Universidad de Alcalá.
Ctra. Madrid-Barcelona, km. 33,6. 28871 Alcalá de Henares (Madrid), Spain.
{mdolores, angel}@aut.uah.es

[2] Departamento de Informática. Universidad Autónoma de Madrid.
Ctra. de Colmenar Viejo, km 15. 28049 Cantoblanco (Madrid), Spain.
david.camacho@uam.es

Abstract

In the last decade three main ideas have revolutionized the Artificial Intelligence (AI) planning methods: the plan graph search structure, planning as satisfiability and the search guided by a heuristic function calculated from a relaxed plan. In this paper we present a new planner, named HPP (Heuristic Progressive Planner), that has been implemented using several ideas extracted from other heuristic planners. However, HPP includes a new module (*analysis reachability module*) that is able to exclude irrelevant domain-dependent operators for the planning process. The analysis performed by this module avoids to expand several parts of the search tree allowing to solve more problems. Finally, the HPP planner has been incorporated into a distributed multi-agent system to allow exploring in parallel subsets of the search space.

The experimental results show how the HPP planner outperforms FF and state of the art planners evaluated in the number of problems solved or the number of steps.

1 Introduction

The AI planning community has seen in the last years an important increment of both, the efficiency and quality of the solutions provided by the available planners. From the GPS [9] to the STRIPS planner [5] that used a stack of goals and the operators that could achieve them, new contributions has been made to solve two main problems in AI plannig. On the one hand, how to represent a particular problem, and on the other hand, how to implement reasoning processes that automatically can solve these problems.

In the knowledge representation for planning, STRIPS has been the more widely used representation, it has evolved into other richer representations, such as the ADL representation [10], or the standard planning language PDDL2.2 [4]. Currently, other approaches that have been used in the planning area is the HTN formalism, ontologies for planning, temporal logic, situational calculus or the propositional logic.

The second main problem in planning, how to implement automatic reasoning techniques, has been studied through the development of three main ideas: to implement the searching process of the solutions using a graph-based search structure; through the generation of a set of propositional clauses from the planning problem that is checked for satisfiability; and finally guiding the searching process using a heuristic function calculated from a relaxed plan. In the first case we are referring to GRAPHPLAN [1] and its descendants, in the second case to SAT-based planners [12] and in the third case to the Heuristic Search Planners (HSP) [2, 8] family.

Despite of these advances, the number of explore nodes by any state of the art planner is still very high. And one of the reasons is that in the planning process they use a higher number of instantiated operators and facts than suffice to solve the problem.
For planners based on heuristic search, the main reason for inefficiency is that state space search is not a very efficient way of traversing big search spaces because states are considered one at a time.

This paper describes HPP (Heuristic Progressive Planner), a FF descendant planner whose main contribution is the design of an *analysis reachability module* that is able to exclude some of the operators that are irrelevants in the planning process. Once these irrelevant operators have been rejected, the search tree is expanded using an heuristic search planner. HPP has been designed to implement a multi-agent search module to allow exploring in parallel subsets of the search space.

The paper is structured as follows. Section 2 briefly describes some basic concepts about Heuristic Search Planners. Section 3 presents the HPP architecture and the algorithms used to improve the efficiency of the planning process. Section 4 shows several experiments that have been carried out to measure the behaviour of the HPP planner. Finally, section 5 summarizes the conclusions and future work.

2 Heuristic Search Planners

The Heuristic Search Planners (HSP) transform planning problems into problems of heuristic search by automatically extracting heuristics functions (h) from the problem, instead of introducing them manually [2].

Considering a relaxed problem in which all delete lists are ignored, from any state s, the optimal cost $h'(s)$ for solving the relaxed problem can be shown to be a lower bound on the optimal cost $h^*(s)$ for solving the original problem. As a result, the heuristic function $h'(s)$ can be used as an admissible heuristic for solving the original problem.

Figure 1 shows the general architecture for a heuristic search planner. This architecture is implemented by three main modules that can be summarized as follows:

- The *Analysis & Processing* module is the responsible to analyse and process all the information that comes from the domain and the initial state of the problem. Usually, a table or a vector is created with all the possible instantiated operators in the problem. To gain efficiency during the search process, predicates and variables are codified into numbers.
- The *Heuristic Computation* module, that computes the cost of applying a determined node in the search process. Some of the (usual) heuristics that can be used to estimate the cost are:
  - To calculate the cost of a set of atoms assuming that subgoals are independent. This heuristic is called the additive heuristic h_{add}.
  - To combine the cost of atoms by a maximisation operation. This is called the *max* heuristic, i.e., replacing sums by maximisation.
  - To calculate the number of actions in an explicit solution from a relaxed GRAPHPLAN algorithm.
- Finally, the *Search* module, that directly depends on the heuristic computation, uses a search algorithm or a combination of them, according to the objectives that we want to achieve: fast solutions, solutions with the less number of steps, etc.

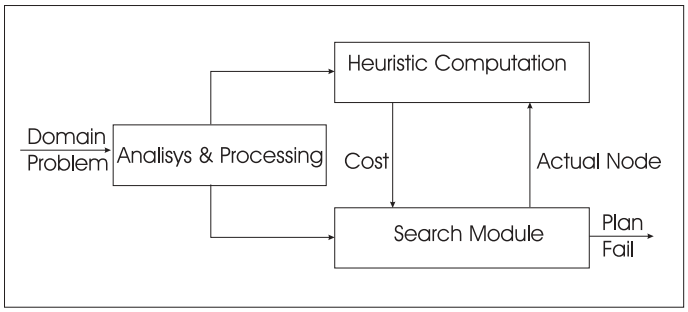

Figure 1: HSP general structure.

Most of the research effort in this area has been mainly focused on the *Search* and *Heuristic* modules, specially in the last one. However, in the following section, we will show how the *Analysis & Processing* module can be modified to help (and improve) the search process.

3 The Heuristic Progressive Planner: HPP

HPP is a forward heuristic planner that bases its *Heuristic* module on FF, and includes new modifications in the *Search* and *Analysis* modules with the purpose of decreasing the number of visited nodes. Our planner includes a new module that is the responsible of reducing the number of operators that can be used during the search process. Figure 2 shows the different modules that compose the HPP architecture. The next sections explain each module in detail.

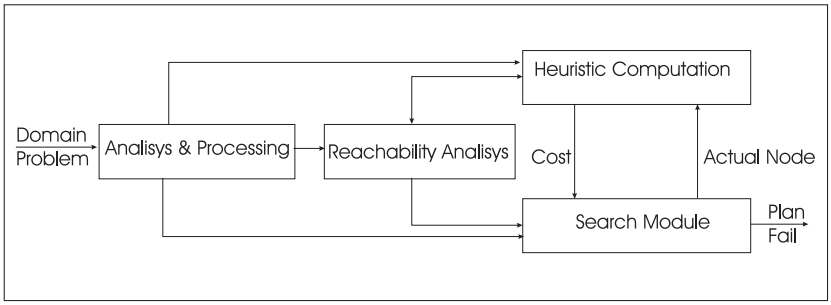

Figure 2: HPP general structure.

3.1 The Reachability Analysis and Analysis & Processing Modules

Previously to the new *Reachability Analysis* module, HPP like FF and other similar planners, processes and loads all the information in different classes. To decrease the computing time, the strings are coded by numbers and keep this information in memory. Then, the static and non-static information is separated. This step is very important to have the first operators instantiation.

In the second step, the instantiation variables will be bound by the all constant combinations that the problem can have. We will call this combination *A vector*, that is, the *A vector* will contain all the possible operators instantiation.

For example, if we consider the blocksworld domain (see Figure 3) and the problem of Figure 4, the number of possible applicable operators in the A *vector* is 24 (this value is obtained by substituting each variable in the operators by the objects in the problem). At a first glance, we can see that there are 6 unnecessary combinations, that is, when the instantiated variables in the *unstack* and *stack* operators are equal (i.e. *(unstack A A)*, *(unstack B B)*, etc). But even eliminating these 6 combinations (HPP does it), the combination is still high (in the solution just 6 operators were needed). Table 1 shows the elements of this vector.

When the *Reachability Analysis* module is applied, the HPP planner generates two new sets of instantiated operators that contains less elements than in the *A vector*. Both vectors will be created using two well-known relaxed heuristics in order to reduce the number of operators.

Table 1: *A Vector* for the problem of Figure 4.

pick-up A	pick-up B	pick-up C	putdown A	putdown B
putdown C	stack A B	stack B A	stack A C	stack C A
stack B C	stack C B	unstack A B	unstack B A	unstack A C
unstack C A	unstack B C	unstack C B	-	-

```
(:action pick-up
 :parameters (?x)
 :precondition (and (clear ?x)
                    (ontable ?x)
                    (handempty))
 :effect(and (not (ontable ?x))
             (not (clear ?x))
             (not (handempty))
             (holding ?x)))

(:action put-down
 :parameters (?x)
 :precondition (holding ?x)
 :effect(and (not (holding ?x))
             (clear ?x)
             (handempty)
             (ontable ?x)))

(:action stack
 :parameters (?x ?y)
 :precondition (and (clear ?)
                    (holding ?x))
 :effect (and (not (holding ?x))
              (not (clear ?y))
              (clear ?x)
              (handempty)
              (on ?x ?y)))

(:action unstack
 :parameters (?x ?y)
 :precondition (and (on ?x ?y)
                    (clear ?x)
                    (handempty))
 :effect (and (holding ?x)
              (clear ?y)
              (not (clear ?x))
              (not (handempty))
              (not (on ?x ?y))))
```

Figure 3: The Blocksworld domain coded in PDDL.

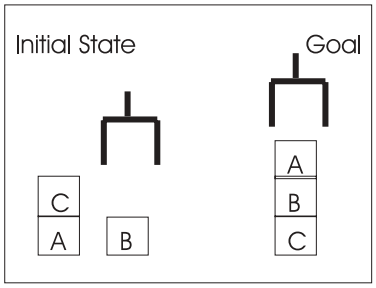

Figure 4: A problem in the Blocksworld domain.

The first new vector, *B vector*, will be generated thanks to a relaxed GRAPHPLAN heuristic, that it will be also used in the *Heuristic* module (see Section 3.2) to compute the costs. Each operator is relaxed by simply eliminating its delete list. The relaxed plan graph is similar to that produced by the GraphPlan, except that it does not contain any mutual exclusion relations. The fact level is built when all the applicable operators in a determined level are instantiated. When any operator is selected in the action level, the goals satisfiability will be checked. If the goals have been achieved, the process stops. All the operators used in the different action levels will be collected and saved in the *B vector*. Then, for the previous example, B *vector* will contain 14 operators as Table 2 shows. The relaxed plan graph consists of four fact layers and three action layers as Figure 5 shows.

Table 2: *B Vector* for the problem of Figure 4.

pick-up A	pick-up B	pick-up C	putdown A	putdown B
putdown C	stack A B	stack B A	X	stack C A
stack B C	stack C B	X	X	X
unstack C A	unstack B C	unstack C B	-	-

The second new vector, that we have called *C vector*, is generated using an additive heuristic h_{add} as the one used in HPP for computing the heuristic cost. Considering the subgoal independence under this heuristic, *C vector* will contain all the operators that are part of the relaxed plan. For the the blocksworld example, C *vector* only contains **9 operators** (see Table 3). The relaxed heuristic is shown on Figure 6.

For this simple example it is possible to compare the number of operators that other planners will generate for the same blocksworld example, the results are shown on Table 4. In this example the

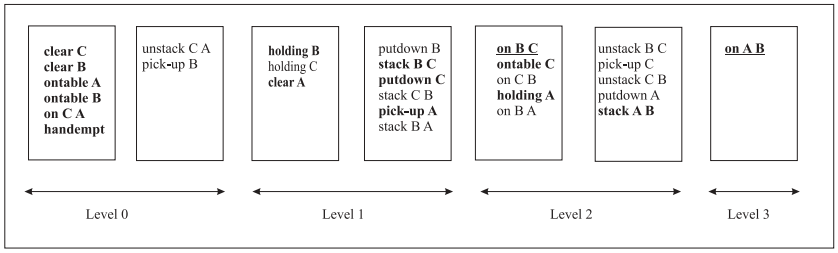

Figure 5: Relaxed Graphplan.

Table 3: *C Vector* for the problem of Figure 4.

pick-up A	pick-up B	pick-up C	putdown A	putdown B
putdown C	stack A B	X	X	X
stack B C	X	X	X	X
unstack C A	X	X	-	-

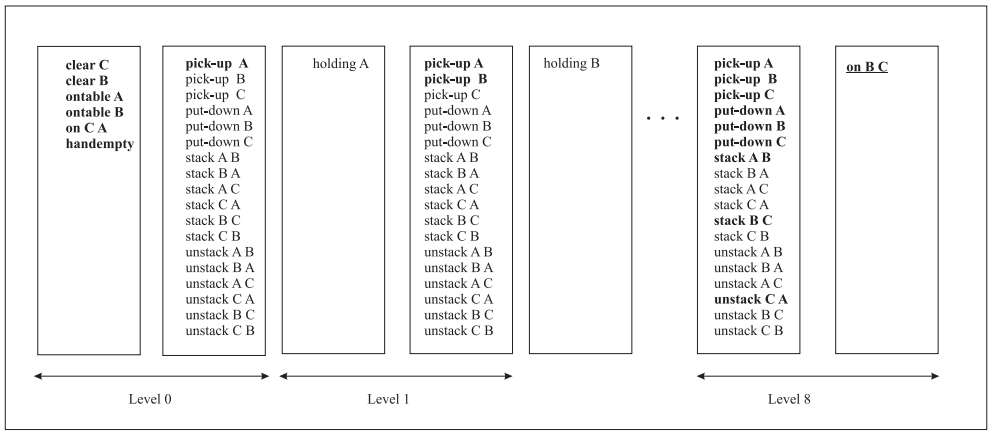

Figure 6: Additive Heuristic used for the *C Vector*.

improvement on the plan length (number of steps) and time execution are not very important due to the simplicity of the problem.

Table 4: N. of operators (*A Vector*) used by other planners in the blocksworld example.

Planner	METRIC-FF	FF-SPEED	LPG	SAPA	HSP2.0	ALTALT
N. ops	24	18	18	24	18	24

3.2 The Heuristic Module

The heuristic computation is probably the main problem for any planner that pretends to reduce the search. This module performs the main differences among most of HSP planners. We can devise a good search algorithm in the search module that branches more or less branches on the search tree, that analyses more or less nodes, but it is the distance estimation the main factor that makes that a planner is better than another.

Why we do believe the estimation is the most important factor? The reason is quite simple: *"if the estimation is closed to the real distance, more branches will be excluded, and then fewer nodes will be processed. Therefore, less search effort time will be spent"*.

For our first HPP version, we have used the same FF *Heuristic* module [8]. As mentioned before, each operator is relaxed by simply eliminating its delete list. The relaxed plan graph generated does not contain any mutual exclusion relations, starting from the initial state to the goals. To generate the relaxed plan, HPP starts with the goal propositions and selects a *no-op* or *action* from next action layer that achieves it, preferring *no-ops* over *actions*. This module stops evaluating actions as soon as it finds an action whose goal state has a better heuristic value.

3.3 The Search Module

The Search Module in HPP can be considered as a Collaborative Multi-Agent system (see Figure 7) with agents working in parallel and in a subset of the search space. Each agent will use each of the vectors generated in the Analysis & Processing module performing the Enforced Hill-Climbing (EHC) algorithm when looking for a solution.

When any agent finds the solution, it will communicate the success to the rest of the agents, and immediately it will stop the search process. When any agent has exhausted its search space, it will communicate the failure to the rest. Therefore, if all the active agents return fail is because the EHC algorithm for all the agents that were collaborating during the search have pruned the branch or branches that drove to the solution.

Using the idea of FF to provide completeness to the *search* module, another agent starts searching using the *A vector* but in this case with the Best First Search (BFS) algorithm. Thanks to this algorithm it is possible to save the list of nodes previously expanded and sorted by the less cost. So if there is a solution to the problem, this will be found.

Another important difference between FF (and other HSP *Search* modules) and our approach, is that it does not have any reference to previous visited nodes. HPP contains information of all the visited nodes and their costs during the search process. If a node has been previously expanded, HPP will save time in generating all the previous expanded branches that end in a fail search, with the considerably time and space improvement (of course this difference will be more obvious in hard problems).

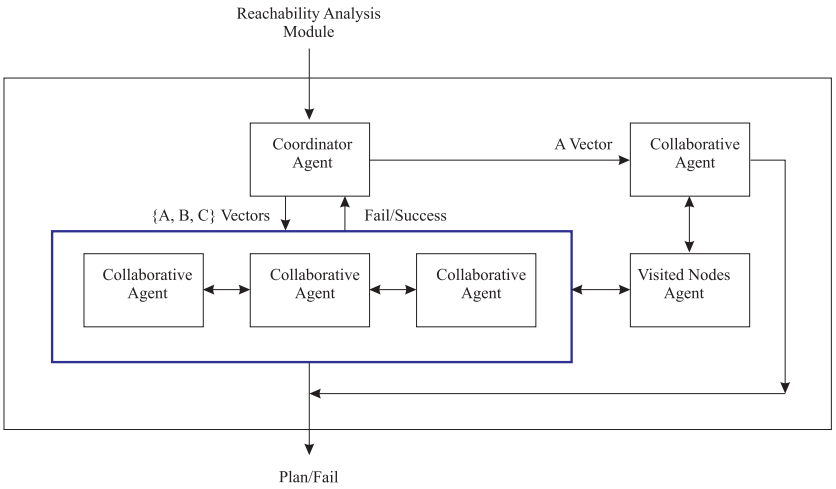

Figure 7: Search Module Architecture.

4 Experiments

In order to compare the behaviour and efficiency of the HPP planner, we have selected three planning domains from the last IPC, and several well-known planners. For these domains, several features (such as number of problems solved, number of steps or execution time) have been evaluated. Since HPP is an heuristic-based planner we have chosen two well-known FF variants: Metric-FF and Speed-FF. HSP could not be used because it did not supported the domains of the IPC'04 Competition.

To provide a complete experimental evaluation of HPP, other two planners were selected. Both planners, LPG [6] and SAPA [3] implement a hybrid heuristic *parallel* technique to implement the search process. These planners were selected because they are planners that use an heuristic module to estimate the cost and they were good competitors in the IPC'03 and IPC'04. Other planners as SATPLAN [12] or SGPlan [13] that were good competitors in the last IPC, have not been chosen because they are not heuristic search planners and we want to compare our system against similar planners to be fair with the comparison and test the new module added to the planner. CRIKEY [7], although is a heuristic planner and incorporates a STP for time reasoning, we have not compared HPP against CRICKEY because HPP does not yet incorporate this feature.

The main features of these planners can be summarized as follows:

- LPG search strategy uses Temporal Action graphs (TA-graph) and some heuristics to solve inconsistencies from the current TA-graph. The evaluation is based on the estimated number of search steps required to reach a solution (a valid plan), its estimated makespan, and its estimated execution cost. Given that LPG bases its search in a non-deterministic local search, we have run five times each problem, and considered the third best solution found. It will be considered the baseline to compare the results.

- SAPA is a domain-independent heuristic forward chaining planner that can handle durative actions, metric resource constraints, and deadline goals. It employs a set of distance based heuristics to control its search. It uses admissible heuristics for objective functions based on makespan and slack to consider optimisation factors. The heuristics are derived from the *relaxed temporal planning graph structure*, which is a generalization of planning graphs to temporal domains. The temporal graph is built by increasing incrementally the time (makespan value) of the graph.

From the seven domains proposed in the International Planning Competition of 2004 (IPC'04), we have discarded: UMTS and Settlers. In the UMTS domain the objective is to minimise the time and it is oriented to optimal planners (HPP is not). The Settle domain was a difficult domain for all the IPC'04 planners so most of them solved very few problems.

From the other five domains, we have just randomly selected three: *Pipesworld*, the *Satellite* and the *PSR* domain because of the lack of space to show the results in all the domains. The time bound for all the problems is 150 seconds.

In the Pipesworld domain, we have not considered the complex domain version, that is, with goal deadlines, because HPP does not support yet the PDDL2.2 version. In this domain the objective is to control the flow of oil through a pipeline network, using several constraints. Table 5 shows the number of problems solved by each planner. As it can be seen, HPP solves the fifty problems (100%).

Table 5: Number of problems solved in the pipeline_notankage_nontemporal domain by each planner.

Planner	Metric-FF	FF-Speed	LPG	SAPA	HPP
Percentage	72%	98%	50%	12%	100%

Figure 8 shows the results when the planners are compared by the number of steps. HPP finds better solutions in the 54% of the problems, FF-Speed in the 24%, Metric-FF in the 16% and LPG in the 6%.

From the results shown on Figure 9, we see that HPP finds faster solutions in the 38% of the problems, Metric-FF in the 44% and FF-Speed in the 18%.

The second domain, *Satellite*, it was already used in the IPC'03 competition, the goal is to collect image data with the satellites available in the problem. Table 6 shows the number of problems solved by

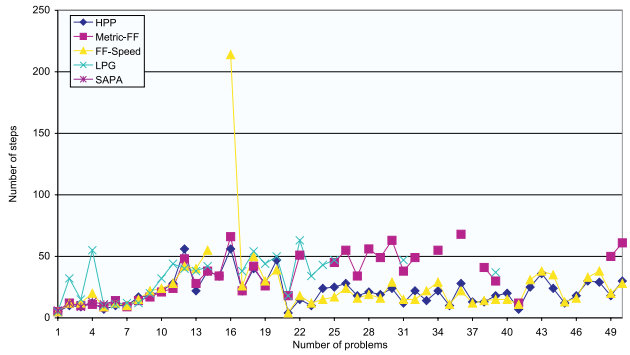

Figure 8: Results in the pipesworld domain comparing the number of steps.

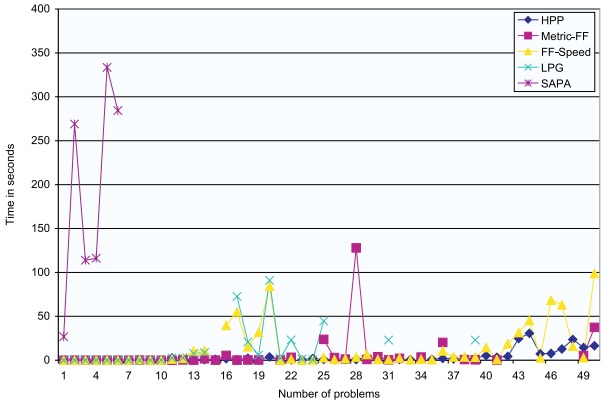

Figure 9: Results in the pipesworld domain comparing the execution time.

each planner. HPP solves the higher number of problems, 27 of a total of 36 problems as Metric-FF, then FF-Speed and LPG and finally, SAPA.

Table 6: Number of problems solved in the satellite domain by each planner.

Planner	Metric-FF	FF-Speed	LPG	SAPA	HPP
Percentage	75%	72%	72%	61,11%	75%

Figure 10 shows the results when the planners are compared by the number of steps. HPP finds the better solutions in all the problems solved.

As mentioned before, the time bound given to each planner to solve the problems is 150 seconds. From the results shown on Figure 11, we see that LPG finds faster solutions in the 66% of the problems, Metric-FF in the 22% and HPP in the 12%. But it is worth while to mention that in the 50% of the problems (until problem 19) in all the planners except SAPA, the execution time differs in a **few milliseconds**.

The last domain is *PSR*. It consists of a number of lines in an electricity network where the flow of electricity through the network is given by a transitive closure over the network connections that depends on the states of the switches and electricity supply devices.

Table 7 shows the number of problems solved by each planner. SAPA is not shown because it does not solve any problem. HPP solves again the higher number of problems 44 of a total of 50 problems. Then, Metric-FF, LPG and FF-Speed.

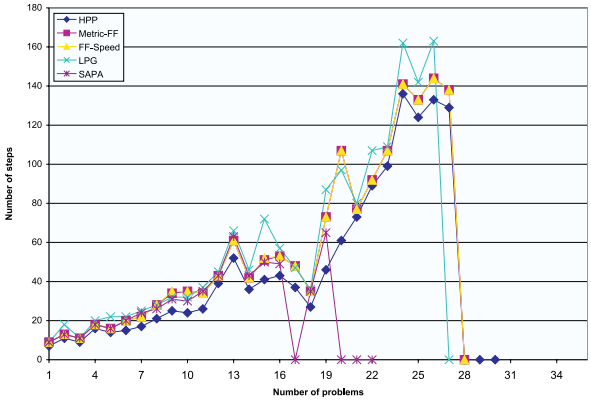

Figure 10: Results in the satellite domain comparing the number of steps.

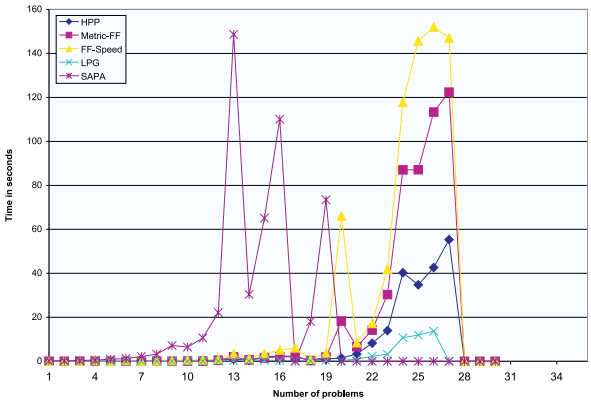

Figure 11: Results in the satellite domain comparing the execution time.

Table 7: Number of problems solved by each planner in the PSR domain.

Planner	Metric-FF	FF-Speed	LPG	HPP
Percentage	86%	26%	54%	88%

The results when the planners are compared by the number of steps in this domain are as follows. HPP, FF-Seed and Metric-FF find the same number of steps in all the problems solved by them. LPG presents the worse performance.

Finally, Figure 12 shows the solutions found comparing the execution time.

From the results, we see that Metric-FF finds faster solutions in the 64% of the problems, FF-Speed in the 18%, LPG in the 10%, and HPP in the 8%. Again, it is worth while to mention that in the 53% of the problems, in all the planners the execution time differs in a **few milliseconds**.

Previous experimental results shows both main experimental contribution. On the one hand, HPP is able to solve more problems than the rest of the considered planners, and on the other hand, HPP is able to find the best quality solution founded by any other planner.

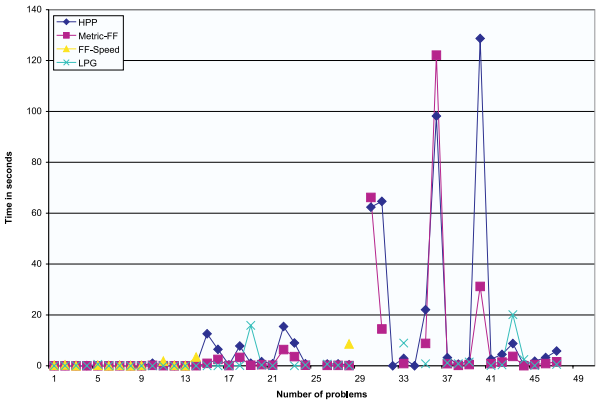

Figure 12: Execution time in the PSR domain.

5 Conclusions

This paper has presented a new heuristic planer, named HPP. This new fast heuristic planner is able to solve more and complex problems that its progenitor FF. The only module that heritage exactly as its father is the *Heuristic* module. HPP adds a new module called *Reachability Analysis* in charge of reducing the number of invalid operators that can be applied in each search step. The module uses the relaxed GRAPHPLAN of the *Heuristic* module to this purpose. Another new feature that HPP incorporates in the *search* module is the collaborative multi-agent system that explores in parallel subsets of the search space. HPP also has the capability to *remember* past situations that can help to discard already failed visited nodes.

From the experimental results shown in Section 4, HPP outperforms in the parameters measured (number of problems solved and the quality of the solution measured using the number of steps or plan length) to all the considered planners in the three selected domains from the IPC 2004 competition.

Currently, HPP has been implemented in C# under the .NET Windows platform. We believe that the execution time can decrease several magnitude orders under the Linux Operative Systems (note that the other planners we run under Linux). As soon as the Mono version will be available, we will repeat these experiments and compare HPP and the other planners. We are sure that we can also outpeform the execution time if HPP is run under the same conditions.

Finally, in the next future, we want to add other features like time and resource reasoning capabilities to HPP following the ideas sketched in the IPSS planner [11]. IPSS is a hybrid planner composed of two systems executing in parallel. The IPSS planner reasoner is a HSP predecessor so we want to substitute this module by HPP and influence the planner search to prune choices that lead to either temporal or resource inconsistencies.

Acknowledgements

This work has been funded by the UAH project PI2005/084. The authors would express their acknowledge to Javier Andrés for his work in the implementation of the initial version of the HPP planner.

References

[1] Blum, A., and Furst, M. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence 90* (1997), 281–300.

[2] Bonet, B., and Geffner, H. Planning as Heuristic Search: New results. In *Procs. of the ECP-99. Springer* (1999).

[3] Do, M. B., and Kambhampati, S. *Jounal of Artificial Intelligence Research 20* (2003), 155–194.

[4] Edelkamp, S., and Hoffmann, J. pddl2.2: The Language for the Classical Part of the 4th International Planning Competition. Tech. Rep. Technical Report No. 195, 2004.

[5] Fikes, R., and Nilsson, N. strips: A new Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence 2* (1971), 189–208.

[6] Gerevini, A., Saetti, A., and Serina, I. Planning through Stochastic Local Search and Temporal Action Graphs. *Jounal of Artificial Intelligence Research 20* (2003), 239–290.

[7] Halsey, K., Long, D., and Fox, M. crikey - A temporal Planner Looking at the Integration of Scheduling and Planning. In *Procs. of the Workshop on Integrating Planning into Scheduling.* (2004).

[8] Hoffmann, J. The Metric-ff Planning System: Translating Ignoring Delete Lists to Numerical State Variables. *Journal of Artificial Intelligence Research* (2002).

[9] Newell, A., and Simon, H. *GPS: A Program that Simulates Human Thought.* in Computers and Thought. Ed. Feigenbaum, E. and Feldman, J. McGraw-Hill., 1963.

[10] Pednault, E. adl: Exploring the Middle Ground between strips and the Situation Calculus. In *Procs. of the 1st Conference on Principles of Knowledge Representation and Reasoning* (1989), pp. 324–332.

[11] R-Moreno, M. D. *Representing and Planning tasks with time and resources.* PhD thesis, Universidad de Alcalá, Spain, 2003.

[12] Selman, B., and Kautz, H. Unifying SAT-based and Graph-based Planning. *Procs. of the IJCAI-99.* (1999).

[13] Wa, B. W., and Chen, Y. Subgoal Partitioning and Global Search for Solving Temporal Planning Problems in Mixed Space. *Internaional J. of Artificial Intelligence Tools, 13*, 4 (December 2005), 767–790.