Variable resolution planning through predicate relaxation

Moisés Martínez
Universidad Carlos III de Madrid
Avenida de la Universidad, 30
28911 Leganés, Spain
moises.martinez@uc3m.es

Fernándo Fernández
University Carlos III de Madrid
Avenida de la University, 30
28911 Leganés, Spain
fernando.fernandez@uc3m.es

Daniel Borrajo
University Carlos III de Madrid
Avenida de la University, 30
28911 Leganés, Spain
dborrajo@ia.uc3m.es

Abstract

Planning a sequence of actions is particularly difficult in stochastic environments, where actions execution might fail, which in turn prevents the execution of the rest of the plan. Also, in many domains, agents cannot wait a long time for plan generation. In this paper, we propose the use of Variable Resolution Planning (VRP). The main idea is based on the fact that if the domain is stochastic it is usually not worth computing a valid (sound) long plan, since most of it will not really be used. So, VRP generates a plan where the first k actions are applicable if the environment does not change, while the rest of the plan might not be necessarily applicable, since it has been generated using an abstraction by removing some predicates. Also, planning with abstraction requires less computation time than computing regular applicable plans. Experimental results show the advantages of this approach in several domains over a standard planning approach.

Introduction

Stochastic environments are challenging for Automated Planning, where a planner has to generate a plan of actions, and the plan execution may yield unexpected states. This process can be prohibitively expensive for most real world scenarios. Besides, depending on the stochasticity level of the domain, most of the actions of the generated plan will not be even applied when the execution of the previous actions in the plan generate an unexpected state. In the worst case an unexpected state can be a failure state from which the rest of the plan cannot be successfully executed.

There are different ways to approach planning and execution in stochastic domains. Different techniques can be applied depending of the information known about the environment. If we have information about the dynamics of the environment (failure in the actuators of a robot, the structure of the terrain, accuracy of sensors, etc), we can define a domain model with probabilistic information (such as in PPDDL). Then, we can build a conditional plan (Peot and Smith 1992) where the plan takes into account all possible alternative problems, or generate a set of policies by solving it as a Markov Decision Process (MDP) as shown by LAO* (Hansen and Zilberstein 2001), or LRTDP (Bonet and Geffner 2004). But, usually, the dynamics of the environment is not known. Then, in turn we have two alternatives. First, we can learn the dynamics. However, usually the learning effort is impractical except for small problems (Zettlemoyer, Pasula, and Kaelbling 2005). The second solution, and most used one, consists of using a deterministic domain model and replan when a failure in execution is detected.

A common solution is using a repair or re-planning system (Fox et al. 2006; Yoon, Fern, and Givan 2007; Borrajo and Veloso 2012). In re-planning, the planner generates an initial applicable plan and executes it, one action at a time. If an unexpected state is detected, the system generates a new plan. This process is repeated until the system reaches the problem goals. Therefore, at each planning (re-planning) step, including the initial one, the system is devoting a huge computational effort on computing a valid plan (applicable plan that achieves the goals), when most of it will not be applied. On the other extreme, reactive systems require much less computational effort. They greedily select the next action to be applied according to some configured - or learned - knowledge. They are "mostly" blind with respect to the future; they usually ignore the impact of the selected action on the next actions and states. Thus, they often get trapped in local minima, or dead-ends.

We advocate here for an approach in which a deterministic planner devotes less time to compute a valid first portion of the plan, and checks that there is a potential continuation of the plan by means of abstractions, trying to avoid dead-ends and/or decreasing computational effort. One of the advantages of our approach is that it requires less planning time than traditional planning approaches that compute a valid complete plan, while retaining their capability of reasoning into the future. As explained before, we assume lack of information about the dynamics of the environment, so we use a deterministic STRIPS domain model and replan upon failure.

To design this approach, we have been inspired by the work of Zickler and Veloso (Zickler and Veloso 2010), where a motion planning technique is presented. It generates a collision-free trajectory from an initial state to a goal state in dynamic environments. They introduced the concept of Variable Level-Of-Detail (VLOD), which focuses search on obtaining accurate short-term motion planning, while con-

sidering the far future with a different level of detail, by selectively ignoring the physical interactions with dynamic objects. This technique decreases the computational cost of the motion planning process, so that information about different elements of the environment is not used to search a path to reach all goals. We introduce Variable Resolution Planning (VRP) through predicate relaxation, a new approach to reduce the computational overhead of planning in stochastic environments. It is based on two ideas: some effort is devoted to compute a valid head of the plan of length k; and the rest of the plan is checked for potential reachability by relaxing the complexity of actions and decreasing domain details through abstraction. Our approach is based on selectively ignoring some domain details, in our case predicates.

This paper is organized as follows: first in Section 2, we formally define the representation of the planning problem. In Section 3 we define our abstraction mechanism. In Section 4 we describe how to implement Variable Resolution Planning (VPR). In Section 5 we show an experimental evaluation of our techniques over different domains. Section 6 presents some work related with our approach. Finally, in Section 7 we conclude and introduce future work.

Planning Framework

In this work, we focus on the use of a classical Automated Planning approach under stochastic environments. A classical planning problem is defined in PDDL using a lifted representation in predicate logic, but most current planners always perform first a grounding transformation. Under that transformation, a planning problem can be defined as a tuple $P = (F, A, I, G)$, where:

- F is a finite set of relevant grounded literals (also known as facts), functions and fluents.
- A is a finite set of grounded actions derived from the action schemes of the domain, where each action $a_i \in A$ can be defined as a tuple $a_i = (Pre, Add, Del)$, where $Pre(a_i), Add(a_i), Del(a_i) \subseteq F$, $Pre(a_i)$ are the preconditions of the action, $Add(a_i)$ are its add effects, and $Del(a_i)$ are the delete effects. $\mathit{Eff}(a_i) = Add(a_i) \cup Del(a_i)$ are the effects of the action.
- $I \subseteq F$ is the initial state.
- $G \subseteq F$ is a set of goals.

A plan π for a problem P is a set of actions (in the common case a sequence) $\pi = (a_1, \ldots, a_n), \forall a_i \in A$, that transforms the initial state I into a final state S_n where $G \subseteq S_n$. This plan π can be executed if the preconditions of each action are satisfied in the state in which it is applied, i.e. $\forall a_i \in \pi\, Pre(a_i) \subseteq S_{i-1}$ ($S_0 = I$).

Abstractions

An abstraction can be defined as a function that transforms a planning problem into another (simpler) one, where low-level details are ignored. Usually, abstractions help on reducing the problem complexity. In this work, we generate abstractions by removing some predicates from the lifted representation of the domain. Thus, since planners work at the propositional (instantiated) level, it is equivalent to remove literals from the preconditions and effects of actions in the grounded representation. We will now define some concepts related to our abstraction mechanism.

The first definition defines a mapping between a predicate in the PDDL domain definition and its corresponding groundings for problem P.

Definition 1 *$g(p, P)$ is the set of propositions (facts) of predicate p in problem P.*

For instance, in the Blocksworld domain, if p is `(ontable ?x)` and a PDDL problem P_1 defines three blocks (A, B, and C) that are on the table, then:

$$g(p, P_1) = \{\texttt{(ontable A), (ontable B), (ontable C)}\}$$

In this approach, an abstraction of an action a over a predicate p removes all propositions that are groundings of predicate p – $g(p, P)$ – from the preconditions and effects of action a. In order to select which predicates to remove, we split the set of facts F into three subsets: $F_d \subseteq F$ is the set of dynamic facts, or facts that appear in the effects of at least one action ($F_d = \{f \in F \mid \exists a_i \in A, f \in \mathit{Eff}(a_i)\}$). $F_s \subseteq F$ is the set of static facts, or facts that do not appear in the effects of any action; and $F_f \subseteq F$ is the functions set, or non-boolean facts that are groundings of PDDL functions instead of PDDL predicates. We are not going to remove all domain predicates (or their corresponding facts), so we define the candidate subset of facts to be potentially removed, $C^{abs} \subseteq F$. We will not use all predicates to generate abstractions. First, static predicates are not added or deleted during the planning process. Therefore, applying an abstraction over a static predicate does not reduce the number of actions to achieve the goals. Deleting an static predicate only increases the number of actions that may be selected during the search process to reach the goal state. Second, predicates which are part of the goal subset cannot be removed, because the planner would not reach a solution. And, finally, we are only considering STRIPS domains in this paper. C^{abs} is composed of all facts except for the static facts (F_s), the functions facts (F_f) and facts that are part of the problem goals (G). Thus:

$$C^{abs} = F \setminus (F_s \cup G \cup F_f)$$

Next, we include our definitions of abstractions:

Definition 2 *An abstraction of an instantiated action $a \in A$ over a predicate p in problem P is defined by function $f(a, p) = (Pre_p^{abs}(a), Add_p^{abs}(a), Del_p^{abs}(a))$, where:*

- $Pre_p^{abs}(a) = Pre(a) \setminus g(p, P)$
- $Add_p^{abs}(a) = Add(a) \setminus g(p, P)$
- $Del_p^{abs}(a) = Del(a) \setminus g(p, P)$.

For instance, in the Blocksworld domain, given a problem P_1 previously defined, if a_1 is `pick-up(A)` and predicate p_1 is `(ontable ?x)`. If we select to remove p_1, $g(p_1, P_1) = \langle (ontableA), (ontableB), (ontableC) \rangle$.

- $Pre_p^{abs}(a_1) = \langle (clearA)(ontableA)(handempty) \rangle \setminus \langle (ontableA), (ontableB), (ontableC) \rangle$

- $Add_p^{abs}(a_1) = \langle(holdingA)\rangle \setminus \langle(ontableA), (ontableB), (ontableC)\rangle$
- $Del_p^{abs}(a_1) = \langle(ontableA), (clearA), (handempty)\rangle \setminus \langle(ontableA), (ontableB), (ontableC)\rangle$.

Definition 3 *An abstraction of an instantiated action a over a set of predicates $L = \{p_1, \ldots, p_n\}$ in problem P is the result of iteratively abstracting action a over each predicate $p_i \in L$. The process can be recursively defined as:*

- $f(a, \{p\}) = f(a, p)$
- $f(a, \{p_1, \ldots, p_n\}) = f(f(a, \{p_2, \ldots, p_n\}), p_1)$

L will always be composed of PDDL predicates (in the lifted representation) whose corresponding facts (groundings) belong to C^{abs}; i.e. candidate elements of L will be in the set of candidate predicates $CL = \{p \mid g(p, P) \subseteq C^{abs}\}$.

Definition 4 *An abstraction of a PDDL problem P over a set of predicates $L \subseteq CL$, called P_L^{abs}, is the result of abstracting all components of P over L:*

$$P_L^{abs} = (F_L^{abs}, A_L^{abs}, I_L^{abs}, G_L^{abs})$$

where

- $F_L^{abs} = F \setminus \cup_{p \in L} g(p, P)$
- $A_L^{abs} = \{a^{abs} \mid a \in A, a^{abs} = f(a, L)\}$
- $I_L^{abs} = I \setminus \cup_{p \in L} g(p, P)$
- $G_L^{abs} = G$, *given that the predicates in the goals are not candidates to abstract*

Definition 5 *An abstract plan Π_L^{abs} that solves a PDDL problem P is an action sequence $\Pi_L^{abs} = \{a_0, \ldots, a_{k-1}, a_k, \ldots a_{n-1}\}$. The k first actions are applicable in P if the actions $a_i, i = 0 \ldots k$, are all in A, and there exists a sequence of states $(s_0, \ldots, s_k)$, such that $s_0 = I$, $Pre(a_i) \subseteq s_i$ and $s_{i+1} = s_i \cup Add(a_i) \setminus Del(a_i)$ for each $i = 0, \ldots, k-1$. The rest of actions from a_k are applicable in an abstract P_L^{abs} if the actions $a_i, i = k \ldots n$, are all in A_L^{abs}, and there exists a sequence of states $(s_k, \ldots, s_n)$, such that $Pre^{abs}(a_i) \subseteq s_i$ and $s_{i+1} = s_i \cup Add^{abs}(a_i) \setminus Del^{abs}(a_i)$ for each $i = k, \ldots, n$. Thus, a partial abstract plan is composed of a standard sound plan from the initial state s_0 to state s_k and an abstract plan from state s_k to a goal state.*

Variable Resolution Planning

The goal of VRP in stochastic domains is to quickly come up with a plan whose first k actions are sound and whose rest of actions could potentially solve the problem by adding the removed details. Thus, we will generate a search tree where nodes of depth less or equal to k use the original definition of the problem, and nodes below that depth will use an abstracted version of the problem. In order to use current planners under this approach, modifications in the planners code are necessary. In this work, Metric-FF (Hoffmann 2003) has been used as a base to implement our approach. We call the new planner Abstract-MFF. Apart from the standard inputs of Metric-FF, Abstract-MFF also receives k and L as input. Once the initial instantiation is performed (computing the instantiated problem and all its components, $P = (F, A, I, G)$), Abstract-MFF generates data structures to use horizon h and abstractions. It creates two different spaces for instantiated actions. The first space S is composed of the actions that can be executed for the standard problem P. The second space S^{abs} is composed of the actions that can be executed for the abstracted problem P^{abs}. Next, the planner starts the search process using ωA* (ω=5), when the depth of a node is greater than the Temporal Horizon, k, the abstract space, P^{abs}, is activated for all the nodes in the subtree of that node while the original space, P is deactivated. The implementation basically changes the space corresponding to the original problem P for the one corresponding to the abstracted problem P^{abs}. So, the nodes above depth k use S and the ones below that threshold use S^{abs}. Finally, the planner generates an abstract plan.

We used, without success, alternative ways of performing this implementation through compiling the dynamic change from the original to the abstract space into the PDDL domain model and searching on that space. For space reasons, we cannot expand here on why it did not work, but it had to do with generating bigger problem spaces (complexity), or representation capabilities of current planners (expressivity).

Experimentation

To evaluate the approach presented in this paper, we have performed a planning-execution-replanning loop on some domains and problems. We have used MDPSim for simulation of plans execution.[1] We provide it with a probabilistic version of each domain in PPDDL (Younes and Littman 2004), where actions can generate unexpected states with different probabilities. In our case an state is considered as an unexpected state when an action could not be executed in the environment and the execution returns the previous state. We have chosen the most simple case to define failures, because we did not have access to a simulator that is able to introduce exogenous effects and MDPSim does not offer a way to include them. For instance, in the Rovers domain when a robot moves from a waypoint1 to a waypoint3, three possibilities can be expected in the real world. First, the robot moves to the right waypoint (the one on the effects of the deterministic version of the actions). Second, the robot does not move, so it stays in the same waypoint. Finally, it can move to another waypoint close to the origin or destination ones.

The overall system works as shown in the algorithm in 1. Given a planning task (consisting of a deterministic PDDL domain and problem descriptions), L (the set of predicates to be removed) and k (the temporal horizon), the planner generates an abstract plan (where the first k actions are non-abstracted actions and the rest are abstracted actions). Next, it sends every action to MDPSim. MDPSim executes each action (with the stochastic model of the PDDL domain). If the resulting state is not the expected one according to the

[1] It was developed for the First Probabilistic Planning Competition (Younes et al. 2005).

Algorithm 1: Description of the execution process in the stochastic environment.

Data: Problem: $\Pi = (F, A, I, G)$, predicates L, horizon h

begin

$plan = planning(\Pi, L, h)$

$lastState = I$

while *plan is empty at the current state do* **do**

$action \longleftarrow getAction(plan)$

$expected \longleftarrow$ $generateState(action, lastState)$

$state \longleftarrow sendActionToMDPSIM(action)$

if $state <> excepted$ **then**

$\Pi \longleftarrow$ $generatePlanningProblem(\Pi, lastState)$

$plan = planning(\Pi, L, h)$

else

$lastState \longleftarrow state$

end

deterministic version of the domain, we generate a new plan from that state. This process is repeated as a re-planning loop until a goal state is reached. We compare Abstract-MFF with the standard Metric-FF planner.

Experiments have been done on an Intel Xeon 2.93 GHZ Quad Core processor (64 bits) running under Linux. The maximum available memory for the planners was set to 6 GB, the maximum planning time for a problem has been set to 1000 seconds and the maximum execution time has been set to 30000 seconds. Each problem has been executed fifteen times until goals are reached (we call it a run). The following provides an evaluation of this approach over four benchmark domains: Rovers, DriverLog, Depots and Satellite domains from IPC-3. First, we present a deep analysis on the Rovers domain to study the different effects of this technique changing several characteristics. And second we show the results on the rest of domains.

Rovers Domain

The technique presented in this paper has been designed to be applied in stochastic environments. For this reason, the initial testing of this technique has been performed on a domain generated for a real world problem, the Rovers Domain. It was designed for the sequential track of IPC-3 (2002) and was inspired on the Mars exploration rovers missions where an area of the planet is represented as a grid of cells, called waypoints. They contain samples of rock or soil that can be collected by the robots. Each robot can traverse across different waypoints and can perform a set of different actions (analyze rock or soil samples or take pictures of a specific waypoint). All data collected by robots has to be sent to the lander, that is placed in a specific waypoint. Taking in account the large number of possibilities, we have selected a subset of significant values for the different parameters (horizon and predicate) and of experiments variables (probability of any action failure) to test our technique:

- We have considered four predicates for removal: `have_image`, `have_soil_analysis`, `have_rock_analysis` and `at`. In this paper, we only show results for predicates `have_rock_analysis` and `at`, because the results generated with the predicates `have_image`, `have_soil_analysis` are similar to those obtained by removing the predicate `have_rock_analysis`.
- We have selected four different horizons for each predicate. $k = 3, 5, 10$ and 20.
- The probability of an action execution failure is included in the definition of the action in then PPDDL domain. We show results with 60% and 30% of failure probability. Note that Abstract-MFF does not have access to the probabilistic version of the domain.

In the Rovers domain, Abstract-MFF greatly reduces the computational cost over Metric-FF when predicate `have_rock_analysis` is removed. Table 1 and 2 compares the planning time (average of all runs of the sum of all planning times during the whole re-planing loop of each run) when using abstraction with different values of k and a different probability of failure.

On one hand, we have observed that removing a predicate that is part of the preconditions of the actions that achieve the problem goals provides better performance in difficult problems, regardless of the level of stochasticity of the environment. This reduces the difference on time between the original planner and Abstract-MFF. Abstracting this kind of predicates allows the planner to achieve goals more easily, reducing the number of actions required to find a state where goals are true. For instance, if we remove the predicate `have_rock_analysis` in the action comunicate_rock_data presented in Figure 1, the number of actions necessary to get a rock sample and send it to the lander will decrease, because the robot does not have to move to the waypoint where the rock is and it does not have to pick the rock sample up to analyze it; the rover only needs to send information to the lander.

```
(:action comunicate_rock_data
   :parameters (?r - rover ?l - lander ?p - waipoint
?y - waypoint)
   :precondition
   (and
      (at ?r ?x) (at_lander ?l ?y)
      (have_rock_analysis ?r ?p) (visible ?x ?y)
      (available ?r) (channel_free ?l)
   )
   :effects
   (and
      (not (available ?r)) (not (channel_free ?l))
      (chanel_free ?l) (communicate_image_data ?o ?m)
      (available ?r)))
```

Figure 1: Action comunicate_rock_data in PDDL.

On the other hand, in Tables 3 and 4 compare planning time when predicate `at` is removed. In this case, results are worse than those for the other predicate. Our initial assump-

tion was that predicates like `at` in domains where a robot or a vehicle has to move through different positions would offer better performance than the rest of predicates. But, in the Rovers domain this is not the case. Thus, the decision on which predicates to remove is important to obtain a good performance.

Regarding the value of k, the results shown in the different tables for the Rovers domain indicate that there is no general rule about the influence of the value of k in the performance of the planner. In some problems, lower values of k are better and in others it is the contrary. We hypothesize that the efficiency also depends on the percentage of action failures or the problem to solve. Initially, we expected that small values of k tend to increase the number of re-planning steps, since a bigger part of the plan is abstracted, but planning time will be less, given that the plan is shorter on each planning episode. However, if the percentage of actions failures is high, the value of k does not affect much the number of planning episodes. If a failure appears on the first actions of the plan, the value of k only decreases the planning time, but it generates a similar number of planning episodes.

In general, results obtained in the Rovers domain confirm our initial hypothesis: in problems where planning time is high, abstractions significantly reduce planning time. The first group of problems are moderately difficult and abstraction does not show a significant reduction of planning time. But, in hard problems as 25 or 27, Abstract-MFF obtains a reduction of the total planning time of 60% when removing predicate `have_rock_analysis`. In addition, when two predicates (`have_rock_analysis, have_image`) are used to generate abstractions in the Rovers domain 5, the planning time is reduced in an order of magnitude. We would like to explore in the future other combinations of predicates to remove.

Results in Other Domains

We report the total planning time, as well as the mean of replanning steps. We have selected some benchmark domains: Depots, DriverLog and Satellite domains from IPC-3 and Gold-Miner domain from the learning track of the IPC-6. For each domain, we have selected two problems, two predicates to remove and four different horizons ($k = 3, 5, 10$ and 20). The probability of an action execution failure has been set to 30%.

Table 6 shows the results. Our approach obtains good results in easy and difficult problems. On one hand, it has good performance in easy problems (13 and 16) in the Depots domain; it needs less planning time if it abstracts predicates `in` and `available`. A similar behaviour can be observed in the Satellite domain, whose results present a similar performance as in the Rovers domain, where it obtained good results in hard problems.

On the other hand, in the Gold Miner domain, abstractions can solve problems avoiding dead-ends. When predicate `robot-at` is removed, the search process can avoid dead-ends when the abstraction is applied before the dead-end appears. Removing this predicate allows the robot to execute any action at every location, decreasing the complexity of the problem. However, if predicate `holds-bomb` is removed, problems can only be solved when the value of k is high, because the predicate `holds-bomb` indicates when a robot is holding a bomb. If the number of actions to hold a bomb is greater than k, the robot can enter in a cycle using the abstraction to reach the goals in the last part of the plan.

Finally, these abstractions do not offer good results in all problems. If we remove the predicate `in` in the DriverLog domain, the cost of solving the problem increases, because predicate `in` defines the position of the trucks and how they move among different locations. When it is deleted, it is possible to move to any location from every other location. Thus, we hugely increase the branching factor. We observed a similar behavior in the Gold Miner Domain, when predicate `robot-at` is removed.

Related Work

This work focuses on applying abstractions over Automated Planning to reduce the computational overhead in stochastic or dynamic environments. There have been already many approaches that generate abstractions. We review some works based on abstraction in classical automated planning and motion planning, which is closely related to our work. The first work in abstractions (Sacerdoti 1972) extended the work of Newell and Simon on GPS and used it to develop Abstrips, which combined abstraction with STRIPS. They defined abstraction levels by assigning criticalities to predicates, which define the difficulty of achieving them. The planner used these criticalities to iteratively generate successive abstract plans using only predicates in the corresponding abstract space. Alpine (Knoblock 1991) automatically generates abstraction hierarchies, using the preconditions of operators. In both cases abstractions are used to generate an abstract plan. This is refined incrementally until the lowest level of criticality is expanded and goals have been satisfied. Instead, our approach generates an abstract plan the first k where actions are valid ones, while the rest of the plan is not necessarily valid.

In recent years, abstractions have been used to generate heuristic techniques. Hoffmann and Nebel (Hoffmann 2003) used abstractions to compute the heuristic value of nodes by building a relaxed plan to guide the search process, where the delete effects of actions are ignored. Another use of abstractions to build heuristics are pattern databases (PDBs), which have been shown to be very useful in several hard search problems (Culberson and Schaeffer 1998) and Automated Planning (Edelkamp 2001). VRP has been designed to generate abstract plans in stochastic environments by decreasing computation time regardless of the technique used to guide the search process.

More recently, changes in the representation have been used to automatically generate finite-state controllers from models (Bonet, Palacios, and Geffner 2009). This represents a kind of contingent problems where actions are deterministic and some fluents are observable. The controllers could be considered general solvers in the sense that they do not solve only the original problem, but also can include changes regarding the size of the problem or the probabilities of the action effects.

Prob	Metric-FF			AMFF (k=3)			AMFF (k=5)			AMFF (k=10)			AMFF (K=20)		
	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT
20	91	6	2	68	10	1	80	10	2	**53**	4	1	57	6	1
21	299	45	12	292	54	3	269	20	3	**91**	5	3	167	7	4
22	819	35	15	**272**	14	4	308	43	3	311	15	4	349	18	4
23	2676	60	32	**687**	190	8	542	55	7	975	76	6	1149	61	7
24	408	259	17	292	58	5	307	29	5	**231**	21	7	271	20	4
25	19414	2759	263	**4392**	269	33	4712	567	74	6340	715	78	7751	398	60
26	2522	229	25	1970	327	21	2125	587	14	**648**	42	16	895	43	52
27	24062	5984	237	6981	1961	61	6975	1681	80	**4858**	1215	42	4879	777	34
28	11848	543	228	3308	313	35	3641	559	38	**3565**	465	32	4451	518	35
Total	62143			18257			18967			**17086**			20293		

Table 1: Planning time for the Rovers domain when removing predicate `have_rock_analysis` with a 60% probability of failure. The first column corresponds to one IPC problem of the Rovers domain, the second column corresponds to Metric-FF and the rest corresponds to Abstract-MFF with different values of k. For each planner, each column shows the average of the sum of planning times of each run in seconds, the standard deviation (SD) in seconds and the time of the first planning process (FT) in seconds. In bold, we highlight the best results per row.

Prob	Metric-FF			AMFF (k=3)			AMFF (k=5)			AMFF (k=10)			AMFF (K=20)		
	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT
20	32	4	2	**20**	3	1	26	6	2	21	10	1	27	9	1
21	**52**	16	12	97	9	3	64	20	3	52	40	3	69	22	4
22	192	26	15	100	11	4	**83**	14	3	99	46	4	108	39	4
23	528	193	32	260	29	8	**153**	21	7	177	113	6	170	71	8
24	136	10	17	104	10	5	90	12	6	**85**	49	7	104	37	5
25	9657	932	263	1880	464	35	**1564**	219	72	1567	813	78	1796	621	58
26	680	42	25	637	42	23	**599**	160	15	662	329	16	617	292	54
27	4581	674	237	2187	438	62	1874	585	78	**1554**	887	44	1989	621	35
28	3132	223	168	1076	250	33	949	175	39	**929**	461	31	1181	362	36
Total	18983			6365			5385			**5149**			6065		

Table 2: Planning time for the Rovers domain when removing predicate `have_rock_analysis` with a 30% probability of generating an unexpected state. The meaning of columns is the same as previous table.

The concept of abstraction has been explored across many other areas of AI. Our work is inspired by the work of Zickler and Veloso (Zickler and Veloso 2010) in motion planning. This technique generates a collision-free trajectory from an initial state to a goal state in dynamic environments. This work considers the far future with a different level of detail, selectively ignoring the physical interactions with dynamic objects of environment. We try to apply a similar idea in automated planning and analyze effects of ignoring any far future information about the environment and decrease complexity of the problem to get a better performance.

Conclusions

In this paper, we have presented Abstract-MFF, a planner based in Metric-FF that uses an abstraction mechanism that dynamically removes some predicates during the planning process based on a temporal horizon, in order to improve planning performance in stochastic environments. The main contribution of this work consists on understanding the effects of this idea over the planning process, and how it can be used to improve the application of planning techniques in real environments.

Regarding the experimentation, we observe that the planning time is most of the time less than the time required by Metric-FF. Also, the time needed to generate the first plan is always lower, which in real environments is critical, as it allows the execution to begin earlier. Besides, the cost of the execution (number of executed actions) using the presented technique is roughly the same as the one of Metric-FF, with an overall improvement of a 5%. Furthermore, the experiments hint that combining different abstracted predicates may lead to an increase in efficiency. As further work, we would like to explore which combinations of predicates are useful to be removed, automatic ways of selecting those, and changing the temporal horizon dynamically. Also, it would be interesting to select a set of problems that cannot be solved by current planners like Metric-FF (Hoffmann 2003) or LAMA (Richter and Westphal 2008) and analyze if our technique can solve these problems applying abstractions. We would also like to apply this technique to robotics. In this case, we should define the interactions between the deliberative and reactive levels taking into account the capa-

Prob	Metric-FF			AMFF (k=3)			AMFF (k=5)			AMFF (k=10)			AMFF (K=20)		
	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT
20	91	6	6	42	10	1	**39**	5	1	53	1	1	573	1	2
21	299	45	12	314	54	3	282	39	3	**121**	7	3	177	15	2
22	819	35	15	445	14	4	450	43	4	**333**	29	4	355	28	4
23	2676	60	32	1428	190	8	1530	55	4	**965**	30	6	1509	88	7
24	408	259	7	308	58	5	327	29	5	**254**	31	6	255	33	6
25	19414	2759	263	18108	269	31	12652	567	80	**6370**	285	73	7791	151	60
26	2522	129	25	1493	327	19	1418	587	18	**668**	53	12	2563	3658	52
27	24062	5984	237	7427	1961	62	10815	1681	81	**4718**	449	41	4799	382	34
28	11848	543	228	10973	313	35	9491	274	37	**3505**	313	32	4391	335	42
Total	62143			40544			37004			**16987**			22413		

Table 3: Planning time for the Rovers domain when removing predicate `at` with a 60% probability of failure. The meaning of the columns is the same as previous table.

Prob	Metric-FF			AMFF (k=3)			AMFF (k=5)			AMFF (k=10)			AMFF (K=20)		
	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT
20	32	4	2	**20**	3	1	26	6	2	21	10	1	27	9	1
21	**52**	16	12	97	9	3	64	20	3	52	40	3	69	22	4
22	192	26	15	100	11	4	**83**	14	3	99	46	4	108	39	4
23	528	193	32	312	34	18	321	84	19	**297**	42	14	301	59	15
24	136	10	17	124	11	12	129	72	13	**103**	29	10	119	46	12
25	9657	932	263	6721	424	113	6544	328	104	**5567**	743	98	5996	832	97
26	680	42	25	797	87	30	691	315	28	671	279	25	**624**	79	37
27	4581	674	237	3971	428	188	3874	498	166	**2954**	731	110	2973	621	137
28	**3132**	223	168	4184	328	179	3949	315	142	4128	537	138	3997	512	168
Total	18983			16290			15681			**13892**			14214		

Table 4: Planning time for the Rovers domain when removing predicate `at` with a 30% probability of failure. The meaning of columns is the same as previous table.

bilities and constraints of the robots and the environment, as in (Lematre and Verfaillie 2007). In our case, it will be necessary to analyze how abstractions are done when dealing with different levels of reasoning.

Acknowledgments

We extend our thanks to Vidal Alcázar and Raquel Fuentetaja for their helpful discussions. This research is partially supported by the Spanish MICINN projects TIN2008-06701-C03-03, TIN2011-27652-C03-02, TRA-2009-008 and Comunidad de Madrid - UC3M(CCG10-UC3M/TIC-5597). The first author is supported by a PhD grant from Universidad Carlos III de Madrid.

References

Bonet, B., and Geffner, H. 2004. A probabilistic planner based on heuristic search. In *In Proceedings of the Fourth International Planning Competition*.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *19th International Conference on Automated Planning and Scheduling*. Thessaloniki, Greece: AAAI Press.

Borrajo, D., and Veloso, M. 2012. Probabilistically reusing plans in deterministic planning. In *Proceedings of ICAPS'12 workshop on Heuristics and Search for Domain-Independent Planning*. Atibaia (Brazil): AAAI Press.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S. 2001. Planning with pattern databases. In *In Proceeding of the sixth European Conference on Planning*, 13–24.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling*, 212–221.

Hansen, E. A., and Zilberstein, S. 2001. Lao* : A heuristic search algorithm that nds solutions with loops. *Articial Intelligence* 129(1-2):35–62.

Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Knoblock, C. 1991. Characterizing abstraction hierarchies for planning. In *In Proceedings of the Ninth National Conference of Artifical Intelligence*.

Lematre, M., and Verfaillie, G. 2007. Interaction between

Prob	Metric-FF			AMFF (k=3)			AMFF (k=5)			AMFF (k=10)			AMFF (K=20)		
	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT	Time	SD	FT
23	2676	193	32	**532**	74	3	560	76	3	474	56	4	787	51	4
25	19414	932	163	**2244**	321	16	2468	342	31	3173	293	29	4531	534	29
27	24062	674	167	**2735**	294	11	2893	301	11	3171	264	11	5028	253	13
28	11848	223	127	**1102**	104	8	1455	123	11	1885	103	13	2343	157	14
Total	18983			**6613**			6966			8703			12689		

Table 5: Planning time for the Rovers domain when removing predicate `have_image` and `have_rock_analysis` with a 60% probability of failure. The meaning of columns is the same as previous table.

Domain	Problem	Predicate	Metric-FF	AMFF (k=3)	AMFF (k=5)	AMFF (k=10)	AMFF (K=20)
			Time	Time	Time	Time	Time
Depots	13	In	**0.90**	1.20	1.85	2.15	1.60
Depots	13	Available	**0.90**	1.31	1.78	2.21	1.92
Depots	16	In	220.10	45.30	**42.10**	108.21	131.43
Depots	16	Available	220.10	**85.25**	86.25	134.92	149.12
Satellite	27	Calibrate	12854.32	4828.85	4320.78	3745.83	**3397.15**
Satellite	27	Power On	12854.32	3951.60	2769.55	**3289.32**	3653.10
Satellite	29	Calibrate	16542.91	**4467.32**	4582.43	4832.12	5104.34
Satellite	29	Power On	16542.91	3934.42	**3582.43**	4232.12	4604.34
Driver Log	14	Empty	10.98	**4.67**	5.32	6.32	5.12
Driver Log	14	In	10.98	8.73	9.21	**8.19**	9.42
Driver Log	19	Empty	1828.52	**460.80**	878.34	793.29	620.31
Driver Log	19	In	**1828.52**	8720.43	8892.82	9621.64	9053.53
Goldminer	12	robot-at	-	308.82	235.74	**82.10**	-
Goldminer	12	holds-bomb	-	-	-	-	**45.32**
Goldminer	15	robot-at	-	395.45	318.09	**75.53**	-
Goldminer	15	holds-bomb	-	-	-	-	**89.68**

Table 6: Planning time for different domains and problems. The columns correspond to the domain, problem number, the removed predicate, the results of Metric-FF and the results of Abstract Metric-FF with different values of k. Each row corresponds to a problem of a domain. For each planner, we provide the average time of execution in seconds. In bold, we highlight the best results.

reactive and deliberative tasks for on-line decision-making. In *17th International Conference on Automated Planning and Scheduling*. Providence, Rhode Island, USA: AAAI Press.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In Kaufmann, M., ed., *In Proceedings of the First International Conference on Artificial Intelligence*, 189197.

Richter, S., and Westphal, M. 2008. The lama planner using landmark counting in heuristic search. In *Proceedings of the International Planning Competition*.

Sacerdoti, E. D. 1972. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 2(5):115–135.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*.

Younes, H. L. S., and Littman, M. L. 2004. Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects. *In Technical Report CMU-CS-04-162*.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligente Research* 24:851–887.

Zettlemoyer, L. S.; Pasula, H.; and Kaelbling, L. P. 2005. Learning planning rules in noisy stochastic worlds. In *In Proceedings of the Twentieth National Conference on Artificial Intelligence*, 911–918.

Zickler, S., and Veloso, M. 2010. Variable level-of-detail motion planning in environments with poorly predictable bodies. In *In Proceeding of the nineteenth European Conference on Artificial Intelligence*.