Selective Abstraction in Automated Planning

Moisés Martínez MOISES.MARTINEZ@UC3M.ES
Fernando Fernández FERNANDO.FERNANDEZ@UC3M.ES
Daniel Borrajo DBORRAJO@IA.UC3M.ES
University Carlos III de Madrid, Avenida de la Universidad, 30 28911 Leganés, Spain

Abstract

One of the most studied areas of human reasoning from a computational point of view has been planning. Classical Automated Planning (AP) deals with the generation of ordered sets of actions that, when correctly executed, allow an agent (or set of agents) to transit from an initial state to a final state where a set of goals are satisfied. We are interested on building intelligent systems that sense, reason (plan) and execute plans in the real world. Since most real environments are stochastic and/or dynamic, we focus on planners that handle the extra complexity of considering non-determinism and/or dynamism. Currently, the two major approaches to deal with these environments consist on explicit reasoning on probabilistic domain models, or, in the other extreme, reasoning only with full deterministic models (removing all probabilistic information), execution of generated plans, and replanning when needed. In this paper, we propose a variation of the second approach. Our technique, Variable Resolution Planning (VRP), performs a detailed analysis of the near future (first actions in the plan). After a given planning horizon, it abstracts away details as it reasons about the far future. The advantages of this approach are that it: is faster than regular full-fledged planning (both in the probabilistic or deterministic settings); does not spend much time on the far future possibilities that will not be reached anyway, since in most cases it will need to replan before finding the end of the plan; takes into account some main components of the far future (as an improvement over pure reactive systems); and, from a cognitive perspective, plausibly is closer to how humans plan in these kinds of environments.

1. Introduction

One of the distinct cognitive abilities of humans relates to our capability of long-term reasoning. Thus, from the early works on computer science and artificial intelligence, the computational models of AP have been widely studied in combination with general problem solving methods (Simon & Newell, 1969). Planning was modeled as a search in a problem space defined as a set of states, and a set of operators, that configured the domain model. Within a given domain, a set of planning problems could be defined, each one composed of an initial state and a set of goals. Given that fully enumerating potentially very large domains was not feasible, expressive languages have been used for concisely representing large problem spaces. Those languages have been largely based on predicate logic. As a prominent example, we can cite the currently used standard Planning Domain Description Language (PDDL) (Fox & Long, 2003). Planners take as input a domain and a problem and compute a plan (ordered set of instantiated operators, also called actions) that solve the problem;

the execution of the plan allows the executing agent to reach a state where the goals are true from the initial state.

We are interested on planning-execution cycles, as in robotics or videogames, where actions in the plan have to be executed by an agent. A common approach to deal with the planning-execution cycle consists of letting a planner fully solve the planning task for some (usually very large) time and then execute the actions in the plan. This scheme presents some disadvantages, specially in environments that present some level of uncertainty. First, AP could be prohibitively expensive for most real world scenarios. In the presence of uncertainty, the computational complexity of planning has been shown to be EXPTIME-complete (Littman, Goldsmith, & Mundhenk, 1998).

Therefore, the agent (perhaps a robot) will do nothing, or something random, while waiting for a solution to be generated. This is usually unacceptable. Second, depending on the uncertainty level, most actions of the generated plan will not be even applied if the execution of the previous actions in the plan fail (generate an unexpected state, disallowing the execution of further actions in the plan). So, most planning effort, specially the one devoted to reasoning on actions at the end of the plan, will be wasted. Third, during the execution of the actions new information about the environment could be discovered, changing the structure of the problem. Finally, if we consider human planning, we do not usually devote a huge time to planning all potential details of issues far into the future, specially in the presence of uncertainty.

There are different ways to approach planning and execution in this kind of environments. If we have models on the dynamics of the environment (failures model of a robot, structure of the terrain, accuracy of sensors, etc.), we can define a domain model with probabilistic information (such as in Probabilistic PDDL (Younes & Littman, 2004) or RDDL (Sanner, 2011)). Then, a planner can build a conditional plan (Peot & Smith, 1992) where the plan takes into account all possible alternative scenarios, or generate a set of policies by solving it as a Markov Decision Process (MDP) as shown by LAO* (Hansen & Zilberstein, 2001), or LRTDP (Bonet & Geffner, 2004). Usually, the dynamics of the environment are not fully known, but learning could be used to first acquire it (Jiménez, Fernández, & Borrajo, 2013). Also, these approaches require a huge computational effort and they do not scale really well to real world problems. Therefore, a alternative solution, consists of using a deterministic domain model (remove all probabilistic information) and replan when a failure in execution is detected (Yoon, Fern, & Givan, 2007).

On an extreme point of planning, we find reactive systems, which 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, ignoring the impact of the selected actions on the future actions and states. Thus, they often get trapped in local minima, or dead-ends. For instance, they can consume resources that cannot be re-generated that might be needed in the future.

Instead, we advocate here for a planning approach in the deterministic setting (no use of probabilistic information), Variable Resolution Planning (VRP). It computes a valid head of the plan (first actions in the plan) with all their details, since the first actions in the plan have a high probability to be executed correctly. Then, starting at a given horizon, which is defined manually, VRP plans consider only partial knowledge on the domain, abstracting away some details. We have been inspired by work on search in classical games (search horizon), the use of abstraction in planning (Sacerdoti, 1972; Knoblock, Tenenberg, & Yang, 1990) and the Variable Level-Of-Detail approach (VLOD) (Zickler & Veloso, 2010). The main difference with search techniques in game playing (as alpha-beta) is that we keep searching after the horizon, but in a smaller problem space, instead of using an evaluation function to replace search. In the related work section we will cover the main differences with previous work on abstraction in planning. The relation to VLOD is closer. VLOD is a motion planning technique, so there is no explicit domain model as in the case of AP. VLOD focuses search on obtaining accurate short-term motion planning, while considering 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. In our case, details are ignored by removing some predicates from the search.

The advantages of VRP are that: it can generate plans faster than regular planning (both in the probabilistic or deterministic settings) if the right predicates are abstracted; it avoids spending much time on the plan details in the far future and all related potential scenarios that will not be reached, since we believe that in most cases it will need to replan before finding the end of the plan; nevertheless, it takes into account key aspects of the far future (as an improvement over pure reactive systems) not to be trapped in execution dead-ends (e.g. consuming all fuel in the first part of the plan); and, from a cognitive perspective, we believe it is closer to how humans plan in these kinds of environments (this remains to be a hypothesis within this paper).

This paper is organized as follows. In Section 2 presents some work related with our approach. Section 3, we formally define the planning task. Next, in Section 4 we define our abstraction mechanism. In Section 5 we describe how to implement Variable Resolution Planning in a planning-execution environment. Section 6 shows an experimental evaluation of our technique over different domains. Finally, in Section 7, we conclude and introduce future work.

2. Related Work

This work focuses on applying abstractions over AP to reduce the computational overhead in stochastic or dynamic environments. There have already been many approaches that generate abstractions in AP. The first work in abstractions (Sacerdoti, 1972) extended the work of Newell and Simon on GPS (Simon & Newell, 1969) and used it to develop Abstrips, which combined abstraction with STRIPS. It defined abstraction levels by assigning criticalities to predicates as 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, where the first k where actions are valid ones, and the rest of the plan is not necessarily valid.

A more cognitive perspective of planning task called opportunistic planning can be found in (Hayes-Roth & Hayes-Roth, 1979). In this work, the plan exists at different levels of abstractions simultaneously and the planner alternates between the different levels of detail. Their planning paradigm is based on the idea of considering human planning activity largely as opportunistic. This means that at each point in the process (planning, monitoring, execution), the planner's current decisions and observations provide several opportunities for plan development. Commonly these opportunities are related with the state of the environment or the information known about it. The work presented in this paper explores a similar idea using different level of abstractions. In contrast, our approach only use two levels, one with complete information about environment and another one abstracting some details to decrease the cost of the planning task. These details represent information that can change in the future during execution making the plan generated previously fails.

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 & 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, in (Haslum et al., 2007) its authors exploited a new form of abstraction, in which domain transition graphs (DTG) (Helmert, 2006) are generated together (to merge them) and then shrunk by abstracting nodes that share the same relaxed distance to the goal. In these cases abstractions have been used to compute a heuristic value, that is used to look for the best path to the goal. More recently, changes in the representation have been used to automatically generate finite-state controllers from models (Bonet, Palacios, & 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 they can also include changes regarding the size of the problem or the probabilities of the action effects.

In other hand

3. Planning Framework

In this work, we focus on the use of a deterministic, STRIPS AP approach (Fikes & Nilsson, 1971). A classical planning problem is defined in PDDL using a lifted representation in predicate logic. Most current planners always perform a grounding transformation first. 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 *(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. *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) $\pi = (a_0, \ldots, a_{n-1})$, $\forall a_i \in A$, 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. $\forall a_i \in \pi\ \mathit{Pre}(a_i) \subseteq S_{i-1}$ ($S_0 = I$).

4. Abstractions

In VRP, an abstraction can be defined as a function that transforms a planning problem into another, where some details (values of some fluents) are ignored. Usually, abstractions help to reduce 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 (grounded) level, this is equivalent to removing literals from the preconditions and effects of actions in the grounded representation. Next, we define some concepts related to our abstraction mechanism.

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

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

For instance, in the classical Blocksworld domain, if p_1 is `(ontable ?x)` and a PDDL problem, P_1, contains three blocks (A, B, and C) on the table, then:

$$g(p_1, P_1) = \{(\texttt{ontableA}), (\texttt{ontableB}), (\texttt{ontableC})\}$$

In our approach, an abstraction of an action a over a predicate p removes all propositions that are groundings of p ($g(p, P)$) from the preconditions and effects of a. We define the candidate subset of facts to be potentially removed, $C^{abs} \subseteq F$, as:

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

where $F_s \subseteq F$ is the set of static facts, or facts that do not appear in the effects of any action (applying an abstraction over a static predicate does not reduce the number of actions to achieve the goals); $F_f \subseteq F$ is the functions set, or non-boolean facts that are groundings of PDDL functions instead of predicates; and F_G is the set of the dynamic predicates which are part of the problem goals (G) (predicates which are part of the goal subset cannot be removed because the planner would not reach a solution).

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

$$\begin{array}{rcl} Pre_p^{abs}(a) & = & Pre(a) \setminus g(p, P) \\ Add_p^{abs}(a) & = & Add(a) \setminus g(p, P) \\ Del_p^{abs}(a) & = & Del(a) \setminus g(p, P) \end{array}$$

For instance, following the previous example in the Blocksworld domain, given problem P_1 and predicate p_1 previously defined, suppose a_0 is `pick-up(A)`. If VRP selects to remove p_1:

$$\begin{array}{rcl} Pre^{abs}_{p_1}(a_0) & = & \{(clearA), (ontableA), (handempty)\} \setminus \{(ontableA), (ontableB), (ontableC)\} \\ & = & \{(clearA), (handempty)\} \\ Add^{abs}_{p_1}(a_0) & = & \{(holdingA)\} \setminus \{(ontableA), (ontableB), (ontableC)\} \\ & = & \{(holdingA)\} \\ Del^{abs}_{p_1}(a_0) & = & \{(clearA), (ontableA), (handempty)\} \setminus \{(ontableA), (ontableB), (ontableC)\} \\ & = & \{(clearA), (handempty)\} \end{array}$$

Definition 3 *An Abstraction of a PDDL problem P over a predicate p, called P^{abs}_p, is the result of abstracting all components of P over p:*

$$P^{abs}_p = (F^{abs}_p, A^{abs}_p, I^{abs}_p, G^{abs}_p)$$

where

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

Definition 4 *An Abstract Plan, Π^{abs}_p, that solves a PDDL problem P removing a predicate p starting at a horizon k is an action sequence:*

$$\Pi^{abs}_p = \{a_0, \dots, a_{k-1}, a_k, \dots a_{n-1}\}$$

The k first actions are applicable in P if the actions $a_i, i = 0 \dots k-1$, are all in A, and there exists a sequence of states $(s_0, \dots, s_k)$, 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 $i = 0, \dots, k-1$. The rest of actions starting from a_k are applicable in P^{abs}_p, if the actions $a_i, i = k \dots n-1$, are all in A^{abs}_p, and there exists a sequence of states $(s_k, \dots, s_n)$, 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 $i = k, \dots, n-1$. Thus, a partial abstract plan is composed of a standard sound plan from the initial state s_0 to state s_{k-1} and an abstract plan from state s_k to a goal state s_n. In this paper definitions use only one predicate, but they can be extended using a set of predicates.

5. Variable Resolution Planning

Variable Resolution Planning is based on two ideas: it is not always possible to collect all the information about the dynamics of a real world environment or if it is possible to get it, the amount of information that describes the environment cannot be managed. According with these ideas, VRP ignores some information about the environment from a horizon k. The abstraction that we perform during the planning process is showed in Figure 1. The abstract plan generated, in bold in

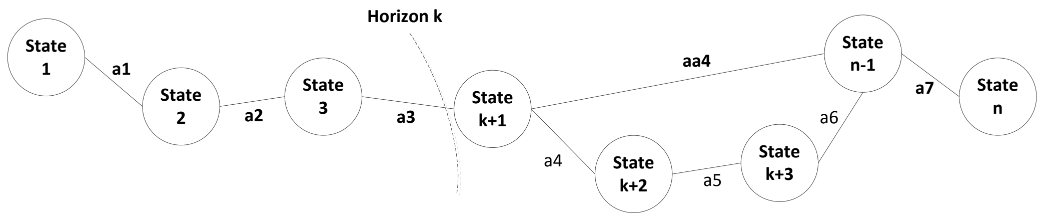

Figure 1. Representation of the abstract planning process.

the figure, will be composed of less actions than the standard plan. It computes a valid head of the plan (before horizon k) with all the information known about the environment. Thus, this part of the plan will be composed of the actions that most probably will be executed in the environment. Next, starting at a given horizon, the planning process solves the problem by ignoring some details about the environment. We will first present how planning works in VRP and then how to integrate planning and execution in VRP.

5.1 Planning

There are different ways to implement the VRP idea in current planners, depending on the PDDL version planners can handle. If the planner supports functions and numeric preconditions, VRP can automatically compile a PDDL domain into one with functions to define the different abstractions (we do not detail this compilation here). Unfortunately, very few planners handle numeric preconditions. Therefore, we have to modify the planners code to perform domain-independent predicate abstractions. In this work, Metric-FF (Hoffmann, 2003) has been used as a base to implement our approach. This planner provides greater expressiveness than other planners allowing numeric preconditions and effects in actions, which can more accurately describe some elements from real environments. We call the new planner Abstract-MFF. Apart from the standard inputs of Metric-FF, Abstract-MFF also receives as input: k (the horizon) and p (the predicate to abstract at time step k). 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 the horizon k and the predicate p. Thus, Abstract-MFF generates a search tree where nodes of depth less than or equal to k use the original definition of the problem, and nodes below that depth use an abstracted version of the problem.

By abstracting p, Abstract-MFF instantiates a new version of the problem. From the original set of actions, Abstract-MFF creates three sets. The first set A_1 contains the actions unaffected by the abstraction. The second set A_2 contains the actions that have p as precondition or effect. The third set A_3 contains the actions derived from applying the abstraction to the actions in A_2. After computing the abstraction, Abstract-MFF searches using a modified version of A^*, such that when the depth of a node is lower than the horizon k, the actions used are $A_1 \cup A_2$. Otherwise the actions are $A_1 \cup A_3$. Finally, the output of the planner is an abstract plan.

5.2 Planning and Execution

We have implemented a planning-execution-replanning loop on some domains and problems to evaluate the approach presented in this paper. We have used the simulator MDPSim (Younes et al., 2005) to simulate the execution of plans.[1] To simulate failures, we provide a probabilistic version of each domain in PPDDL to MDPSim, where actions can generate unexpected states with different probabilities. Note that the planner does not have any probabilistic knowledge on the domain. We assume we do not have that information available. This probabilistic version of the domain is given to MDPSim so that MDPSim can act as the real world. In our case, a state is considered as an unexpected state when the next action in the plan cannot be executed in the environment. In those cases, MDPSim returns the previous state (the action could not be executed). We have chosen the simplest case to define failures, because we did not have access to a simulator that was able to introduce more complex uncertainty scenarios, as those with exogenous effects.

Algorithm 1 shows the planning and execution cycle. Given a planning task (consisting of a deterministic PDDL domain and problem descriptions), p (the predicate to be removed) and k (horizon), the planner generates an abstract plan (where the first k actions are non-abstracted actions and the rest are abstracted actions). We assume we do not know the probabilistic model of the world.

Next, the planner 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 deterministic version of the domain, Abstract-MFF generates a new plan from that state. This process is repeated as a re-planning loop until a goal state is reached.

```
Data: Problem: $P = (F, A, I, G)$, predicate $p$, horizon $k$
1  begin
2      $plan \leftarrow$ planning$(P, p, k)$;
3      $lastState \leftarrow I$;
4      while $plan$ is not empty at the current state do
5          $action \leftarrow$ getAction$(plan)$;
6          $expectedState \leftarrow$ generateState$(action, lastState)$;
7          $environmentState \leftarrow$ sendActionToMDPSIM$(action)$;
8          if $environmentState = expectedState$ then
9              $lastState \leftarrow environmentState$;
10         else
11             $P \leftarrow$ generatePlanningProblem$(P, lastState)$;
12             $plan =$ planning$(P, p, k)$;
```

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

1. MDPSim was developed for the First Probabilistic Planning Competition.

6. Experiments and Results

In the experiments, we compare Abstract-MFF (AMFF) with the standard Metric-FF planner and a reactive version of Metric-FF we have built, called Reactive Metric-FF (RMFF). RMFF generates a plan composed by only one action each time. It uses a deterministic domain and selects the best next action acording with the heuristic used by the planner. RMFF has been designed to analyze the effects of a reactive model that observes the environment and generates the next action to execute in the environment.

Experiments were performed 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 was set to 18000 seconds. Each problem was executed fifteen times until goals are reached (we call it a run). The probability of an action execution failure has been included in the definition of the action in the PPDDL domain description. We show results with 30% of failure probability. No planner in the comparison (AMFF, Metric-FF or RMFF) had access to the probabilistic version of the domain.

The following sections provide an evaluation of VRP over two benchmark domains from the International Planning Competition (IPC) set:[2] Rovers from the IPC-3 and Gold-Miner from the learning track of the IPC-6. We used Rovers, because it is a domain similar to the ones used for robotics tasks that perfectly match the kind of domains where we believe VRP will work best. And we have used Gold-Miner, because it is also a robotic task, but it presents dead-ends which are hard to solve by reactive behaviours or greedy approaches.

6.1 Rovers Domain

The Rovers domain 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 the data collected by the robots has to be sent to the lander, that is placed at a fixed waypoint. VRP needs the definition of two parameters: horizon and predicate for abstracting. Taking into account the large number of alternatives, we have selected a subset of significant values for them. We have considered four predicates for removal: `at`, `have_rock_analysis`, `have_soil_analysis` and `have_image`. In this paper, we only show results for predicates `at` and `have_rock_analysis`, because the results generated with the other predicates are similar to those obtained when predicate `have_rock_analysis` is removed. We have selected four different horizons for each predicate. $k = 1, 5, 10$ and 20. Table 1 reports the planning time for the Rovers domain when removing predicate `have_rock_analysis` with a 30% probability of failure in a set of 10 problems. The first column defines the problem number, the second column describes the results obtained by Reactive Metric-FF, the third column describes the result obtained by Metric-FF, and the rest of columns correspond to the results of Abstract-MFF with different values of k. For each planner, each column shows the average of the

2. ipc.icaps-conference.org

Table 1. Planning time for the Rovers domain when removing predicate `have_rock_analysis` with a 30% probability of failure. The first column corresponds to one IPC problem of the Rovers domain, the second column corresponds to Metric-FF and the remaining columns correspond 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, the standard deviation (SD) and the time of the first planning process (FT) (all times measured in seconds). In bold, we highlight the best results per row.

Prob	RMFF		Metric-FF		AMFF (k=5)		AMFF (k=10)		AMFF (k=20)	
	Time	FT	Time	FT	Time	FT	Time	FT	Time	FT
20	48 ± 8	1	32 ± 4	2	**26** ±6	2	21 ± 10	1	27 ± 9	1
21	61 ± 9	1	52 ±16	12	64 ± 20	3	**52** ±40	3	69 ± 22	4
22	110 ± 6	1	192 ± 26	15	**83** ±14	3	99 ± 46	4	108 ± 39	4
23	271 ± 69	1	528 ± 193	32	**153** ±21	7	177 ± 113	6	170 ± 71	8
24	129 ± 18	1	136 ± 10	17	90 ± 12	6	**85** ±49	7	104 ± 37	5
25	2978 ± 348	1	9657 ± 932	263	**1564** ±219	72	1567 ± 813	78	1796 ± 621	58
26	**398** ± 24	1	680 ± 42	25	599 ±160	15	662 ± 329	16	617 ± 292	54
27	**1268** ± 475	1	4581 ± 674	237	1874 ± 585	78	1554 ±887	44	1989 ± 621	35
28	1154 ± 187	1	3132 ± 223	168	949 ± 175	39	**929** ±461	31	1181 ± 362	36
Total	6417		18983		5385		**5149**		6065	

sum of the planning times of each run in seconds and the standard deviation (SD) in seconds and the time of the first planning process (FT) in seconds.

The results reported in Table 1 show that removing a predicate that is part of the preconditions of actions that are used to achieve the problem goals, like `have_rock_analysis`, provides better performance in difficult problems. In some cases, Reactive MFF offers better results in terms of execution time, but the actions used to solve the problem are many more compared to other planners. Abstracting this kind of predicate 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 `communicate_rock_data`, the number of actions needed to obtain a rock sample and send it to the lander will decrease. This is 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. Thus, solving the path planning part is not relevant when planning for all actions. It will be needed when in fact the rover has to actually move to a specific waypoint. And, at that time, it will compute those details (they will be in the head part of the next plan).

Table 2 reports the planning time for the Rovers domain when removing predicate `at` with a 30% probability of failure in a set of 10 problems. In this case, results are not so impressive as those for predicate `have_rock_analysis`, although Abstract MFF outperforms Metric-FF in most problems. Our initial assumption was that predicates like `at` would offer better performance than the rest of predicates in domains where a robot or a vehicle has to move through different positions. However, this did not happen in the Rovers domain.

Table 2. Planning time for the Rovers domain when removing predicate `at` with a 30% probability of failure. The meaning of columns is the same as in the previous table.

Prob	RMFF		Metric-FF		AMFF (k=5)		AMFF (k=10)		AMFF (k=20)	
	Time	FT	Time	FT	Time	FT	Time	FT	Time	FT
20	50 ± 3	1	32 ± 4	2	26 ± 6	2	**21** ±10	1	27 ± 9	1
21	65 ± 6	1	52 ±16	12	64 ± 20	3	**52** ±40	3	69 ± 22	4
22	104 ± 9	1	192 ± 26	15	**83** ±14	3	99 ± 46	4	108 ± 39	4
23	**271** ±69	1	528 ± 193	32	321 ± 84	19	297 ±42	14	301 ± 59	15
24	136 ± 36	1	136 ± 10	17	129 ± 72	13	**103** ±29	10	119 ± 46	12
25	**3174** ± 453	1	9657 ± 932	263	6544 ± 328	104	5567 ± 743	98	5996 ± 832	97
26	**410** ± 47	1	680 ± 42	25	691 ± 315	28	671 ± 279	25	624 ± 79	37
27	**1739** ± 621	1	4581 ± 674	237	3874 ± 498	166	2954 ± 731	110	2973 ± 621	137
28	**2316** ± 509	1	3132 ± 223	168	3949 ± 315	142	4128 ± 537	138	3997 ± 512	168
Total	**8265**		18983		15681		13892		14214	

In general, results obtained in the Rovers domain confirm our initial hypothesis: in problems where planning time is high, abstractions significantly reduce such time. The first group of problems are moderately difficult and abstraction does not show a significant reduction of planning time. But, in harder problems as 25 or 27, Abstract-MFF obtains a reduction of the total planning time of 30% when removing the predicate `have_rock_analysis`. Results also showed that the most important element in the abstraction is the predicate. When the information represented by the predicate in the domain has a great influence over the solution, abstraction will produce a modification in the complexity of the problem, changing the number . Using this kind of abstraction can increase or decrease the number of nodes expanded and the branch factor during search, as it can be observed for problem 28. In this problem, while the abstraction of the predicate `have_rock_analysis` decreases the complexity of the problem and the number of nodes expanded to find a solution decreasing the time to solve it, the abstraction of the predicate `at` decrease the number of nodes expanded during search, but increase the branch factor diminishing the initial advantages of removing the predicate `at`. Moreover, abstractions do not work equally well with all predicates. In the case of the Rovers domain, there are some dynamic predicates which do not offer advantages over traditional planning. For instance predicate `calibrated`, needed for the rovers cameras, does not decrease the planning time when removed, because the process to achieve groundings of this predicate implies the execution of only one action and does not have a big influence over the search.

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 the opposite is true. 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

Table 3. Planning time for the Gold-Miner domain when removing predicate `robot_at` with a 30% probability of failure. The meaning of columns is the same as in the previous table.

Prob	RMFF		Metric-FF		AMFF (k=5)		AMFF (k=10)		AMFF (k=20)	
	Time	FT	Time	FT	Time	FT	Time	FT	Time	FT
10	-	-	542 ± 112	45	93 ± 9	8	**65** ± 6	13	491 ± 45	21
11	-	-	661 ± 73	54	120 ± 15	11	**95** ± 12	18	598 ± 61	30
12	-	-	598 ± 72	48	235 ± 36	7	**82** ± 14	13	570 ± 92	28
13	-	-	-	-	280 ± 25	15	**91** ± 17	11	-	-
14	-	-	-	-	309 ± 32	14	**72** ± 74	16	529 ± 94	26
15	-	-	-	-	318 ± 27	7	**65** ± 18	9	548 ± 52	23
16	-	-	798 ± 93	69	95 ± 17	7	**69** ± 11	9	-	-
17	-	-	-	-	218 ± 29	11	**93** ± 15	12	-	-
18	-	-	910 ± 101	76	92 ± 21	14	**87** ± 11	16	-	-
20	-	-	-	-	**264** ± 18	23	430 ± 19	27	570 ± 73	33
Total	-		3509		2024		**1194**		3306	

part of the plan is abstracted, but planning time will be less, given that the plan is shorter on each planning episode.

6.2 Gold-Miner Domain

The Gold-Miner Domain was designed for the learning track of IPC-6 (2008) and was inspired on the gold extraction process in a mine. In this domain, there is a robot in a mine and it has to reach a location that contains gold to extract it. The mine is defined as a grid with each cell either being hard or soft rock. The robot can use two different tools to destroy rocks: bombs and a laser cannon. The laser cannon can shoot through both hard and soft rock, whereas the bomb can only penetrate soft rock. However, the laser cannon also will destroy the gold if used to uncover the gold location. The bomb will not destroy the gold. For this domain, initially we had considered three predicates for removal: `robot_at`, `holds_bomb` and `hold_laser`. However, abstracting the predicate `hold_laser` does not offer advantages over the normal planning process. Thus. we do not report the results for abstracting this predicate. We have selected five different horizons for each predicate. $k = 1, 5, 10$ and 20. Table 3 reports the planning time for the Gold-Miner domain when removing predicate `robot_at` for a set of 10 problems..

A main characteristic of Gold Miner is that reactive approaches typically fall into dead-ends, as it can be observed in Tables 3 and 4. However, when executing Abstract Metric-FF, the search process can avoid dead-ends when the abstraction is applied after the dead-end appears. For instance, removing predicate *robot-at* allows the robot to execute any action at every location, decreasing the complexity of the problem. Given that the robot does not need to destroy the rocks to generate a path to the gold, it can move to the position where the gold is picked up. In this case the planning process is performed partially while the robot is performing actions on the environment. Table 4

Table 4. Planning time for the Gold-Miner domain when removing predicate `holds_bomb` with a 30% probability of failure. The meaning of columns is the same as first table.

Prob	RMFF		Metric-FF		AMFF (k=5)		AMFF (k=10)		AMFF (k=20)	
	Time	FT	Time	FT	Time	FT	Time	FT	Time	FT
10	-	-	542 ± 112	45	-	-	**75** ± 18	9	82 ± 7	11
11	-	-	661 ± 73	54	-	-	91 ± 14	7	**89** ± 11	9
12	-	-	598 ± 72	49	-	-	**71** ± 12	8	128 ± 18	11
13	-	-	-	-	-	-	**68** ± 12	6	176 ± 32	15
14	-	-	-	-	-	-	-	-	**212** ± 23	12
15	-	-	-	-	-	-	-	-	**205** ± 28	11
16	-	-	798 ± 93	68	-	-	**94** ± 13	7	164 ± 15	17
17	-	-	-	-	-	-	**101** ± 23	9	193 ± 26	13
18	-	-	910 ± 101	76	-	-	**87** ± 9	6	142 ± 28	12
20	-	-	-	-	-	-	-	-	**210** ± 34	10
Total	-		8509		-		**587**		1601	

reports the planning time for the Gold-Miner domain when removing predicate `holds_bomb` for a set of 10 problems.

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. This domain shows interesting results, because the abstraction is not enough to avoid the dead-end, but when combined with a high horizon k, dead-ends are avoided, decreasing the execution time of the problem.

7. Conclusions

In this paper, we have presented a new planner, Abstract-MFF. It uses an abstraction mechanism, VRP, that dynamically removes some predicates during the planning process based on a temporal horizon, in order to improve planning performance in stochastic and dynamic environments like robotics systems. The main contribution of this work consists on defining VRP, and analyzing how it can be used to improve the application of planning techniques to real environments. VRP tries to simulate human cognitive processes by selectively abstracting information about the environment when planning and replanning if needed. Planning time is reduced with respect to the time required by regular planning processes, specially on ignoring details on future components of plans that will most probably not be needed anyway. Also, VRP generates the first plan much faster, which in real environments is critical, as it allows the execution to begin earlier. Besides, the quality of the solutions (number of executed actions) using VRP is roughly the same as the one of the base planner, with an overall improvement of a 5%.

There are several future research lines. On one hand, a key aspect is how to select the predicates that generate abstractions. There are three properties to be analyzed: the relevance of the predicate into the problem; the difficulty of generating the predicate, as effects of actions, during the search process; and whether the removal of the predicate can lead a dead-end. Besides, another way to abstract information from the states consists of abstracting a sub-set of grounded predicates, not necessarily all corresponding to the same predicate. For example, in the Blocksworld domain, it is possible to remove grounded predicates for some blocks while leaving the same predicate for other blocks. Another key parameter of VRP is the selection of the value of k. It can be even dynamically chosen, depending of how much does the information about the environment change, or what predicates are removed during search. In short, the usefulness of removing a predicate and the selection of the value k may depend on the stage the search process is in. Changing the abstraction accordingly may be beneficial, which means that both the abstracted predicates and the value k should not be fixed at the beginning but rather vary dynamically during search.

Finally, we are interested in applying this technique to other kinds of problems which have more information encoded in predicates and actions about environment, like robotics, video-games or automatic road transportation systems. In particular, several of these types of problems are currently out of reach of state-of-the-art domain-independent planners, which which can be used to generated abstractions to decrease the complexity of the planning process.

Acknowledgements

We extend our thanks to Vidal Alcazar for his helpful discussions. This research is partially supported by the Spanish MICINN projects TIN2011-27652-C03-02, TIN2012-38079-C03-02 and TRA-2009-008.

References

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

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

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

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

Fikes, R. E., & Nilsson, N. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *3-4*, 189–208.

Fox, M., & Long, D. (2003). Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, *20*, 2003.

Hansen, E. A., & Zilberstein, S. (2001). Lao* : A heuristic search algorithm that ïňĄnds solutions with loops. *ArtiïňĄcial Intelligence*, *129*, 35–62.

Haslum, P., Botea, A., Helmert, M., Bonet, B., & Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. *In Proceeding of the twenty-second Conference on Artificial Intelligence* (pp. 1007–1012).

Hayes-Roth, B., & Hayes-Roth, F. (1979). A cognitive model of planning. *Cognitive Science*, *3*, 275.310.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

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

Jiménez, S., Fernández, F., & Borrajo, D. (2013). Integrating planning, execution and learning to improve plan execution. *Computational Intelligence Journal*, *29*, 1–36.

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

Knoblock, C., Tenenberg, J., & Yang, Q. (1990). Learning abstractions hierarchies for problem solving. *In Proceedings of the Eighth National Conference of Artifical Intelligence* (pp. 223–228). Boston, MA.

Littman, M. L., Goldsmith, J., & Mundhenk, M. (1998). The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, *9*, 1–36.

Peot, M. A., & Smith, D. E. (1992). Conditional nonlinear planning. *In Proceedings of the First International Conference on Artificial Intelligence* (p. 189âĂŞ197). College Park, Maryland.

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

Sanner, S. (2011). Relational dynamic influence diagram language (rddl): Language description. *Proceedings of the Seventh International Planning Competion*.

Simon, H., & Newell, A. H. (1969). GPS: A case study in generality and problem solving. *Artificial Intelligence*, *2*, 109–124.

Yoon, S. W., Fern, A., & Givan, R. (2007). Ff-replan: A baseline for probabilistic planning. *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*. Providence, Rhode Island, USA.

Younes, H. L. S., & 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., & Asmuth, J. (2005). The first probabilistic track of the international planning competition. *Journal of Artificial Intelligente Research*, *24*, 851–887.

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