Robotics and Autonomous Systems 83 (2016) 214–230

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Planning and execution through variable resolution planning

Moisés Martínez *, Fernando Fernández, Daniel Borrajo

Computer Science Department, Universidad Carlos III de Madrid, Avenida de la Universidad, 30, Leganés 28911, Madrid, Spain

HIGHLIGHTS

- A novel technique for planning and execution in dynamic and stochastic environments.
- When planning, the technique removes information far into the future.
- Planning information is abstracted by selecting several predicates.
- Planning and execution performance are improved by computing plans very fast.

ARTICLE INFO

Article history:
Received 12 May 2015
Received in revised form
9 December 2015
Accepted 22 April 2016
Available online 11 May 2016

Keywords:
Task planning
Planning and execution
Abstract representation
Cognitive robotics

ABSTRACT

Generating sequences of actions – plans – for robots using Automated Planning in stochastic and dynamic environments has been shown to be a difficult task with high computational complexity. These plans are composed of actions whose execution might fail due to different reasons. In many cases, if the execution of an action fails, it prevents the execution of some (or all) of the remainder actions in the plan. Therefore, in most real-world scenarios computing a complete and sound (valid) plan at each (re-)planning step is not worth the computational resources and time required to generate the plan. This is specially true given the high probability of plan execution failure. Besides, in many real-world environments, plans must be generated fast, both at the start of the execution and after every execution failure. In this paper, we present Variable Resolution Planning which uses Automated Planning to quickly compute a reasonable (not necessarily sound) plan. Our approach computes an abstract representation – removing some information from the planning task – which is used once a search depth of k steps has been reached. Thus, our approach generates a plan where the first k actions are applicable if the domain is stationary and deterministic, while the rest of the plan might not be necessarily applicable. 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 actions that probably will not be executed, since in most cases it will need to replan before executing the end of the plan; and takes into account some information of the far future, as an improvement over pure reactive systems. We present experimental results on different robotics domains that simulate tasks on stochastic environments.

1. Introduction

Automated Planning (AP) is the branch of Artificial Intelligence that studies the generation of an ordered set of actions – plan – that allows a system to transit from a given initial state to a state where a set of goals have been achieved. AP has been successfully used to solve real world problems such as planning Mars exploration missions [1] or controlling underwater vehicles [2]. Despite of these examples, the application of AP systems to stochastic and dynamic environments still presents some challenges, mainly because these scenarios increase the complexity of the planning and execution process: (i) new information about the environment can be discovered during action execution, modifying the structure of the planning task; (ii) actions' execution can fail which in turn prevents the execution of the rest of the plan; (iii) the execution of the actions in the plan can generate states from which the rest of the plan cannot be successfully executed (dead-ends); and (iv) plans may need to be generated quickly to offer a real time interaction between the AP system and the environment. For these reasons, the process of generating a plan of actions can be prohibitively expensive for this kind of scenarios.

There are two main (extreme) approaches to solve problems in stochastic and dynamic scenarios: deliberative and reactive. At

* Corresponding author.
E-mail addresses: moises.martinez@uc3m.es (M. Martínez), fernando.fernandez@uc3m.es (F. Fernández), dborrajo@ia.uc3m.es (D. Borrajo).

http://dx.doi.org/10.1016/j.robot.2016.04.009
0921-8890/

one extreme, we find deliberative systems which are based on interleaving AP and execution with full or partial observability. If we have information about the dynamics of the environment (failures in the actuators of a robot, the structure of the terrain, accuracy of sensors), we can define a domain model with probabilistic information with full observability (such as in PPDDL [3] or RDDL [4]). Then, one alternative consists on building conditional plans [5] where plans take into account all possible outcomes. Another approach consists on generating a set of policies by solving the problem as a Markov Decision Process (MDP) [6–8].

But, usually, the dynamics of the environment are not known or cannot be easily modeled. Then, in turn, we have two alternatives. First, we can learn the dynamics and then apply the previous approaches. However, the learning effort is huge except for small tasks [9]. Another solution, and the most used one, consists of using a deterministic domain model and replan or repair the plan when a failure in execution is detected (e.g. the robot is not in the expected place). When replanning [10], 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 from scratch. 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 (an applicable plan that achieves the goals), when most of it will not be used. When repairing a running plan [11–13], the planner generates an initial applicable plan and executes it. If an unexpected state is detected, the system generates a new plan by reusing the plan generated previously and adding/removing some actions. In general, deliberative systems require a huge computational effort to generate a complete and sound plan. Depending upon the dynamics of the environment, most probably the plan will not be executed fully.

On the other extreme, there are several approaches that solve problems in stochastic and dynamic scenarios using reactive techniques. These systems are based in greedily selecting the next action to be applied according to some knowledge which has been programmed or learned previously. If the knowledge about the environment is only used to select the next action, we can consider a pure reactive system without deliberation, where the system perceives and generates the next action in a continuous cycle. Systems based on the Subsumption architecture [14,15] are built using a control layer set, where different layers are interconnected with signals. During each execution step, one layer is chosen depending on the information perceived. Other reactive approaches are based on building reactive behavioral navigation controllers using neural networks [16,17] or fuzzy logic [18,19]. In general, reactive systems require much less computational effort and 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.

In this paper, we propose Variable Resolution Planning (VRP) for interleaving planning and execution in stochastic and dynamic environments. Our research has been inspired by the work of Zickler and Veloso [20], where a motion planning technique is used to generate a collision-free trajectory from an initial state to a goal state. They consider the far future with a different level of detail, by selectively ignoring the physical interactions with dynamic objects. Similarly, VRP is based on two main concepts: (i) most planning effort is devoted to compute a valid plan head of length k; and (ii) the rest of the plan is only generated by checking for potential reachability by relaxing the actions' model. Actions are simplified by removing some domain details to decrease the computational effort avoiding dead-ends. The main advantage of our approach is that it requires much less search time than traditional planning approaches that compute a valid complete plan (improving over pure deliberative approaches), while retaining their capability of reasoning into the future (improving over pure reactive approaches). In addition, our technique can be easily parameterized by appropriately setting a value for k so that its behavior gradually transits from a more deliberative approach (large values of k) to a more reactive approach (small values of k). In the extremes, if $k = 1$, VRP becomes an almost pure reactive system, while if $k = \infty$, VRP behaves as a standard deliberative planner.

This paper is organized as follows: first in Section 2, we formally define the representation of the planning task in classical planning. Section 3 presents an overview of VRP. Section 4 introduces the concept of predicate abstraction and how it can be deployed in AP. Section 5 describes the algorithms related to VRP. Section 6 presents a description of the planning and execution architecture used to deploy VRP. Section 7 shows experimental evaluation of VRP in five different domains. Section 8 presents some works related with our approach. Finally, Section 9 concludes and introduces future work.

2. Planning formalization

There are different types of planning tasks defined in the literature. In this paper, we consider the sequential classical planning task which is encoded in the propositional fragment of Planning Domain Description Language (PDDL) 2.2. It includes advanced features like numeric fluents, ADL conditions, effects and derived predicates (axioms).

Definition 1 (*Planning Task*)**.** A planning task can be defined as a tuple $\Pi = (F, A, I, G)$, where:

- F is a finite set of grounded literals (also known as facts or atoms).
- A is a finite set of grounded actions derived from the action schemes of the domain.
- $I \subseteq F$ is a finite set of grounded predicates that are true in the initial state.
- $G \subseteq F$ is a finite set of goals.

Any state s is a subset of facts that are true at a given time step. Each action $a_i \in A$ can be defined as a tuple $a_i = (Pre, Add, Del)$, where $Pre(a_i) \subseteq F$ are the preconditions of the action, $Add(a_i) \subseteq F$ are its add effects, and $Del(a_i) \subseteq F$ are the delete effects. $Eff(a_i) = Add(a_i) \cup Del(a_i)$ are the effects of the action. Actions can also have a cost, $c(a_i)$ (the default cost is one). An action a is applicable in s_i, if $Pre(a) \subseteq s_i$. Then, the result of applying an action a in state s_i generates a new state that can be defined as: $s_{i+1} = (s_i \setminus Del(a)) \cup Add(a)$. A plan π for a planning task Π is an ordered set of actions (commonly, a sequence) $\pi = (a_1, \ldots, a_n)$, $\forall a_i \in A$, that transforms the initial state I into a 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. $\forall a_i \in \pi$, $Pre(a_i) \subseteq s_{i-1}$ such that state s_i results from executing the action a_i in the state s_{i-1}. s_0 is the initial state I. The cost of the solution is the sum of the action costs.

In PDDL [21], planning tasks are described in terms of objects of the world (robots, locations, rocks, etc.), predicates which describe static or dynamic features of these objects or relations among them (e.g. locations are connected by roads), actions that manipulate those relations (a robot can move from one location to another, a package can be grasped by a robot), an initial state that describes the initial situation before plan execution, and a goal definition which describes the objectives that must be reached by the solution plan. Commonly, this information is provided in two input files: a domain and a problem. The domain file contains a