Learning–Based Planning

Sergio Jiménez Celorrio
Universidad Carlos III de Madrid, Spain

Tomás de la Rosa Turbides
Universidad Carlos III de Madrid, Spain

INTRODUCTION

Automated Planning (AP) studies the generation of action sequences for problem solving. A problem in AP is defined by a state-transition function describing the dynamics of the world, the initial state of the world and the goals to be achieved. According to this definition, AP problems seem to be easily tackled by searching for a path in a graph, which is a well-studied problem. However, the graphs resulting from AP problems are so large that explicitly specifying them is not feasible. Thus, different approaches have been tried to address AP problems. Since the mid 90's, new planning algorithms have enabled the solution of practical-size AP problems. Nevertheless, domain-independent planners still fail in solving complex AP problems, as solving planning tasks is a PSPACE-Complete problem (Bylander, 94).

How do humans cope with this planning-inherent complexity? One answer is that our experience allows us to solve problems more quickly; we are endowed with learning skills that help us plan when problems are selected from a stable population. Inspire by this idea, the field of learning-based planning studies the development of AP systems able to modify their performance according to previous experiences.

Since the first days, Artificial Intelligence (AI) has been concerned with the problem of Machine Learning (ML). As early as 1959, Arthur L. Samuel developed a prominent program that learned to improve its play in the game of checkers (Samuel, 1959). It is hardly surprising that ML has often been used to make changes in systems that perform tasks associated with AI, such as perception, robot control or AP. This article analyses the diverse ways ML can be used to improve AP processes. First, we review the major AP concepts and summarize the main research done in learning-based planning. Second, we describe current trends in applying ML to AP. Finally, we comment on the next avenues for combining AP and ML and conclude.

BACKGROUND

The languages for representing AP tasks are typically based on extensions of first-order logic. They encode tasks using a set of actions that represents the state-transition function of the world (the planning domain) and a set of first-order predicates that represent the initial state together with the goals of the AP task (the planning problem). In the early days of AP, STRIPS was the most popular representation language. In 1998 the Planning Domain Definition Language (PDDL) was developed for the first International Planning Competition (IPC) and since that date it has become the standard language for the AP community. In PDDL (Fox & Long, 2003), an action in the planning domain is represented by: (1) the action preconditions, a list of predicates indicating the facts that must be true so the action becomes applicable and (2) the action post-conditions, typically separated in *add* and *delete* lists, which are lists of predicates indicating the changes in the state after the action is applied.

Before the mid '90s, automated planners could only synthesize plans of no more than 10 actions in an acceptable amount of time. During those years, planners strongly depended on speedup techniques for solving AP problems. Therefore, the application of search control became a very popular solution to accelerate planning algorithms. In the late 90's, a significant scale-up in planning took place due to the appearance of the reachability planning graphs (Blum & Furst, 1995) and the development of powerful domain independent heuristics (Hoffman & Nebel, 2001) (Bonet & Geffner, 2001). Planners using these approaches could often synthesize 100-action plans just in seconds.

At the present time, there is not such dependence on ML for solving AP problems, but there is a renewed interest in applying ML to AP motivated by three factors: (1) IPC-2000 showed that knowledge-based planners significantly outperform domain-independent planners. The development of ML techniques that automatically define the kind of knowledge that humans put in these planners would bring great advances to the field. (2) Domain-independent planners are still not able to cope with real-world complex problems. On the contrary, these problems are often solved by defining ad hoc planning strategies by hand. ML promises to be a solution to automatically defining these strategies. And, (3) there is a need for tools that assist in the definition, validation and maintenance of planning-domain models. At the moment, these processes are still done by hand.

LEARNING-BASED PLANNING

This section describes the current ML techniques for improving the performance of planning systems. These techniques are grouped according to the target of learning: search control, domains-specific planners, or domain models.

Learning Search Control

Domain-independent planners require high search effort, so search-control knowledge is frequently used to reduce this effort. Hand-coded control knowledge has proved to be useful in many domains, however is difficult for humans to formalize it, as it requires specific knowledge of the planning domains and the planner structure. Since AP's early days, diverse ML techniques have been developed with the aim of automatically learning search-control knowledge. A few examples of these techniques are macro-actions (Fikes, Hart & Nilsson, 1972), control-rules (Borrajo & Veloso, 1997), and case-based and analogical planning (Veloso, 1994).

At the present, most of the state-of-the-art planners are based on heuristic search over the state space (12 of the 20 participants in IPC-2006 used this approach). These planners achieve impressive performance in many domains and problems, but their performance strongly depends on the definition of a good domain-independent heuristic function. These heuristics are computed solving a simplified version of the planning task, which ignores the delete list of actions. The solution to the simplified task is taken as the estimated cost for reaching the task goals. These kinds of heuristics provide good guidance across the wide range of different domains. However, they have some faults: (1) in many domains, these heuristic functions vastly underestimate the distance to the goal leading to poor guidance, (2) the computation of the heuristic values of the search nodes is too expensive, and (3) these heuristics are non-admissible so heuristics planners do not find good solutions in terms of plan quality.

Since evaluating a search node in heuristic planning is so time consuming, (De la Rosa, García-Olaya & Borrajo, 2007) proposed using Case-based Reasoning (CBR) to reduce the number of explored nodes. Their approach stores sequences of abstracted state transitions related to each particular object in a problem instance. Then, with a new problem, these sequences are retrieved and re-instantiated to support a forward heuristic search, deciding the node ordering for computing its heuristic value.

In the last years, other approaches have been developed to minimize the negative effects of the heuristic through ML: (Botea, Enzenberger, Müller & Schaeffer, 2005) learned off-line macro-actions to reduce the number of evaluated nodes by decreasing the depth of the search tree. (Coles & Smith, 2007) learned on-line macro-actions to escape from plateaus in the search tree without any exploration. (Yoon, Fern & Givan, 2006) proposed using an inductive approach to correct the domain-independent heuristic in those domains based on learning a supplement to the heuristic from observations of solved problems in these domains.

All these methods for learning search-control knowledge suffer from the utility problem. Learning too much control knowledge can actually be counterproductive because the difficulty of storing and managing the information and the difficulty of determining which information to use when solving a particular problem can interfere with efficiency.

Learning Domain-Specific Planners

An alternative approach to learning search control consists of learning domain-specific planning programs. These programs receive as input a planning problem of a fixed domain and return a plan that solves the problem.