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.

The first approaches to learn domain-specific planners were based on supervised inductive learning; they used genetic programming (Spector, 1994) and decision-list learning (Khardon, 1999), but they were not able to reliably produce good results. Recently, (Winner & Veloso, 2003) presented a different approach based on generalizing an example plan into a domain-specific planning program and merging the resulting source code with the previous ones.

Domain-specific planners are also represented as policies, i.e., pairs of state and the preferred action to be executed in the state. Relational Reinforcement Learning (RRL) (Dzeroski, Raedt & Blockeel, 1998) has aroused interest as an efficient approach for learning policies for relational domains. RRL includes a set of learning techniques for computing the optimal policy for reaching the given goals by exploring the state space though trial and error. The major benefit of these techniques is that they can be used to solve problems whether the action model is known or not. In the other hand, since RRL does not explicitly include the task goals in the policies, new policies have to be learned every time a new goal has to be achieved, even if the dynamics of the environment has not changed.

In general, domain-specific planners have to deal with the problem of generalization. These techniques build planning programs from a given set of solved problems so cannot theoretically guarantee solving subsequent problems.

Learning Domain Models

No matter how efficient a planner is, if it is fed with a defective domain model, it will return defective plans. Designing, encoding and maintaining a domain model is very laborious. At the time being, planners are the only tool available to assist in the development of an AP domain model, but planners are not designed specifically for this purpose. Domain model learning studies ML mechanisms to automatically acquire the planning action schemas (the action preconditions and post-conditions) from observations of action executions.

Learning domain models in deterministic environments is a well-studied problem; diverse inductive learning techniques have been successfully applied to automatically define the actions schema from observations (Shen & Simon, 1989), (Benson, 1997), (Yang, Wu & Jiang, 2005), (Shahaf & Amir, 2006). In stochastic environments, this problem becomes more complex. Actions may result in innumerable different outcomes, so more elaborated approaches are required. (Pasula, Zettlemoyer & Kaelbling, 2004) presented the first specific algorithm to learn simple stochastic actions without conditional effects. This algorithm is based on three levels of learning: the first one consists of deterministic rule-learning techniques to induce the action preconditions. The second one relies on a search for the set of action outcomes that best fits the execution examples, and; the third one consists of estimating the probability distributions over the set of action outcomes. But, stochastic planning algorithms do not need to consider all the possible actions outcomes. (Jimenez & Cussens 2006) proposed to learn complex action-effect models (including conditions) for only the relevant action outcomes. Thus, planners generate robust plans by covering only the most likely execution outcome while leaving others to be completed when more information is available.

In deterministic environments, (Shahaf & Amir, 2006) introduced an algorithm that exactly learns STRIPS action schemas even if the domain is only partially observable. But, in stochastic environments, there is still no general efficient approach to learn action model.

FUTURE TRENDS

Since the appearance of the first PDDL version in IPC-1998, the standard planning representation language has evolved to bring together AP algorithms and real-world planning problems. Nowadays, the PDDL 3.0 version for the IPC-2006 includes numeric state variables to support quality metrics, durative actions that allow explicit time representation, derived predicates to enrich the descriptions of the system states, and soft goals and trajectory constraints to express user preferences about the different possible plans without discarding valid plans. But, most of these new features are not handled by the state-of-the-art planning algorithms: The existing planners usually fail solving problems that define quality metrics. The issue of goal and trajectory preferences has only been initially addressed. Time and resources add such extra complexity to the search process that a real-world problem becomes extremely difficult to solve. New challenges for the AP community are those related to developing new planning algorithms and heuristics to deal with these kinds of problems. As it is very difficult to find an efficient general solution, ML must play an important role in addressing these new challenges because it can be used to alleviate the complexity of the search process by exploiting regularity in the space of common problems.

Besides, the state-of-the-art planning algorithms need a detailed domain description to efficiently solve the AP task, but new applications like controlling underwater autonomous vehicles, Mars rovers, etc. imply planning in environments where the dynamics model may be not easily accessible. There is a current need for planning systems to be able to acquire information of their execution environment. Future planning systems have to include frameworks that allow the integration of the planning and execution processes together with domain modeling techniques.

Traditionally, learning-based planners are evaluated only against the same planner but without learning, in order to prove their performance improvement. Additionally, these systems are not exhaustively evaluated; typically the evaluation only focuses on a very small number of domains, so these planners are usually quite fragile when encountering new domains. Therefore, the community needs a formal methodology to validate the performance of the new learning-based planning systems, including mechanisms to compare different learning-based planners.

Although ML techniques improve planning systems, existing research cannot theoretically demonstrate that they will be useful in new benchmark domains. Moreover, for time being, it is not possible to formally explain the underlying meaning of the learned knowledge (i.e., does the acquired knowledge subsumes task decomposition? a goal ordering? a solution path?). This point reveals that future research in AP and ML will also focus on theoretical aspects that address these issues.

CONCLUSION

Generic domain-independent planners are still not able to address the complexity of real planning problems. Thus, most planning systems implemented in applications require additional knowledge to solve the real planning tasks. However, the extraction and compilation of this specific knowledge by hand is complicated.

This article has described the main last advances in developing planners successfully assisted by ML techniques. Automatic learned knowledge is useful for AP in diverse ways: it helps planners in guiding search processes, in completing domain theories or in specifying particular solutions to a particular problem. However, the learning-based planning community can not only focus on developing new learning techniques but also on defining formal mechanisms to validate its performance against other generic planners and against other learning-based planners.

REFERENCES

Benson, S. (1997). Learning Action Models for Reactive Autonomous Agents. PhD thesis, Stanford University.

Blum, A., & Furst, M. (1995). Fast planning through planning graph analysis. In C. S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, volume 2, pages 1636–1642, Montreal, Canada, August 1995. Morgan Kaufmann.

Bonet, B. & Geffner, H. (2001). Planning as Heuristic Search. *Artificial Intelligence*, 129 (1-2), 5-33.

Borrajo, D., & Veloso, M. (1997). Lazy Incremental Learning of Control Knowledge for Efficiently Obtaining Quality Plans. *AI Review Journal. Special Issue on Lazy Learning.* 11 (1-5), 371-405.

Botea, A., Enzenberger, M., Müller, M. & Schaeffer, J. (2005). Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *Journal of Artificial Intelligence Research (JAIR)*, 24, 581-621.

Bylander, T., The computational complexity of propositional STRIPS planning. (1994). *Artificial Intelligence*, 69(1-2), 165–204.

Coles, A., & Smith, A. (2007). Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, 28, 119–156.

De la Rosa, T., García Olaya, A., & Borrajo, D. (2007) Using Utility Cases for Heuristic Planning Improvement. *Procceedings of the 7th International Conference on Case-Based Reasoning*, Belfast, Northern Ireland, Springer-Verlag.

Dzeroski, S., Raedt, L. D., & Blockeel, H., (1998) Relational reinforcement learning. In *International*

Workshop on Inductive Logic Programming, pages 11–22.

Fikes, R., Hart, P., & Nilsson, N., (1972) Learning and Executing Generalized Robot Plans, *Artificial Intelligence*, 3, pages 251-288.

Fox, M. & Long, D, (2003) PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20, 61–124.

Hoffmann J. & Nebel B. (2001) The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*,14, 253–302.

Jiménez, S. & Cussens, J. (2006). Combining ILP and Parameter Estimation to Plan Robustly in Probabilistic Domains. *In Conference on Inductive Logic Programming. Santiago de Compostela, ILP2006.* Spain.

Khardon, R. (1999) Learning action strategies for planning domains. *Artificial Intelligence*, 113, 125–148,

Pasula, H. Zettlemoyer, L. & Kaelbling, L. (2004) Learning probabilistic relational planning rules. *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling, ICAPS04.*

Samuel, A. L., (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3), 211–229.

Shahaf, D & Amir, E. (2006). Learning partially observable action schemas. *In Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06).*

Shen, W. & Simon. (1989). Rule creation and rule learning through environmental exploration. In *Proceedings of the IJCAI-89*, pages 675–680.

Spector, L. (1994) Genetic programming and AI planning systems. In *Proceedings of Twelfth National Conference on Artificial Intelligence*, Seattle,Washington,USA, AAAI Press/MIT Press.

Veloso, M. (1994). *Planning and learning by analogical reasoning*. Springer Verlag.

Winner, E. & Veloso, M. (2003) Distill: Towards learning domain-specific planners by example. In Proceedings of Twentieth International Conference on Machine Learning (ICML 03), Washington, DC, USA.

Yang, Q, Wu, K & Jiang, Y. (2005) Learning action models from plan examples with incomplete knowledge. *In Proceedings of the 2005 International Conference on Automated Planning and Scheduling, (ICAPS 2005) Monterey, CA USA*, pages 241–250.

Yoon, S., Fern, A., & Givan, R., (2006). Learning heuristic functions from relaxed plans. *In International Conference on Automated Planning and Scheduling (ICAPS-2006)*.

KEY TERMS

Control Rule: *IF-THEN* rule to guide the planning search-tree exploration.

Derived Predicate: Predicate used to enrich the description of the states that is not affected by any of the domain actions. Instead, the predicate truth values are derived by a set of rules of the form **if** formula(x) **then** predicate(x).

Domain Independent Planner: Planning system that addresses problems without specific knowledge of the domain, as opposed to *domain-dependent planners,* which use domain-specific knowledge.

Macro-Action: Planning action resulting from combining the actions that are frequently used together in a given domain. Used as control knowledge to speed up plan generation.

Online Learning: Knowledge acquisition during a problem-solving process with the aim of improving the rest of the process.

Plateau: Portion of a planning search tree where the heuristic value of nodes is constant or does not improve.

Policy: Mapping between the world states and the preferred action to be executed in order to achieve a given set of goals.

Search Control Knowledge: Additional knowledge introduced to the planner with the aim of simplifying the search process, mainly by pruning unexplored portions of the search space or by ordering the nodes for exploration.