Improving Automated Planning with Machine Learning

Susana Fernández Arregui, Sergio Jiménez Celorrio and Tomás de la Rosa Turbides
Departamento de Informática. Universidad Carlos III de Madrid,
Avda. de la Universidad, 30. 28911-Leganés, MADRID ESPAÑA
(Phone) +34 - 91 624 62 59, (Fax) +34 - 91 624 91 29
sfarregu@inf.uc3m.es, sjimenez@inf.uc3m.es, trosa@inf.uc3m.es

ABSTRACT

This chapter reports the last Machine Learning techniques for the assistance of Automated Planning. Recent discoveries in Automated Planning have opened the scope of planners, from toy problems to real-world applications, making new challenges come into focus. The planning community believes that Machine Learning can assist to address these new challenges. The chapter collects the last Machine Learning techniques for assisting automated planners classified in: techniques for the improvement of the planning search processes and techniques for the automatic definition of planning action models. For each technique, the chapter provides a deep analysis of their domain, advantages and disadvantages. Finally, the chapter draws the outline of the new promising avenues for research in learning for planning systems.

INTRODUCTION

Automated Planning (AP) is the branch of Artificial Intelligence (AI) that studies the computational synthesis of sets of actions to carry out a given task (Ghallab et al., 2004). AP appeared in the late '50s from converging studies into state-space search, theorem proving and control theory to solve the practical needs of robotics. The STanford Institute Problem Solver STRIPS (Fikes & Nilsson, 1971), developed to be the planning component for controlling the autonomous robot Shakey (Nilsson, 1984), perfectly illustrates the interaction of these influences. Since the Shakey's days up to now, AP has created efficient planners and well accepted standards for representing and solving the planning tasks. In the last years, AP systems have been successfully applied for controlling underwater vehicles (McGann et al 2008), for the management of fire extinctions (Castillo et al., 2006), for the planning of space missions (Bresina et al., 2005), etc . But despite of all these successful examples, the application of AP systems to real-world problems is still complicated, mainly due to the following two factors. First, the search for a solution plan in AP is a PSpace-complete[1] problem (Bylander, 1991). One can easily find planning problems that overwhelm the capacities of the off-the-shelf planners. Second, AI planners need to be fed with accurate description of the planning tasks. These

[1]PSpace-complete problems are those problems that are PSPACE (memory needs linear with respect to the length of the problem specification) and every problem in PSPACE can be reduced to it in polynomial time [Sipser, 1997]

descriptions consist of a model of the actions that can be carried out in the environment together with a specification of the state of the environment and the goals to achieve. Knowing in advance this information is unfeasible in most of the real-world problems.

The following sections describe different approaches for applying Machine Learning (ML) to assist AP in order to overcome these two problems. The chapter is organized as follows. The first section explains the basic concepts of AP. Second section describes the common issues about using ML for AP. Third section reviews the last techniques for automatically learning AP control knowledge. Fourth section reviews the last advances for the learning of AP action models. The fifth section depicts the current challenges of ML for AP. And finally, section sixth discuses some conclusions.

BACKGROUND

An AP task is defined by two elements: (1) a set of actions that represents the state-transition function of the world (*the planning domain*) and (2), a set of facts that represent the initial state together with the goals of the AP task (*the planning problem*). These two elements are typically represented in languages coming from the first-order logic. 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). Since that date, PDDL has become the standard representation language for the AP community. According to the PDDL specification (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 effects, which is a list of predicates indicating the changes in the state after the action application. Like STRIPS, PDDL follows the closed world assumption to solve the frame problem. Regarding this assumption, what is not currently known to be true, is false.

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, such as learning control knowledge, for solving interesting AP problems. In the late 90's, a significant scale-up in planning took place due to the appearance of the reachability planning graph (Blum & Furst, 1995) and the development of powerful domain independent heuristics (Hoffman & Nebel, 2001; Bonet & Geffner, 2001). Planners using these advances can often synthesize 100-action plans just in seconds. So, at the present time, there is not such dependence on ML for solving AP problems. However, there is a renewed interest in learning for AP motivated by two factors: (1) IPC-2000 showed that knowledge-based planners scale up dramatically better than 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. With this aim, the IPC-2008 introduced a specific track for learning-based planners. And, (2) there is a need for tools that assist in the definition, validation and maintenance of the AP models. At the moment, these processes are still done by hand. Since 2005, the International Competition on Knowledge Engineering for Planning Systems (ICKEPS) takes place every two years with the aim of encouraging the development of tools for modeling AP tasks.

LEARNING ISSUES WITHIN AUTOMATED PLANNING

Some common issues has to be considered when designing a learning process for the assistance of a given AP system. The most relevant ones are: (1) How to represent the learned knowledge, (2) which algorithm is suitable for learning this knowledge, and (3) how to collect examples for learning this knowledge. Though there are no general answers, this section gives clues to these questions.

The Representation Language

AP tasks are commonly defined in predicate logic because it allows one to represent problems in a compact way and to capture relations between objects. However, other representation languages have been used to express the learned knowledge in order to make the learning processes more efficient. Languages for describing object classes has been shown to provide a useful learning bias for many AP domains. These languages include C*oncept Language* (Martin & Geffner, 2000) or *Taxonomic Syntax* (McAllester & Givan, 1993). These encoding have the expressive power of standard first order logic with a syntax that is suited for representing and reasoning with classes of objects. For example, the useful concept of block "well-placed" in the Blocksworld domain[2] can be defined using these languages in a very compact way: *A block is well placed when it is on the right block or table and all blocks beneath it are "well-placed" too*.

When the AP task is compiled into another form of problem solving, e.g., Constraint Satisfaction Problems (CSP), Boolean Satisfiability Problems (SAT) or Model Checking, the representation language of the learned knowledge must suit to the corresponding compilation. In the case of Model Checking planners, like the TLPlan system (Bacchus & Ady, 2001), control knowledge is represented as new plan requirements expressed by temporal logic formulas. In the case of planners based on CSP, like the system GP-CSP (Do & Kambhampati, 2000), search failures were stored as variable-value assignations with the meaning of not being part of the solution.

The selection of a *feature space* able to capture the significant knowledge of the domain is also a key issue. Traditionally, this *feature space* has consisted on predicates for describing the *current state* and the *goals* of the planning task. The PRODIGY system (Minton 1988) enriched the *feature space* with extra predicates, called meta-predicates (Borrajo & Veloso, 1997). Metapredicates captured useful knowledge of the planning context, like the current applicable operators or the goals pending to be solved. Recent works on learning general policies has extended this definition of meta-predicates including predicates for capturing the actions of the relaxed plan in the current state (Yoon et al., 2007) or the current helpful actions (de la Rosa et al. 2008).

The learning algorithms

[2] The blocksworld is a classic domain in AP which consists of a set of blocks, a table and a gripper: the blocks can be on other blocks or on the table, a block that has nothing on it is clear and the gripper can hold one block or be empty. Because its simpleness and clarity it is by far the most frequently domain used in the AI planning literature.

One of the most significant issue that arises in applying ML to AP is the learning algorithm. Next, there is a description of the ML mechanisms that best suit the AP task.

Inductive Logic Programming

Inductive Logic Programming (ILP) arises from the intersection of ML and logic programming and is concerned about the development of inductive algorithms to learn a given target logic concept, from examples and background knowledge. Formally, the inputs to an ILP system are:

- A set of ground learning examples *E* of the target concept, consisting of true examples (positive examples) *E+* and false examples (negative examples) *E-* of the concept.
- A representation language *L*, specifying the syntactic restrictions of the domain predicates.
- Background knowledge *B*, which provides additional information to argument the classification of the learning examples

With these three inputs, the ILP systems try to find a hypothesis *H*, expressed in *L* which agrees with the examples *E* and the background knowledge *B*.

ILP has traditionally being considered a binary classification task for the given target concept. However, in the recent years, ILP techniques have broadened to cover the whole spectrum of ML tasks such as regression, clustering or association analysis. Particularly, there is a successful new trend in ILP consisting on extending the classical propositional ML algorithms to the relational framework:

- Learning relational decision trees. A very well-known approach for multi-class classification consists of finding the smallest decision tree that fits a given data set. The common way to find these decision trees is following the *Top-Down Induction of Decision Trees* (TDIDT) algorithm (Quinlan, 1986). This algorithm builds the decision tree by splitting the training examples according to the values of a selected attribute that minimizes a measure of variance along the prediction variable. The leaves of learned trees are labeled by a class value for the target concept that fit the examples that satisfy the conditions along the path from the root of the tree to that leaf. Relational decision trees are the first-order logic upgrade of the classical decision trees (Blockeel & De Raedt, 1998). Unlike the classical ones, relational trees work with examples described in a relational language such as predicate logic. This means that each example is not described by a single feature vector but by a set of logic facts. Therefore, the nodes of the tree do not contain tests about the examples attributes, but logic queries about the facts holding in the examples.

- Relational instance based learning. Unlike tree learning methods that explicitly construct a classifier from learning examples, these techniques perform what is called a *lazy learning*, i.e. they simply store the learning examples and delay the generalization until the moment a new example have to be classified. One disadvantage of these methods is that the cost of classifying an instance can be high because it implies all the generalization computations. On the other hand the generalization of these methods is local to the neighborhood of the new example which allows one to achieve better

performance when the classification function is very complex. The generalization in all these methods is done by analyzing instances similar to the new query instance ignoring the ones that are very different. Thereby a critical issue is the definition of an accurate similarity measure among the instances described in predicate logic (M. Sebag, 1997), (Jan Ramon and Maurice Bruynooghe, 2001).

Explanation Based Learning

EBL explains the learning examples using the domain theory. An explanation is a justification for why a subset of learning examples merits its assigned class. Other learning examples are used to evaluate these conjectured explanations. The informational effect of a confirmed explanation is equivalent to all examples covered by the explanation. Since this can be significantly larger than the learning set, the learner can draw much stronger statistical conclusions than would be warranted from the learning examples alone. Formally, the inputs to an EBL system are:

- A set of learning examples *E*.
- A domain theory *D* consisting of correct predicate definitions.
- The current hypothesis *H* which is incorrect or should be redefined in other terms.
- A representation language *L* specifying the syntactic constraints on the predicate definitions.

The EBL task is formally defined as learning a new hypothesis *H'* such as *H'* is complete and consistent with the learning examples and the domain theory *D*.

Reinforcement Learning

ILP and EBL shared a common feature: the learning success depended on outside expertise (1) to decide what is worth to learn, i.e., the learning target and (2) to collect significant examples of the learning target e.g., the set of positive and negative examples in ILP for learning a given logic concept. But such outside expertise is not always available. Reinforcement Learning (RL) poses a different approach to deal with this limitation: RL tries to learn how to take a decision in a given environment by trial and error. According to this, RL reduces the external supervision to a reward signal guidance.

Most of the work on RL has focused on propositional representations for the state and the actions. Because of that, traditional RL algorithms are not applicable to domains with relational structure, like the AP ones, without extensive engineering effort. Recently, there are coming up numerous attempts to naturally bring together RL and the expressiveness of relational representations for states and actions. They are included in a new AI field called Relational Reinforcement Learning (RRL). Currently, the different approaches of RRL to generalize RL to the relational setting consist of:

- Relational learning of the value function. This approach consists of using relational regression tools to generalize the value function. So far, three different relational regression algorithms have been used:
    - Relational regression trees (Dzeroski et al., 2001). For each goal, a regression tree is build from examples consisted on pairs (state,q-value). The test nodes of the induced tree represent the set of facts that have to be holding for the prediction to be true. The leaf nodes of tree represent the different predictions of the q-value.

- Instance based algorithm with relational distance (Driessens & Ramon, 2003). In this case a k-nearest neighbor prediction is done. It computes a weighted average of the Q-values of the examples stored in memory where the weight is inversely proportional to the distance between the examples. This distance measure copes with the relational representations of the states and actions of the examples.
- Relational Kernels methods (Gartner et al. 2003). They use the incremental regression algorithm *Gaussian Processes* to approximate the mappings between q-values and state-action pairs. In order to employ *Gaussian Processes* in a relational setting they make use graph kernels as the covariance function between state-action pairs.

- Relational learning of policies. An important drawback of the previous methods is that the value function can be very hard to predict in stochastic tasks. An alternative approach is trying to directly learn the policies and only implicitly represent the value function. That is, given an explicit representation of a policy *pi*, the implicitly represented value function is the value obtained by executing *pi* repeatedly from each state. Note that in this case, a classifier, like relational decision trees (Dzeroski et al., 2001), is needed to learn the optimal *pi* rather than the regression learners previously used. The advantage of this alternative approach is that usually it is easier to represent and learn suitable policies for structured domains than to represent and learn accurate value functions.

The Generation of Learning Examples

The success of the ML algorithms strongly depends on the learning examples used. In the case of AP, these learning examples are extracted from the experience collected solving training problems. Therefore, the quality of the learning examples will depend on the quality of the problems used for training. Traditionally, these training problems are obtained by random generators with some parameters to tune the problems difficulty. This approach presents two limitations. First, is not trivial to guarantee that a given AP problem is solvable. Formally, proving that from a given initial state, one can reach a state where goals are true is as hard as the original AP task. Second, the parameters of the AP problem generators are domain-dependent (the number and type of the objects of the world are different for different AP domains). Tuning these parameters for generating good quality learning examples implies domain expertise.

In order to avoid the domain expertise for obtaining good learning examples, (Yoon et al., 2004) introduced *Random Walks*. *Random Walks* is an automatic and domain-independent strategy for "bootstrapping" the learning process by using automatically generated problems of gradually increasing walk length. First, it generates a random initial state s_0. Second, starting at s_0 it takes n uniformly random actions (only select random actions from the set of applicable actions in a state s_i,) to produce the state sequence $(s_0,s_1,..,s_n)$. Finally, it returns the planning problem with initial state s_0 and the set of goals s_n. This approach only requires the availability of an action simulator for the planning domain. However, in domains where complexity depends on the number of world objects this approach is not enough.

An alternative approach consists of generating training problems using a previous one (Fuentetaja & Borrajo, 2006). In this case the learning process has the bias imposed by this

previous problem, but it allows generating a random exploration of problems of increasing difficulty in domains where complexity depends on the number of world objects. This approach also presents a limitation. When the domain is not symmetrical (i.e. for each instantiated operator *oi*), there must be a sequence of instantiated operators, such that, if applied, they return to the planning state before applying *oi*, otherwise, it could easily generate unsolvable problems.

LEARNING PLANNING CONTROL KNOWLEDGE

On one hand, the planning systems that are able to exploit domain-specific knowledge can perform orders of magnitude faster than the off-the-shelf planners. On the other, real-world applications make use of control knowledge to find solution plans fitting the user preferences. In both cases, hand-coding this domain-specific control knowledge is tricky because it implies expertise on of both the planning domain and the planning system. This section revises the different techniques for automatically learning search control knowledge according to the target of their learning process: planning sequences, control rules, planning policies, hierarchies or evaluation function.

Learning Planning Sequences

The target of learning sequences refers to capture portions of planning episodes that can occur in other problems; therefore a previous recorded situation may help in the process of solving the new problem. If the sequences becomes directly from solution plans they are called macro-actions. If the sequences are abstractions of decisions taken during the planning search, they can take different means, but generally called planning cases.

Learning MacroActions

Macro-actions are the first attempt to speed-up the planning process. They are extra actions added to a given domain theory resulting from combining the actions that are frequently used together. Figure 1 shows an example of the macro-action UNLOAD-DROP induced by the MacroFF (Botea et al., 2005) system for the Depots domain[3].

```
(:action UNLOAD
  :parameters (?x - hoist ?y - crate ?t - truck ?p - place)
  :precondition (and (in ?y ?t) (available ?x) (at ?t ?p) (at ?x ?p))
  :effect (and (not (in ?y ?t)) (not (available ?x)) (lifting ?x ?y)))

(:action DROP
  :parameters (?x - hoist ?y - crate ?s - surface ?p - place)
  :precondition (and (lifting ?x ?y) (clear ?s) (at ?s ?p) (at ?x ?p))
  :effect (and (available ?x) (not (lifting ?x ?y)) (at ?y ?p)
           (not (clear ?s)) (clear ?y) (on ?y ?s)))
```

[3] The Depots domain consists of actions to load and unload trucks, using hoists that are available at fixed locations. The loads are all crates that can be stacked and unstacked onto a fixed set of pallets at the locations.

```
(:action UNLOAD-DROP
  :parameters (?h - hoist ?c - crate ?t - truck ?p - place ?s - surface)
  :precondition (and (at ?h ?p) (in ?c ?t) (available ?h)
                (at ?t ?p) (clear ?s) (at ?s ?p))
  :effect (and (not (in ?c ?t)) (not (clear ?s))
            (at ?c ?p) (clear ?c) (on ?c ?s)))
```

Figure1. Example of macro-action induced by MacroFF for the Depots domain.

The use of macro-actions reduces the depth of the search tree. However, this benefit decreases with the number of new macro-actions as they enlarge the branching factor of the search tree causing the utility problem. To decrease this negative effect filters deciding the applicability of the macro-actions should be defined.

Since the beginning of AP the use of macro-actions has been widely explored. Traditionally, most of the techniques use an off-line approach to generate and filter macro-actions before using them in search. Early works on macro-actions began with a version of the STRIPS planner (Fikes et al., 1972). It used previous solution plans and segments of the plans as macro-actions to solve subsequent problems. Later, (Korf, 1985) extended this approach by adding some filtering heuristics to prune the generated set of macro-actions. This approach distinguished between two types of macro-actions: S-macros that occur frequently during search and T-macros, those that occur less often, but model some weakness in the heuristic. The REFLECT system took the alternative approach of forming macro-actions based on pre-processing of the domain (Dawson & Silklossly, 1977). All sound pairwise combinations of actions were considered as macro-actions and filtered through some basic pruning rules. Due to the small size of the domains with which the planner was reasoning, the number of macro-actions remaining following this process was sufficiently small to use in planning.

More recent works on macro-actions include the IPC4 competitor MacroFF (Botea et al., 2005). In this work macro-actions are extracted in two ways: from solution plans and by the identification of statically connected abstract components. Besides, an off-line filtering technique is used to prune the list of macro-actions. Marvin also competed in IPC4 using action-sequence-memorization techniques to generate macro-actions on-line that allow the planner escape from plateaus without any exploration (Coles & Smith, 2007). Recently, (Newton, 2007) uses a genetic algorithm to generate a collection of macro-actions independently of the source planner, and then filters this collection through an off-line filtering technique similar to that used by MacroFF.

Learning Planning Cases

Planning Cases consist of traces of past solved planning problems. Unlike macro-actions, cases can memorize goals information. PRODIGY/ANALOGY (Veloso & Carbonell, 1993) introduced the application of derivational analogy and abstraction to planning. This system stored planning traces to avoid failure paths in future exploration of the search tree. To retrieve similar planning traces, PRODIGY/ANALOGY indexed them using the minimum preconditions

to reach a set of goals. Recently, typed sequences cases have been used in heuristic planning for node ordering during the plan search (de la Rosa et al., 2007). They have shown that a reduction in the number of heuristic evaluations can improve the planner performance in terms of time. Contrary to macro-actions, this technique is object-centered, useful for domains in which different goal situations determine the plan construction.

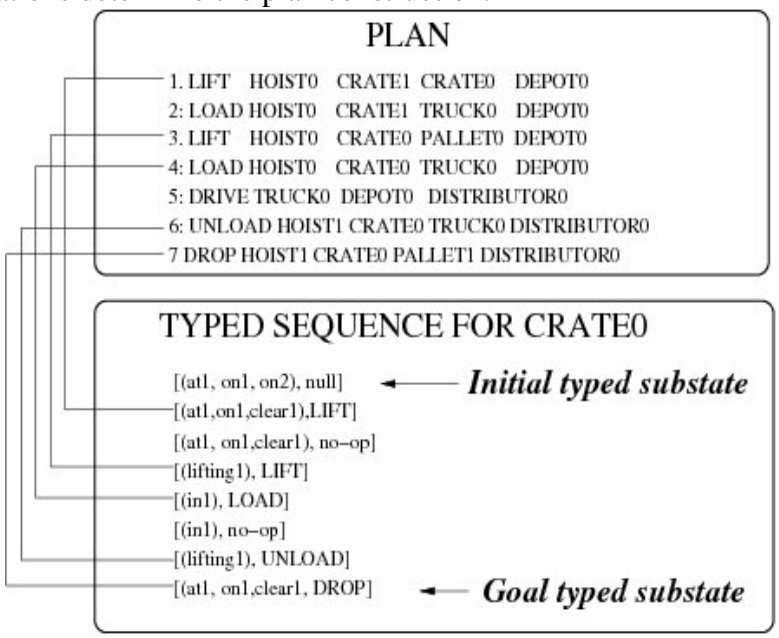

Figure 2. Example of a planning case learned for the Depots domain.

Learning Control Rules

A control rule is an *IF-THEN* rule for guiding the exploration of the planning search tree. Control rules can guide the exploration in two different ways: for node pruning or for node ordering during the tree exploration. Figure 3 shows and example of a control rule for node punning learned for the PRODIGY (Minton 1988) system in the Depots domain.

```
(control-rule select-load-for-available-hoist
   (if (and (current-goal (available <h1>))
            (type-of-object <c1> crate)
            (type-of-object <h1> hoist)
            (type-of-object <l1> place)
            (type-of-object <s1> surface)))
   (then select operator load))
```

Figure 3. Example of a control rule learned by PRODIGY for the logistics domain.

Control-rules enrich the planning language with extra predicates, called meta-predicates. Meta-predicates allow the planner to capture specific knowledge of the given domain, e.g., the applicable operator, the current goals the planner is trying to solve. However, the knowledge captured depends strongly on the quality of the learning examples used. When the learning

examples are not significant enough the induced control rules may mislead the search process of the planner. Besides, control-rules also suffer from the utility problem.

On one hand, there are systems that inductively learn control rules. Among these systems Inductive Learning Programming (ILP) is the most popular learning technique. The Grasshopper system (Leckie & Zukerman, 1991) used the Foil (Quinlan & Cameron-Jones, 1995) ILP system to learn control rules that guide the PRODIGY planner when selecting goals, operators and binding operators. Purely inductive systems need a big number of learning examples to acquire efficient control knowledge. On the other hand, there are analytical systems. Static that obtains control rules without any training example just using EBL to analyze the relations between actions' preconditions and effects (Etzioni, 1993). The main disadvantage of these methods is that, as there are no training examples, there is no measure of the learned knowledge utility. With the aim of solving the problems of the purely inductive and purely analytical approaches some researchers have tried to combine them: the pioneering systems based on this principle are LEX-2 (Mitchell et al., 1982) and Meta-Lex (Keller, 1987). AxA-EBL combined EBL + induction (Cohen, 1990) . It first learns control rules with EBG and then refines them with training examples. Dolphin was an extension of AxA-EBL which used Foil as the inductive learning module (Zelle & Mooney, 1993). The Hamlet system combines deduction and induction in a incremental way (Borrajo & Veloso, 1997). First it learns control rules with EBL that usually are too specific and then uses induction to generalize and correct the rules. EvoCK applies the genetic programming paradigm to solve the learning problems caused by the hypothesis representation language (Aler et al., 2002) .

Learning General Policies

A policy is a mapping between the world states and the preferred action to be executed in order to achieve a certain set of goals. A general policy it is a mapping for the problem instances of a given domain, i.e. the diverse combinations of initial state and goals, into the preferred action to be executed in order to achieve the goals. Thereby, a good general policy is able to solve all the possible problems instances of a given domain. However, though no general method is known for computing such general policies, they can be defined by-hand for many domains. For example, a general policy for the Blocksworld domain can be given as follows:

```
(1) put all blocks on the table, and then
(2) move block x on block y when x should be on y,
    both blocks are clear, and y is well placed
```

The problem of learning general policies was first studied in (Khardon, 1999). Khardon induced general policies for both the Blocksworld and the Logistics domain by extending the decision list learning algorithm (Rivest, 1987) to the relational setting. This first approach presented two important weaknesses: (1) it relied on human-defined background knowledge that expressed key features of the domain, e.g. the predicates *above(block1,block2)* or *in_place(block)* for the Blocksworld, and (2) the learned policies do not generalize well when the problem size is increased. Martin and Geffner solved the weaknesses of Khardon's approach in the Blocksworld domain by changing the representation language of the general policies (Martin & Geffner,

2000). Instead of representing the current state and the problems goals using predicate logics they used a Concept Language. This representation allows them to learn rules of the form:

```
IF an object is in a certain class,
THEN do a certain action on the object.
```

In the last years the scope of general policy learning has been augmented over a diversity of domains making this approach to be competitive with the state-of-the-art planners. This achievement is due to the assimilation of two new ideas: (1) The policy representation language is enriched with new predicates. Meta-predicates are introduced to capture more effective domain specific knowledge. (2) The learned policies are not longer greedily applied but combined with heuristic planning algorithms. Specifically (Yoon et al., 2007) compliments the information of the current state with the relaxed planning graph and the learned policies are used during node expansions in a best-first search heuristic algorithm. At each node expansion of the best-first search, they add to the search queue the successors of the current best node as usual, but they also add the states encountered by following the policy from the current node for a given horizon.

Recently, the system Roller has defined the problem of learning general policies as a two-step standard classification process (de la Rosa et al., 2008). As a consequence, an off-the-shelf relational classifier can be used for general policy learning. At the first step the classifier captures the preferred operator to be applied in the different planning contexts. At the second step the classifier captures the preferred instantiation for each operator in the different planning contexts of a given domain. These contexts are defined by the set of helpful actions extracted from the relaxed planning graph of a given state, the goals remaining to be achieved, and the static predicates of the planning task. Additionally, Roller implemented two methods for guiding the search of a heuristic planner with the learned policies. The first one consists of using the resulting the policy in a Depth First Search algorithm. The second one consists of ordering the node evaluation of the Enforced Hill Climbing algorithm with the learned policy.

Learning Hierarchical Knowledge

Hierarchical planning combines hierarchical domain-specific representation of the planning models together with domain-independent search for problem solving. One of the best-known approaches for modeling hierarchical knowledge about a planning domain is Hierarchical Task Network (HTN). Current HTN planners can outperform the state-of-the-art ones and provide a natural modeling framework for many real-world applications like fire extinctions (L. Castillo, J. Fdez.-Olivares, O. García-Pérez & F. Palao, 2006), evacuation planning (Muñoz-Avila et al, 1999) or manufacturing.

In HTN planning, complex tasks are decomposed into simpler tasks until a sequence of primitive actions is generated. Therefore, the input to an HTN planner includes a set of operators similar to the ones used in classical planning (primitive actions) and a set of methods describing how tasks should be decomposed into subtasks in this particular domain. Figure 5 shows the method for the task *transport-crate(crate,place)* from HTN description of the Depots domain.

```
(:method
  ; head
      (transport-crate ?crate ?destination)
  ; precondition
      (and
         (truck ?truck)
         (at ?truck ?place1)
         (at ?crate ?place2)
         (different ?place1 ?place2))
  ; subtasks
       (:ordered (drive ?truck ?place1)
                 (load ?crate ?truck)
                 (drive ?truck ?place2)
                 (unload ?crate ?truck)))
```

Figure 4. Example of a method for hierarchical planning in the Depots domain.

The specification of these hierarchical methods by hand is a hard task as expert knowledge is needed. And defining a general algorithm for automatically learning them for any domain is still an open issue. The ALPINE system completely automates the generation of abstraction hierarchies from the definition of a problem space (Knoblock, 1990). Each abstraction space in a hierarchy is formed by dropping literals from the original problem space, thus it abstracts the preconditions and effects of operators as well as the states and goals of a problem. Concerning abstraction in planning, there also exists the system PARIS that stores each case in different levels of abstraction (Bergmann & Wilke, 1992). To solve new problems, the problem will be compare with cases of the hierarchy of abstractions and the planner is used to refine the abstract case and to adapt it to the new problem. X-learn is a system that uses a generalize-and-test algorithm based on ILP to learn goal-decomposition rules (Reddy & Tadepalli, 1997). These (potentially recursive) rules are 3-tuples that tell the planner how to decompose a goal into a sequence of subgoals in a given world state, and therefore are functionally similar to methods in HTN domains. X-learn training data consists of solutions to the planning problems ordered in an increasing order of difficulty (authors refer to this training set as an exercise set, as opposed to an example set, which is a set of random training samples without any particular order). This simple-to-hard order in the training set is based on the observation that simple planning problems are often subproblems of harder problems and therefore learning how to solve simpler problems will potentially be useful in solving hard ones. Hicap (Hierarchical Interactive Case-based Architecture for Planning) integrates the SHOP hierarchical planner together with a case-based reasoning (CBR) system named NaCoDAE to assist with the authoring of plans for noncombatant evacuation operations (H. Muñoz-Avila et al. 1999). It interacts with users to extract a stored similar case that allows one to solve the current problem. The system CaseAdvisor also integrates CBR and hierarchical planning (C. Carrick, Q. Yang, I. Abi-Zeid & L. Lamontagne, 1999). It uses plans previously stored to obtain information of how to solve a task instead of choosing the refining method. The KnoMic (Knowledge Mimic) is a ML system which extracts hierarchical performance knowledge by observation to develop automated agents that intelligently perform complex real world tasks (M. van Lent & J. Laird, 2001). The

knowledge representation learned by KnoMic is a specialized form of Soar's production rule format (P. S. Rosenbloom, A. Newell & J. E. Laird, 1993). The rules implement a hierarchy of operators with higher level operators having multiple sub-operators representing steps in the completion of the high level operator. These rules are then used by an agent to perform the same task. Langley and Rogers describes how ICARUS, a cognitive architecture that stores its knowledge of the world in two hierarchical categories of concept memory and skill memory, can learn these hierarchies by observing problem solving in sample domains (P. Langley & D. Choi 2006).

Recent works attempt to automatically learn HTN domain descriptions: CaMeL uses an extended version of the Candidate Elimination incremental algorithm to learn expansions methods in a HTN planner by observing plan traces. It is designed for domains in which the planner is given multiple methods per task, but not their preconditions. That is, the structure of the hierarchy is known in advance and the learning task is to identify under what conditions different hierarchies are applicable. The problem with this work is that it required very many plan traces to converge (completely determine the preconditions of all methods). CaMeL++(O. Ilghami, D. Nau, H. Muñoz-Avila and D. W. Aha, 2005) is also an algorithm for learning preconditions for HTN methods that enables the planner to start planning before the method preconditions are fully learned[referencia]. By doing so, the planner can start solving planning problems with a smaller number of training examples than is required to learn the preconditions completely with insignificant cost of few incorrect plans. Camel required all information about the methods except for their preconditions to be given to the learner in advance, so that the only information for the learner to learn was the methods preconditions. Besides, each input plan trace for Camel needed to contain a lot of information so that the learner could learn from it: At each decomposition point in a plan trace, the learner needed to have all the applicable method instances, rather than just the one that was actually used. The HDL algorithm introduced in (Okhtay Ilghami, Dana S. Nau & Hector Muñoz-Avila,2006) starts with no prior information about the methods, HDL does not need most of that information. At each decomposition point, it only needs to know about one or at most two methods: The method that was actually used to decompose the corresponding task, and one (if there are any) of the methods that matched that task but whose preconditions failed (to serve as a negative training sample).The system HTN-MAKER receives even less input information (C. Hogg, H. Munoz-Avila & U. Kuter). HTN-Maker is able to produces an HTN domain model from a STRIPS domain model, a collection of STRIPS plans, and a collection of task definitions.

Apart from learning decomposition task methods, learning has also been applied to hierarchical planning for other purposes. Lotem and Nau build a method for extracting knowledge on HTN planning problems for speeding up the search (A. Lotem & D. Nau). This knowledge is gathered by propagating properties through an AND/OR tree that represents disjunctively all possible decompositions of an HTN planning problem. This knowledge is used during the search process of GraphHTN[referenica] planner, to split the current refined planning problem into independent subproblems.

Learning Heuristic Functions

A heuristic function *H(s;A;g)* is a function of a state *s*, an action set *A*, and a goals set *g* that estimates the cost of achieving the goals *g* starting from *s* and using the actions from *A*. In case the estimation provided by the heuristic function is accurate, a greedy application of the actions recommended by the heuristic will achieve the goals without search. However, when a heuristic is imperfect, it must be used in the context of a heuristic search algorithm, where the accuracy of the heuristic impacts the search efficiency.

The current domain-independent heuristics for AP are very expensive to compute. As a consequence, heuristic planners suffer from scalability problems. This effect becomes more prominent in domains where the heuristic function is less accurate because, in these domains heuristic planners compute useless node evaluations, e.g. Blocksworld domain. With the aim of avoiding this undesirable effect one can try to directly learn the heuristic function for a given domain: Regarding this approach, (Yoon, Fern, & Givan 2006; Xu, Fern, & Yoon 2007) build a state generalized heuristic function through a regression process. The regression examples consist of observations of the true distance to the goal from diverse states, together with extra information from the relaxed planning graph. The obtained heuristic function provide a more accurate estimation that captures domain specific regularities expressed as a weighted linear combinations of features, that is,

$$H(s;A;g) = wi*fi(s;A;g),$$

where the *wi* are the weights and the *fi* represent the different features of the planning context. The main disadvantage of this leaning approach is that the result of the regression is poorly understandable by humans making the verification of the correctness of the acquired knowledge difficult.

LEARNING PLANNING ACTION MODELS

This section reviews the planning systems that use ML techniques for automatically acquiring knowledge about domain models. Because AP is a model-based form of problem solving it requires complete and correct definition of actions models. However, building action models from scratch is a difficult and time-consuming task, even for planning experts. The following ML techniques learn the parameters, preconditions and/or the effects of planning actions for the automatic definition of actions' models.

The LIVE system is an extension of the GPS problem solving framework with a learning component for rule learning (W.M. Shen & Simon, 1989). LIVE alternates problem solving with the rule learning for the automatic definition of STRIPS-like operators while exploring the world. The decision for alternation mainly depends on *surprises*, i.e., situations where an action's consequences violate its predicted models. When no rule can be found for solving the problem, LIVE will generate and execute an exploration plan, or a sequence of actions, seeking for surprises to extend the rule set. When new rules are learned, problem solving is resumed and a solution plan may be constructed through means-ends analysis. Since the correctness of a solution plan cannot be guaranteed, learning is inevitable at execution time. When the prediction

of a rule fails during execution, LIVE will revise the rule set and then plan another solution for the problem.

The EXPO system refines incomplete planning operators(are missing some preconditions and effects) when the monitoring of a plan execution detects a divergence between internal expectations and external observations (Y. Gil, 1997). Plans are executed while the external environment is monitored, and differences between the internal state and external observations are detected by various methods each correlated with a typical cause for the expectation failure. The methods also construct a set of concrete hypotheses to repair the knowledge deficit. After being heuristically filtered, each hypothesis is tested in turn with an experiment. After the experiment is designed, a plan is constructed to achieve the situation required to carry out the experiment. The experiment plan must meet constraints such as minimizing plan length and negative interference with the main goals.

Unlike the previous works that refined planning operators by an active exploration of the environment, OBSERVER learns PRODIGY operators by observing expert agents and subsequent knowledge refinement in a learning-by-doing paradigm (X. Wang, 1994) . The observations of the expert agent consist of: (1) the sequence of actions being executed, (2) the state in which each action is executed, and (3) the state resulting from the execution of each action. Planning operators are learned from these observation sequences in an incremental fashion utilizing a conservative specific-to-general inductive generalization process. In order to refine the new operators to make them correct and complete, the system uses the new operators to solve practice problems, analyzing and learning from the execution traces of the resulting solutions or execution failures.

The TRAIL system limits its operators to an extended version of Horn clauses so that ILP can be applied (S.S. Benson, 1997) . TRAIL learns the *preimage* or precondition of an action for a TOP operator using ILP. The examples used require the positive or negative examples of propositions held in states just before each actions application. This enables a concept for the *preimage* of an action to be learned for each state just before that action. TRAIL learned well when the positive and negative examples of states before all target actions are given.

The LOPE system achieves learning planning operators by observing the consequences of executing planned actions in the environment (R. Garcia-Martinez & D. Borrajo, 2000) . At the beginning, the system have no knowledge, it perceives the initial situation, and selects a random action to execute in the environment. Then it loops by (1) executing an action, (2) perceiving the resulting situation of the action execution and its utility, (3) learning a model from the perception and (4) planning for further interaction with the environment in case the execution of the plan is finished, or when the system observes a mismatch between the predicted situation the situation perceived. The planning component of LOPE does not explicitly receives a goal input given that LOPE creates its own goals from the situations with the highest utility.

Besides all these systems, the techniques previously revised for the automatic definition of decomposition methods for HTN planning are also considered as learning action's schema techniques

In domains with uncertainty less work has been done: (H. Pasula and L. Zettlemoyer & L. Kaelbling, 2007) presented the first specific algorithm for learning probabilistic planning operators. (S. Jiménez, F. Fernández and Daniel Borrajo, 2008) introduced a mechanism for automatically complete a STRIPS action model with situation-dependent probabilities learned from plan executions. The ARMS system learns preconditions and effects of domain operators only from initial state, goal state and plan in domains where no or partial intermediate states are given (Q. Yang, K. Wu and Y. Jiang, 2005) . In (E. Amir, 2006) introduced an algorithm to exactly learn actions in partially observable STRIPS domains. But still, there is no general efficient approach yet to cope with the problem of partial observability and stochastic environment.

FUTURE TRENDS

Since the STRIPS days, the planning representation languages have evolved to bring together AP algorithms and real-world problems. Nowadays, the PDDL version for the IPC-2008 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. Nevertheless, most of these new features are not efficiently handled by the state-of-the-art planners: they add such extra complexity to the search process that problems becomes extremely difficult to solve. Besides, off-the-shelf planners still fail to scale-up in domains with strong goals interaction, such as the Blocksworld. As it is very difficult to find an efficient general solution to all these challenges, ML must play an important role in addressing them because it can alleviates the complexity of the search process by exploiting regularity in the space of common problems.

Otherwise, the state-of-the-art planning algorithms need a detailed domain description to efficiently solve the AP tasks, but real-world applications like controlling autonomous vehicles, emergency evacuations, etc, imply planning in environments where the dynamics model may be not easily accessible. There is a current need for planning systems capable of progressively acquiring and refining planning models of the environment as more information is available. Otherwise, this issue has already been extensively studied in other AI areas such as RL.

CONCLUSIONS

We have described in this chapter the state-of-the-art in ML for assisting AI planning. Automatic learned knowledge is useful for AP in two different ways: (1) it helps planners guiding their search processes and (2), it assists planning domain designers on the definition of planning action models.

As shown in the paper there is a whole bunch of ML techniques that successfully help planning to solve particular problems. However, research in learning based planning is not mature from a general problem solving point of view. Unlike off-the-shelf automated planners, the learning based systems are usually not able to obtain good performance in a diversity of domains, they frequently implement ad-hoc learning algorithms so they can not benefits from the last advances in ML, and moreover, they lack of theoretical mechanisms to evaluate the utility of the learning knowledge. The specific IPC track for learning control knowledge together with the ICKEPS, promise to settle the formal basis for learning-based AP. These basis involve finding methodologies for system comparisons, building frameworks for rapid algorithm development and test, and keeping the community interest on new challenging problems.

REFERENCES

Aler, R., Borrajo, D. & Isasi, P. (2002). Using genetic programming to learn and improve control knowledge. In, *Artificial Intelligence* (v.141, pp. 29-56).

Amir, E. (2006). Learning Partially Observable Action Schemas. In, *Proceedings of the 21st AAAI Conference on Artificial Intelligence*.

Bacchus, F. & Ady, M. (2001). Planning with Resources and Concurrency: A Forward Chaining Approach. In, *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (pp. 417-424).

Bacchus, F. and Kabanza, F. (2000) Using Temporal Logics to Express Search Control Knowledge for Planning. In, *Artificial Intelligence Journal*, vol. 116, pp 123-191

Bergmann, R. & Wilke, W. (1996). PARIS: Flexible plan adaptation by abstraction and refinement. In, *Workshop on Adaptation in Case-Based Reasoning, ECAI06.*

Benson, S. (1997). Learning Action Models for Reactive Autonomous Agents. In, *phD Thesis Stanford University.*

Blockeel, H. & De Raedt, L., (1998). Top-Down Induction of First-Order Logical Decision Trees. In, *Artificial Intelligence* (v.101, pp. 285-297).

Blum, A.L. & Furst, M. L. (1995). Fast Planning Through Planning Graph Analysis. In, *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (pp. 1636-1642). Montreal, Canada.

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

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

Botea, A., Enzenberger, M., Muller, M. & Schaeffer, J. (2004). Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. In, *Journal of Artificial Intelligence Research* (v.24 pp. 581--621).

Bresina J., A. Jonsson, P.Morris & K. Rajan. (2005). Activity Planning for the Mars Exploration Rovers. In, *Proceedings of the International Conference on Automated Planning and Scheduling.*

Bylander, T. (1991). Complexity Results for Planning. In, *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (pp. 274-279). Sydney, Australia.

Castillo, L., Fdez.-Olivares, J., García-Pérez, O. & Palao, F. (2006). Bringing users and planning technology together. Experiences in SIADEX. In, *Proceedings of the 16th International Conference on Automated Planning and Scheduling.*

Carrick, C., Yang Q., Abi-zeid, I. & Lamontagne, L. (1999). Activating CBR systems through autonomous information gathering. In, *Proceedings of the 3rd International Conference on Case-Based Reasoning*..

Cohen, W. W. (1990). Learning approximate control rules of high utility. In, *Proceedings of the International Conference on Machine Learning*. (pp. 268-276)

Coles, A. & Smith, A. (2007). Marvin: A Heuristic Search Planner with Online Macro-Action Learning. In, *Journal of Artificial Intelligence Research* (v.28 pp. 119--156).

Dawson, C. & Silklossly, L. (1977). The role of preprocessing in problem solving system. In, *Proceedings of the 5th International Joint Conference on Artificial Intelligence* (pp. 465-471).

Do, M. B. & Kambhampati, S. (2000). Solving planning graph by compiling it into a CSP. In, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS00)*.

Driessens, K. & Ramon, J. (2003). Relational Instance Based Regression for Relational Reinforcement Learning. In, *Proceedings of the International Conference on Machine Learning*.

Dzeroski, S., De Raedt, L. & Driessens, K. (2001). Relational reinforcement learning. In, *Machine Learning* (v.43 pp. 7-52).

Etzioni, O., (1993). Acquiring Search-Control Knowledge via Static Analysis. In, *Artificial Intelligence* (v.62 pp. 255-301).

Fikes, R. E. & Nilsson, N. J., (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In, *Artificial Intelligence* (v.2 pp. 189-208).

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

Fox, M. & Long, D. (1998). The Automatic Inference of State Invariants in TIM. In, *Journal of Artificial Intelligence Research* (v.9, pp. 367-421).

Fox, M. & Long, D. (2002). Extending the Exploitation of Symmetries in Planning. In, *International Conference on AI Planning and Scheduling*.

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

Fuentetaja, R., & Borrajo, D. (2006). Improving Control-Knowledge Acquisition for Planning by Active Learning. In, *Proceedings of the European Conference on Machine Learning.*

Ghallab, M. Nau, D. & Traverso, P. (2004) Automated Task Planning. Theory & Practice, Morgan Kaufmann

Garcia-Martinez R. & Borrajo, D. (2000). An Integrated Approach of Learning, Planning, and Execution. In, *Journal of Intelligent and Robotics Systems* (v.29 pp. 49-78).

Gartner, T., Driessens, K. & Ramon, J. (2003). Relational Instance Based Regression for Relational Reinforcement Learning. In, *Proceedings of the International Conference on Machine Learning*.

Gil, Y. (1992). Acquiring Domain Knowledge for Planning by Experimentation. In, *phD Thesis Carnegie Mellon University.*

Hogg, C., Munoz-Avila, H. & Kuter, U. (2008). *HTN-MAKER: Learning HTNs with Minimal Additional Knowledge Engineering Required. In, Proceedings of the 23rd AAAI Conference on Artificial Intelligence*.

Hoffmann, J. & Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. In, *Journal of Artificial Intelligence Research* (v.14 pp. 253-302).

Huffman, S.B., Pearson, D.J., & Laird, J.E. (1992). Correcting imperfect domain theories: A knowledge level analysis. In, *Machine Learning:Induction, Analogy and Discovery*.

Ilghami, O., Nau, D. Muñoz-Avila, H & Aha, D.W. (2002). CaMel: Learning Method Preconditions for HTN planning. In, *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS02)*.

Ilghami, O. Nau, D., & Héctor Muñoz-Avila, H. (2006). Learning to Do HTN Planning. In, *Proceedings of the 16th International Conference on Automated Planning and Scheduling.*

Jiménez, S., Fernández, F. & Borrajo, D. (2008). The PELA architecture: integrating planning and learning to improve execution. In, *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*.

Keller, R. (1987). The Role of Explicit Contextual Knowledge in Learning Concepts to Improve Performance. In, *Rutgers University Technical Report.*

Khardon, R., (1999). Learning Action Strategies for Planning Domains. In, *Artificial Intelligence* (v.113 pp. 125-148).

Knoblock, C. (1990). Learning Abstraction Hierarchies for Problem Solving. In, *Seventh International Workshop on Machine Learning.*

Korf, R. E., (1985). Macro-operators: A weak method for learning. In, *Artificial Intelligence* (v.26 pp. 35-77).

Langley, P. Choi, D. (2006). A Unified Cognitive Architecture for Physical Agents. In, *Proceedings of the 21th AAAI Conference on Artificial Intelligence (AAAI'2006)*.

Leckie, C. & Zukerman I. (1991). Learning Search Control Rules for Planning: An Inductive Approach. In, *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 422-426).

van Lent M. & Laird, J. (2001). Learning Procedural Knowledge through Observation. In, *Proceedings of the International conference on Knowledge Capture.*

Lotem, A. & Nau, D. (2000). New Advances in GraphHTN: Identifying Independent *Subproblems in Large HTN Domains. In, Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems. (pp. 206-215)*

Martin, M. & Geffner, H. (2000). Learning generalized policies in planning using concept languages. In, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS00)*.

Mcallester, D. & Givan, R. (1993). Taxonomic syntax for first order inference. In, *Journal of the ACM.* (v.40, pp. 289-300).

McGann, C., Py, F., Rajan, K., Ryan, H., Henthorn, R.. (2008). Adaptative Control for Autonomous Underwater Vehicles. In, *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*.

Minton, S. (1988) *Learning Effective Search Control Knowledge: An Explanation Based Approach*, Kluwer Academic Publisher.

Mitchell T., Utgoff, T. & Banerjiartin, R. (1982). Learning problem solving heuristics by experimentation. In, *Machine Learning: An Artificial Intelligence Approach*.

Muggleton, S. & Feng, C. (1990). Efficient induction of logic programs. In, *Proceedings of the 1st Conference on Algorithmic Learning Theory*. (p. 368-381)

Muggleton, S. (1995). Inverse Entailment and Progol. In, *New Generation Computing, Special issue on Inductive Logic Programming*. (v.13, p. 245-286)

Muñoz-Avila, H. Aha, D., Breslow, D. & Nau, D. (1999). HICAP: An interactive case based planning architecture and its application to non-combatant evacuation operations. In, *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence.*

Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, W., Wu, D., & F.Yaman. (2003). Shop: An HTN planning system. In, *Journal of Artificial Intelligence Research*, 20, 379-404.

Nebel, B., Dimopoulos, Y. & Koehler, J. (1997). Ignoring Irrelevant Facts and Operators in Plan Generation. In, *Proceedings of the European conference on Planning.*

Newton, M., Levine,J., Fox, M., and Long, D. (2007). Learning Macro-Actions for Arbitrary Planners and Domains. *In, Proceedings of the International Conference on Automated Planning and Scheduling.*

Nilsson, N. J. (1984). Shakey the robot. Technical Report 323, AI Center, SRI International, Menlo Park, CA.

Pasula, H., Zettlemoyer, L. & Kaelbling, L. (2007). Learning Symbolic Models of Stochastic Domain. In, *Journal of Artificial Intelligence Research*, 29, pp. 309-352.

Quinlan, J. R. (1986). Induction of decision trees. In, *Machine Learning* (v.1 pp. 81-106).

Quinlan, J. R. & Cameron-Jones, R. M. (1995). Introduction of logic programs: {FOIL} and related systems. In, *New Generation Computing, Special issue on Inductive Logic Programming* (v.13 pp. 287--312).

Ramon, J., and Bruynooghe, M. (2001). A polynomial time computable metric between point sets. In, *Acta Informatica* (v.10 pp. 765-780).

Reddy, R. and Tadepalli, P. (1997). Learning goal-decomposition rules using exercises. In, *Proceedings of the International Conference on Machine Learning*. (pp. 278-286)

Rivest, R. L., (1987). Learning Decision List. In, *Machine Learning* (v.2 pp. 229-246).

de la Rosa, T., Garcia-Olaya, A & Borrajo D. (2007). Using Cases Utility for Heuristic Planning Improvement. In, *Proceedings of the 7th International Conference on Case-Based Reasoning*. Belfast, Northern Ireland.

de la Rosa, T., Jimenez, S. & Borrajo D. (2008). Learning Relational Decision Trees for Guiding Heuristic Planning. In, *Proceedings of the International Conference on Automated Planning and Scheduling*.

Rosenbloom, P.S., Newell, A & Lairdamon, J.E. (1993). Towards the knowledge level in Soar: the role of the architecture in the use of knowledge . In, T*he Soar papers: research on integrated intelligence* (v.2 pp. 897--933).

Sevag, M. (1997). Distance Induction in First Order Logic. In, *Proceedings of the 7th International Workshop on Inductive Logic Programming*.(p.264-272)

Shen, W.M & Simon (1989). Rule creation and rule learning through environmental exploration. In, *Proceedings of the International Joint Conference on Artificial Intelligence (pp. 675--680).*

Sipser, M. (1997). Introduction to the Theory of Computation. PWS Publishing.

Yang, Q., Wu, K. & Jiang Y. (2005). *Learning Action Models from Plan Examples with Incomplete Knowledge.* In*, Proceedings of the International Conference on Automated Planning and Scheduling*.

Yoon, S., Fern, A. & Givan, R. (2004). Learning domain-specific control knowledge from random walks. In, *Proceedings of the International Conference on Automated Planning and Scheduling.*

Yoon, S., Fern, A. & Givan, R. (2006). Learning Heuristic Functions from Relaxed Plans. In, *Proceedings of the International Conference on Automated Planning and Scheduling.*

Yoon, S., Fern, A. & Givan, R. (2007). Using Learned Policies in Heuristic-Search Planning. In, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (pp. 1636-1642).*

Veloso, M. & Carbonell, J., (1993). Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. In, *Machine Learning* (v.10 pp. 249-278).

Veloso, M., Carbonell, J., Pérez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. In, *JETAI*, 7(1):81–120.

Wang, X. (1994). Learning Planning Operators by Observation and Practice. In, *International Conference on AI Planning Systems*. (pp.335-340)

Xu, Y., Fern A. & Yoon, S. (2007). Discriminative Learning of Beam-Search Heuristics for Planning. In, *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (pp. 2041-2046).

Zelle, J. & Mooney, R. (1993). Combining FOIL and EBG to Speed-up Logic Programs. In, *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 1106-1113).

KEY TERMS & DEFINITIONS

The Frame problem: The problem of expressing a dynamic system using logic without explicitly specifying which conditions are not affected by an action.

Closed World Assumption: Any formula non explicitly asserted in a state is taken to be false, which allows one to avoid the explicit specification of negated literals. This assumptions presupposes that actions only change a small part of the world.

Model Checking: consist of determining whether a given property, usually described as a temporal logic formula, holds in a given model of a system. Traditionally, this problem has been studied for the automatic verification of hardware circuits and network protocols.

Metapredicates: are extra predicates used to reason about the state of the search process in the planner, e.g., the goals that the planner is currently working on, the operators being considered, etc.

Concept language: also known as description logics, is a representation language with the expressive power of fragments of standard first-order logic but with a syntax that is suited for representing and reasoning with classes of objects.

Utility problem: It is a drawback that arise when using learned knowledge cost of using overwhelms its benefit because the difficulty of storage and management the learned information and because of determining which information to use to solve a particular problem.

Bootstrap effect: The fact of learning to solve a problem a teacher can not solve by observing how the teacher solves more simple problems.

Problem distribution: A set of problems belonging to a given domain generated with the same number of world objects and problem goals.