Learning action durations from executions

Jesús Lanchas, Sergio Jiménez, Fernando Fernández and **Daniel Borrajo**
Departamento de Informática.
Universidad Carlos III de Madrid.
Avda. de la Universidad, 30. Leganés (Madrid). Spain.
jesus.lanchas@inbox.com, sjimenez@inf.uc3m.es, ffernand@inf.uc3m.es and dborrajo@ia.uc3m.es

Abstract

Accurate action models are essential for efficiently solving automated planning tasks. An accurate action model allow the planner to precisely foresee the consequences of executing actions in a given environment and therefore to find robust and good quality plans. But when addressing planning tasks in the real world, even hand-coding a simple STRIPS action model is complex, thus defining action models capturing further features, like the execution duration or costs, becomes more difficult. Moreover, if these features can be captured at a given instant they may vary over time. In this paper we automatically model the duration of action execution as relational regression trees learned from observing plan executions. And we show how planners find better plans after incorporating these models to their domain definition.

Introduction

Traditionally, the automated planning community assumes complete and correct actions models for solving the planning tasks. But when addressing real world problems this assumption is far from being true. As a result, diverse mechanisms that automatically define action models have been developed: LIVE (Shen & Simon 1989) learned simple operators with quantified variables while incrementally exploring the world; (Wang 1994) learned PRODIGY operators generalizing observations of actions executions; TRAIL (Benson 1997) used ILP to induce operators represented as an extended version of Horn clauses; LOPE (Garcia-Martinez & Borrajo 2000) incrementally learned reactive planning operators; ARMS (Yang, Wu, & Jiang 2005) and (Amir 2006) learned STRIPS actions in partially observable domains; and (Pasula, Zettlemoyer, & Kaelbling 2004) learned probabilistic planning operators for stochastic domains. All of these works focused on learning action models consisting only in actions preconditions and outcomes.

In the last years, with the aim of bringing together the theoretic algorithms and the real-world problems, the action models for planning have become more complex. Nowadays, PDDL includes numeric variables, explicit time representation and derived predicates that enrich the descriptions of the world. The new planners are designed to cope with all these new features. However, they need accurate definitions of these features and knowing in advance features like the execution cost, duration or relevant derived predicates is unfeasible in many real-world planning tasks. As an example, in domains like workflow or education, the duration and cost spent to perform an activity depend on many variables as who is doing it, the day, the time,... And the cited mechanisms for automatically action modelling were not designed to cope with the new PDDL features.

In this paper we propose a mechanism to automatically learn action-duration models from the observations of plan executions. Specifically we use relational regression tree learning to acquire patterns of the environment state that affect the actions duration. The work described in the paper is focused on learning models for actions durations, but this same approach can be directly applied to learn models for any *fluent* defined in the domain model, like action cost, rewards, etc.

System overview

We propose a three-phase process for the modelling of the action duration: (1) **knowledge acquisition**, where plans are automatically generated, executed and observations of these executions are stored; (2) **model learning**, where the observations are used to induce the models of the duration of the actions; and (3) **redefinition of the action model**, where the induced duration models are incorporated to the action model. Planning with this redefinition of the action model allows to obtain solutions of smaller execution time. Figure 1 shows an overview of the whole modelling process and how its three phases are integrated. The following sections are a detailed description of each of the three modelling phases: knowledge acquisition, model learning and redefinition of the action model.

Phase 1: Knowledge acquisition

During this phase plans are generated, executed and observations of these executions are stored. Although plans can be generated using any off-the-shelf planner we decide using LPG (Gerevini & Serina 2002) because its stochastic behaviour provided us with assorted experience. The plans are executed action by action and after each action execution the corresponding observation is stored. Each execution observation is stored as a tuple of the form (s_n, a_i, f_{n+1}), where:

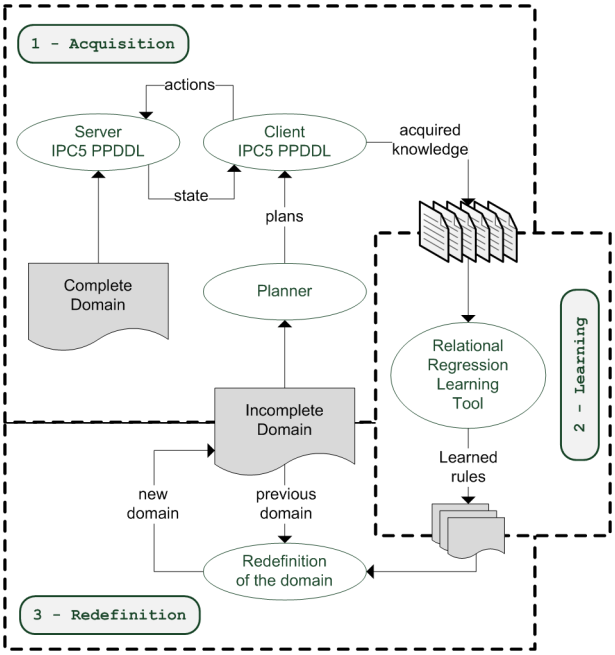

Figure 1: The action duration modelling process.

- s_n is a conjunction of literals representing the facts holding before the action a_i execution.
- a_i is the action executed.
- f_{n+1} is the value of the *fluent* we want to model after the action execution, i.e., in the state s_{n+1}. In our case this fluent is the action duration, but it could be any other *fluent* of the domain.

The Figure 2 shows an observation stored from the execution of the action `pick-up(b1, b2)` from a 3-blocks blocksworld domain. It contains the literals representing the state in which the action was executed, the action, and the time spent in the execution (5 units of time). We tag all the literals with the same *id* (`observation1`) to indicate they all belong to the same observation.

```
% previous state
  on-table(observation1,b0).
  on(observation1,b2,b0).
  on(observation1,b1,b2).
  clear(observation1,b1).
  emptyhand(observation1).
% action executed
  pick-up(observation1,b1,b2).
% fluents values
  spent-time(observation1,5).
```

Figure 2: Knowledge acquired after the execution of the action `pick-up(b1, b2)`.

The outcome of executing the plans is simulated by the software of the probabilistic track of IPC. This software implements a client/server design to simulate the execution of actions according to a given PPDDL action model. **Our approach implements the client side which has a STRIPS action model with no knowledge about the actions durations** (incomplete domain). And the execution simulator implements the server side fed with a PPDDL domain description with the duration model of the actions (complete domain). When the outcome of an execution is unexpected according to the incomplete domain, our planner replans, i.e., the planner is recalled to solve the planning task in this new scenario.

Phase 2: Model learning

At this phase an off-the-shelf system for learning relational regression trees (Kramer 1996) is used to model the execution duration of the actions in the planning domain.

Inducing regression trees is a well-known approach to build models for numeric variables. As decision trees, regression trees are generated by splitting the training examples according to the values of their attributes that minimize a measure of variance along the prediction variable. The leaves of regression trees are labelled by regression formulas that fit the examples that satisfy the conditions along the path from the root of the tree to that leaf. Relational trees are the first-order logic upgrade of these classical trees. Unlike traditional regression trees, relational regression 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, as the example shown in Figure 2. Thus, the nodes of the tree do not contain tests about the examples attributes, but logic queries about the facts holding in the examples.

Specifically, we use the TILDE system (Blockeel & Raedt 1998), a relational tree learning tool that allows the construction of both relational decision and regression trees implementing the *Top-Down Induction of Decision Trees* algorithm (Quinlan 1986). TILDE takes two inputs:

- The knowledge base, containing the examples of the target concept and the background knowledge. Our examples of the target concepts are the values of the fluents (the f_{n+1} of the stored observations) and our background knowledge are the actions executed and the literals holding in the state (the a_i and s_n of the stored observations).
- The settings file, specifying the parameters of the system and the language bias. This *bias* consists of the types of the variables in the predicates and the restrictions about their instantiation. This knowledge is automatically extracted from the definition of the planning domain.

For every action a_i defined in the planning domain we induce a regression tree that models the execution duration of a_i. The nodes of the tree contain the conditions under which the estimation of duration is true. And the leaf nodes contain situation-dependent estimations of the action durations. Thus, the deeper a leaf is in the tree the more specific is its prediction. Figure 3 shows a regression tree induced for the action `pick-up` in a blocksworld where the robot arm can get blocked resulting in a longer execution time. The information of the leaf nodes means: (1) the estimation of

the *fluent* spent-time in square brackets; (2) the number of examples classified in that leaf and (3) the relative error committed when classifying the training examples in that leaf, in square brackets too.

```
spent-time(-A,-B)
[5.52] 75.0
arm-blocked(A) ?
  +--yes: [10.0] 27.0 [0.0]
  +--no:  [5.0] 48.0 [0.0]
```

Figure 3: Regression tree induced for the action pick-up.

Phase 3: Redefinition of the action model

This phase incorporates the induced regression trees into the domain model. Each regression tree induced for a particular action of the planning domain is transformed into conditional effects of this action. The path of test nodes from the root of the regression tree to each leaf corresponds to the conjunction of literals of the condition in the conditional effect. The estimation of the leaf node corresponds to an increase effect. Figure 4 shows the PDDL conditional effects resulting from the transformation of the regression tree of Figure 3. As the outcome of this phase is standard PDDL code it can be used to feed an off-the-shelf numeric planner.

```
(when (arm-blocked)
      (increase (spent-time) 10))
(when (not (arm-blocked))
      (increase (spent-time) 5))
```

Figure 4: The resulting PDDL conditional effects.

Experiments

This section describes the experiments carried out to determine whether the induced duration models lead planners towards better solutions.

The domain

We have used the probabilistic Blocksworld domain from IPC5. Besides the four traditional actions of the blocksworld: pick-up, pick-up-from-table, put-on-block and put-down-on-table, this domain defines the following new set of actions: pick-tower, to pick up two blocks together; put-tower-on-block, to leave a two-blocks tower on a block; and put-tower-down, to leave a two-blocks tower on the table. For all the experiments our system initially does not know the duration of the actions since it is fed with a simple STRIPS action model (Figure 5). And the target of the modelling process is the *fluent* spent-time, that represents the time spent by an action execution.

In the execution simulator we have defined three configurations of the probabilistic blocksworld in order to cover different durations models:

```
(:action pick-up
  :parameters (?b1 ?b2 - block)
  :precondition (and (not (= ?b1 ?b2))
                     (emptyhand)
                     (clear ?b1)
                     (on ?b1 ?b2))
  :effect (and (holding ?b1)
               (clear ?b2)
               (not (emptyhand))
               (not (clear ?b1))
               (not (on ?b1 ?b2))))
```

Figure 5: Example of STRIPS action considered by our system.

1. *Deterministic durations*. This is the simplest configuration, where actions duration is a fixed constant. In this case the actions in the simulator deterministically increase the *fluent* spent-time. As an example, the Figure 6 shows the action pick-up as it is defined in the simulator. In this configuration, the execution of the action pick-up always takes 3 units of time.

```
(:action pick-up
  :parameters (?b1 ?b2 - block)
  :precondition (and (not (= ?b1 ?b2))
                     (emptyhand)
                     (clear ?b1)
                     (on ?b1 ?b2))
  :effect (and (holding ?b1)
               (clear ?b2)
               (not (emptyhand))
               (not (clear ?b1))
               (not (on ?b1 ?b2))
               (increase (spent-time) 3)))
```

Figure 6: Action pick-up in the deterministic configuration of the simulator.

2. *Situation-dependent duration*. In this configuration, the actions durations depend on the state of the robot-arm and the blocks handled. Now the actions in the simulator incorporate conditional effects to increase the fluent (spent-time) according to the state of the robot-arm (arm-blocked) or the handled blocks (is-heavy ?block). Figure 7 shows the definition of the action pick-up in the simulator. In this definition when the robot arm is blocked or is handling heavy blocks it takes more time to complete the execution of the actions than usual. Besides, a probabilistic effect has been introduced to randomly modify the state of the robot arm after the action execution.
3. *Stochastic duration*. In this configuration, the actions have situation-dependent and probabilistic outcomes. It represents the domains where the action duration depends also on circumstances of the environment that are not captured within the domain predicates. Figure 8 shows the definition of the action pick-up in the simulator. According to this schema, one out of three times the execution takes less time without any observable reason.

```
(:action pick-up
  :parameters (?b1 ?b2 - block)
  :precondition (and (not (= ?b1 ?b2))
                     (emptyhand)
                     (clear ?b1)
                     (on ?b1 ?b2))
  :effect
   (and
       (holding ?b1)
       (clear ?b2)
       (not (emptyhand))
       (not (clear ?b1))
       (not (on ?b1 ?b2))
       (when (and (not (is-heavy ?b1))
                  (not (arm-blocked)))
             (increase (spent-time) 3))
       (when (and (not (arm-blocked))
                  (is-heavy ?b1))
             (increase (spent-time) 20))
       (when (and (not (is-heavy ?b1))
                  (arm-blocked))
             (increase (spent-time) 8))
       (when (and (is-heavy ?b1)
                  (arm-blocked))
             (increase (spent-time) 30))
        (probabilistic
             1/2 (arm-blocked)
             1/2 (not (arm-blocked)))))
```

Figure 7: Action `pick-up` in the situation-dependent configuration of the simulator.

The induced models

We have induced action duration models from the observations gathered after solving 50 random 5-blocks problems with the planner LPG. The experiments show that the induced models faithfully correspond to the actual models for each of the three configurations of the simulator.

1. In the deterministic configuration, our models exactly capture the time consumed by the execution of each action. As shown in Figure 9, the induced trees consists of a single leaf node with the exact numerical value for the *fluent* `spent-time` and relative error of 0.0.
2. In the situation-dependent configuration our models successfully captured the conditions of the action durations so the relative error in the leaf nodes is 0.0 too. The Figure 10 shows the logical regression tree induced for the `pick-up` action. The first query tests whether the arm is blocked or not. In both answers the following question is if a heavy (`is-heavy`) block (`C` or `E`) exists. In affirmative answer it is verified if that block is the one that is tried to pick up from the table.
3. In the stochastic configuration our models also captured successfully these conditions of the durations. But in this case, the relative error of the leaf nodes is not 0.0 because the leaf nodes do not represent an exact duration but the average of the durations observed in the examples. Figure 11 shows the logical decision tree induced by TILDE for the action `pick-up` in this configuration.

```
(:action pick_up
  :parameters (?b1 ?b2 - block)
  :precondition (and (not (= ?b1 ?b2))
                     (emptyhand)
                     (clear ?b1)
                     (on ?b1 ?b2))
  :effect
   (and
    (holding ?b1)
    (clear ?b2)
    (not (emptyhand))
    (not (clear ?b1))
    (not (on ?b1 ?b2))
    (when (and (not (is-heavy ?b1))
               (not (arm-blocked)))
          (probabilistic
            2/3 (increase (spent-time) 3)
            1/3 (increase (spent-time) 2)))
    (when (and (not (arm-blocked))
               (is-heavy ?b1))
          (probabilistic
            2/3 (increase (spent-time) 20)
            1/3 (increase (spent-time) 14)))
    (when (and (not (is-heavy ?b1))
               (arm-blocked))
          (probabilistic
            2/3 (increase (spent-time) 8)
            1/3 (increase (spent-time) 5)))
    (when (and (is-heavy ?b1)
               (arm-blocked))
          (probabilistic
            2/3 (increase (spent-time) 30)
            1/3 (increase (spent-time) 20)))
    (probabilistic
          1/2 (arm-blocked)
          1/2 (not (arm-blocked)))))
```

Figure 8: Action `pick-up` in the stochastic configuration of the simulator.

Evaluation of the induced models

Since the aim of our approach is to discover knowledge about the actions execution duration we evaluate the benefits of our approach by comparing the quality of the plans generated with and without incorporating the induced duration models to the initial STRIPS domain theory. We have evaluated the models obtained with the the three configurations of the simulator (deterministic, situation-dependent and stochastic) using a test set of 30 problems of increasing difficulty randomly generated. And we have measure the average value of the metric `spent-time` after solving each of these test problems 15 times. The graphs of the Figure 12 show the obtained results. According to them the domains incorporating the duration models (LPG with ex-

```
spent-time(-A,-B)
[3.0] 2280.0 [0.0]
```

Figure 9: Tree induced by TILDE for the action `pick-up` in the deterministic configuration of the simulator.

```
spent-time(-A,-B)
arm-blocked(A) ?
+--yes: is-heavy(A,-C) ?
|       +--yes: pick-up(A,C,-D) ?
|       |       +--yes: [30] 152.0 [0.0]
|       |       +--no:  [8] 225.0 [0.0]
|       +--no:  [8.0] 9.0 [0.0]
+--no:  is-heavy(A,-E) ?
        +--yes: pick-up(A,E,-F) ?
        |       +--yes: [20.0] 183.0 [0.0]
        |       +--no:  [3.0] 364.0 [0.0]
        +--no:  [3.0] 20.0 [0.0]
```

Figure 10: Tree induced by TILDE for the action `pick-up` in the situation-dependent configuration of the simulator.

```
spent-time(-A,-B)
[9.0935] 1080.0
arm-blocked(A) ?
+--yes:[12.0711] 464.0
|      is-heavy(A,-C) ?
|      +--yes:[13.0779] 385.0
|      |      pick-up(A,C,-D) ?
|      |      +--yes:[27.2174] 115.0 [0.4197]
|      |      +--no: [7.05555] 270.0 [0.0849]
|      +--no: [7.1645] 79.0 [0.1523]
+--no: [6.8506] 616.0
       is-heavy(A,-E) ?
       +--yes:[7.6964] 514.0
       |      pick-up(A,E,-F) ?
       |      +--yes:[18.0714] 168.0 [0.2168]
       |      +--no: [2.6589] 346.0 [0.0255]
       +--no: [2.5882] 102.0 [0.0489]
```

Figure 11: Tree induced by TILDE for the action `pick-up` in the stochastic configuration of the simulator.

perience and Metric-FF with experience) allow to generate plans with smaller execution time.

For the three configurations of the simulator, LPG was run with a limit for planning of '3 solutions or 30 seconds time' and, in the case of *LPG with experience*, with the optimization option to minimize the metric (spent-time). The results obtained by LPG with experience (using the induced models) are overall better. But this is not always true (problems 3, 25 or 29 of the deterministic configuration) because of the stochastic behaviour of this planner. For the three configurations of the simulator, Metric-FF was run with 2 hour time limit for planning and in the case of Metric-FF with experience with the optimization option to minimize the metric (spent-time). The results obtained by Metric-FF are overall better using the induced models but in some problems like problems 1, 16 and 33 of the stochastic configuration, the induced model adds such complexity to the domain theory that Metric-FF optimizing the metric is not able to find any solution in the time limit.

This approach is useful for any domain where goals can be achieved by different plans and the quality of the plans is an important issue. For example, in the probabilistic blocksworld domain. In this domain, problems can be

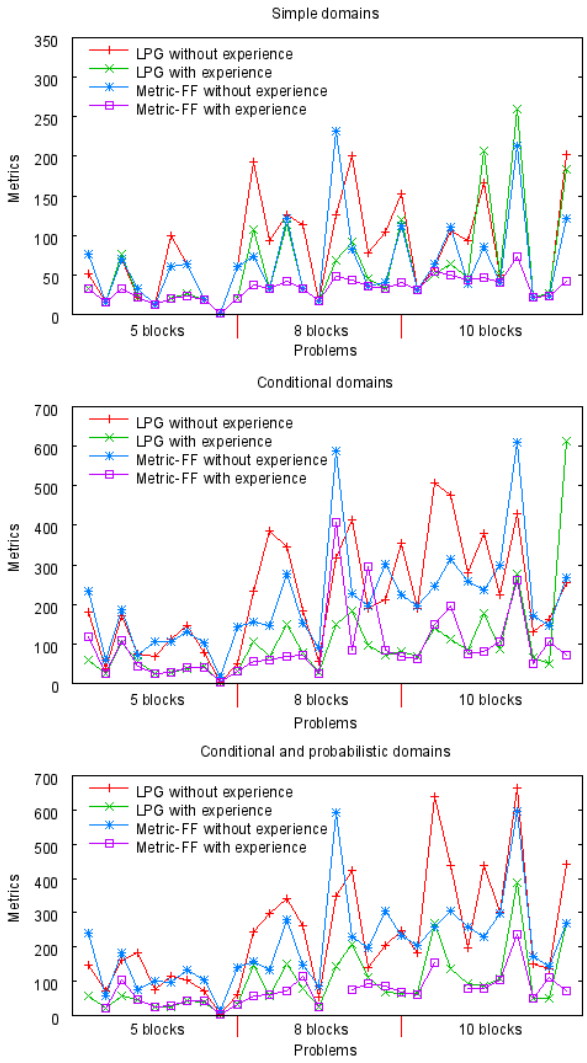

Figure 12: Experimental plans durations.

solved managing blocks individually or handling towers of them. At the beginning our system had no information about which way of handling the blocks was better. After the modelling process we obtained PDDL models for the actions execution duration in the environment that an off-the-shelf numeric planner can directly use to generate good quality plans. The experimental results have shown that because the induced models are correct, considering them to plan provide solutions with smaller execution time. The work described in the paper focuses on learning models for actions durations, but the same approach can be directly applied to learn models for any *fluent* defined in the domain model.

Related work

The relational version of regression trees have not been previously applied to planning action modelling, but the classical versions of both regression and decision trees have being

used for action modelling in autonomous robots. (Haigh & Veloso 1999) used decision tree learning to acquire rules that prioritize the activities of the robot ROGUE according to the values of its sensors. (Balac, Gaines, & Fisher 2000) learned regression trees that proposed the next action for a mobile robot according to the sensed current state of the environment. In both cases, since the learning techniques are propositional the internal nodes of the trees consist only of tests over numerical values. So they were not able to predict according to relational representations of the state, like the ones used in automated planning.

Otherwise, relational tree learning have also been used to induce action policies for reactive systems (Dzeroski, Raedt, & Blockeel 1998). As in our approach, the predictions depended on conditions of the state described relationally. But in this case, the target of the learning process is not an action model but a policy. So unlike our approach, every time a new problem with different goals has to be achieved, new relational trees have to be learnt again, even if the dynamics of the environment do not change.

Conclusions and future work

In this paper we have presented and evaluated an inductive relational machine learning approach to model the duration of planning actions execution. Our approach generates accurate models of the durations of the action execution when they are deterministic, situation-dependent or stochastic and these models can be used by any off-the-self planner. This approach is useful for planning task where goals can be achieved by different plans, and the quality of the plans matters but it is 'a priori' unknown.

In many domains, the duration of an action execution is a function over others *fluents*. In these cases, the estimations of the tree node leaves should not be a numeric value but a mathematical formula. Given that TILDE can learn regression models in the leaf nodes, this tool could be also used for modelling this kind of knowledge. Moreover, our approach assumes the total observability of the environment so the observations obtained after an action executions are always perfect. Further work has to be done in order to acquire models for action durations in environments where the observations could be wrong or incomplete. Besides, the implementation of incrementally learning strategies to on-line incorporate the new learnt knowledge is still in progress.

References

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

Balac, N.; Gaines, D. M.; and Fisher, D. 2000. Using regression trees to learn action models. In *IEEE Systems, Man and Cybernetics Conference, Nashville*.

Benson, S. S. 1997. *Learning Action Models for Reactive Autonomous Agents*. Ph.D. Dissertation, Stanford University.

Blockeel, H., and Raedt, L. D. 1998. Top-down induction of first-order logical decision trees. *Artificial Intelligence* 101(1-2):285–297.

Dzeroski, S.; Raedt, L. D.; and Blockeel, H. 1998. Relational reinforcement learning. In *International Workshop on Inductive Logic Programming*, 11–22.

Garcia-Martinez, R., and Borrajo, D. 2000. An integrated approach of learning, planning, and execution. *Journal of Intelligent and Robotics Systems* 29:47–78.

Gerevini, A., and Serina, I. 2002. LPG: a planner based on local search for planning graphs. *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*.

Haigh, K. Z., and Veloso, M. M. 1999. Learning situation-dependent rules. In *Improving Task Planning for an Incompletely Modelled Domain. 1999 AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information.*

Kramer, S. 1996. Structural regression trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 812–819. Cambridge/Menlo Park: AAAI Press/MIT Press.

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

Quinlan, J. 1986. Induction of decision trees. *Machine Learning* 1:81–106.

Shen, W.-M., and Simon. 1989. Rule creation and rule learning through environmental exploration. In *Proceedings IJCAI-89*, 675–680.

Wang, X. 1994. Learning planning operators by observation and practice. In *Proceedings of the Second International Conference on AI Planning Systems, AIPS-94*, 335–340. Chicago, IL: AAAI Press, CA.

Yang, Q.; Wu, K.; and Jiang, Y. 2005. Learning action models from plan examples with incomplete knowledge. In *In Proceedings of the ICAPS 2005. Monterey, CA USA*, 241–250.