Learning and Transferring Relational Instance-Based Policies

Rocío García-Durán, Fernando Fernández y Daniel Borrajo
Universidad Carlos III de Madrid
Avda de la Universidad 30, 28911-Leganés (Madrid), España
rgduran@inf.uc3m.es, ffernand@inf.uc3m.es y dborrajo@ia.uc3m.es

Abstract

A Relational Instance-Based Policy can be defined as an action policy described following a relational instance-based learning approach. The policy is represented with a set of state-goal-action tuples in some form of predicate logic and a distance metric: whenever the planner is in a state trying to reach a goal, the next action to execute is computed as the action associated to the closest state-goal pair in that set. In this work, the representation language is relational, following the ideas of Relational Reinforcement Learning. The policy to transfer (the set of state-goal-action tuples) is generated with a planning system solving optimally simple source problems. The target problems are defined in the same planning domain, have different initial and goal states to the source problems, and could be much more complex. We show that the transferred policy can solve similar problems to the ones used to learn it, but also more complex problems. In fact, the policy learned outperforms the planning system used to generate the initial state-action pairs in two ways: it is faster and scales up better.

Introduction

Traditionally, first order logic has been used for general problem solving, especially in classical planning (Ghallab, Nau, and Traverso 2004). Classical Reinforcement Learning, however, has used attribute-value representations (Kaelbling, Littman, and Moore 1996). Nevertheless, in the last years both areas, Reinforcement Learning and Planning, are getting close due to two main reasons. On the one hand, the automated planning community is moving towards more realistic problems, including reasoning about uncertainty, as shown by the probabilistic track of the IPC (International Planning Competition) since 2006. On the other hand, Relational Reinforcement Learning (RRL) is using richer representations of states, actions and policies (Dzeroski, Raedt, and Driessens 2001; Driessens and Ramon 2003). Therefore, RRL can serve as a bridge between classical planning and attribute-value reinforcement learning, since both can use first order logic to represent the knowledge about states and actions.

This bridge between the automated planning and reinforcement learning areas opens a wide range of opportunities, since the knowledge generated by one of them could be transferred to the other. For instance, plans generated with automated planners could be used as an efficient guide to reinforcement learning learners. In theory, relational reinforcement learning algorithms based on the learning of the value function could also learn general policies for different problems, and the idea of including the goal as part of the state was introduced early (Dzeroski, Raedt, and Driessens 2001; Fern, Yoon, and Givan 2004). However, most current approaches learn problem/task oriented value functions. That means that the value function learned depends on the task solved, since it depends on the reward of the task. Then, the policy derived from it is only useful for that task, and "ad-hoc" transfer learning approaches are required to reuse the value function or the policy among different tasks. As another example of the potential usefulness of the combination of automated planning and reinforcement learning is that policies learned through reinforcement learning could be used as control knowledge of a planner.

In this work we use an automated planning system for policy learning and transfer. We can use any automated planning system that can easily solve simple problems optimally. However, most of the domain-independent planning systems have problems scaling-up, given that the complexity of planning problems can be PSPace complete (Bylander 1994). From the plans obtained by a planner in different problems (with different goals) we can easily extract sets of state-action pairs. Since we use relational representations, the goal can be added and we can generate a set of state-goal-action tuples. These tuples represent optimal decisions to those simple problems (if the planner generated optimal plans).

In this paper we show that such set of tuples can represent a general policy that can be transferred to solve new potentially more complex problems: given a new problem with a specific goal, g, in any given state, s, we can look for the closest state-goal-action tuple stored, (s, g, a). The action returned, a, will be the action that the planner should execute next to achieve g from s. As we said before, g, s and a are represented in first order logic. This has two implications. The first one is that the transfer to different problems in the same domain can be done without any mapping, since

the language is the same independently of the problem or the complexity of such problem. For instance, in the Keepaway task(Taylor, Stone, and Liu 2007), when we transfer from the 3 vs 2 scenario to the 5 vs 4, the state space size increases with classical RL, because we need to add new attributes with information of the new players. Therefore, a mapping from the source state space to the target one is required. However, with relational RL we can use a relational language which is independent of the number of players, as used in some approaches(L. Torrey and Maclin 2007).

The second implication of the use of a relational representation is that a relational distance must be used. There are several in the literature, and we use a variation of the Relational Instance Based Learning distance (Kirsten, Wrobel, and Horváth 2001).

One problem with this approach is that the size of the set of state-goal-action tuples can increase as we solve more problems, suffering from the utility problem (Minton 1988): the time to retrieve the right policy can increase up to the level in which the time to make the decision based on the stored policy is bigger than the one needed to search for a good alternative. To solve this problem, we use a reduction algorithm to select the most representative set of tuples. The algorithm, called RNPC (Relational Nearest Prototype Classification), selects the most relevant instances of the training set. The selected prototypes, together with the distance metric, compose the policy.

We show empirically the success of this approach. We use a classical planning domain, *Zenotravel*, and an automated planning system, Sayphi (De la Rosa, García-Olaya, and Borrajo 2007). The transferred policy solves the same problems than Sayphi in less time. Additionally, the policy scales up and solves complex problems that Sayphi can not solve. However, as is the case of most current task planners, the learning process does not guarantee completeness nor optimality.

The paper is organized as follows. The next section describes the relational language used to represent states, actions, plans and policies, and introduces the distance metric. Section 3 describes the learning process, how the policy is generated with the RNPC algorithm, and how the policy can be efficiently used. Section 4 shows the experiments, and Section 5 summarizes the main conclusions and future work.

A planning policy

Planning is a problem solving task that consists on, given a domain model (set of actions) and a problem (initial state and set of goals), obtaining a plan (set of instantiated actions). That plan, when executed, transforms the initial state into a state where all goals are achieved. More formally, a planning problem is usually represented as a tuple $< S, A, I, G >$, such that S is a set of states, A is a set of action models, $I \subseteq S$ is the initial state, and $G \subseteq S$ is a set of goals. In general, planning is PSPACE complete, so learning can potentially help on obtaining plans faster, plans with better quality, or even generating domain models (action descriptions) (Zimmerman and Kambhampati 2003).

Planning domains are represented in the Planning Domain Description Language (PDDL), which has become a standard for the representation of planning domain models. We use the original STRIPS formulation in this work, where actions are represented in a form of predicate logic with their preconditions and effects. The effects of an action consist of deleting or adding predicates to transform the current search state into a new one. Most current planners instantiate the domain before planning by generating the set of all possible ground literals and actions from the problem definition. Suppose that we are solving problems in the Zenotravel domain. The Zenotravel domain, introduced in the third IPC, consists of flying people among different cities by aircrafts, which need different levels of fuel to fly. The domain actions are: *board* a person into an aicraft in a city; *debark* a person from an aircraft in a city; *fly* an aircraft from a city to other city using one level of fuel; *zoom* fly using two levels of fuel; and *refuel* the aircraft in a city. So, if the definition of states includes a predicate `at(?x - person ?y - city)` to represent the fact that a given person is in a given city, the planners will generate a ground literal for each person-city combination (e.g. a six persons, three cities problem will generate 18 ground literals). And they also perform that instantiation with the actions. Therefore, even if the input language is relational, they search in the space of propositional representations of states, goals and actions.

A plan is an ordered set of instantiated actions, $< a_0, \ldots, a_{n-1} >$ such that, when executed, all goals are achieved. The execution of a plan generates state transitions that can be seen as tuples $< m_i, a_i >$. $a \in A$ is an instantiated action of the plan. And, $m_i \in M$ are usually called meta-states given that they contain relevant things about the search that allow making informed decisions (Veloso et al. 1995; Fernández, Aler, and Borrajo 2007). In our case, each m_i is composed of the state $s_i \in S$ and the pending goals $g_i \in S$. So, M is the set of all possible pairs (s, g). Other authors have included other features in the representation of meta-states as previously executed actions (Minton 1988), alternative pending goals in the case of backward search planners (Borrajo and Veloso 1997), hierarchical levels in the case of hybrid POP-hierarhical planners (Fernández, Aler, and Borrajo 2005), or the deletes of the relaxed plan graph (Yoon, Fern, and Givan 2006). In the future, we would like to include some of these alternative features in the meta-states to understand the implications of the representation language of meta-states.

Relational Instance-Based Policies

A *Relational Instance-Based Policy* (RIBP), π, is defined by a tuple $< L, P, d >$. P is a set of tuples, $t_1, \ldots t_n$ where each tuple t_i is defined by $< m, a >$, where $m \in M$ is a meta-state, and $a \in A$. Each t_i can be considered as an individual suggestion on how to make a decision, i.e. the action a that should be executed when the planner is in state s and tries to achieve the goals g. L defines the language used to describe the state and the action spaces. We assume that the state and action spaces are defined using PDDL. And, d is a distance metric that can compute the relational distance between two different meta-states. Thus, a *Relational Instance Based Policy*, $\pi : M \rightarrow A$ is a mapping from a meta-state to an action.

This definition of policy differs from the classical reinforcement learning definition, since the goal is also an input to the policy. Therefore, a Relational Instance Based Policy can be considered an universal policy for the domain, since it returns an action to execute for any state and any goal of the domain. Given a meta-state, m, the policy returns the action to execute following an instance-based approach, by computing the closest tuple in P and returning its associated action. To compute the closest tuple, the distance metric d is used as defined in equation 1.

$$\pi(m) = \arg_a \min_{(<m',a'>\in P)} (dist(m, m')) \quad (1)$$

Next subsection describes the distance metric used in this work, although different distance metrics could be defined for different domains. In this work, the distance metric is based on previously defined metrics for Relational Instance Based Learning approaches, the RIBL distance (Kirsten, Wrobel, and Horváth 2001).

The RIBL distance

To compute the distance between two meta-states, we follow a simplification of the RIBL distance metric, which has been adapted to our approach. Let us assume that we want to compute the distance between two meta-states, m_1 and m_2. Also, let us assume that there are K predicates in a given domain, $p_1, \ldots, p_K$. Then, the distance between the meta-states is a function of the distance between the same predicates in both meta-states, as defined in equation 2.

$$d(m_1, m_2) = \sqrt{\frac{\sum_{k=1}^{K} w_k d_k(m_1, m_2)^2}{\sum_{k=1}^{K} w_k}} \quad (2)$$

Equation 2 includes a weight factor, w_i for $i = 1, \ldots, K$, for each predicate. These weights modify the contribution of each predicate to the distance metric. And $d_k(m_1, m_2)$ computes the distance contributed by predicate p_k to the distance metric. In the Zenotravel domain, there are 5 different predicates that define the regular predicates of the domain, plus the ones referring to the goal ($K = 5$): *at, in, fuel_level, next, goal_at*. There is only one goal predicate, *goal_at*, since the goal in this domain is always defined in terms of the predicate *at*.

In each state there may exist different instantiations of the same predicate. For instance, two literals of predicate *at*: *(at p0, c0)* and *(at pl0, c0)*. Then, when computing $d_k(m_1, m_2)$ we are, in fact, computing the distance between two sets of literals. Equation 3 shows how to compute such distance.

$$d_k(m_1, m_2) = \frac{1}{N} \sum_{i=1}^{N} \min_{p \in P_k(m_2)} d'_k(P_k^i(m_1), p) \quad (3)$$

where $P_k(m_i)$ is the set of literals of predicate p_k in m_i, N is the size of the set $P_k(m_1)$, $P_k^i(m_i)$ returns the ith literal from the set $P_k(m_i)$, and $d'_k(p_k^1, p_k^2)$ is the distance between two literals, p_k^1 and p_k^2 of predicate p_k. Basically, this equation computes, for each literal p in $P_k(m_1)$, the minimal distance to every literal of predicate p_k in m_2. Then, the distance returns the average of all those distances. Finally, we only need to define the function $d'_k(p_k^1, p_k^2)$. Let us assume the predicate p_k has M arguments. Then,

$$d'_k(p_k^1, p_k^2) = \sqrt{\frac{1}{M} \sum_{l=1}^{M} \delta(p_k^1(l), p_k^2(l))} \quad (4)$$

where $p_k^i(l)$ is the lth argument of literal p_k^i, and $\delta(p_k^1(l), p_k^2(l))$ returns 0 if both values are the same, and 1 if they are different.

Given these definitions, the distance between two instances depends on the similarity between the names of both sets of objects. For instance, the distance between two meta-states that are exactly the same but with different object names is judged as maximal distance. To partially avoid this problem the object names of every meta-state are renamed. Each object is renamed by its type name and an appearance index. The first renamed objects are the ones that appear as parameters of the action, followed by the objects that appear in the goals. Finally, we rename the objects appearing in literals of the state. Thus, we try to keep some kind of relevance level of the objects to find a better similarity between two instances.

The learning process

The complete learning process can be seen in Figure 1. We describe it in three steps:

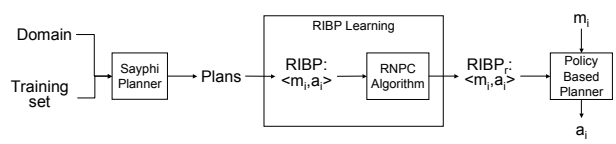

Figure 1: Scheme of the learning process.

1. Training: we provide a planner a set of simple and random training problems to be solved.
2. Relational Instance Based Policy Learning (RIBPL): from each resulting plan, $\{a_0, a_1, \ldots, a_{n-1}\}$, we extract a set of tuples $< m_i, a_i >$. All these tuples from all solution plans compose a policy (RIBP). However we must reduce the number of tuples $< m_i, a_i >$ of the policy to obtain a reduced one (RIBP$_r$). The higher the number of tuples, the higher the time needed to reuse the policy. If this time is too high, it would be better to use the planner search instead of the learned policy. To reduce the number of tuples, we use the Relational Nearest Prototype Classification algorithm (RNPC) (García-Durán, Fernández, and Borrajo 2006), which is a relational version of the original algorithm ENPC (Fernández and Isasi 2004). There are two main differences with that work: RNPC uses a relational representation; and the prototypes are extracted by selection as in (Kuncheva and Bezdek 1998). The goal is to obtain a reduced set of prototypes P that generalizes the data set, such that it can predict the class of a new instance faster than using the complete data set and with

an equivalent accuracy. The RNPC algorithm is independent of the distance measure and different distance metrics could be defined for different domains. In this work we have experimented with the RIBL distance described in the previous section.

3. Test: we test the obtained RIBP$_r$ using a new set of target problems. These problems are randomly generated, with similar and higher difficulty as those used in the training step. For each current meta-state ($m_i = (s_i, g_i)$, where s_i is the current state of the search and g_i is the set of goals that are not true in s_i), and all the applicable actions in s_i, we compute the nearest prototype $p = (m, a)$ from the learned set P. Then, we execute the action a of p, updating the state s_{i+1} (state after applying a in s_i) and g_i (remove those goals that became true in s_{i+1} and add those that were made false by the application of a). Then we perform these steps until we find a solution to the problem ($g \subseteq S$), or a given time bound is reached.

Experiments and results

We have used the SAYPHI planner (De la Rosa, García-Olaya, and Borrajo 2007) which is a reimplementation of the METRIC-FF planner (Hoffmann 2003), one of the most efficient planners currently. Since SAYPHI implements many search techniques, we used the EHC (Enforced Hill Climbing) algorithm. This algorithm is a greedy local search algorithm that iteratively evaluates each successor of the current node with the relaxed plan graph heuristic until it finds that one is better than the current node and search continues from that successor. If no successor improves the heuristic value of the current node a breadth-first search is performed locally in that node to find a node in the subtree that improves the current node.

The chosen domain is Zenotravel, widely used in the AI planning field, and the RIBL distance. The source problems are 250 randomly generated problems: the first 50 with one person and one goal, one plane and three cities; the next 100 problems with two persons and goals, two planes and three cities; and the last 100 problems with three persons and goals, two planes and three cities. All of them with seven levels of fuel. The planner has been executed using a branch-and-bound algorithm to select the optimal solution, giving 120 seconds as time bound. We extracted a total of 1509 training instances from the solution plans. After applying the RNPC algorithm 10 times, since it is a stochastic algorithm, we reduced the instances to an average of 18.0 prototypes, which form the tuples $< m, a >$ of the learned RIBP$_r$.

We have used two target or test problem sets. The first test set contains 180 random problems of different complexity. It is composed of nine subsets of 20 problems: (1,3,3), (1,3,5), (2,5,10), (4,7,15), (5,10,20), (7,12,25), (9,15,30), (10,17,35) and (12,20,40), where (pl,c,p) refers to number of planes (pl), numbers of cities (c) and numbers of persons and goals (p). All these problems have seven levels of fuel. The time bound given to solve them is 180 seconds.

The second test set is composed of the 20 problems from the third IPC. They have different complexity: from one plane, two persons, three cities and three goals; to five planes, 25 persons, 22 cities and 25 goals. All of them have seven levels of fuel. For this set we have let 1800 seconds as time bound.

#Problems (#goals)	Approach	Solved	Time	Cost	Nodes
20 (3)	Sayphi	20	0.46	166	683
	RIBP$_r$	20	1.47	275.5	296.1
20 (5)	Sayphi	20	0.49	236	868
	RIBP$_r$	20	1.4	291.3	314.2
20 (10)	Sayphi	20	5.84	535	3277
	RIBP$_r$	20	8.11	719.5	743.8
20 (15)	Sayphi	20	82.54	749	10491
	RIBP$_r$	20	40.69	1285.3	1306.9
20 (20)	Sayphi	20	607.32	1293	21413
	RIBP$_r$	20	133.25	2266.6	2286.9
20 (25)	Sayphi	13	629.06	961	26771
	RIBP$_r$	20	112.41	1880.6	1896.5
20 (30)	Sayphi	8	221.92	712	9787
	RIBP$_r$	20	72.22	1340	1353
20 (35)	Sayphi	0	–	–	–
	RIBP$_r$	15			
20 (40)	Sayphi	1	30.2	120	1420
	RIBP$_r$	6.2	25.82	298.6	301.3

Table 1: Results in the first target problem set in Zenotravel domain using the Sayphi planner and the learned RIBP$_r$.

The results of solving the first set of target problems using the planner (*Sayphi*) and the learned RIBP$_r$ are shown in Table 1. We show four different variables: the number of solved problems of the testing set (*Solved*), the acumulated time used to solved them (*Time*), the accumulated cost measured in number of actions in the solution plan (*Cost*) and the accumulated number of evaluated nodes by the planner in the search of the solution (*Nodes*). The last three columns are computed only for the common solved problems of the two approaches. In the case of using the learned RIBP$_r$ we show the average of the 10 different runs of the RNPC algorithm.

Analyzing these results, we can observe that not all the problems are solved in each case. In the case of the most simple problems both approaches solve all problems. However, when the complexity increases, the number of solved problems decreases, specially in the case of the Sayphi planner. In all the cases, the obtained cost for the common solved problems is better in the case of the planner, although the RIBP$_r$ solves them in less time, except in the two easier cases. The number of evaluated nodes is always lower in the case of the learned policy.

	solved	time	cost	nodes
Sayphi	18	2578.84	512	30023
RIBP	20	2761.69	1081	1102
RIBP$_r$	20	111.62	1248.0	1267.8

Table 2: Results in the IPC test set in Zenotravel domain using the Sayphi planner, the RIBP and the learned RIBP$_r$.

In Table 2 we can see the results of solving the IPC set of test problems using the planner, the resulting *RIBP* without the reduction, i. e. the 1509 tuples from the training step, and the learned RIBP$_r$. This Table follows the same format as Table 1. In this test set of problems the planner only solves 18 problems giving 1800 seconds as time bound. However both policies solve all the problems. Although the

accumulated cost for the 18 problems is greater in the case of using the learned RIBP$_r$, the accumulated time and the accumulated evaluated nodes needed to solve them is one order of magnitude less than using the planner. The cost of the RIBP is better than the one obtained by the RIBP$_r$, while the number of evaluated nodes is in the same order of magnitude as the RIBP$_r$. This is reasonable given that it covers many more meta-states when making the decisions, so the decisions will be usually more correct than using the compact representation obtained by the RIBPL.

In the case of time to solve, if the policy is not compact, such as the RIBP, it can even be higher than the one employed by the planner. That is because of the high number of required comparisons to the training tuples. However, if the policy is compact, as the RIBP$_r$, decision time should be less that deciding with the planner, given that the planner has to compute the heuristic for some nodes until it finds a node that is better than its parent (EHC algorithm). Using the learned policy is even greedier than EHC (which in itself is one of the most greedy search algorithms), since it does not evaluate the nodes, and instead relies on the source planning episodes. So, one would expect that it would lead the planner to not solve the problem due to a very long path, by applying actions that do not focus towards the goal. However, this is not the case, and we were even able to solve more or equal number of problems than EHC. Another kind of potential problem when using policies, usually less harmful in the case of EHC given the use of a lookahead heuristic (the relaxed plan graph computation), are dead-ends: arriving to states where no action can be applied. Zenotravel does not have dead-ends, given that planes can always be refueled, and actions effects can be reversed. So, in the future we would like to explore the use of the RIBPL in domains where dead-ends can be found.

Conclusions and future work

In this paper we have described how a policy learned from plans that solve simple problems can be transfered to solve much more complex problems. The main contributions are i) the use of plans to generate state-goal-action tuples; ii) the use of the nearest neighbour approach to represent the policies; iii) the use of the RNPC algorithm to reduce the number of tuples, i.e. reduce the size of the policy; and iv) the direct transfer of the learned policy to more complex target problems.

We show empirically the success of this approach. We apply the approach to a classical planning domain, *Zenotravel*. In this domain, the automated planning system used, Sayphi, solves problems of up to 15 goals. We use the same planner to generate optimal plans for problems of less than 5 goals. From these plans, more than 1500 state-goal-action tuples are generated. Then, the RNPC algorithm is used to select the most relevant prototypes (less than 20) which are used as the transferred policy. This reduced RIBP solves the same problems than Sayphi in less time. Additionally, the policy solves problems that Sayphi can not solve. However, the learning process does not guarantee completeness nor optimality, although no planning system can guarantee completeness nor optimality in a given time bound.

The approach has been tested in a deterministic domain. The same ideas can be extended in the future to probabilistic domains where automated planning systems have more problems to scale-up, and where policies show their main advantage over plans.

Acknowledgements

This work has been partially supported by the Spanish MEC project TIN2005-08945-C06-05, a grant from the Spanish MEC, and regional CAM-UC3M project CCG06-UC3M/TIC-0831. We would like to thank Tomás de la Rosa and Sergio Jiménez for their great help.

References

Borrajo, D., and Veloso, M. 1997. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning* 11(1-5):371–405. Also in the book "Lazy Learning", David Aha (ed.), Kluwer Academic Publishers, May 1997, ISBN 0-7923-4584-3.

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

De la Rosa, T.; García-Olaya, A.; and Borrajo, D. 2007. Using cases utility for heuristic planning improvement. In Weber, R., and Richter, M., eds., *Case-Based Reasoning Research and Development: Proceedings of the 7th International Conference on Case-Based Reasoning*, volume 4626 of *Lecture Notes on Artificial Intelligence*, 137–148. Belfast, Northern Ireland, UK: Springer Verlag.

Driessens, K., and Ramon, J. 2003. Relational instance based regression for relational reinforcement learning. In *Proceedings of the 20th International Conference on Machine Learning*.

Dzeroski, S.; Raedt, L. D.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning* 43.

Fern, A.; Yoon, S.; and Givan, R. 2004. Approximate policy iteration with a policy language bias. In Thrun, S.; Saul, L.; and Schölkopf, B., eds., *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press.

Fernández, F., and Isasi, P. 2004. Evolutionary design of nearest prototype classifiers. *Journal of Heuristics* 10(4):431–454.

Fernández, S.; Aler, R.; and Borrajo, D. 2005. Machine learning in hybrid hierarchical and partial-order planners for manufacturing domains. *Applied Artificial Intelligence* 19(8):783–809.

Fernández, S.; Aler, R.; and Borrajo, D. 2007. Transferring learned control-knowledge between planners. In Veloso, M., ed., *Proceedings of IJCAI'07*. Hyderabad (India): IJCAI Press. Poster.

García-Durán, R.; Fernández, F.; and Borrajo, D. 2006. Nearest prototype classification for relational learning. In *Conference on Inductive Logic Programming*, 89–91.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Task Planning. Theory & Practice*. Morgan Kaufmann.

Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *International Journal of Artificial Intelligence Research* 4:237–285.

Kirsten, M.; Wrobel, S.; and Horváth, T. 2001. *Relational Data Mining*. Springer. chapter Distance Based Approaches to Relational Learning and Clustering, 213–232.

Kuncheva, L., and Bezdek, J. 1998. Nearest prototype classfication: Clustering, genetic algorithms, or random search? *IEEE Transactions on Systems, Man, and Cybernetics*.

L. Torrey, J. Shavlik, T. W., and Maclin, R. 2007. Relational macros for transfer in reinforcement learning. In *Proceedings of the 17th Conference on Inductive Logic Programming*.

Minton, S. 1988. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Boston, MA: Kluwer Academic Publishers.

Taylor, M. E.; Stone, P.; and Liu, Y. 2007. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

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

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine* 24(2):73–96.