An Enhancement of Relational Reinforcement Learning

Renato R. da Silva, Claudio A. Policastro, and Roseli A.F. Romero

***Abstract*— Relational reinforcement learning is a technique that combines reinforcement learning with relational learning or inductive logic programming. This technique offers greater expressive power than that one offered by traditional reinforcement learning. However, there are some problems when one wish to use it in a real time system. Most of recent research interests on incremental relational learning structure, that is a great challenge in this area. In this work, we are proposing an enhancement of TG algorithm and we illustrate the approach with a preliminary experiment. The algorithm was evaluated on a Blocks World simulator and the obtained results shown it is able to produce appropriate learn capability.**

I. Introduction

Reinforcement learning(RL) addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals [1]. The RL offers a general structure and several methods to make it improve its behavior while it actions in the environment. However, the conventional representation, propositional or attribute-value (AV), used by RL methods on the real enviroment with large search space are not appropriate, once the learning algorithm can take a large time to converge [2].

Relational learning is the problem of learning structured concept definitions from structured examples. Relational representations use a first-order model to describe the problem domain and examples are given to the learner in terms of objects and object relations rather than by simple propositions [3].

In recent times, relational learning is studied under different names, including inductive logic programming, relational data mining and probabilistic relational modeling. Reinforcement learning is also studied under multiple guises of which neuro-dynamic programming and decision theoretic planning are the most recognizable [4].

Relational Reinforcement Learning (RRL) [5] combines RL and relational learning or Inductive Logic Programming [6]. It combines a standard RL algorithm (Q-learning) with a relational regression algorithm, as TILDE-RT [7]. TILDE-RT can be seen as a first−order extension of the C4.5 decision tree algorithm [1]. TILDE-RT is used as a representational tool to store the $Q-function$ in a first−order logic decision tree, which is called a Q−tree. RRL collects experience in the form of state-action pairs with corresponding Q−values. During an episode, actions are taken according to the current policy, according to the current Q-tree. All newly encountered state−action pairs are stored, while the values of already encountered pairs are updated according to the Q-learning algorithm [2].

Although RRL is an effective tool for relational RL, it suffers from a number of problems. First, after each episode, a new Q−tree is induced from the examples, which is clearly not efficient. Second, the set of examples is constantly growing; all state-action pairs have to be memorized. Third, updating values for already experienced state−action pairs requires searching through the whole example set. Fourth, generalizing is not done in an efficient way. When updating a value for one state−action pair, the values of all pairs in that leaf should be updated, but in standard RRL only those pairs are updated that are experienced exactly again [2]. Because this problems another different regression algorithms are proposed.

Another three different regression algorithms have been used in this context so far: an incremental regression tree algorithm called *TG* [8], a relational instance based algorithm *RIB* [9] and a regression algorithm based on Gaussian Processes and kernels for structural domains called *KBR* [10].

Fetching use all the power of the approach RRL in real applications are necessary structures and methods that allow the use incremental process during the episode of learning, so, improving its efficiency and reducing resource.

These improvements are important to use in a robotic architecture in the real enviroment where it will be considered the ability of the shared attention [11].

This article reports an ongoing work aimed at developing an enhanced of RRL TG algorithm. For this, a new algorithm is proposed and discussed.

This article is organized in the following way. In Section II, we present the Knowledge Representation problem. In Section III, we briefly introduce some concepts of RRL. In Section IV, we present the proposed algorithm that is an enhanced of TG algoritm. In Section V, the main results from a set of experiments carried out to evaluate the proposed algorithm are discussed. Finally, in Section VI, the conclusions and future works are presented.

II. Knowledge Representation

Knowledge representation is a key element in artificial intelligence. However, much of the research in machine learning [1] [12] has been concerned with propositional representations only, and is very much concerned with statistical approaches. An exception is formed by inductive logic programming (ILP) [13][6] that is mainly concerned with learning logical descriptions, but largely did not consider probabilistic aspects of learning. The recent direction of

Renato R. da Silva is with the Department of Computer Science, University of Sao Paulo, Sao Carlos, Brazil; email: ramos@icmc.usp.br. Claudio A. Policastro is with the Department of Computer Science, University of Sao Paulo, Sao Carlos, Brazil; email: capoli@icmc.usp.br. Roseli A.F. Romero is with the Department of Computer Science, University of Sao Paulo, Sao Carlos, Brazil; email: rafrance@icmc.usp.br.

978-1-4244-1821-3/08/$25.00©2008 IEEE

probabilistic logic learning (PLL) [14] [15] tries to reconcile statistical and logical approaches to deal with large, relational and uncertain domains. RRL has to be seen as fitting in this direction, and extending it with a utility-based framework. Reinforcement learning in relational worlds involves dealing with logical abstractions, with uncertainty, and with utilities [2].

Gordon Plotkin showed early on how the inductive process of generalization can be formalized within predicate or first-order logic [16] and [17]. Logic-based learning produces learning results in a (restricted) predicate logic. Predicate logic allows to formulate relations in an elegant way which user (human experts from others faculties than computer science) can easily understand. Moreover, the learning result can be interpreted by the system [18].

Because of the similarities in ideas by ILP, PLL and RRL, most of the reasons for moving to relational representations apply to all fields equally. In many cases (e.g. for finite domains), intrinsically relational domains can be modeled in terms of a propositional representation. For example, the game of chess can in principle be represented in propositional form by using propositions such as **whiteKingOnSameLineAsBlackKing** and **NumberOfBlackPawnsIsNotFive**. However, the game is more naturally represented in a relational form, using for example **on(blackKing, A4)**, **on(whiteKing, A7)**, and **sameLine(blackKing, whiteKing)** [2]. This example show us the relational form uses predicate.

Furthermore propositionalization comes with a number of drawbacks. First, the number of objects should be fixed, and propositions for all relations between all objects should be constructed. This generates a huge number of propositions. Second, no generalization is possible over objects or relations. Third, an order has to be imposed on the objects and relations [2]. More details on the relations between propositional and relational representations can be found in [19] and [20]

There are a number of additional reasons for modelling RL problems using relational formalisms. Lookup tables or propositional representations are not able to represent the structural aspects of states and actions in relational domains such as Blocks World and chess. Relational representations also allow for a general and intuitive way of specifying and using knowledge. Many standard agent architectures and theories use subsets or extensions of first−order logic (e.g. such as modal logic) [21]. And as some argue [22], a representation should enable representing and reasoning about objects. We argue that this is even more evident when relating to existing agent architectures and programming languages. One has to be able to represent objects and relations in our language if an intentional stance [23] is taken, identifying cognitive notions like beliefs, desires, intentions and emotions [2].

III. Relational Reinforcement Learning

The RRL is concerned with reinforcement learning in domains that exhibit structural properties and in which different kinds of related objects exist. These kind of domains are usually characterized by a very large and possibly unbounded number of different possible states and actions. In this kind of environment, most traditional reinforcement learning techniques break down.

One reason of traditional RL fail is that it stores the learned *Q-values* in an explicit state action table, with one value for each possible combination of the state and action. The RRL method introduced by [24] and [5] modifies this aspect of reinforcement learning in order to introduce variables and to increase the ability of the agent to generalize learned knowledge to unseen parts of the state space. Rather than using an explicit state−action table, RRL stores the *Q-values* in a logical regression tree [25].

The RRL task is very similar to the traditional reinforcement learning task except that a relational representation is used for the states and actions [26].

The RRL task can be defined as follows [26],
Given:

- A set of possible states S (represented in a relational format),
- a set of possible actions A (represented in a relational format),
- an unknown transition function δ: S x $A \rightarrow S$ (this function can be nondeterministic)
- a real-valued reward function r: S x $A \rightarrow \Re$.
- Background knowledge.

Find a policy for selecting actions π^*: $S \rightarrow A$ that maximizes a value function $V^\pi(s_t)$ for all $s_t \in S$.

The optimal policy π^* will always select the action that maximizes the sum of the immediate reward and the value of the immediate successor state, i.e.,

$$\pi^*(s) = argmax_a(r(s,a) + \gamma V^{\pi^*}(\delta(s,a))) \quad (1)$$

The Q-function for policy π is defined as follows :

$$Q^\pi(s,a) = r(s,a) + \gamma V^\pi(\delta(s,a)) \quad (2)$$

Knowing Q^* , the $Q - function$ for the optimal policy, allows us to rewrite the definition of π^* as follows

$$\pi^*(s) = argmax_a Q^*(s,a) \quad (3)$$

An approximation to the Q-function, $\hat{Q}$, in the form of a look-up table, is learned by the following way. The agent learns through continuous interaction with the environment, during which it exploits what it has learned so far, but also explores. In practice, this means that the current approximation Q is used to select an action most of the time. However, in a small fraction of cases an action is selected randomly from the available choices, so as to explore and evaluate unseen (state,action) pairs [27].

In the RRL, however, more specific rules govern the learning process. States are represented as sets of first-order logical facts, and the algorithm can only see one state at a time. Actions are also represented relationally as predicates

describing the action as a relationship between one or more variables. Because not all actions are possible in all states, the algorithm can only consider actions that are possible given the current state, and it must be able to determine from the state description whether a given action is possible. Also, because of the relational representation of states and actions and the inductive logic programming component of the algorithm, there must exist some body of background knowledge which is generally true for the entire domain, as well as a language bias determining how the learned policy will be represented [25].

Perhaps the most important modification, however, is the method of storage of the learned *Q-values*. In RRL, sets of state−action pairs and their corresponding *Q-values* are used to induce a logical regression tree after each training session. This tree then provides the *Q-values* for state-action pairs for the following training session [25].

Figure 1 gives an example of a first order regression tree. It is a binary tree in which every internal node contains a test which is a conjunction of first−order literals and every leaf (terminal node) of the tree contains a real valued prediction [26].

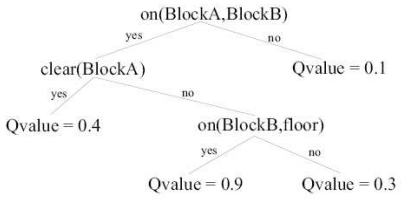

Fig. 1. A relational regression tree.

In the next section, we present one of the main algorithms founding in RRL literature.

A. The RRL system

The RRL system is a union of stantard relational reinforcement learning and relational regression tree algorithms. Now, we will describe this two methods.

After to initialize the *Q-function*, the algorithm then starts running learning episodes like any standard *Q-learning* algorithm [28] [1] [29].

During the learning episode, all the encountered states and the selected actions are stored, together with the rewards connected to each encountered (state, action) pair. At the end of each episode, when the system encounters a terminal state, it uses reward back-propagation and the current *Q-function* approximation to compute the appropriate *Q-value* approximation for each encountered (state, action) pair.

The algorithm then presents the set of (state, action, qvalue) triplets to a relational regression engine, which will use this set of examples to update the current estimate *Q-function*, whereafter the algorithm continues executing the next learning episode.

The steps that compose the stantard RRL algorithm can be seen in Algorithm 1.

Algorithm 1 The Relational Reinforcement Learning Algorithm

initialize the Q-function hypothesis $\hat{Q}_0$
$e \leftarrow 0$
repeat
  $Examples \leftarrow \emptyset$
  generate a starting state s_0
  $i \leftarrow 0$
  repeat
    choose a_i for s_i using a policy derived from the current hypothesis $\hat{Q}_e$
    take action a_i, observe r_i and s_{i+1}
    $i \leftarrow i + 1$
  until s_i is terminal
  for $j = i - 1$ to 0 **do**
    generate example $x = (s_j, a_j, \hat{q_j})$ where $\hat{q_j} \leftarrow r_j + \gamma max_a \hat{Q}_e(s_{j+1}, a)$
    $Examples \leftarrow Examples \cup \{x\}$
  end for
  Update $\hat{Q}_e$ using Examples and a relational regression algorithm to produce $\hat{Q}_{e+1}$
  $e \leftarrow e + 1$
until no more episodes

This algorithm is essentially identical to a traditional *Q-learning* algorithm, except in its method of updating the *Q-function*, Q_e (represented as a regression tree after an episode e).

The TG algorithm, described in Algorithm 2, is the first of three relational regression algorithms that have been developed by Driessens [26] for use in the RRL system. It is an incremental first order regression tree algorithm, and also an extension of the G algorithm proposed by Chapman and Kaelbling [30].

Algorithm 2 TG Algorithm

initialize by creating a tree with a single leaf with empty statistics
for each learning example that becomes available **do**
  sort the example down the tree using the tests of the internal nodes until it reaches a leaf
  update the statistics in the leaf according to the new example
  if the statistics in the leaf indicate that a new split is needed **then**
    generate an internal node using the indicated test
    grow 2 new leafs with empty statistics
  end if
end for

The TG algorithm uses a relational representation language for describing the examples and the tests that can be used in the regression tree [26].

The main drawback in this current version of TG algorithm is the use of an intermediate structure to store the examples learned during a learning episode. Another possible drawback is the use of a binary tree that may limit its application on larger problem domains, since a binary tree support only two leaf nodes for each state configuration, that is, only two responses for each state.

IV. Proposed RRL algorithm

In this work, we are proposing an enhanced of the current version of TG algorithm by incorporating a hybrid regression tree, with may store several responses for each state representation. Additionally, our enhancement uses the regression tree during the learning episode, saving the use of auxiliar structures and speeding up the knowledge learning and generalization.

Our algorithm starts by using the same initializing the Q-function of TG algorithm, and creates an empty regression tree.

For each episode, randomly choose an initial and final state. The agent observed the actual state, choose and take an action. This process changes the state and the agent receives its reward. The reward can be either positive (equals to 1) or negative (equals to 0). Another point is that the Q−Value is free, we do not use a maximum value. After this occurred, presents the set of (state, action, qvalue) triplets to relational regression engine. Do this until the final state is equal to the initial state. All process can be found in on Algorithm 3.

Algorithm 3 The Proposed RRL Algorithm

initialize the *Q-function* hypothesis $\hat{Q}_0$ and create a tree with a single leaf
$e \leftarrow 0$
repeat
  generate a starting state s_0
  generate a target state s_t
  $i \leftarrow 0$
  repeat
    choose a_i for s_i using a policy derived from the current hypothesis $\hat{Q}_e$
    take action a_i, observe r_i and s_{i+1}
    Update $\hat{Q}_e$ using $x = (s_j, a_j, \hat{q_j})$ where $\hat{q_j} \leftarrow r_j + \gamma max_a \hat{Q}_e(s_{j+1}, a)$ and a relational regression algorithm to produce $\hat{Q}_{e+1}$
    $i \leftarrow i + 1$
  until s_i is the same of s_t
  $e \leftarrow e + 1$
until no more episodes

The relational regression engine receives a set of (state, action, $qvalue$) and tests the internal nodes if the state already exists. Case this performance is false, the state is inserted in the tree and the leaf receives the action with $qvalue$, forming a new branch. Otherwise, update the $qvalue$ for respective action in leaf.

In a leaf node, more than one action can occur. For an easy access to the best action, those actions are ordered according to their $qvalue$ when the example is inserted or updated.

In Algorithm 4, the relational regression algorithm proposed by us is presented. It is an adaptation of Algorithm 2 used in Algorithm 3.

Algorithm 4 Proposed Three Relational Regression Algorithm

repeat
  sort the example down the tree using the tests of the internal nodes until it reaches a leaf or null
  if reaches a leaf **then**
    update the *Q-value* for the action in the leaf according to the example
  else if reaches null **then**
    generate an internal node
  end if
until the example in a brainch
if necessary **then**
  order actions
end if

V. Experiments

In order to evaluated the proposed algorithm, we carried out a set of experiments employing a known blocks world simulator [31] [32] [33].

The blocks world simulator, as used in this work, consists of a constant number of blocks and a floor large enough to hold all the blocks. Blocks can be on the floor or can be stacked on one another. Only states with neatly stacked blocks are considered, i.e. it is not possible for a block to be on top of two different blocks.

The actions in the blocks world consist of moving a clear block, i.e. a block that has no other block on top of it, onto another clear block or the floor.

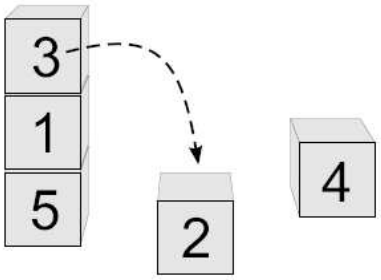

Fig. 2. An example state of a blocks world with 5 blocks. The action is indicated by the dotted arrow.

Figure 2 shows a possible blocks world state and action pair. The shown state is in a blocks world with 5 blocks. The blocks are labeled with a number to establish their identity. In this figure, the action of moving block 3 onto block 2 is represented by the dotted arrow [26].

A. Experimental setup

We conducted the experiments with three different number of blocks, we have used 3, 4 and 5-blocks world. In the each number block set we performed 50 total runs of 5000 episodes sessions. In each episode we configure a new initial and final state.

The current state was described using the following primitive predicates:

- clear(A): true if block A has no other blocks on top of it
- on(A,B): true if block A is directly on top of block B. B may be a block or the table.

We can represent the blocks world of Figure 2 by using this predicates as shown the Figure 3.

on(1,5).	clear(2).
on(2,floor).	clear(3).
on(3,1).	clear(4).
on(4,floor).	
on(5,floor).	move(3,2).

Fig. 3. The blocks world state represented as a relational representation.

In this work, we used three kind of specific goals. The goal task was random selected with three different options. It can put all blocks on floor or build a specific tower (e.g. Put block 3 on top of block 1, put block 1 on top of block 2 and it on floor) or else put on top one block on another that it on the floor (e.g. Put block 3 on top of block 1 and it on floor, and all others blocks must be on the floor).

One point that should be noted is that when a goal is chosen, it will never be equal to the initial state, due to a way that is generated in the proposed algorithm.

B. Experimental results

Before starting any experiment, is very important the choice of a metric that is made for a good assessment of what is proposed.

The calculation of the average of reinforcement obtained in each episode of the 50 total runs is adopted as criteria for the evaluation of the proposed algorithm.

The adoption of this metric enables us to see how much, on average, is needed to strengthen for each episode and thus trace the evolution of the amount of reinforcements over the turns.

In Figure 4, it is showed us the results of the tests performed and we can see the reinforcement curves are growing up to a threshold indicating the acquisition of knowledge by agent. When the curve reaches the threshold, the agent has the optimal reply for this problem.

Through the turns, the average values of reinforcement is increasing to a threshold, this is because the agent is optimizing the intermediate actions from an initial to the final state. This occurs because the only mode to get a positive value is when the agent finds a target and the maximum value

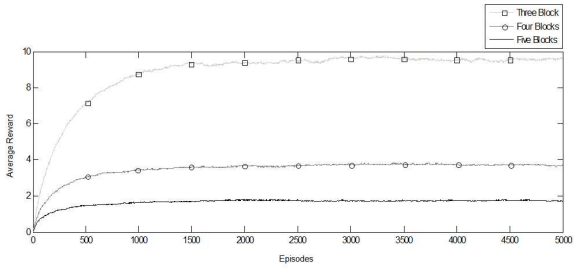

Fig. 4. Reinforcement curves of blocks world for differents numbers blocks.

is attained when the number of intermediary actions between the initial and final states is minimal.

When the minimum value of intermediate action is attained the curve stabilizes.

VI. Conclusions

In this paper, it was proposed an enhancement of TG algorithm for acquisition of knowledge. This work is the step for developing of efficient learning algorithms for applications in real world. The evaluation showed that the proposed algorithm is able to produce appropriate learning from blocks world simulator, and to generalize the learned knowledge. This algorithm could be helpful in different kinds of environments and applications.

As a next step we will compare this algorithm with conventional TG algorithm in the same environment to take the advantages e disadvantagens of both algorithms. This analyse will be done by size of problem, time of solve the question, average of number of step to find the target state and the evolution of reinforcement over the episodes.

We intend also to insert this algorithm into a learning process of a robotic head to yield incremental learning during the interaction with human beings and to improve the Hierarchical Relational Reinforcement Learning [25], comparing its perfomance with other algrotihms.

Acknowledgment

The authors would like to thank CNPq and FAPESP for support received.

References

[1] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.
[2] V. Otterlo, "A survey of reinforcement learning in relational domains," *CTIT Technical Report, TR-CTIT-05-31*, 2005.
[3] D. Roth and W. Yih, "Relational learning via propositional algorithms: An information extraction case study," in *IJCAI*, 2001, pp. 1257–1263.
[4] G. R. Tadepalli, P. and K. Driessens, "Relational reinforcement learning: An overview," in *Proceedings of the ICML'04 workshop on Relational Reinforcement Learning*, Banff, Canada, 2004.
[5] S. Džeroski, L. De Raedt, and K. Driessens, "Relational reinforcement learning," *Machine Learning*, vol. 43, no. 1/2, pp. 7−−52, 2001.
[6] S. Džeroski, *Relational Data Mining*, N. Lavrac, Ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.
[7] H. Blockeel, L. Raedt, and J. Ramon, "Top-down induction of clustering trees," in *15th International Conference on Machine Learning*. Morgan Kaufmann, 1998.

[8] K. Driessens, J. Ramon, and H. Blockeel, "Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner," in *EMCL '01: Proceedings of the 12th European Conference on Machine Learning*. London, UK: Springer-Verlag, 2001, pp. 97–108.
[9] K. Driessens and J. Ramon, "Relational instance based regression for relational reinforcement learning," in *In Proceedings of the Twentieth International Conference on Machine Learning*. AAAI Press, 2003, p. 123130.
[10] T. Gärtner, K. Driessens, and J. Ramon, "Graph kernels and Gaussian processes for relational reinforcement learning," in *ILP03*, ser. LNAI, T. Horváth and A. Yamamoto, Eds., vol. 2835. SV, 2003, pp. 146–163.
[11] R. Z. G. Policastro, C.A.; Romero, "Robotic architecture inspired on behavior analysis," *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pp. 1482−−1487, 12-17 Aug. 2007.
[12] G. F. Page, "Introduction to machine learning, by ethem alpaydin, mit press, 2004, xxx + 415 pp., with index. 160 ref. distributed chapter by chapter. isbn 0-262-01211-1." *Robotica*, vol. 24, no. 1, pp. 143–143, 2006.
[13] S. Muggleton and L. De Raedt, "Inductive logic programming: Theory and methods," *Journal of Logic Programming*, vol. 19/20, pp. 629–679, 1994.
[14] L. De Raedt and K. Kersting, "Probabilistic logic learning," *SIGKDD Explor. Newsl.*, vol. 5, no. 1, pp. 31–48, 2003.
[15] ——, "Probabilistic inductive logic programming," in *In Proceedings of the 15th International Conference on Algorithmic Learning Theory (ALT-2004)*. Padova, Italy: AAAI Press, 2004, p. 1936.
[16] G. Plotkin, "A note on inductive generalization," *Machine Intelligence*.
[17] ——, "A further note on inductive generalization," *Machine Intelligence*.
[18] K. Morik, B.-E. Kietz, W. Emde, and S. Wrobel, *Knowledge Acquisition and Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.
[19] L. De Raedt, "Logical settings for concept-learning," *Artif. Intell.*, vol. 95, no. 1, pp. 187–201, 1997.
[20] W. V. Laer and L. D. Raedt, "How to upgrade propositional learners to first order logic: a case study," pp. 102–126, 2001.
[21] W. Woolridge and M. J. Wooldridge, *Introduction to Multiagent Systems*. New York, NY, USA: John Wiley & Sons, Inc., 2001.
[22] L. Kaelbling, T. Oates, N. Hernandez, and S. Finney, "Learning in worlds with objects," in *In The AAAI Spring Symposium*, 2001.
[23] D. Dennet, "The intentional stance," *The MIT Press*, 1987.
[24] S. Džeroski, L. De Raedt, and H. Blockeel, "Relational reinforcement learning," in *International Workshop on Inductive Logic Programming*, 1998, pp. 11–22.
[25] M. Aycinena, "Hierarchical relational reinforcement learning," August 2002, unpublished. [Online]. Available: http://csail.mit.edu/~aycinena/curis2002/final_paper.pdf
[26] K. Driessens, "Relational reinforcement learning," Ph.D. dissertation, Department of Computer Science, K.U.Leuven, Leuven, Belgium., 2004.
[27] S. Džeroski, L. De Raedt, and K. Driessens, *Relational Reinforcement Learning for Agents in Worlds with Objects*, ch. 559.
[28] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.
[29] L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237−−285, 1996.
[30] D. Chapman and L. Kaelbling, "Input generalization in delayed reinforcement learning: An algorithm and performance comparisions," in *In Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1992, pp. 726−−731.
[31] N. Nilsson, *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.
[32] P. Langley, *Elements of Machine Learning*. Morgan Kaufmann, 1994.
[33] J. Slaney and S. Thiébaux, "Blocks world revisited," *Artif. Intell.*, vol. 125, no. 1-2, pp. 119–153, 2001.