Relational Reinforcement Learning: An Overview

Prasad Tadepalli TADEPALL@EECS.ORST.EDU
School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA

Robert Givan GIVAN@PURDUE.EDU
School of Electrical & Computer Engineering, Purdue University, West Lafayette, IN 47907, USA

Kurt Driessens DRIESSENS@EECS.ORST.EDU
Computer Science Departement, Catholic University of Leuven, B-3001 Leuven, Belgium

Abstract

Relational Reinforcement Learning (RRL) is both a young and an old field. In this paper, we trace the history of the field to related disciplines, outline some current work and promising new directions, and survey the research issues and opportunities that lie ahead.

1. Introduction

Integrating learning with problem solving has been a dream of Artificial Intelligence for a long time. While it might appear to run counter to the principle of divide-and-conquer, there are many good reasons for pursuing this integrated approach. Problem solving is often computationally hard, and without learning the agent will not be able to take advantage of its own experience to make future problem solving more efficient. On the other hand, learning in isolation depends on outside expertise to decide what is worth learning and provide training data. Such expertise is not always available, and when available, is often expensive. Thus, systems like LEX (Mitchell et al., 1983), SOAR (Laird et al., 1986) and Prodigy (Minton et al., 1989) pioneered ways of combining problem solving with learning, albeit with the knowledge of a complete and correct domain model, and in a fully observable and deterministic world.

Reinforcement Learning (RL), which is based on the theory of Markov Decision Processes, relaxes some of these assumptions including determinism and complete knowledge of the domain model. (Kaelbling et al., 1996; Bertsekas & Tsitsiklis, 1997; Sutton & Barto, 1998). Rather than being provided with a complete and correct domain theory, the agent is situated in an environment with which it can interact to gather domain knowledge by taking actions and receiving positive or negative reinforcement. The primary goal of the agent is to learn a method to choose its actions based on its current state, i.e., to learn a *policy* mapping states to actions, which optimizes a performance metric such as the expected average reward received per time-step.

Reinforcement Learning offers a general framework and a battery of methods for constructing intelligent agents that optimize their behavior in stochastic environments with minimal supervision. Unfortunately, however, most current work in RL is based on propositional representations, which makes it difficult to apply it to complex real-world tasks such as information retrieval from the Web, natural language processing, or planning in a rich real-world domain like cooking. Successful applications of propositional RL in such areas is typically possible only with expert human construction of the propositional features used. The states and actions in these domains are more naturally represented in relational form, and it appears that people routinely take advantage of rich relational structure in learning and generalization. The question naturally arises: *how is this done?*

In this overview paper, we motivate RRL as an attempt to answer this question, place it in the historical context, outline some of the promising approaches and new directions, and discuss some open research issues in the field.

Appearing in *Proceedings of the ICML'04 workshop on Relational Reinforcement Learning*, Banff, Canada, 2004. The first author acknowledges the support of NSF under grant number IIS-0329278.

2. Motivation

Consider a typical question answering task on the web (Berners-Lee et al., 2001; Fensel et al., 2002), which might involve accessing and integrating semi-structured information from the web to answer a complex query, e.g., "find a graduate school on the west coast that has affordable housing, multiple faculty and funded research in Artificial Intelligence." Even if the query is posed in a formal query language, answering it requires several skills such as query planning, optimization, information extraction, and information integration in a relational language. Or consider what is involved in learning to cook a meal. While certainly not an exhaustive list, one needs to reason about peoples' tastes and preferences, one's own knowledge of recipes and skills, availability of ingredients, their locations, and procedures to access them, the capacities of utensils and cooking ranges, and the effects of different proportions of ingredients, of cooking temperatures, and of different kinds of cooking processes on the taste and quality of the final product.

It is easy to pose both of these problems as reinforcement learning-problems. In both tasks, we might penalize the system for the time spent and other costs, and reward it for the quality of the final product. What is problematic, however, is that the structure of the web and the reasoning involved in the cooking task are most naturally represented using relational representations. This poses several challenges to the success of RL in these domains.

Function Approximation: The value-function approximation typically used in RL, e.g., neural networks or regression trees, does not generalize well when applied to relational domains. This is partly because these representations are ill-suited to the task of representing relational knowledge. When they are successful, they require a careful choice of propositional features or basis functions which are hand-crafted specifically to a particular task at hand. Designing function-approximation schemes that exploit relational structure when present is a critical challenge.

Generalization Across Objects: RL methods that do not explicitly represent objects and relationships between them are fundamentally limited in their ability to transfer learning about one object to similar related objects. Critical challenges here are identifying classes of objects that are to be considered "similar", across which such generalization is justified, and identifying and representing knowledge suitable for transfer.

Transfer across Tasks: RL programs are usually tested on a single task and do not exhibit transfer of knowledge across tasks. Each task in a given domain, e.g., each query in information retrieval, may look quite different when formulated propositionally, and thus may require separate training to convergence to solve. Relational representations facilitate formulating broad collections of related tasks as a single domain, yielding natural generalization between these related tasks.

Run-time Planning and Reasoning: In most reinforcement learning work, there is no deliberate planning and reasoning at run-time. It is tacitly assumed that either all planning occurs offline or the system relies completely on exploration and learning to construct good plans, reducing the run-time execution to reactive behavior. However, complex dynamic domains require both deliberation and reactivity, as is demonstrated by successful game-playing programs. It appears that the approximate nature of the value function demands a more refined search at the run time to compensate for its errors. Reasoning may also be important in constructing new features for an improved value-function approximation.

Prior Knowledge: RL de-emphasizes the role of prior knowledge in learning and reasoning, thus relying on trial and error learning which can be very inefficient and often does not scale to more complex tasks such as above.

Relational reinforcement learning (RRL) seeks to address all the above problems by generalizing RL to relationally represented states and actions. In fact, both Reinforcement Learning and Relational Learning have a long history. The study of reinforcement learning began with Samuel's pioneering 1959 work on checkers (Samuel, 1990). Work on relational learning started with Winston's work on blocks world learning (Winston, 1975). 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.

Perhaps less obviously, and as alluded to in the introduction, reinforcement learning is also closely related to *speedup learning* and to systems such as SOAR and Prodigy. Indeed, the classic *Readings in Machine Learning* (Shavlik & Dietterich, 1990) classifies Samuel's work under speedup learning. This is justified because in speedup learning, the learning task is viewed as speeding up a brute-force problem solver by learning appropriate control knowledge, i.e., learn what to do when or *a policy*. While the work in speedup learning, following the classical planning paradigm, did not consider dynamic domains and stochasticity, it did utilize relational representations from the start. Another difference between speedup learning and reinforcement learning is that in speedup learning, it is assumed that the domain theory, i.e., action models and immediate reward function are given, while in RL, only a simulation model is available.

Relational Reinforcement Learning (RRL) brings together the expressiveness of relational representations of states and actions, and the dynamic and stochastic nature of reinforcement learning, thus creating a very general and challenging setting for learning.

3. Reinforcement Learning

Reinforcement Learning (RL) is based on the idea of an agent that can interact with its environment using sensors to perceive its world, and effectors to act on the world. In addition to sensors and effectors, the agent also receives *reinforcement* or *reward*. In general, the environment may be stochastic, dynamic, and may only be partially observable. The goal of the agent is to behave in a way that optimizes its expected long-term utility, for different operational definitions of what this means, e.g., optimize expected average reward per step or optimize expected total reward discounted geometrically in each step (Puterman, 1994).

Given this rather grand goal of RL, it is not surprising that most if not all AI problems can be cast in an RL framework. Indeed, RL can be seen as one of a collection of *AI-complete* problems, or problems whose solution would imply success on a broad range of the goals of the field. However, current methods, reviewed briefly next, do not approach handling the environmental richness this would imply. RRL is a critical step towards bridging this gap, allowing RL methods to scale into richly structured domains.

The Reinforcement Learning literature offers several approaches to solving the RL problem. The "policy gradient" approaches directly search a parameterized space of policies in some language guided by an estimate of the gradient of its performance measure with respect to its parameters. One advantage of these approaches are that they are also applicable to partially observable environments under some conditions.

Another class of methods rely more strongly on the observability of the state, and learn an optimal policy indirectly by learning a suitably parameterized real-valued *value function* over states or state-action pairs. The parameters of the value function are locally updated, moving the value of the current state closer to the value of the next state plus any immediate reward. Under strong assumptions that are difficult to meet in practice, these local updates can be shown to cause the value function to converge to the true expected utility achievable from the given state (under the given action, if any), which we call the *optimal* value. If an optimal value function is known, the agent can perform optimally by acting greedily with respect to it, choosing at each state the action that maximizes the expected value of the resulting state. The collection of methods that learn an (approximately or heuristically) correct value function by some variant of the just described local updates are loosely termed *value iteration methods*. When the value function being updated assigns values to state-action pairs, it is termed a *Q-function*, and the methods are also referred to as *Q-learning methods*. In either case, the process of updating the value-function estimate at every state locally, is called *value-function regression*, as it can be seen as regressing the value-function estimate through the action dynamics of the environment by one step.

A related approach to finding an optimal policy, *policy iteration*, is based on a simple theorem: acting greedily with respect to a sub-optimal value function V will obtain at least as much value as indicated by that value function, and more value at at least one state. In other words, the greedy policy with respect to V achieves more value than V, as long as V underestimates optimal value. Policy iteration starts with an arbitrary such V, and computes the improved value function V' for the greedy policy with respect to V; this process can then be repeated, taking V' as V, until an optimal V is found. Convergence requires only a few iterations in practice, but the process of finding the improved V' from V is expensive, though polynomial, in large state spaces. To handle this problem, in large state spaces, V' can be sampled at any given state s by averaging the utilities obtained on a number of sample trajectories from s, acting greedily according to V; this process is called *policy rollout* (Bertsekas & Tsitsiklis, 1997). Given enough such samples of V', machine learning techniques can learn an approximation of V', say a linear combination of statespace features, giving a cheaper way to conduct policy iteration approximately in large state spaces.

4. Relational Reinforcement Learning: State of the Art

In this section, we outline some of the promising current approaches for Relational Reinforcement Learning.

4.1. Relational Regression and Q-learning

Through the use of relational regression, the RRL system (Džeroski et al., 2001) allows the application of almost standard Q-learning to reinforcement learning problems in environments that are characterized by their relational nature.

The use of relational representations of states and actions combined with relational regression for Q-function generalization allows the use of structural information such as the existence of objects with the right properties or relations between objects in the description of the Q-values, and as a consequence in the description of the derived policy. This enables the re-use of experience on smaller but related problems when confronted with more elaborate or simply larger tasks.

Three regression algorithms have been developed for use in this RRL system: the TG algorithm, which incrementally builds first order regression trees, an instance based algorithm called RIB and a kernel based algorithm KBR that uses Gaussian processes as the regression technique.

The TG algorithm (Driessens et al., 2001) is a combination of the Tilde algorithm (Blockeel & De Raedt, 1998) that builds first order classification and regression trees and the G-algorithm (Chapman & Kaelbling, 1991) that uses a number of statistical values concerning the performance of each possible extension in each leaf of the tree to build the tree incrementally. The relational regression trees used by the TG algorithm use conjunction of first order literals as tests in the internal leafs. The test corresponding to a certain leaf is the conjunction of the tests appearing on the path from the root of the tree to the leaf, in which any appearing variable is existentially quantified. The TG algorithm employs a user-defined refinement operator that originated in the Tilde system to generate the possible first order tests that can be used to replace a leaf. The statistics stored by the TG algorithm algorithm in each leaf of the tree consist of the number of examples classified positively or negatively by each possible test and the sum of the Q-values and squared Q-values in each of these cases. This allows the use of an F-test to decide which test to select. For now, no tree restructuring is done by TG. All decisions that the algorithm makes are final.

The instance based algorithm RIB (Driessens & Ramon, 2003) uses k-nearest-neighbor prediction as the regression technique, i.e., 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. The distance used needs to be able to cope with relational representations of states and actions and can be either a general purpose first order distance (Sebag, 1997; Ramon & Bruynooghe, 2001) or an application-specific one, which can usually be computed more efficiently. Because Q-learning generates a continuous stream of learning examples, a number of example selection methods were developed to reduce both the memory and the computational requirements. These selection criteria are based on those used in IB2 and IB3 (Aha et al., 1991) and look at the influence of individual examples on the overall prediction error.

The third algorithm is called KBR (Gärtner et al., 2003a) and uses Gaussian processes as the regression technique. Gaussian processes (MacKay, 1997) require a positive definite covariance function to be defined between the example descriptions. Because of the use of relational representations in the RRL system, kernels for structured data have to be used to fulfill this task. Possible candidates here are the convolution kernel (Haussler, 1999) or kernels defined on graphs (Gärtner et al., 2003b). Because Gaussian processes are a Bayesian technique, the KBR algorithm offers more than just a basic prediction of the Q-value of a new unseen example. It can also give an indication of the expected accuracy of this estimate, which in turn can be used, for example, by the Q-learning algorithm to guide exploration.

One of the major problems that reduces the applicability of Q-learning with relational function abstraction stems from the nature of Q-values themselves, i.e., their implicit encoding of both the distance to and the size of the next reward. These can be very hard to predict in stochastic and highly chaotic tasks. Other approaches such as advantage learning or policy iteration seem more appropriate in such cases.

4.2. Approximate Policy Iteration for RRL

Approximate policy iteration, as described above, can be viewed as moving from value function V to better value function V', or, alternatively, as moving from corresponding greedy policy π to better greedy policy π', and then iterating. Here, π acts greedily with respect to V, and π' acts greedily with respect to V'.

Nearly all uses of approximate policy iteration, until very recently, represent value functions directly, but represent the corresponding policy only implicitly (as greedy action with respect to the directly represented value function). This approach can work well (e.g., in TD-gammon (Tesauro, 1995)) for propositional domains, given extensive expert-human feature engineering, but has not been successful for highly structured, relational domains. For reasons discussed above, it is difficult to find good approximate value-function representations for these highly structured domains.

An alternative approach is to directly represent the policies involved, and only implicitly represent the value functions. Given an explicit representation of a policy π, the implicitly represented value function is the value obtained by executing π repeatedly from each state. Policy rollout can still be used, as described above, to generate samples of π', given π, by drawing suitable trajectories under π starting with each possible alternative action. Note that, in this approach, a supervised-classification learner is used to learn π' rather than the previous use of a regression learner to approximate V'.

The advantage of this alternative approach is that it is often easier to represent and learn suitable policies for structured domains than it is to represent and learn accurate value functions. General-purpose policy languages can be given that leverage decades of work on knowledge representation to allow compact, learnable description of many useful policies (Martin & Geffner, 2000; Khardon, 1999; Yoon et al., 2002). Substantial empirical work has shown (Fern et al., 2003; Fern et al., 2004) that policies can be learned with little or no human assistance for a variety of difficult, structured domains derived from benchmark planning problems used in the first three international planning competitions. Learning systems using this form of approximate policy iteration can learn policies that compete with state-of-the-art deterministic planners in these domains. However, unlike deterministic planners, these systems are robust to the introduction of uncertainty and can be shown to perform well in stochastic variants of the same problems. Also, a learning system learns a policy for the entire planning domain once, and thereafter can solve any instance in the domain by simply executing the learned policy. In contrast, deterministic planners use a new search for each problem instance, transferring no knowledge between instances.

There are many open research issues remaining regarding the approximate-policy-iteration approach to RRL. First, the policy languages explored to date are fairly limited. It remains to be determined if a good, learnable, general-purpose policy language can be found to avoid the need for human redesign of the language when meeting new domains. In particular, policy languages that incorporate memory, rather than simply being reactive to the current state, are only beginning to be explored. Also, current policy languages lack a general ability to incorporate background knowledge about the problem domain into either the policy language or the policy learner. Finally, these techniques have not been extended into partially observable or multi-agent environments, though there are natural ways to do so.

4.3. Symbolic Dynamic Programming

A tantalizing approach to RRL is to take advantage of a symbolic representation of the state-transition model to do a symbolic version of a "Bellman backup." The roots of this approach go back to *Explanation-Based Learning*, (EBL) wherein, subsequent to a successful problem-solving session, a proof is constructed that explicates the reasons behind its success. The proof is then generalized to construct a description of states which can be solved in the same way (Mitchell et al., 1986; DeJong & Mooney, 1986). In state-space problems and MDPs, proofs correspond to showing that a sequence of actions achieves a goal, and EBL corresponds to goal regression over an operator sequence. Indeed, EBL is at the heart of the generalization algorithms used in systems such as Prodigy and SOAR (Minton et al., 1989; Laird et al., 1986) to learn general control rules from specific examples of problem-solving episodes. Dietterich and Flann combined this idea with Reinforcement Learning by associating these generalized state descriptions with values obtained from Bellman backup (Dietterich & Flann, 1997). Thus, one could learn descriptions of states which lead to a win in at most 1 move, 2 moves, 3 moves, etc., and use them to pick the best move in any state.

Boutilier generalized the resulting Explanation-based Reinforcement Learning (EBRL) to stochastic domains, whose reward models are described by structured Bayesian networks, thus making them amenable to symbolic reasoning (Boutilier et al., 2001). An advantage of both of these approaches is that rather than inductively generalizing a set of examples using an unclearly motivated syntactic bias, the generalization is provably correct, starting from the symbolic domain theory. That the domain theory is readily available as a compact, symbolic representation is a standard assumption in decision-theoretic planning. When this is not true, first learning a compact description of the domain theory (Pasula et al., 2004) and using it in goal regression might still be more efficient than directly learning a value function. This is because, more often than not, the domain model lends itself to a compact representation, even when the value function does not. Consider, for example, the rules of chess, or the description of various planning domains in PDDL. The assumption that the domain model is compact is similar to the assumption made in inductive approaches that the policy to be learned is compact; this assumption underlies the syntactic bias used in approximate policy iteration for RRL, for example.

Unfortunately, symbolic dynamic programming (SDP), as this method is sometimes called, is not a panacea. The description of states that share a given value becomes increasingly complex and disjunctive, as these states get farther and farther from the goal. The number of states covered by each conjunctive description reduces dramatically, leading to a large number of low coverage rules. Indeed, this has been observed to be the case in early EBL systems, leading to what is called the "utility problem" (Minton, 1988; Dietterich & Flann, 1997). At some point, it becomes necessary to give up on exact representation of the value function, and approximate it compactly, or else one would spend more time matching all the rules rather than searching for a solution in the original state space. Making such approximations typically involves inductive learning and results in an approach that is qualitatively similar to relational regression.

Some problems can be abstracted to equivalent smaller problems by statespace aggregation. Problems with smaller equivalent problems formed in this way typically yield well to SDP. These problems can also be attacked directly by finding the relevant statespace aggregation using model minimization (Givan et al., 2003) and then solving the resulting smaller problem with any applicable technique (e.g., value iteration).

4.4. Directly Approximating the Value Function

Only very recently has any work addressed the approach of improving our value function representations so that the resulting approximation exploits the relational structure of the domain without substantial human engineering of statespace features (Guestrin et al., 2003). The novelty of this work is that it is able to use an efficient method, i.e., linear programming, to directly approximate the value function.

Unfortunately, this necessitates making several assumptions, some of which are quite severe. One of the strongest assumptions is that the relational properties between objects do not change over time. The need for such a strong assumption underscores the difficulty of directly approximating the value function. While this may not look all that severe in the limited subdomain of Freecraft on which this work is evaluated, note that in every planning benchmark from the international competitions, relations between objects change over time.

Given this limited setting, the global value function is assumed to decompose additively into local value functions for each object. Under the further assumption that objects fall into classes (which can possibly be learned automatically), the local value function approximation is also allowed to vary between classes. The resulting method must find a local value function for each class of objects. Here, the term "local" indicates that the value contributed by a given object can only depend on properties of that object (and possibly those objects immediately related to it); in the work reported, the local value is a linear combination of the local object properties. Finding a good value function then reduces to finding weights for the linear combination to be used for each class of objects.

Given this value-function approximation, the problem of finding appropriate weights can then be cast as an exponentially large linear program, and approximately solved using a constraint-sampling technique. The result can be guaranteed to select weights close to the best possible. If the assumptions about the value function implicit in the approximation hold (which may indeed be a big if), then the method is guaranteed to closely approximate the true value function.

All RRL methods we described so far take advantage of the relational representation to generalize their value functions or policies to similar domains that share properties and objects. In each case, the generalization bias inherent in their knowledge representations determines the effectiveness of the generalization. This point, more than anything else, dramatizes that knowledge representation is really a key issue in RRL as it determines what generalization occurs.

5. Research Issues in RRL

Research on RRL offers many promises, but also opens up many new questions and challenges. Here are *some of* the questions that are immediately apparent and pressing.

Theory of RRL: Unlike the propositional RL literature, the theory of RRL is not as mature and is just being developed (Boutilier et al., 2001; Kersting et al., 2004). While the basic results of finite MDPs carry over to relational domains with finite objects, these results are not as useful since they depend on propositionalization and suffer from the same problems of non-generalizability as the propositional RL. Thus theoretical and practical issues underlying effective function approximation are somewhat more crucial for RRL.

Hierarchical RRL: Hierarchies are important to reduce the complexity of decision making, and allow transfer across different tasks. Hierarchical RL is an active research topic within the propositional setting. Relational setting allows richer hierarchies that include "subtask" as well as "more-specific-than" relationships between tasks. How do these richer hierarchies help learning? How to learn these hierarchies automatically?

Model Learning: One of the issues in Reinforcement Learning is how to represent and learn action models. Explicit enumeration of states is not possible in all but trivial domains. Factored models (e.g., Dynamic Bayes Networks) can represent actions succinctly. Relational settings require even richer representations, for example, probabilistic relational models (PRMs) or probabilistic STRIPS-style operators. How can we learn these richer action models? How can we use them in reasoning and learning of policies? Early work on this topic is reported in (Pasula et al., 2004).

Policy Learning: One of the central debates of Reinforcement Learning is whether policy learning or value-function learning is more appropriate for a given domain. The current work suggests that in some relational domains, including the blocks world, one would get better generalization by policy learning. What are the conditions under which this is the case? Is it possible to improve value function methods so that they can be competitive with policy learning? Or is it more appropriate to incorporate value-function learning into a learnable policy language that can then refer to value in defining the policy? A related issue here is identifying better/ideal general policy languages that can support learning and represent useful policies in a wide range of domains.

Satisficing: Propositional reinforcement learning is concerned with learning optimal policies. Unfortunately in many relational domains, optimal policies are NP-hard or worse, while there exist useful polynomial-time suboptimal policies. How should one make the tradeoff between the optimality and the efficiency of a policy? Work reported above relies on inductive bias to make this tradeoff in a poorly understood manner.

Prior Knowledge: Humans appear to exploit (and develop) domain knowledge in handling large domains. Lack of means to handle prior knowledge may be a key issue in limiting scaling of RL systems. What kind of prior knowledge is easy to discover/communicate and exploit effectively by an RL system? Can relational representations make it easier to do this and at what cost? How can learning and reasoning effectively complement each other?

Reasoning after Solution: The most successful RRL methods to date use inductive methods that limit their ability to guarantee solution quality. Can reasoning be deployed in any tractable manner to verify the quality of a solution after it is found and/or identify problem areas requiring further planning?

6. A Summary of the Rest of the Proceedings

The rest of the proceedings contains descriptions of several on-going research efforts that address a variety of issues discussed above.

Ramon and Driessens explore adding example selection techniques to the kernel-based regression algorithm of Driessens to improve on the memory and computational requirements, but more importantly, to increase the numerical stability of the computations. Walker, Shavlik and Matwin describe an approach of building useful features by randomly sampling a large space of relational features, and using them in regularized kernel regression to predict the value function of a hand-coded policy. They obtain promising results in the Keep-Away subtask of RoboCup domain.

The abstract by Fern, Yoon and Givan summarizes their work on approximate policy iteration and its application to large relationally structured decision-theoretic planning problems. Itoh and Nakamura describe an approach to learn whether or not to use each rule in a hand-coded relational policy with a limited memory in a partially observable domain. They test their algorithm in a maze-like domain where planning is sometimes useful and the problem is to learn when it is useful. The paper by Strens describes an approach to search a parameterized policy space in a partially observable, two dimensional, multi-agent pursuer-evader domain. He shows that using relational policies outperforms either identical policies or joint policies when the number of pursuers is more than 2. Croonenborghs, Ramon, and Bruynooghe consider an approach of building "influence models" that predict rewards starting from the state features. These models, built in the framework of Bayesian logic programs, are used in combination of Q-values to choose actions based on multi-step look-ahead.

Gretton and Thiébaux describe an interesting approach that combines symbolic dynamic programming with an inductive regression method to reap the benefits of both while side-stepping the complex reasoning problems of SDP. Nason and Laird show how reinforcement learning can be integrated into the SOAR architecture by adding rewards and numerical preferences. They discuss the impact of SOAR's architectural assumptions on its ability to learn effectively in relational domains. Langley, Arai and Shapiro describe a cognitive architecture called ICARUS, which combines hierarchical skills and reactive execution, and contrast learning with and without explicit models of actions. A hierarchical approach to relational reinforcement learning with value function approximation is described by Roncagliolo and Tadepalli.

Morales proposes an approach to learn in an abstracted state space using an abstract version of Q-learning. He also describes a way to induce relational actions from traces of human experts. The abstract by Wilson points out the need for expressive policy languages and a bias towards simpler policies. Finally, van Otterlo and Kersting point out a number of challenges for relational reinforcement learning. These include developing a convergence theory, understanding the relative merits of policy-based versus value-function-based approaches, generalization across multiple domains, and exploiting prior knowledge.

7. Conclusions

We hope that we have convinced the reader that relational reinforcement learning offers a variety of challenges and opportunities. We motivated RRL and outlined a slew of research issues and some promising directions. With the heightened interest in relational representations in AI and a deeper understanding of the problems and prospects of reinforcement learning, it appears that the time is ripe to work on a comprehensive framework that includes expressive representations, inference, and action execution to try to solve practical problems of interest. We invite the reader to become a full participant in this adventure.

References

Aha, D., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, *6*, 37–66.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American*.

Bertsekas, D. P., & Tsitsiklis, J. N. (1997). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence*, *101*, 285–297.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first order mdps. *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (pp. 690–697).

Chapman, D., & Kaelbling, L. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisions. *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (pp. 726–731).

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, *1*, 145–176.

Dietterich, T. G., & Flann, N. S. (1997). Explanation-based learning and reinforcement learning: A unified view. *Machine Learning*, *28*, 169–210.

Driessens, K., & Ramon, J. (2003). Relational instance based regression for relational reinforcement learning. *Proceedings of the Twentieth International Conference on Machine Learning* (pp. 123–130). AAAI Press.

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. *Lecture Notes in Computer Science*, *2167*.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*, 7–52.

Fensel, J., Hendler, J., Lieberman, H., & Wahlster, W. (Eds.). (2002). *Spinning the semantic web: Bringing the world wide web to its full potential*. Boston, MA: MIT Press.

Fern, A., Yoon, S., & Givan, R. (2003). Approximate policy iteration with a policy language bias. *Advances in Neural and Information Processing 16*.

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

Gärtner, T., Driessens, K., & Ramon, J. (2003a). Graph kernels and Gaussian processes for relational reinforcement learning. *Inductive Logic Programming, 13th International Conference, ILP 2003, Proceedings* (pp. 146–163). Springer.

Gärtner, T., Flach, P., & Wrobel, S. (2003b). On graph kernels: Hardness results and efficient alternatives. *Proceedings of the 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop*. Springer.

Givan, R., Dean, T., & Greig, M. (2003). Equivalence notions and model minimization in markov decision proc esses. *Artificial Intelligence*, *147*, 163–223.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. (2003). Generalizing plans to new environments in relational mdps. *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Acapulco, Mexico: Morgan Kaufmann.

Haussler, D. (1999). *Convolution kernels on discrete structures* (Technical Report). Department of Computer Science, University of California at Santa Cruz.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Aritificial Intelligence Research*, *4*, 237–285.

Kersting, K., Van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. *Proceedings of the Twenty-First International Conference on Machine Learning*. Banff, Alberta, Canada: AAAI Press.

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

Laird, J., Rosenbloom, P., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

MacKay, D. (1997). Introduction to Gaussian processes. available at http://wol.ra.phy.cam.ac.uk/mackay.

Martin, M., & Geffner, H. (2000). Learning generalized policies in planning domains using concept languages. *Seventh International Conferene on Principles of Knowledge Representation and Reasoning*.

Minton, S. (1988). *Learning search control knowledge*. Boston, MA: Kluwer Academic Publishers.

Minton, S., Carbonell, J., Knoblock, C., Kuokka, D., Etzioni, O., & Gil, Y. (1989). Explanation-Based Learning: a problem solving perspective. *Artificial Intelligence*, *40*, 63–118.

Mitchell, T., Utgoff, P., & Banerji, R. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. Michalski and et al. (Eds.), *Machine learning: An artificial intelligence approach*, vol. 1. Morgan Kaufmann.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, *1*, 47–80.

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

Puterman, M. L. (1994). *Markov decision processes: Discrete dynamic stochastic programming*. John Wiley.

Ramon, J., & Bruynooghe, M. (2001). A polynomial time computable metric between point sets. *Acta Informatica*, *37*, 765–780.

Samuel, A. L. (1990). Some studies in machine learning using the game of checkers. In J. Shavlik and T. Dietterich (Eds.), *Readings in machine learning*, 535–554. Morgan Kaufmann. Originally published in *IBM Journal*, 3(3),July 1959.

Sebag, M. (1997). Distance induction in first order logic. *Proceedings of the Seventh International Workshop on Inductive Logic Programming* (pp. 264–272). Springer.

Shavlik, J., & Dietterich, T. (1990). *Readings in machine learning*. San Mateo, CA: Morgan Kaufmann.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: The MIT Press.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Comm. ACM*, *38*, 58–68.

Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision*. New York: McGraw Hill.

Yoon, S., Fern, A., & Givan, R. (2002). Inductive policy selection for first-order MDPs. *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence* (pp. 568–576).