Chapter 2
Reinforcement Learning

This chapter provides an overview of the field of reinforcement learning and concepts that are relevant to the proposed work. The field of reinforcement learning is not very well-known and although the learning paradigm is easily understandable, some of the more detailed concepts can be difficult to grasp. Accordingly, reinforcement learning is presented beginning with a review of the the fundamental concepts and methods. This introduction to reinforcement learning is followed by a review of the three major components of the reinforcement learning method: the environment, the learning algorithm, and the representation of the learned knowledge. Some of the terminology used herein may be slightly different from other fields, though this is done to be consistent with the reinforcement learning literature.

Note that in this work reinforcement learning is considered from the artificial intelligence or computer science perspective on solving sequential decision making problems. Sequential decision making problems, however, are also the focus of other fields with difference perspectives, including control theory and operations research (Kappen 2007; Powell 2008). Each of these fields uses slightly different methods that have been developed for, or have been successful in, solving types of problems with unique characteristics that are specific to each field, though there may be considerable overlap in the types of problems solved by each community. The operations research community uses approaches such as simulation-optimization, forecasting approaches for rolling-horizon problems, and dynamic programming methods (Powell 2008), whereas the control theory community uses integral control and related methods based on plant models (Kappen 2007). The field of reinforcement learning (from the artificial intelligence perspective) is not only related to other computational and mathematical approaches for solving similar problems, but it is also a well-accepted and fundamental physiological model of learning in the neuroscience community (Rescorla and Wagner 1972; Dayan and Niv 2008) with conceptual intersections between the two fields (Maia 2009; Niv 2009).

Portions of this chapter previously appeared as: Gatti & Embrechts (2012).

C. Gatti, *Design of Experiments for Reinforcement Learning,*
Springer Theses, DOI 10.1007/978-3-319-12197-0_2

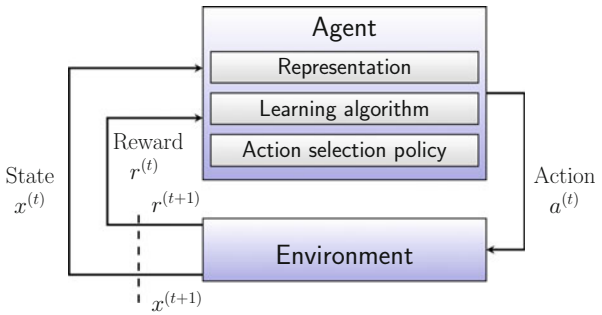

Fig. 2.1 The reinforcement learning paradigm consists of an agent interacting with an environment. The agent observes the state of the environment $x^{(t)}$ at time t, selects an action $a^{(t)}$ based on its action selection policy, and transitions to state $x^{(t+1)}$ at time $t+1$, at which point the environment issues a reward $r^{(t+1)}$ to the agent. Over time, the agent learns to pursue actions that lead to the greatest cumulative reward.

Reinforcement learning is based on an *agent*, an entity that is capable of learning and making decisions, that repeatedly interacts with an *environment* (or *domain*) (Fig. 2.1). Interactions between the agent and the environment proceed by the agent observing the state of the environment, selecting an action which it believes is likely to be beneficial, and then receiving *feedback*, or a *reward*, from the environment that provides an indication of the utility of the action (i.e., whether the action was good or bad). More specifically, in the most basic sense, the agent selects actions based on its perception of the value of the subsequent state. The feedback provided to the agent is used to improve its estimation of the value of being in each state. In the general reinforcement learning paradigm, rewards may be provided after each and every action. After repeated interaction with the environment, the agent's estimation of the true state values slowly improves, which enables the selection of more optimal actions in future interactions.

This method was largely developed as a machine learning technique by Sutton and Barto (1998). Many formulations of reinforcement learning have been developed in order to either accommodate various types of environments or to utilize slightly different learning mechanisms. One of the fundamental reinforcement learning methods is the temporal difference algorithm. This algorithm uses information from future states, and effectively propagates this information back through time in order to improve the agent's valuation of each state that was visited during an episode of agent-environment interactions. Improvements to the estimation of the state values therefore have a direct impact on the action selection processes as well. This algorithm is particularly useful in scenarios that do not necessarily follow the schema shown in Fig. 2.1, in which rewards are provided following every action. In some scenarios, with board games being a prime example, feedback is only provided at the end of the game in the form of a win, loss, or draw. The temporal difference algorithm can be used to determine the utility of actions played early in the game based on the outcome of the game, which results in a refined action selection procedure.

Reinforcement learning is more formally described as a method to determine optimal action selection policies in sequential decision making processes. The general framework is based on sequential interactions between an agent and its environment, where the environment is characterized by a set of states X, and the agent can pursue actions from the set of admissible actions A. The agent interacts with the environment and transitions from state $x^{(t)} = x$ to state $x^{(t+1)} = x'$ by selecting an action $a^{(t)} = a$. Transitions between states are often based on some defined probability $\mathcal{P}^{a}_{xx'}$. In this particular section, the time index of variables is denoted using a parenthetic-superscript or using prime notation (i.e., apostrophe), whichever is clearest or notationally cleanest, and subscripts are reserved for denoting other entities. The time index denotes the particular time step at which states are visited, actions are pursued, or rewards are received, and thus the sequences of states and actions can then be thought of as a progression through time.

The sequential decision making process therefore consists of a sequence of states $x = \{x^{(0)}, x^{(1)}, \ldots, x^{(T)}\}$ and a sequence of actions $a = \{a^{(0)}, a^{(1)}, \ldots, a^{(T)}\}$ for time steps $t = 0, 1, \ldots, T$, where state $x^{(0)}$ is the initial state and where $x^{(T)}$ is the terminal state. Feedback may be provided to the agent at each time step t in the form of a reward $r^{(t)}$ based on the particular action pursued. This feedback may be either rewarding (positive) for pursuing actions that lead to beneficial outcomes, or they may be aversive (negative) for pursuing actions that lead to worse outcomes. The terms *feedback* and *reward* will be used interchangeably, and it is important to note that a reward may be positive or negative, despite its conventional positive connotation. Processes that provide feedback at every time step t have an associated reward sequence $r = \{r^{(1)}, r^{(2)}, \ldots, r^{(T)}\}$. A complete process of agent-environment interaction from $t = 0, 1, \ldots, T$ is referred to as an *episode* consisting of the set of states visited, actions pursued, and rewards received. The total reward received for a single episode is the sum of the rewards received during the entire decision making process, $\mathcal{R} = \sum_{t=1}^{T} r^{(t)}$.

For the general sequential decision making process define above, the transition probabilities between states $x^{(t)} = x$ and $x^{(t+1)} = x'$ may be dependent on the complete sequences of previous states, actions, and rewards:

$$Pr\left\{x^{(t+1)}{=}x', r^{(t+1)}{=}r \mid x^{(t)}, a^{(t)}, r^{(t)}, x^{(t-1)}, a^{(t-1)}, r^{(t-1)}, \ldots, x^{(1)}, a^{(1)}, r^{(1)}, x^{(0)}, a^{(0)}\right\} \tag{2.1}$$

Processes in which the state transition probabilities depend only the most recent state information $(x^{(t)}, a^{(t)})$ satisfy the Markovian assumption. Under this assumption, the state transition probabilities can be expressed as:

$$Pr\left\{x^{(t+1)} = x', r^{(t+1)} = r \mid x^{(t)}, a^{(t)}\right\}$$

which is equivalent to the expression in Eq. 2.1 because all information from $t = 0, 1, \ldots, t-1$ has no effect on the state transition at time t.

If a process can be assumed to be Markovian and can be posed as a problem following the general formulation defined above, this process is amenable to both

modeling and analysis. The Markov decision process (MDP) is a specific process which entails sequential decisions and consists of a set of states X, a set of actions A, and a reward function $R(x)$. Each state $x \in X$ has an associated true state value $V^*(x)$. The decision process consists of repeated agent-environment interactions, during which the agent learns to associate states visited during the process with outcomes of entire process, thus attempting to learn the true value of each state. However, the true state values are unobservable to the agent, and thus the agents' estimates of the state values $V(x)$ are merely approximations of the true state values.

There are many cases where the Markovian assumption may not be valid. Such cases include those where the agent either cannot perfectly observe the state information, in which case the problem is referred to as a partially-observable Markov decision processes (POMDPs) (Singh et al. 1994), or if there is a long temporal dependence between states and the feedback provided. Approaches to solving these types of problem often include retaining some form of the state history (Wierstra et al. 2007), such as by using recurrent neural networks or a more complex variant that relies on long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997). These approaches, however, are not considered in this work.

The transition between states $x^{(t)} = x$ and $x^{(t+1)} = x'$ when pursuing action a may be represented by a transition probability $\mathcal{P}^a_{xx'} = Pr\{x^{(t+1)} = x \mid x^{(t)} = x,\, a^{(t)} = a\}$. For problems in which the transition probabilities between all states are known, the transition probabilities can be used to formulate an explicit model of the agent-environment interaction process. A policy π is defined to be the particular sequence of actions $a = \{a^{(0)},\, a^{(1)},\, \ldots,\, a^{(T)}\}$ that is selected throughout the decision making process. Similarly, the expected reward for transitioning from state x to state x' when pursuing action a may be represented by $\mathcal{R}^a_{xx'} = \mathbb{E}\{r^{(t+1)} \mid x^{(t)} = x,\, a^{(t)} = a,\, x^{(t+1)} = x'\}$. An optimal policy π^* is considered to be the set of actions pursued during the process maximizes the total cumulative reward $\mathcal{R}$.

The optimal policy can be determined if the true state values $V^*(x)$ are known for every state. The true state values can be expressed using the Bellman optimality equation (Bertsekas 1987), which states that the value of being in state x and pursuing action a is a function of both the expected return of the current state and the optimal policy for all subsequent states:

$$V^*(x) = \max_{a \in A} \mathbb{E}\left\{r^{(t+1)} + \gamma\ V^*\left(x^{(t+1)}\right) \mid x^{(t)} = x,\, a^{(t)} = a\right\} \tag{2.2}$$

$$= \max_{a \in A} \sum_{x^+} \mathcal{P}^a_{xx'} \left[\mathcal{R}^a_{xx'} + \gamma\ V^*(x')\right] \tag{2.3}$$

The Bellman equation provides a conceptual solution to the sequential decision making problem in that the value of being in a state is a function of all future state values. The solution to reinforcement learning problems is often regarded as a policy π^*, or an action selection procedure, that leads to an optimal outcome. When implementing reinforcement learning however, the Bellman equation does not have to be explicitly solved. Rather, the identification of optimal policies in reinforcement learning problems is based on the notion that future information is relevant to the valuation of current states, and that accurate estimations of state values can be used to determine the optimal policy.

As previously mentioned, if the transition probabilities $\mathcal{P}_{xx'}$ between states are explicitly known, these probabilities can be used to formulate a model of the system. However, if these transition probabilities are unknown, transitions between states must proceed by what could be referred to as empirical sampling in a simulation-like manner. In this case, the structure of the environment can be used with the allowable actions to essentially generate the transitions between states and constrain the behavior of the agent, where the structure of the environment refers to anything that governs or has an effect on the dynamics of the environment. When state transition probabilities are unknown, the agent-environment system is considered to be *model-free*. This model-free type of problem is what classical reinforcement learning studies, and this is also the focus of this work.

Whereas supervised learning problems have a static mapping between the input space X and their labels Y, and sequence learning problems have a sequence-dependent mapping between the input space X and a set of labels Y, the only true mapping in reinforcement learning is between some states $x^* \subset X$, where x^* denotes a goal or terminal states, and their associated rewards R, which may be provided very sporadically. The classical reinforcement learning problem can be defined by an agent that interacts with an environment, and that attempts to learn the utility of states by the receipt of (possibly infrequent) rewards, in order to select a sequence of actions that maximizes the reward received. While there are many extensions, modifications, and special cases to this learning paradigm, this definition is sufficient for this work, and any exceptions will be noted.

2.1 Applications of Reinforcement Learning

Reinforcement learning has been successfully applied to many different problem areas. As the methods and extensibility of this learning approach are still being understood, a large majority of the literature uses benchmark problems. Beyond these benchmark problems, the application of reinforcement learning to games represents the next largest category, followed finally by a comparably small amount of works that apply reinforcement learning to real-world problems. In all of these applications, the definition of a *successful* application or implementation of reinforcement learning is subject to interpretation, and it is important to keep this in mind. Initial or incremental steps made in a particular application may be viewed as progress by some; others, however, may view such work as a failure if an optimal solution is not found or perfect performance is not achieved.

2.1.1 Benchmark Problems

Benchmark problems and domains are often used in reinforcement learning to evaluate novel approaches under very well-understood domain conditions. These domains can have either discrete or continuous state spaces that are often low dimensional ($\sim$ 2), allowing for visualization of functions based on their state space. These

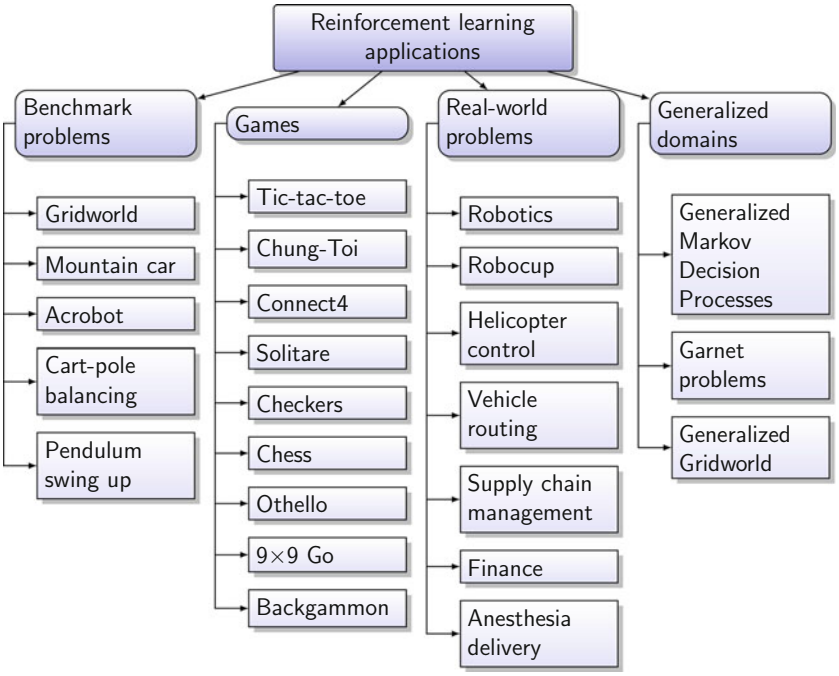

Fig. 2.2 Reinforcement learning has been applied to benchmark problems, games, real-world problems, and generalized domains. This figure shows, for each category, examples of the types of problems to which reinforcement learning has been applied.

domains are almost exclusively based on a single agent (non-adversarial domains). Though these problems are widely used and accepted, the extensibility of the learning method novelties applied to these problems is often not fully explored.

The most widely used benchmark problem in reinforcement learning is Gridworld (Sutton and Barto 1998; Fig. 2.3). In its most basic form, this domain consists of a discrete planar grid with finite dimensions. An agent is placed at some set starting location, and then selects cardinal actions (up, down, left, right) to move within the grid with the goal of reaching some specific goal grid location. Modifications to this domain include the addition of diagonal moves, the addition of penalty or hole grid locations, changes to the dimensions of the grid, and the addition of a stochastic 'wind' component that acts on the agent (Sutton and Barto 1998). Further extensions include using a large gridspace with multiple rooms, which has been used for evaluating hierarchical learning strategies through the use of subgoals (Bakker and Schmidhuber 2004; Şimşek and Barto 2004). All of these modifications are purely environmental and have an effect on the dynamics of the domain, which has downstream effects on the actions of the agent, the efficacy of the learning algorithm, and finally on the knowledge acquired by the agent and its performance in the domain.

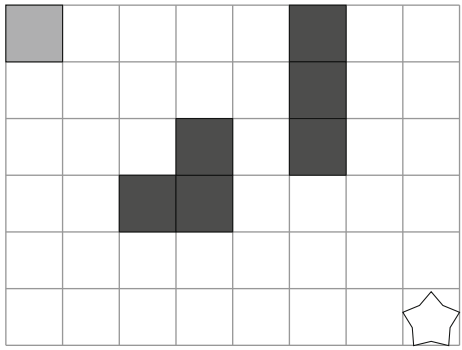

Fig. 2.3 Example Gridworld domain: The goal of this domain is to move from the starting location (*gray square*) to the goal (*star*) by taking cardinal actions and avoiding the walls (*black squares*).

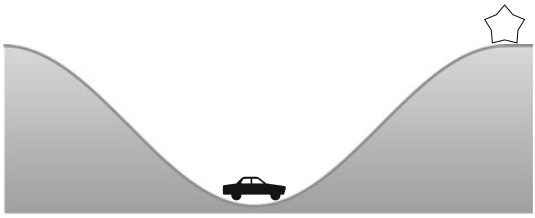

Fig. 2.4 Mountain car problem: The goal of this domain is to have the underpowered car reach the top of the mountain (*star*) by building momentum using forward or reverse actions or using no action.

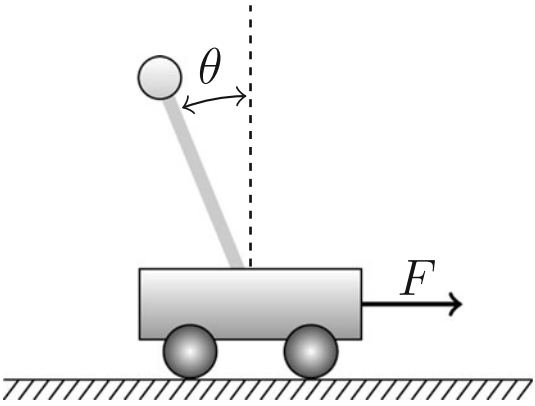

Fig. 2.5 Cart pole-balancing task: The goal of the cart-pole balancing task is to keep the pole vertically-oriented by applying left- or right-directed forces to the cart.

Additional benchmark problems include low-dimensional and continuous control problems that are based on well-defined dynamics of physical systems. Examples of these domains include the mountain car domain (Moore 1990; Singh and Sutton 1996; Riedmiller 2005; Fig. 2.4), acrobot (Sutton and Barto 1998), cart-pole balancing (Barto et al. 1983; Riedmiller 2005; Fig. 2.5), and the pendulum swing-up task (Doya 1996, 2000). The dynamics of the environment (i.e., equations of motion) are unknown to the agent, though it must learn how to behave in the environment only through selecting control actions and the receipt of rewards or penalties.

2.1.2 Games

Games represent a large proportion of the applications of reinforcement learning. In almost all of these applications, the domains are based on two-agent, adversarial, and zero-sum situations. Games in which reinforcement learning have been applied include: Tic-Tac-Toe (Wiering 1995; Ghory 2004; Patist and Wiering 2004; Konen and Beielstein 2008, 2009; Gatti and Embrechts 2012), Chung-Toi (Gatti et al. 2011a), Connect4 (Ghory 2004), Solitare (Yan et al. 2004), checkers/draughts (Schaeffer et al. 2001; Patist and Wiering 2004; Wiering et al. 2007), Chess (Thrun 1995; Baxter et al. 1998a; Mannen and Wiering 2004; Wiering et al. 2007; Veness et al. 2009), Othello (Binkley et al. 2007; van Eck and van Wezel 2008; Yoshioka et al. 1999; Skoulakis and Lagoudakis 2012), 9×9 Go (Schraudolph et al. 1994; Silver et al. 2012), and backgammon (Tesauro 1995; Wiering et al. 2007; Papahristou and Refanidis 2011). Note that these games span a large range of difficulty and that they all have unique environmental characteristics. It is also very important to note that the large majority of these applications to games have not resulted in agents that can play perfect games against any level of opponent. A truly successful application of reinforcement learning to games is often considered one that is capable of matching or beating the performance of human master players or (non-reinforcement learning trained) computer programs. Some of these successes include those to chess (Veness et al. 2009), checkers (Schaeffer et al. 2001), 9×9 Go (Silver et al. 2012), and backgammon (Tesauro 1995). In most circumstances, reinforcement learning applications to games are evaluated against computer opponents that either use a simplistic action-selection policy or are computer programs that have been developed for the same game (e.g., Schraudolph et al. 1994; Patist and Wiering 2004; Silver et al. 2012). Far fewer applications evaluate the performance of a reinforcement learning agent against human opponents, such as in Tesauro (1995) or Gatti et al. (2011b).

The most notable and most cited application of reinforcement learning is the work of Tesauro (1995) who trained a neural network to play the board game of backgammon that could challenge and win against human grandmasters in world championship play. The reasons why this applications was so powerful are not quite understood; similarly, the reasons why no other application has seen similar success is not understood as well. Some attribute the success of this application to the speed of play, representation smoothness, and stochasticity (Baxter et al. 1998b), while others question whether it is actually a true success and claim that its success is not due to reinforcement learning but rather the dynamics and co-evolution of the game (Schraudolph et al. 1994; Pollack and Blair 1996).

2.1.3 Real-World Applications

Applications of reinforcement learning to real-world problem are much less cited, though there are still numerous applications in a variety of domains. In the field of robotics, Smart and Kaelbling (2002) and Smart (2002) used Q-learning to train

mobile robots to navigate through different environments. Kohl and Stone (2004) used policy gradient reinforcement learning to train quadruped robots to walk with a fast, stable, and robust gait. Kwok and Fox (2004) used least squares reinforcement learning to train RoboCup (robot soccer) robots how to use sensors to accurately detect, gain possession, and kick a ball. Peters and Schaal (2006) used policy gradient methods for robot control with extensions to motor control strategies (Peters and Schaal 2009). Reinforcement learning for motor control has also been explored by Coulom (2002a, b).

Control is another area in which there have been successes, most notably to the control of autonomous helicopters for learning how to fly under challenging conditions (Bagnell and Schneider 2001; Ng et al. 2004). In the field of operations research, Proper and Tadepalli (2006) use reinforcement learning in a toy vehicle routing problem, and Gabel and Riedmiller (2007) use multi-agent reinforcement learning to solve a benchmark job-shop scheduling problem. A reinforcement learning algorithm named SMART has been used in a number of OR-related applications, including to optimizing industrial manufacturing processes (Mahadevan and Theocharous 1998), airline revenue management (Gosavi et al. 2002), and supply chain management (Pontrandolfo et al. 2002). Reinforcement learning has also been used for a number of problems in finance and economics. Moody and colleagues use reinforcement learning to trade financial instruments such as currencies, stocks and stock indices, and treasury bills (Moody et al. 1998; Moody and Saffell 2001), and Gorse (2011) has also applied reinforcement learning to trading stock indices. Lee (2001) applied the basic reinforcement learning algorithm TD(0) to predict stock prices. Similar work using Q-learning has been applied to stock trading and asset allocation (Nevmyvaka et al. 2006; O et al. 2006). Li and colleagues used a reinforcement learning algorithm called least-squares policy iteration (LSPI) to learn policies for exercising American stock options (Li and Schuurmans 2008; Li et al. 2009). Moore et al. (2014) applied the Q-learning reinforcement learning algorithm (with some minor adjustments) to controlling the delivery of anesthesia. In this work, an *in silico* simulation was used to develop an initial control strategy, which was subsequently refined in a real study.

The number of successful applications of reinforcement learning, however, is far fewer than one would expect since the field's inception and significant foundational works in the early 1980s (Barto et al. 1983; Sutton 1984). Success of this learning method is dependent on the *interaction* between an environment, a learning algorithm, and some form of knowledge representation (Langley 1988), and in general, the reasons why reinforcement learning works in some situations and not in others is not well understood. Successful applications are often rely on heuristics and ad-hoc implementations that require practical and theoretical knowledge about the environment, learning algorithm, and form of knowledge representation, instead of being based on formal proofs, and this approach may be the reason for such mixed results (Tsitsiklis and Roy 1996; Konen and Beielstein 2008). Formal proofs have had a place in reinforcement learning, however (see Sects. 2.2.2 and 2.2.3.2), though the assumptions on which they are based limit their applicability to very simplistic situations (Tsitsiklis and Roy 1996). Because of this, simple problem domains are still used to understand how environmental and problem characteristics affect learning.

These problems are simpler than their real-world and end-goal counterparts, but the insights that can be gained from them can be valuable in achieving the end-goal. For example, the multi-armed bandit problem is where an agent is attempts to maximize its reward from multiple single-armed bandits (i.e., slot machines), where each bandit has a different expected reward. Learning how to maximize performance in this domain requires both exploitation of knowledge and exploration of potentially better actions. This problem scenario may be used to gain an understanding of the dynamics of a problem, and such knowledge has been used to aid in the application of similar methods to problems in energy management or robotics (Galichet et al. 2013; Karnin et al. 2013).

2.1.4 Generalized Domains

There is another class of domains to which reinforcement learning has been applied that deserves attention. These domains are not based on any physical, real-world, or game-like domain. Rather they could instead be considered *generalized domains* that are either completely abstract environments or are environments that have parameterized characteristics. The term *abstract* in this sense refers to an environment which lacks a physical representation and is merely a numerical representation with defined properties, dynamics, and constraints. While all of the environments mentioned in this section have not been used with reinforcement learning specifically, they have been used with some form of sequential decision making-related algorithm.

One of the original purely abstract generalized domains was for infinite horizon problems developed by Archibald et al. (1995) out of a need for a more flexible and widely available platform for assessing the characteristics of Markov decision processes. This domain is highly parameterized, using 14 variables that determine the dynamics of the domain; one of the novelties of this work was that it allowed for the control of the mixing speed of the problem (Aldous 1983), which is related to the relative importance of the starting state of the process. However, this type of domain is based on enumerated states that are used with look-up table representations, rather than state vectors that are commonly used in present day problems.

Garnet (Generalized Average Reward Non-stationary Environment Testbed) problems are a similar type of generalized abstract domain that have been developed more recently, though these domains are for episodic learning problems (Bhatnagar et al. 2009; Castro and Mannor 2010; Awate 2009), as is considered in the present work. The domains generated by Bhatnagar et al. (2009) are based on a simpler, five parameter domain that is defined by: (1) the number of states, (2) the number of actions, (3) the branching factor, (4) the standard deviation of the reward(s), and (5) the stationarity. Two additional parameters are the dimensionality and the number of non-zero components of the state vectors. The domains created by Awate (2009) and Castro and Mannor (2010) omit the stationarity parameter, but otherwise use the same domain construction method.

Springer

http://www.springer.com/978-3-319-12196-3

Design of Experiments for Reinforcement Learning

Gatti, C.

2015, XIII, 191 p. 46 illus., 25 illus. in color., Hardcover
ISBN: 978-3-319-12196-3