The Reinforcement Learning Competitions (preprint)

Shimon Whiteson Brian Tanner Adam White

Abstract

This article reports on the reinforcement-learning competitions, which have been held annually since 2006. In these events, researchers from around the world developed reinforcement-learning agents to compete in domains of various complexity and difficulty. We focus on the 2008 competition, which employed fundamentally redesigned evaluation frameworks that aimed to systematically encourage the submission of robust learning methods. We describe the unique challenges of empirical evaluation in reinforcement learning and briefly review the history of the previous competitions and the evaluation frameworks they employed. We describe the novel frameworks developed for the 2008 competition as well as the software infrastructure on which they rely. Furthermore, we describe the six competition domains, present selected competition results, and discuss the implications of these results. Finally, we summarize the 2009 competition, which used the same evaluation framework but different events, and outline ideas for the future of the competition.

1 Introduction

Competitions have a long history in artificial intelligence. Events such as RoboCup, the Netflix Prize, the Trading Agent Competition, the International Planning Competition, the General Game Playing Competition, the AAAI Computer Poker Competition, and the DARPA Grand Challenge, have succeeded in raising awareness and stimulating research about their respective topics. The empirical, problem-oriented nature of these competitions can be an important counterweight to traditional research efforts, which often focus more on theoretical results and algorithmic innovation. Competition results provide a barometer for which approaches are popular, effective, and scalable to challenging problems. The competitive nature of the events provide an incentive to transform theoretical ideas into practical tools, enabling the field to reap the benefits of its scientific progress.

The field of *reinforcement learning* (RL) (Kaelbling, Littman, and Moore 1996; Sutton and Barto 1998) is ripe for such a transformation. Broadly speaking, RL researchers aim to develop online algorithms for optimizing behavior in *sequential decision problems* (SDPs), wherein agents interact with typically unknown environments and seek behavior that maximizes their long-term reward. Many challenging and realistic domains can be cast in this framework (e.g., robot control, game-playing, and system optimization), so RL algorithms contribute to the broad goals of artificial intelligence. In recent years, many advances have been made in RL theory and algorithms, particularly in areas such as exploration vs. exploitation, function approximation, policy search, hierarchical methods, model-based approaches, partial observability, and multiagent systems. To improve the RL community's expertise in developing practical learning systems, researchers recently began organizing reinforcement-learning competitions.[1]

This article reports on these events, in which researchers from around the world developed reinforcement-learning agents to compete in domains of various complexity and difficulty. We focus on the 2008 competition, which employed fundamentally redesigned evaluation frameworks that aimed to systematically encourage the submission of robust learning methods.

The rest of this article is organized as follows. Section 2 gives a brief background on reinforcement learning, Section 3 describes the challenges of empirical evaluation in RL, and Section 4 reviews the recent history of RL competitions. Section 5 introduces the evaluation frameworks developed for the 2008 competition and Section 6 describes the competition's software infrastructure. Section 7 describes the six competition domains used in 2008, Section 8 presents selected competition results from 2008, and Section 9 summarizes the 2009 competition. Finally, Section 10 discusses the implications of the competition results and Section 11 outlines ideas for the future of the competition.

2 Reinforcement Learning

The field of reinforcement learning aims to develop algorithms for solving sequential decision problems (SDPs), in which an autonomous *agent* strives to maximize a scalar reward signal by choosing actions in response to observations at a series of timesteps. At each timestep, the *environment* generates observations and rewards in response to the actions emitted by the agent. The observations are correlated with the true state of the system, which evolves over time in response to the agent's actions according to a fixed *transition function*. The agent's behavior is determined by its *policy*, π, a mapping from observations to actions. Figure 1 depicts the interaction between the agent and its environment.

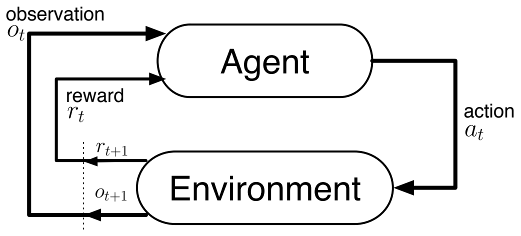

Figure 1: Interaction between an agent and an environment.

RL is distinct from traditional supervised machine learning in two fundamental ways. First, supervised learning is passive while RL is active. In supervised learning, a learning algorithm is presented with a set of observation-response pairs and its objective is to learn an appropriate mapping from observations to responses. In reinforcement learning, the agent must actively select actions in response to observations from the environment at each step. The correct answer is not explicitly provided; instead the agent receives a reward for the specific action that was selected.

Second, RL has a temporal structure that is not present in supervised learning. The agent strives to maximize the expected sum of future rewards, as opposed to greedily choosing the action with the greatest immediate reward. Which action is most desirable depends on the expected availability of rewards in the future.

3 Evaluating Reinforcement Learning Algorithms

The success and fairness of any competition depends critically on how the performance of the participants is evaluated. Developing an effective framework for evaluations for the 2008 RL Competition was challenging because we elected to focus on *online* performance, i.e., how much reward is collected as the agent explores the environment and learns how to behave. Consequently, it is not sufficient to simply measure the quality of the final policy that is produced by a given RL agent.

Focusing on online performance has the advantage of measuring an agent's ability to effectively balance exploration (trying out novel behaviors) and exploitation (using the best policy found so far), which many researchers consider a central challenge of RL. However, the online nature of our evaluation criteria introduces some challenges into the competition process.

On one hand, we want to make the competition environments available to the participants so that they can experiment with them and improve their agents before the end of the competition. On the other hand, we want those environments to be unfamiliar to the agents at the end of the competition, so we can evaluate their ability to efficiently explore them and learn online, not merely employ pre-learned policies. In other words, we need to give the participants an opportunity to train, but still save some surprises for testing.

In traditional supervised learning, this issue is typically resolved by using separate *training data* and *testing data*. The training data is a fixed supply of information that limits the learner's experience before being tested. The algorithm can later be evaluated using the testing data. This process is suitable for supervised learning because it is passive: the training and testing data are static observation-response pairs, which can be stored in a data file. The

active, temporal aspects of RL require that the SDPs be represented as interactive computer programs, called *environment programs*, instead of fixed data sets.
Previous RL competitions, which we review in the next section, have used various evaluation frameworks to address these challenges. In the subsequent section, we describe the fundamentally redesigned evaluation frameworks used in the 2008 competition, which more systematically address the difficulties described here.

4 RL Competition History

The process leading to the 2008 competition began in 2004 with a workshop at the Neural Information Processing Systems Conference (NIPS), called "Reinforcement Learning Benchmarks and Bake-Offs". The goals of this workshop included assembling a list of candidate benchmark problems, brainstorming specifications for implementing such problems, and planning a future benchmarking event. In 2005, the first benchmarking event occurred at another NIPS workshop, "Reinforcement Learning Benchmarks and Bake-offs II". This event, as well as all later events, relied on RL-Glue (White 2006;
Tanner and White 2009), a platform- and language-independent interface for connecting agent, environment, and experiment programs. The organizers selected six simple, well-known RL problems: three with continuous observations (Mountain Car, Puddle World, and Cart-Pole) and three with discrete observations (Blackjack, Sensor Network, and Taxi). The event also involved two triathlons, in which a single agent ran on all three continuous or discrete domains.
Performance was measured online, i.e., based on the cumulative reward the agent received across a fixed number of episodes. There was no separation between training and testing and no attempt to restrict the agent's prior knowledge of the domain dynamics: the agents did not have to learn during the competition.
A second benchmarking event occurred in 2006 with a workshop at ICML, called "Reinforcement Learning and Benchmarking Event". Like its predecessor, this event featured several simple, well-known problems (Mountain Car, Cart-Pole, Blackjack, and Cat-Mouse). However, it also featured a new, large-scale problem: controlling a simulated octopus arm (Yekutieli et al. 2005). This problem, which has high-dimensional continuous observation and action spaces, was designed to challenge RL researchers to demonstrate methods that scale up to more challenging settings.
Unlike the previous event, evaluation was separated into training and testing phases. Participants were allowed unlimited runs on training SDPs but were evaluated on a different, but qualitatively similar, set of test SDPs. Only a final window of the testing episodes were counted toward final scores, allowing agents to learn without penalty in early episodes. Hence, the event measured online performance after a period of free exploration with the test SDPs.
These workshops and benchmarking events led to the "First Annual Reinforcement Learning Competition", held at NIPS at the end of 2006. Unlike the previous benchmarking events, this event was a real competition with official winners who were awarded small prizes and invited to present their approaches at the associated workshop. As in previous years, the event featured well-known benchmark problems or variations thereof: three with discrete observations (Cat-Mouse, GARNET and Tetris) and three with continuous observations (Cart-pole, Nonstationary Mountain Car, and Puddle World).
The competition featured the first attempts to force participants to submit more general learning algorithms through use of randomization and nonstationary dynamics, i.e., the transition probabilities changed over time. For example, in Cat-Mouse, the algorithms were tested on several randomly generated maps. In GARNET, the transition function was randomly altered at regular intervals. In Nonstationary Mountain Car, the force of gravity was periodically changed. These domain characteristics made it less feasible for competitors to succeed using pre-learned policies in place of online learning algorithms. The competition also featured a pentathlon for which only two of the five domains were available in advance for training. This approach forced participants to submit general-purpose methods that could discover good policies in arbitrary environments.
All of these events and the evaluation frameworks they employed laid the foundation for the 2008 Reinforcement Learning Competition. The next section describes the new evaluation frameworks designed for this competition.

5 Evaluation Frameworks

As described in Section 3, the interactive nature of RL creates new challenges when designing an evaluation framework to measure the performance of learning algorithms. The first challenge is the need for a platform- and language-independent interface connecting agent and environment programs. Fortunately, this problem has already been addressed by RL-Glue (White 2006;
Tanner and White 2009), an open source software project that has been used in every benchmarking event since its first release in 2005.

The second challenge concerns the conflicting goals of providing participants with SDPs to develop and debug their learning algorithms but later evaluating the online learning performance of those same algorithms on unfamiliar SDPs. This issue was not systematically addressed in earlier events. Thus, a primary goal of the 2008 competition was to design fundamentally new evaluation frameworks that would encourage use of learning algorithms that continue to learn in the evaluation phase instead of only re-using what was learned from the training phase.

For the 2008 competition, we addressed this problem using evaluation frameworks with separate training and test phases containing different sets of SDPs. We created multiple, related SDPs for each domain and split them into training and test sets. The training SDPs were released at the beginning of the competition and participants were invited to use them for unlimited training. However, final performance in the competition was based solely on cumulative reward accrued on the test SDPs, which were hidden from the participants until the end of the competition. In this way, the need for online learning was preserved because agents were evaluated based on their interactions with environment programs whose details they had not seen before. At the same time, the opportunity to train on related problems gave participants enough prior knowledge to make learning practical. This approach was employed in two different evaluation frameworks, the *generalized framework* and the *altered framework*, which are detailed in Sections 5.1 and 5.2, respectively.

We also employed a third evaluation framework which takes a more direct approach to encouraging online learning: eliminating the training phase altogether. Since no training SDPs are released, participants must develop general-purpose learning algorithms that will learn online on previously unseen SDPs during the test phase. The resulting *unknown framework* is described in Section 5.3.

The primary goal of these frameworks was to encourage submission of robust learning methods. A secondary goal was to motivate participants to begin preparing their agents early and to continually improve those agents throughout the competition. To this end, we augmented the training and testing phases with a third, intermediate *proving phase*, inspired by a similar setup for the Netflix Prize. As with the training and testing phases, the proving phase has its own set of SDPs. However, participants are allowed to conduct runs on these SDPs only once per week. While the results do not count toward the official final scores, they are posted automatically to publicly accessible *leaderboards* on the web.

Note that in none of the phases of any of these frameworks was there any attempt to limit or even measure the computational resources used by the participants. Instead, the competition focused on *sample complexity*, i.e., how many interactions with its environment an agent needs to find a good policy. We made this choice for two reasons. First, sample complexity is more important than computational complexity in many real-world problems, since computational resources are getting continually cheaper but interacting with a real environment remains expensive and dangerous. Second, it is more practical, as measuring or controlling the computational resources used by participants would require a significantly different and more complicated software infrastructure.

The remainder of this section describes in more detail the three frameworks used in the 2008 competition. Table 1 summarizes the differences between them and lists the domains used with each framework. The domains are further described in Section 7.

Table 1: The three evaluation frameworks used in the 2008 competition.

5.1 Generalized Framework

As mentioned above, one way to encourage learning in the evaluation phase is to use different SDPs for training, proving, and testing. To do so, we need a way to generate sets of related SDPs. In the generalized framework, we formally describe the dimensions along which these SDPs can vary and define a distribution over the resulting space (Whiteson et al. 2009). The training, proving, and testing SDPs are then formed by drawing independent samples from this distribution. As a result, the SDPs in each of the three phases are *independent and identically distributed* (IID).

Formally, a generalized domain $\mathcal{G} : \Theta \mapsto [0, 1]$ is simply a probability distribution over a set of SDPs Θ. Given such a domain, we can form a training, proving, and test sets by sampling repeatedly from $\mathcal{G}$. The learner's score on each set is thus an average across multiple runs, with each run conducted on a different SDP from the set. Note that $\mathcal{G}$ is not known to the participants. However, they can try to estimate it from the SDPs in the training set. Furthermore, on each proving or testing run, the learner's interactions with the environment provide information about the specific SDP, drawn from $\mathcal{G}$, that it faces.

To excel in a generalized domain, a learner must be robust to the variation represented by $\mathcal{G}$. Except in degenerate cases, no fixed policy will perform well across many SDPs. Consequently, for strong performance, learning is required during testing. No matter how much learning is done with the training set, the agent must still learn on each SDP in the test set in order to excel. By including the variation that we deem important in $\mathcal{G}$, we can ensure that only appropriately robust learners will perform well.

5.2 Altered Framework

An advantage of the generalized framework is that, by formalizing the way that SDPs can vary, it is possible to generate independent training, proving, and test sets. However, the generalized framework is not always feasible because the competition designers must have a thorough understanding of the domain in order to choose $\mathcal{G}$ such that the resulting SDPs are interesting. Furthermore, complex environment programs may be so computationally intense that evaluating an agent on a single SDP takes days or even weeks. Therefore, it can be impractical for training, proving, and test sets to each contain multiple SDPs, as in the generalized framework.

When the generalized framework is impractical, we employ the altered framework instead. Only three SDPs are used in total: one each for training, proving, and testing. These SDPs are not formally sampled from a distribution and in principle they can be arbitrarily different. In practice, the competition designers create qualitatively similar training, proving, and test SDPs with variation in the specific details. These changes ensure that learning during evaluation will be valuable to the agent's performance.

The altered framework was inspired by the General Game Playing Competition, in which a different set of games is used for training and testing, with no formal relationship between the two. Participants cannot rely on any IID guarantees, but instead must reason about the factors that the competition designers are likely to find important and try to design appropriately robust learning agents.

Since only three SDPs need to be generated, the altered framework is more practical for the competition administrators for problems that run slowly or are not well understood. In addition, since no distribution needs to be chosen, the proving and testing SDPs do not have to be finalized before releasing the training SDP and can even be designed based on feedback about competitors' experience with the training SDP. Furthermore, the altered framework is useful for domains where, for technical reasons, it is not practical to hide the source code from the competitors. If the generalized framework was used in such cases, competitors could easily deduce $\mathcal{G}$.

5.3 Unknown Framework

The unknown framework eliminates any promise of similarity between the training and testing SDPs, with the goal of encouraging the development of truly general-purpose agents that can learn with little prior knowledge about the domains that they face. Therefore, the training SDPs in the unknown framework serve simply as a technical check so that participants can ensure their algorithms will not crash during testing. This directly contrasts with both the generalized and altered frameworks, which encourage participants to leverage knowledge gained in the training phase to improve performance in the test phase.

In the proving and test phases, the agents face a suite of SDPs that can be arbitrarily different from each other. Unlike in the generalized framework, these SDPs are not drawn from any fixed distribution. Unlike in the altered framework, there is no assumption that the various SDPs will be qualitatively similar: participants should aggressively test the generality of their agents. Since it is easy to devise SDPs that are significantly different from each other, this is a challenging framework for participants.

To ensure it remained feasible, all SDPs in the unknown event of the 2008 competition had observation and action spaces of known, fixed dimensionality. These constraints made it practical to program one agent that could learn on all SDPs. Nonetheless, the transition and reward dynamics of each domain varied widely, requiring agents to be robust in order to perform well in testing.

6 Software Infrastructure and Logistics

Various methods of distributing software and conducting test runs have been used in previous events, ranging from distributing source code and letting participants submit data files with results, to giving participants accounts on a server and requiring them to log in to conduct test runs. For the 2008 competition we developed a new infrastructure designed to meet several criteria.

The most important criterion is secrecy. The evaluation frameworks described in Section 5 necessitate a software design in which it is possible to keep some information, e.g., the nature of the SDPs in the testing set, hidden from participants. Otherwise, there is no way to ensure that agents are actually learning during the test phase of the competition. It is also important that participants cannot easily modify the software to give themselves undeserved advantages. The software infrastructure should also permit large experiments to run within a reasonable time. For example, an experiment of tens of millions of steps should be feasible to complete within 24 hours. Furthermore, the infrastructure should be minimally restrictive in terms of what programming languages or software libraries can be used, and should scale well with the number of competitors so that the competition organizers do not need to buy new computers or hire additional staff to manage the infrastructure.

For the 2008 competition, we elected to allow participants to download the competition software and run it on their own machines. The software was designed to automatically report the results of proving and test runs to a central competition server, which automatically updates the leaderboards and a database of test results. This approach gives participants maximal flexibility when developing their agents, as they are free to use whatever hardware or software libraries they have available. It also simplifies administration of the competition, as no extra machines or user accounts need to be managed.

We distributed the software in the form of compiled Java classes because they are portable to almost every platform and can be bundled into compressed packages called JAR files. Because the software uses RL-Glue, participants could still create their agents in the programming language of their choice. Java also lets us leverage RL-Viz[2], an extension to RL-Glue that adds visualization and dynamic configuration options. There were two main concerns about distributing Java classes. First, not all competition domains were previously available in Java. We addressed this problem by translating as many of them to Java as possible. Second, malicious participants might be able to de-compile the classes and thus examine or alter the source code. Therefore, we ran a source code obfuscator over the JAR files before distributing them. We also considered digitally signing the JAR files so we would know if they had been altered but eventually decided it was unnecessary, though that may change in future competitions. To add extra security, our software deferred downloading the proving and testing SDPs until they were required. The files were downloaded, used, and deleted by our automated testing software without every being seen by the participants.

7 2008 Competition Domains

The 2008 competition featured six domains with various characteristics and overall difficulty. Each domain has at least one characteristic that is considered challenging for current RL methods. These challenges include continuous and/or high-dimensional observations, large numbers of available actions, dangerous actions, partial observability, and noisy observations and actions. In this section, we briefly describe each domain and its salient characteristics.

7.1 Mountain Car

In the Mountain Car problem (Boyan and Moore 1995;
Sutton 1996;
White 2006), depicted in Figure 2, the agent must drive an underpowered car out of a valley. Because the car's engine is not powerful enough to drive straight out of the valley, it must instead drive back and forth to gain enough momentum to drive down one side of the valley and out the other side. The agent strives to reach this goal in as few steps as possible, using only three actions: forward, backward and neutral.

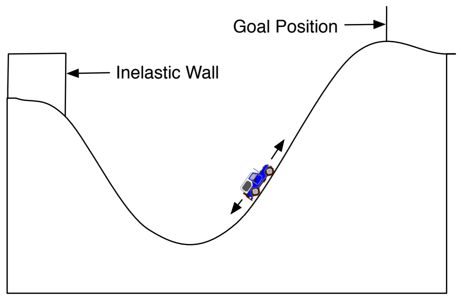

Figure 2: The Mountain Car domain, in which an underpowered car must drive out of a valley.

The Mountain Car domain is one of the most frequently used RL testbeds because it has several interesting aspects that are still relevant today. Mountain Car is a classic example of a delayed reward problem: the agent receives a constant negative reward on each time step. This reward scheme makes it difficult for the agent to accurately assign credit to individual actions over a long trajectory. The reward scheme also produces an interesting exploration problem. Because all actions provide the same reward, the agent cannot learn to improve its policy until the car reaches the top of the hill for the first time. Random exploration strategies require a significant amount time to reach the goal for the first time because the base of the hill is an attractor. A long sequence of mostly optimal actions is necessary to reach the top of the hill.
The Mountain Car domain was generalized by perturbing the observations and the outcomes of actions. The Mountain Car SDPs featured scaled, translated and noisy observations and stochastic actions.

7.2 Tetris

The Tetris domain (Bertsekas and Tsitsiklis 1996;
Demaine, Hohenberger, and Liben-Nowell 2003;
Szita and Lörincz 2006), depicted in Figure 3, is based on the popular falling-blocks puzzle game. At each step a piece falls one grid position. The agent may change the orientation or position of the piece with six actions: rotate left, rotate right, move left, move right, fall to bottom, and drop one space (no action). The board is represented as a bit map indicating which squares on the wall are occupied. The agent receives a positive reward for each line eliminated, with exponentially more reward for multiple lines removed at a time. There is no penalty for filling the screen (ending an episode).
One of the most interesting aspects of Tetris, as a research problem, is the detail of the observation data. Humans can look at a Tetris image and quickly abstract high-level information about the game needed to strategize and plan. Tetris has an enormous number of possible board configurations, so some abstraction over the board image is required. Most successful agents use hand-designed, expert game features or function approximation.
The Tetris domain was generalized by parameterizing the width and height of the board and the probability distribution used to generate new blocks. The reward function was also parameterized: SDPs provided different bonus points for eliminating multiple rows at a time.

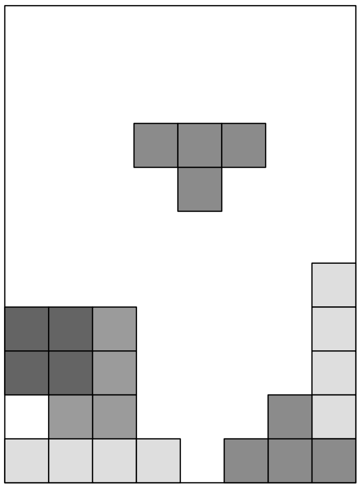

Figure 3: The Tetris domain, based on the popular falling blocks puzzle.

7.3 Helicopter Hovering

In the Helicopter Hovering domain (Bagnell and Schneider 2001;
Ng et al. 2004), the agent must hover a simulated helicopter in 3-dimensional space for a fixed number of steps. The helicopter is controlled via four continuous actions representing the pitch of the main and tail rotors. The reward is a direct function of the stability of the helicopter's hovering regime. A crash, however, incurs a large negative reward. The helicopter simulator is based on data collected during actual flights of an XCell Tempest helicopter, shown in Figure 4.

Figure 4: An autonomous XCell Tempest helicopter from Stanford University. The simulator used in the Helicopter Hovering domain is based on real data captured from such helicopters.

Helicopter Hovering is challenging for several reasons. First, the transition dynamics are extremely complex. Second, both the observation and action spaces are continuous and high-dimensional. The latter is particularly problematic as many traditional RL algorithms, e.g., *value function based methods* (Sutton 1988), require enumerating the action space in order to select a maximizing action. Third, the domain involves high risk, as bad policies can crash the helicopter, incurring catastrophic negative reward. As a result, exploration must be conducted with extreme care. The Helicopter Hovering domain was generalized by parameterizing the wind velocity and adding noise to the helicopter model simulation.

7.4 Keepaway

Keepaway (Stone et al. 2005;
Stone, Sutton, and Kuhlmann 2005) is a simulated robot soccer domain built on the RoboCup Soccer Server (Noda et al. 1998), an open source software platform that has served as the basis of multiple international competitions and research challenges. The server simulates a full 11 vs. 11 soccer game, complete with noisy sensors and actuators. In the keepaway sub-game, depicted in Figure 5, three *keepers* try to retain possession of the ball while two *takers* try to get the ball. A keeper can choose to either pass the ball to another keeper or hold the ball. The game ends if a taker intercepts the ball, touches the keeper with the ball, or if the ball goes out of bounds. The takers follow a simple policy: go directly towards the keeper that currently holds the ball. The learning agent controls only one of the three keepers, while the others follow a hand-coded policy.

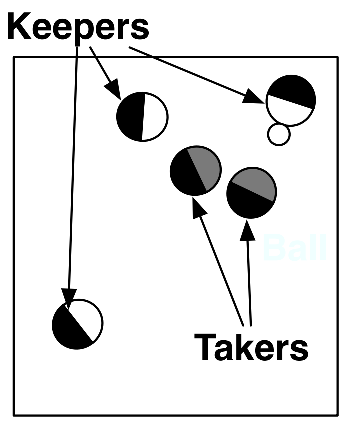

Figure 5: The Keepaway domain, in which three keepers try to keep a ball within a square boundary and away from two takers for as long as possible.

Keepaway soccer presents many challenges to RL methods, including high-dimensional observations, noisy observations and actions, and variable delays in the effects of actions.
The keepaway problem was altered by changing the players' movement speed, the size of the field and adjusting the stochasticity of actions. Compared to the proving version of keepaway, the testing problem featured faster movement, a smaller field and less noise.

7.5 Real-Time Strategy

The Real-Time Strategy (RTS) domain, shown in Figure 6, is a two-player game where each player has a single construction base and two types of units: fighters and workers. The workers harvest and transport resources that are needed for building. The fighters defend the workers and the base, and can attack the opponent's units. Each player can spend harvested resources to create new units at the construction base. The players are restricted by a "fog of war", wherein they can only observe parts of the map that are occupied by their units. The learning agent controls one player and aims to destroy the base of the other player, who is controlled by a fixed policy.
The dynamic size of the observation and action spaces in RTS poses a new challenge to the RL community. The amount of information available to the agent depends on the number and placement of the units. For example, if the agent loses units in combat, it will have fewer actions available to it and less information on which to base its decisions. RTS is interesting because participants can employ learning at various levels of abstraction. One could use learning to determine the best way to search for mineral patches on a variety of game boards or to prefer certain high-level game strategies based on the opponent's behavior.
The RTS domain was altered in several ways for the testing phase of competition. The size and armor of the construction base was increased. The attack strength, armor, walk speed, and cost of fighters was increased. The

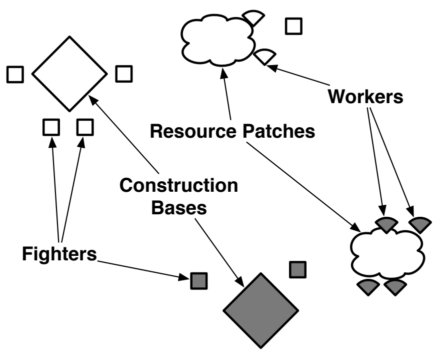

Figure 6: The Real-Time Strategy domain, in which two players compete for resources and survival.

attack strength, walk speed, and carrying capacity of workers was also increased. Finally, the opponent was made more aggressive: it initially attacks the agent with all available units and then resorts to a more defensive strategy if the initial assault fails.

7.6 Polyathlon

In the Polyathlon domain, the agent faces a series of SDPs with continuous observations and discrete actions. The SDPs are each variations and embellishments on well-known environments from the RL literature. The winner of the Polyathlon is the agent that ranks best across all problems, relative to the other entrants.

The Polyathlon is an interesting domain because it challenges competitors to develop general-purpose learning algorithms that do not significantly depend on prior knowledge. Any particular problem in the Polyathlon could feature large observation and action spaces, noisy dynamics or dangerous exploration. The Polyathlon is meant to encourage interest in making current learning methods more robust in practical settings and developing general learning algorithms that make fewer assumptions about the dynamics and structure of the world.

The Polyathlon was the only domain in the unknown framework. The set of testing problems included two *multi-armed bandit problems*(simple and associative) (Bellman 1956), two 2-dimensional navigation domains (continuous gridworld), three Cat-Mouse domains, Mountain Car, Pole Swing Up and the Acrobot problem (DeJong and Spong 1994).

8 2008 Selected Results

Twenty-one different teams participated in the 2008 competition. The competition ended with a workshop at the International Conference on Machine Learning (ICML) in Helsinki, Finland, at which the results were announced. In this section, we highlight the results in two of the competition domains: Helicopter Hovering and Tetris.

8.1 Helicopter Hovering

Helicopter Hovering was selected as a domain for the 2008 competition primarily because the action space is high-dimensional and continuous, characteristics known to be challenging to many RL methods. However, competition results indicate that participants primarily focused on a different aspect of the problem: the difficulty of exploring safely. In Helicopter Hovering, the agent receives small negative rewards at each timestep based on how much it deviates from a target equilibrium position. In addition, if the helicopter deviates from this equilibrium too much, it crashes and receives a very large negative reward. Consequently, performing well in testing depends critically on avoiding crashing at all costs, as a single crash could devastate the total score of an otherwise excellent agent.

Because the domain was generalized, some exploration was necessary, since each test SDP was different from those seen in training. However, it was critical to explore conservatively to minimize the risk of testing a policy that might crash. In practice, this requires beginning with policies that have proven robust on all the training SDPs.

Six teams successfully completed test runs in Helicopter Hovering but only two of these teams managed to avoid ever crashing the helicopter. The winning agent was submitted by Rogier Koppejan from the University of Amsterdam. Another agent, submitted by Jose Antonio Martin H. of the Complutense University of Madrid, achieved almost identical performance to Koppejan's agent on most SDPs. However, three episodes resulted in crashes, relegating this agent to a fourth place finish.

Koppejan's agent (Koppejan and Whiteson 2009) used evolutionary computation (Goldberg 1989) to train a multi-layer feed-forward neural network controller for the helicopter. Before the competition, Koppejan evolved separate specialized controllers for each training SDP. He then tested each of these controllers on all the training SDPs to ensure they were robust enough to avoid crashing. During each test run, the agent spent initial episodes trying out each of these specialized controllers. Whichever controller performed best in that test SDP was then used for the remainder of the run.

8.2 Tetris

Tetris was the most popular and most competitive event during the competition, with thirteen teams active during the proving phase and eight teams completing testing runs. Two aspects of the domain proved the most challenging in practice for the participants.

The first challenging aspect is the difference in reward structure for the various test SDPs. Some SDPs gave much larger bonuses than others for eliminating multiple rows at a time. Eliminating multiple rows at a time introduces greater risk because the agent must first build up several uncompleted rows. Identifying this trade-off in each test SDP and customizing the agent's policy accordingly proved challenging for the participants.

The second challenging aspect is the way the agent's observations are represented: as bit vectors corresponding to spaces on the board and the currently falling piece. This representation is quite detailed and thus difficult to use as a basis for learning. Participants found that it was necessary to either manually extract higher-level features or learn a more complex nonlinear function capable of abstracting information from the bit vector.

The winner of the Tetris domain was a team consisting of Bruno Scherrer, Christophe Thiery, and Amine Boumaza of the French National Institute for Research in Computer Science and Control (INRIA). They used an off-line method to optimize the weights of a linear controller based on logged game data. They started with the same higher-level features as those developed by Colin Fahey, an amateur AI researcher who has studied Tetris extensively.[3] They also added three new features encoding the depth of each hole in the wall, the number of rows with holes, and a diversity feature that is activated if the columns have very different heights. Their agent was trained in an iterative fashion, alternating between playing games to gather data and optimizing off-line using the collected data. Data was collected from the training and proving SDPs and a simulator learned from data.

9 The 2009 Reinforcement Learning Competition

Due to the success of the 2008 competition, the community decided to organize another competition for 2009. Since RL-Glue and the competition software infrastructure ran smoothly in 2008, these were retained in 2009 with only minor updates. Furthermore, because the generalized, altered, and unknown evaluation frameworks were well received by 2008 participants, these were also retained.

Three domains from the 2008 competition were featured: Helicopter Hovering, Tetris, and Polyathlon. However, the organizers changes the way in which the specific SDPs in each domain were varied. Helicopter Hovering added new wind patterns. Tetris added an adversarial component in which the environment chooses the pieces that are expected to minimize the agent's score. Polyathlon added two new underlying problem types: Cart-Pole and continuous Cat-Mouse.

In addition, three new domains were introduced: the classic Acrobot problem (DeJong and Spong 1994); Octopus Arm (Yekutieli et al. 2005), which earlier appeared in the 2006 benchmarking event; and Infinite Mario, a new domain developed by Markus Persson[4] based on the classic video game. Acrobot replaced Mountain Car, and served as an introductory problem that was feasible even for novices in the RL community. Octopus Arm posed a new challenge because it has large, multi-dimensional continuous observation (82 dimensions) and action (32 dimension) spaces. Infinite Mario, like the Real-Time Strategy domain from 2008, has an object-oriented observation space (Diuk, Cohen, and Littman 2008). The number of observations and their structure changes as new elements (enemies, platforms, pits, etc.) are encountered by the agent. Acrobot and Infinite Mario were generalized domains, while Octopus Arm was an altered domain.

The competition results demonstrated that changing the way SDPs are varied within a domain is enough to create significant new challenges. For example, a model-based approach that excelled in the 2008 Helicopter Hovering domain (Koppejan and Whiteson 2009) proved unreliable in 2009, since the new wind patterns violated critical assumptions in the model. However, the results also illustrated how it can be to design appropriate changes to the SDPs. For example, in the Octopus Arm, the location of the goal was changed in the testing SDP in order to create additional uncertainty. However, because the new goal was closer to the agent's initial position, the winning agent, by systematically exploring the space, was able to reach the goal more quickly.

10 Discussion and Conclusions

The 2008 and 2009 competitions were dramatically different from previous events in terms of evaluation frameworks, domain difficulty, and software infrastructure. In this section, we weigh the effects, both positive and negative, of these changes, based on results of the competition itself and feedback given by participants at the concluding workshop.

Overall, the new software infrastructure was well received. The 2008 and 2009 competitions were the first events in which competitors could graphically visualize their agents' performance, track the performance of other teams on leaderboards, discuss competition details with organizers and other competitors via online forums, and benchmark their agents from their own computers via the internet. Previous competitions used a manual approach to team registration, problem distribution and benchmarking.

The new evaluation frameworks also generated positive feedback. In particular, the weekly proving runs made it manageable to deal with the challenges of generalized, altered, and unknown domains. Many teams made significant performance improvements during the proving phase. At the same time, basing performance only on the final test runs allowed last minute entrants to remain competitive. However, some participants disliked the proving runs and requested the option to keep proving results private in future competitions. The primary concern was competitive: participants did not want other teams knowing how strong their agent was. The team that won the Tetris event in 2008 deliberately put their agent into “suicide mode” near the end of each proving run to mislead other participants about the quality of their agent.

Tetris was the most popular domain of the 2008 RL Competition. Several of the agents submitted for testing were capable of eliminating tens of millions of rows per game; these agents may surpass human performance in Tetris, especially considering the additional challenges introduced in the generalized version of the problem. However, the Tetris results also demonstrate how competitive spirit can conflict with the goals of the competition and the field. For example, despite generalization of the domain, none of the top three agents used online learning and the winning entry relied on multiple heuristics that are unlikely to be generally useful for learning. This outcome suggests that Tetris should be more broadly generalized in future competitions. It also underscores the difficulty of selecting an appropriate generalization when setting up a competition.

The Polyathlon was the most popular domain in the 2009 RL competition. The increased participation demonstrates that many of the 2009 competitors were interested in online learning approaches and evaluating the practicality of algorithms from the literature. The 2008 Tetris participants, on the other hand used a combination of simulation, batch algorithms and game heuristics to achieve impressive performance. The shift in interest toward the Polyathlon, where pre-processing data from the proving SDPs is ineffective, suggests that the competitions have promoted the development of practical and robust methods.

The 2008 competition generated substantially more interest than in previous years, with over 200 teams registering and downloading the competition software. However, only 21 teams completed test runs. Similar participation occurred in 2009. In the future, we hope to find ways to increase the fraction of interested teams that ultimately compete. Nonetheless, both the 2008 and 2009 competitions had higher levels of participation than previous events.

In addition, the time and effort invested in developing the competitions will benefit the RL community for some time to come. All of the source code is available as an open source project[5], and even now undergraduate and graduate classes in artificial intelligence and machine learning are reusing these events for teaching tools and class research projects. Given this trend, and the improvements in software infrastructure and evaluation frameworks, we believe the competitions have been a useful step in the progress of empirical RL.

11 Future Competitions

There are many ways in which the competition could be improved or expanded in the future. As the software infrastructure evolves, new challenges can be addressed by the RL community. Since the start of the 2008 competition, several research groups have released the source code for new agents to the RL community. These agents may be the basis for the winning submission to a future competition. RL-Glue, the software on which the competition is based, has also grown to support new languages including Lisp and Matlab, making future competitions even more accessible to the community.

Other changes might involve different evaluation frameworks. For example, Nouri et al. recently proposed an empirical evaluation methodology that focuses on *policy evaluation*, an important subproblem of reinforcement learning (Nouri et al. 2009). Rather than learning online, the agent receives a data set gathered under some policy and is evaluated based on its ability to estimate that policy's value function, i.e., the expected cumulative reward it will accrue from each state. While this approach does not provide a way to evaluate full reinforcement-learning agents, it is easy to employ. It avoids the interactive complications of reinforcement learning and allows evaluations to be based on fixed data sets. Consequently, data need not be tied to a simulator but can be drawn from a real-world setting, such as physical robots, or even contributed by an industrial sponsor.

Future competitions could also include new domains. Since one goal of the competition is to encourage researchers to develop more practical methods, one obvious choice would be to select increasingly difficult domains each year. The domains themselves could be made more difficult or, alternatively, the parameter space of the generalized domains could be broadened. Doing so would require future participants to devise methods that are robust with respect to more and more dimensions of variability. Increasing domain difficulty in this way could encourage researchers to develop robust methods in a bottom-up fashion: by starting with very specific domains and gradually broadening them as much as possible.

The competition could also select a challenge problem, one clearly too difficult for current methods, in the hopes of galvanizing the community into making a big leap forward. This approach has been successful for the DARPA Grand Challenge. One way to find such a problem would be in industry. Businesses currently face numerous challenges that could be addressed using RL if sufficiently robust methods could be found. Drawing on industry for a challenge problem would ensure that the resulting methods have real-world relevance and would also attract sponsors and publicity to the event.

All of these possible extensions must be balanced against the need to keep the competition simple and comprehensible to outsiders and potential new participants. Keeping a low barrier to entry is essential for maintaining momentum, increasing participation, and further establishing the competition's importance within the community.

Acknowledgments

We wish to thank the sponsors of the 2008 and 2009 reinforcement-learning competitions for their generous support, which made possible the prizes awarded to winners and travel grants enabling students to attend the workshops. The 2008 sponsors were: the Alberta Ingenuity Centre for Machine Learning, the Toyota Research Institute of North America, and the Reinforcement Learning and Artificial Intelligence Lab at the University of Alberta. The 2009 sponsors were: the Alberta Ingenuity Centre for Machine Learning, the Reinforcement Learning and Artificial Intelligence Lab at the University of Alberta, International Business Machines, Intel, MathWorks, the University of Southern California, and the Institute for Creative Technologies.

We also wish to thank David Wingate and Carlos Diuk, the chairs of the organizing and technical committees for the 2009 competition, for their tireless efforts. Finally, we wish to thank all the members of the organizing and technical

committees as well as all the participants for helping make the competitions successful.

Notes

1. http://rl-competition.org/
2. http://rl-viz.googlecode.com/
3. http://www.colinfahey.com/tetris/tetris_en.html
4. http://www.mojang.com/notch/mario/
5. http://rl-competition.googlecode.com/

References

Bagnell, J., and Schneider, J. 2001. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the International Conference on Robotics and Automation 2001*, 1615–1620. IEEE.

Bellman, R. E. 1956. A problem in the sequential design of experiments. *Sankhya* 16:221–229.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neural Dynamic Programming*. Belmont, MA: Athena Scientific.

Boyan, J. A., and Moore, A. W. 1995. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, 369–376.

DeJong, G., and Spong, M. W. 1994. Swinging up the acrobot: An example of intelligent control. In *Proceedings of the American Control Conference*, 2158–2162.

Demaine, D. E.; Hohenberger, S.; and Liben-Nowell, D. 2003. Tetris is hard, even to approximate. In *Proceedings of the Ninth International Computing and Combinatorics Conference*, 351–363.

Diuk, C.; Cohen, A.; and Littman, M. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine learning*, 240–247.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

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

Koppejan, R., and Whiteson, S. 2009. Neuroevolutionary reinforcement learning for generalized helicopter control. In *GECCO 2009: Proceedings of the Genetic and Evolutionary Computation Conference*, 145–152.

Ng, A. Y.; Coates, A.; Diel, M.; Ganapathi, V.; Schulte, J.; Tse, B.; Berger, E.; and Liang, E. 2004. Inverted autonomous helicopter flight via reinforcement learning. In *Proceedings of the International Symposium on Experimental Robotics*, 363–372.

Noda, I.; Matsubara, H.; Hiraki, K.; and Frank, I. 1998. Soccer server: a tool for research on multiagent systems. *Applied Artificial Intelligence* 12:233–250.

Nouri, A.; Littman, M. L.; Li, L.; Parr, R.; Painter-Wakeeld, C.; and Taylor, G. 2009. A novel benchmark methodology and data repository for real-life reinforcement learning. In *Proceedings of the 26th International Conference on Machine Learning*. Poster.

Stone, P.; Kuhlmann, G.; Taylor, M. E.; and Liu, Y. 2005. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020. Berlin: Springer Verlag. 93–105.

Stone, P.; Sutton, R. S.; and Kuhlmann, G. 2005. Reinforcement learning in robocup-soccer keepaway. *Adaptive Behavior* 13(3):165–188.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, Massachussets: MIT Press.

Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. *Machine Learning* 3:9–44.

Sutton, R. S. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, 1038–1044.

Szita, I., and Lörincz, A. 2006. Learning tetris using the noisy cross-entropy method. *Neural Computation* 18(12):2936–2941.

Tanner, B., and White, A. 2009. RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research* 10:2133–2136.

White, A. 2006. A standard benchmarking system for reinforcement learning. Master's thesis, University of Alberta.

Whiteson, S.; Tanner, B.; Taylor, M. E.; and Stone, P. 2009. Generalized domains for empirical evaluations in reinforcement learning. In *ICML 2009: Proceedings of the Twenty-Sixth International Conference on Machine Learning: Workshop on Evaluation Methods for Machine Learning*.

Yekutieli, Y.; Sagiv-Zohar, R.; Aharonov, R.; Engel, Y.; Hochner, B.; and Flash, T. 2005. A Dynamic Model of the Octopus Arm. I. Biomechanics of the Octopus Reaching Movement. *Journal of Neurophysiology* 94(2):1443–1458.

Autobiographical Statements

Shimon Whiteson

Shimon Whiteson is an assistant professor at the Informatics Institute at the University of Amsterdam. His research focuses on single- and multi-agent decision-theoretic planning and learning, especially reinforcement learning. He served as the chair of the organizing committee for the 2008 Reinforcement Learning Competition.

Brian Tanner

Brian Tanner is a provisional Ph.D candidate at the University of Alberta. His research focuses on empirical evaluation and comparison in artificial intelligence. He served as the chair of the technical committee for the 2008 Reinforcement Learning Competition and the senior technical advisor in the 2009 competition.

Adam White

Adam White is a provisional Ph.D candidate at the University of Alberta. His research focuses on human-computer interaction using reinforcement learning. He served as the chair of the organizing committee for the 2006 Reinforcement Learning Competition, a member of the organizing committee for the 2008 competition, and a member of the technical committee for the 2009 competition.