In *Machine Learning*,
84(1-2):205-247, 2011.

**Machine Learning manuscript No.**
(will be inserted by the editor)

# Characterizing Reinforcement Learning Methods through Parameterized Learning Problems

**Shivaram Kalyanakrishnan · Peter Stone**

Received: date / Accepted: date

**Abstract** The field of reinforcement learning (RL) has been energized in the past few decades by elegant theoretical results indicating under what conditions, and how quickly, certain algorithms are guaranteed to converge to *optimal* policies. However, in practical problems, these conditions are seldom met. When we cannot achieve optimality, the performance of RL algorithms must be measured empirically. Consequently, in order to meaningfully differentiate learning *methods*, it becomes necessary to characterize their performance on different *problems*, taking into account factors such as state estimation, exploration, function approximation, and constraints on computation and memory. To this end, we propose *parameterized learning problems*, in which such factors can be controlled systematically and their effects on learning methods characterized through targeted studies. Apart from providing very precise control of the parameters that affect learning, our parameterized learning problems enable benchmarking against optimal behavior; their relatively small sizes facilitate extensive experimentation.

Based on a survey of existing RL applications, in this article, we focus our attention on two predominant, "first order" factors: partial observability and function approximation. We design an appropriate parameterized learning problem, through which we compare two qualitatively distinct classes of algorithms: on-line value function-based methods and policy search methods. Empirical comparisons among various methods within each of these classes project Sarsa($\lambda$) and Q-learning($\lambda$) as winners among the former, and CMA-ES as the winner in the latter. Comparing Sarsa($\lambda$) and CMA-ES further on relevant problem instances, our study highlights regions of the problem space favoring their contrasting approaches. Short run-times for our experiments allow for an extensive search procedure that provides additional insights on relationships between method-specific parameters — such as eligibility traces, initial weights, and population sizes — and problem instances.

Shivaram Kalyanakrishnan
E-mail: shivaram@cs.utexas.edu

Peter Stone
E-mail: pstone@cs.utexas.edu

Department of Computer Science, The University of Texas at Austin
1616 Guadalupe St Suite 2.408 Austin Texas 78701 USA

**Keywords** Reinforcement learning · Empirical evaluation · Partial observability · Function approximation · Parameterized learning problem

## 1 Introduction

Sequential decision making from experience, or reinforcement learning (RL), is a well-suited paradigm for agents seeking to optimize long-term gains as they carry out sensing, decision and action in an unknown environment. RL tasks are commonly formulated as Markov Decision Problems (MDPs). The solution of MDPs has benefited immensely from a strong theoretical framework that has been developed over the years. The cornerstone of this framework is the *value function* of the MDP (Bellman, 1957), which encapsulates the long-term utilities of decisions. Control policies can be suitably derived from value functions; indeed several algorithms provably converge to optimal policies in finite MDPs (Watkins and Dayan, 1992; Singh *et al.*, 2000). Also, near-optimal behavior can be achieved after collecting a number of samples that is polynomial in the size of the state space ($|S|$) and the number of actions ($|A|$) (Kearns and Singh, 2002; Brafman and Tennenholtz, 2003; Strehl and Littman, 2005), using a memory bounded in size by $O(|S||A|)$ (Strehl *et al.*, 2006).

Unfortunately a large section of the RL tasks we face in the real world cannot be modeled and solved exactly as finite MDPs. Not only are the traditional objectives of convergence and optimality thereby inapplicable to a predominant number of tasks occurring in practice, in many of these tasks we cannot even ascertain the best performance that can be achieved, or how much training is necessary to achieve given levels of performance. The objective of learning in such practical tasks, which fall beyond the reach of current theoretical modeling, has to be rescaled to realizing policies with "high" expected long-term reward in a "sample efficient" manner, as determined *empirically*.

In a formal sense, the "No Free Lunch" theorems of Wolpert and Macready (1997) establish that for any optimization algorithm, an elevated performance in one class of problems is offset by worse performance in some other class. Even so, the enterprise of machine learning rests on the assumption that classes of problems encountered in practice tend to possess regularities, which can be actively characterized and exploited. Consequently, to the extent that the relationships between *problem* instances and the performance properties of *algorithms* are unclear, it becomes a worthwhile pursuit to uncover them. The need for such research has long been advocated: in an early editorial in this journal, Langley (1988, see p.7) writes:

> "For instance, one might find that learning method A performs better than method B in one environment, whereas B fares better than A in another. Alternatively, one might find interactions between two components of a learning method or two domain characteristics. We believe the most unexpected and interesting empirical results in machine learning will take this form."

The practice of supervised learning has benefitted from a number of empirical studies that seek to identify the strengths and weaknesses of learning methods. For example, Caruana and Niculescu-Mizil (2006) undertake a detailed comparison involving a number of supervised learning methods, test problems, and evaluation metrics. Caruana *et al.* (2008) present empirical results demonstrating that random forests (Breiman,

2001) are typically more effective than several other classification methods on problems with high dimensionality (greater than 4000). Although the canonical boosting algorithm (Freund and Schapire, 1996) enjoys desirable theoretical properties and is predominantly effective in practice, studies comparing it with other ensemble schemes such as bagging (Quinlan, 1996; Bauer and Kohavi, 1999) hint at its vulnerability in the presence of noisy training data. Banko and Brill (2001) advance the case that for problems with very large data sets (for example, natural language applications on the Internet), simple classifiers such as Winnow (Littlestone, 1987) can be the most effective, and that voting-based ensemble schemes do not retain their attractiveness.

By associating problem characteristics with the strengths and weaknesses of supervised learning methods, the studies listed above provide useful "rules of thumb" to a practitioner who must choose a method to apply to a problem. Unfortunately the complex scope of the RL problem leaves the practitioner of RL with few such guidelines. Faced with a sequential decision making problem, not only does a designer need to pick a learning algorithm; he/she has to address the related issues of state estimation, exploration, and function approximation, while possibly satisfying computational and memory constraints. The broad motivation for this article is the eventual development of a "field guide" for the practice of RL, which would both inform the choices made by designers of RL solutions, and identify promising directions for future research.

Ultimately, a field guide would be evaluated based on the extent to which it can expedite the process of designing solutions for full-scale deployed applications. However, such applications are themselves too complex and constrained to provide reliable data from which the principles for a field guide can be inferred. Rather, there is a need for simpler, more transparent problems through which we, as designers, can systematically sort through the complex space of interactions between RL problems and solution strategies. This article joins a growing line of research in this direction (Moriarty *et al.*, 1999; Gomez *et al.*, 2008; Heidrich-Meisner and Igel, 2008a; Whiteson *et al.*, 2010).

The primary thrust of existing work on the subject has been in comparing RL algorithms on standard, benchmarking tasks, with possibly a small number of variations. By contrast, we design a synthetic, *parameterized learning problem*[1] with the explicit purpose of ascertaining the "working regions" of learning algorithms in a space that is carefully engineered to span the dimensions of the task and the learning architecture. The approach we propose enjoys the following merits:

1. The designed task and learning framework are easy to understand and can be controlled precisely.
2. We may examine the effect of subsets of problem parameters while keeping others fixed.
3. We can benchmark learned policies against optimal behavior.
4. The learning process can be executed in a relatively short duration of time, thereby facilitating extensive experimentation.

While the careful design of a synthetic learning problem allows us these liberties, equally it qualifies the extent to which our conclusions may generalize in practice. Thus, the results from our study are to be taken as starting points for further empirical investigation, rather than treated as well-grounded final products in themselves. In this

[1] The term "parameterized learning problem" is quite generic; such problems have been used in the past both in RL and in other fields. For some examples, see our discussion of related work in Section 6. By applying the term here to describe our framework, we aim to underscore that problem parameters are its very crux; they are not secondary as in related work.

sense, the methodology we put forth enjoys a *complementary* relationship with the research strategy of evaluating RL methods on more realistic problems. We proceed to demarcate the scope of our study.

## 1.1 Scope of Study

In order to develop a field guide for solving realistic RL problems, it is first necessary to characterize such problems along the dimensions that distinguish them from well-understood cases such as table-based learning in finite MDPs. Towards this purpose, we undertake a survey of literature describing practical applications of RL. While surveying material from relevant journals and conference proceedings, we apply the criterion that the task application, rather than the learning method employed, be the primary focus of the publication. Based on our findings, we focus our attention on two predominant, "first order" factors that characterize a sizeable fraction of sequential decision making problems in practice: (a) **partial observability**, which arises due to an agent's inability to identify the system state, and (b) **function approximation**, which is necessary for learning in large or continuous state spaces. The ubiquity of these factors in applications is apparent from Table 1, which summarizes our survey.[2]

A majority of the applications listed in Table 1 have to contend with partial observability of state. In complex systems such as stock markets (Nevmyvaka *et al.*, 2006), computer networks (Tesauro *et al.*, 2007), and cellular tissue (Guez *et al.*, 2008), available measurements seldom suffice to capture all the information that can affect decision making. Nearly every agent embedded in the real world (Kwok and Fox, 2004; Ng *et al.*, 2004; Lee *et al.*, 2006) receives noisy sensory information. The inadequacy of the sensory signal in identifying the underlying system state hinders the assumption of a Markovian interaction between the agent and the environment, on which the theoretical guarantees associated with many learning methods rely. Whereas coping with partial observability in a systematic manner is a well-studied problem, it is yet to scale to complex tasks with large, high-dimensional, continuous state spaces (Chrisman, 1992; Cassandra *et al.*, 1994; Bakker *et al.*, 2003; Pineau *et al.*, 2006).

Of the 25 applications listed in Table 1, 15 involve continuous state spaces, which necessitate the use of function approximation in order to generalize. Indeed among the ten applications that have discrete state spaces, too, seven use some form of function approximation to represent the learned policy, as their state spaces are too large for enumeration, and possibly even infinite. The use of function approximation negates the theoretical guarantees of achieving optimal behavior. Often the function approximation scheme used is not capable of representing an optimal policy for a task; even when it is, seldom can it be proven that a learning algorithm will discover such a policy. Although there exist convergence guarantees for certain algorithms that use linear function approximation schemes (Konda and Tsitsiklis, 2003; Perkins and Precup, 2003; Maei *et al.*, 2010), they do not provide effective lower bounds for the values of the learned policies. Further, convergence results rarely extend to situations in which non-linear representations such as neural networks are used to approximate the value

[2] Other independently-compiled surveys of sequential decision making applications corroborate the observations we draw based on Table 1. Langley and Pendrith (1998) describe several RL applications presented at a symposium organized around the topic; Szepesvári lists numerous applications from the control and approximate dynamic programming literature at this URL: `http://www.ualberta.ca/~szepesva/RESEARCH/RLApplications.html`.

**Table 1** Characterization of some popular applications of reinforcement learning. "Policy Representation" describes the underlying representation from which the policy is derived. A "neural network" representation is non-linear, incorporating at least one hidden layer of units. Under tile coding, the number of "features" indicates the number of state variables, rather than the number of individual tiles.

| Task | State Observability | State Space | Policy Representation (Number of Features) |
|---|---|---|---|
| **Backgammon**<br>(Tesauro, 1992) | Complete | Discrete | Neural network<br>(198) |
| **Job-shop scheduling**<br>(Zhang and Dietterich, 1995) | Complete | Discrete | Neural network<br>(20) |
| **Tetris**<br>(Bertsekas and Tsitsiklis, 1996) | Complete | Discrete | Linear<br>(21) |
| **Elevator dispatching**<br>(Crites and Barto, 1996) | Partial | Continuous | Neural network<br>(46) |
| **Acrobot control**<br>(Sutton, 1996) | Complete | Continuous | Tile coding<br>(4) |
| **Dynamic channel allocation**<br>(Singh and Bertsekas, 1997) | Complete | Discrete | Linear<br>(100's) |
| **Active guidance of finless rocket**<br>(Gomez and Miikkulainen, 2003) | Partial | Continuous | Neural network<br>(14) |
| **Fast quadrupedal locomotion**<br>(Kohl and Stone, 2004) | Partial | Continuous | Parameterized policy<br>(12) |
| **Robot sensing strategy**<br>(Kwok and Fox, 2004) | Partial | Continuous | Linear<br>(36) |
| **Helicopter control**<br>(Ng *et al.*, 2004) | Partial | Continuous | Neural network<br>(10) |
| **Dynamic bipedal locomotion**<br>(Tedrake *et al.*, 2004) | Partial | Continuous | Feedback control<br>policy (2) |
| **Adaptive job routing/scheduling**<br>(Whiteson and Stone, 2004) | Partial | Discrete | Tabular<br>(4) |
| **Robot soccer keepaway**<br>(Stone *et al.*, 2005) | Partial | Continuous | Tile Coding<br>(13) |
| **Robot obstacle negotiation**<br>(Lee *et al.*, 2006) | Partial | Continuous | Linear<br>(10) |
| **Optimized trade execution**<br>(Nevmyvaka *et al.*, 2006) | Partial | Discrete | Tabular<br>(2-5) |
| **Blimp control**<br>(Rottmann *et al.*, 2007) | Partial | Continuous | Gaussian Process<br>(2) |
| **9 × 9 Go**<br>(Silver *et al.*, 2007) | Complete | Discrete | Linear<br>(≈1.5 million) |
| **Ms. Pac-Man**<br>(Szita and Lőrincz, 2007) | Complete | Discrete | Rule List<br>(10) |
| **Autonomic resource allocation**<br>(Tesauro *et al.*, 2007) | Partial | Continuous | Neural network<br>(2) |
| **General game playing**<br>(Finnsson and Björnsson, 2008) | Complete | Discrete | Tabular (over<br>part of state space) |
| **Soccer opponent "hassling"**<br>(Gabel *et al.*, 2009) | Partial | Continuous | Neural network<br>(9) |
| **Adaptive epilepsy treatment**<br>(Guez *et al.*, 2008) | Partial | Continuous | Extremely randomized<br>trees (114) |
| **Computer memory scheduling**<br>(İpek *et al.*, 2008) | Complete | Discrete | Tile coding<br>(6) |
| **Motor skills**<br>(Peters and Schaal, 2008) | Partial | Continuous | Motor primitive<br>coefficients (100's) |
| **Combustion Control**<br>(Hansen *et al.*, 2009) | Partial | Continuous | Parameterized policy<br>(2-3) |

function; yet non-linear representations are used commonly in practice, as apparent from Table 1.

Our survey of RL applications suggests that the most common strategy adopted while implementing sequential decision making in practice is to apply algorithms that come with provable guarantees under more restrictive assumptions, and to empirically verify that they remain effective when those assumptions are relaxed. Typically much manual effort is expended in designing schemes to mitigate the adverse effects partial observability and inadequate function approximation. In addition recent lines of research have focused on developing adaptive methods to cope with these factors (Pineau *et al.*, 2006; Whiteson and Stone, 2006; Mahadevan, 2009). While such methods can improve the performance of RL algorithms in practice, their effectiveness is yet to be demonstrated on a wide scale; it remains that even in the situations they apply, the undesirable effects of partial observability and function approximation are only reduced, and not eliminated.

Adopting the view that in practice, partial observability and function approximation will affect learning to varying degrees, we aim to examine the capabilities of learning methods that operate *in their presence*. Specifically we design a framework in which these factors can be systematically controlled to gauge their effect on different learning methods. While these factors can be construed as aspects of an agent's learning apparatus, our study also considers task-specific characteristics such as the size of the state space and the stochasticity of actions. Any fixed setting for the parameters that control these factors determines a *learning problem*, on which different learning methods can be compared.

In our study, we compare learning methods from two contrasting classes of algorithms. The first class corresponds to (model-free) on-line value function-based methods, which learn by associating utilities with action choices from individual states. The second class of algorithms we examine are policy search methods. Rather than learn a value function, policy search methods seek to directly optimize the parameters representing a policy, treating the expected long-term reward accrued as an objective function to maximize.

First we evaluate several methods *within* each of the above classes, and based on their empirical performance, pick one method from each class to further compare across a suite of problem instances. The representatives thus chosen are Sarsa($\lambda$) (Rummery and Niranjan, 1994; Sutton and Barto, 1998) from the class of on-line value function-based methods, and CMA-ES (Hansen, 2009) from the class of policy search methods. In evaluating a method on a problem instance, our experimental framework allows us to extensively search for the method-specific parameters (such as learning rates, eligibility traces, and sample sizes for fitness evaluation) that lead to the method's best performance. Our experiments identify regions of the problem space that are better suited to on-line value function-based and policy search methods, and yield insights about the effect of algorithm-specific parameters.

The remainder of this article is organized as follows. In Section 2, we describe the detailed design of our parameterized learning problem. Section 3 provides brief descriptions of the methods compared in the study. In Section 4, we present detailed results from our experiments, which we follow with a discussion in Section 5. Related work is discussed in Section 6. We summarize and conclude the article in Section 7.

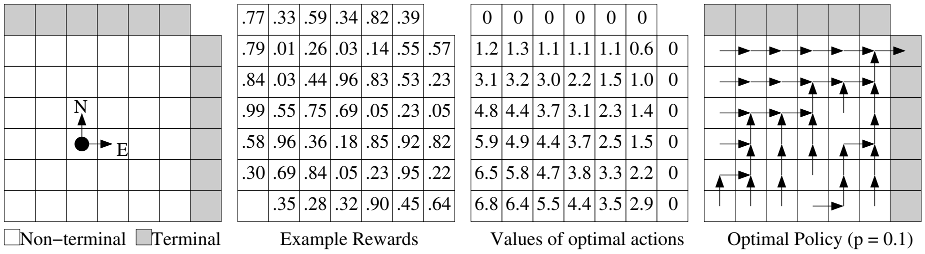
## 2 A Parameterized Sequential Decision Making Problem

In this section, we describe the construction of our parameterized learning problem, which is composed of a task MDP and an accompanying learning framework that incorporates partial observability and function approximation.

## 2.1 Problem Size and Stochasticity

The class of tasks we design consists of simple square grids, each having a finite number of states. An example of such a task is illustrated in Figure 1. The size of the state space is $s^2 - 1$, where $s$, the side of the square, serves as a parameter to be varied. Each episode begins with the agent placed in a start state chosen uniformly at random from among the set of non-terminal states, as depicted in Figure 1(a). The north and east sides of the grid are lined with terminal states, of which there are $2(s-1)$. From each state, the agent can take either of two actions: **North** (**N**) and **East** (**E**). On taking **N** (**E**), the agent moves north (east) with probability $p$ and it moves east (north) with probability $1-p$. The variable $p$, which essentially controls the stochasticity in the transitions, is also treated as a parameter of the task MDP. Note that irrespective of the value of $p$, the agent always moves either north or east on each transition before reaching a terminal state. Consequently episodes last at most $2s-3$ steps.

Through the course of each episode, the agent accrues rewards at the states it visits. Each MDP is initialized with a fixed set of rewards drawn uniformly from $[0,1]$, as illustrated in Figure 1(b). In general the rewards in an MDP can themselves be stochastic, but in our tests, we find that the effect of stochastic rewards on our learning algorithms is qualitatively similar to the effect of stochastic state transitions, which are controlled by the parameter $p$. Thus, we keep the rewards deterministic. Figures 1(c) and 1(d) show the optimal values and the actions to which they correspond under the reward structure shown in Figure 1(b) (assuming $p = 0.1$). We do not discount rewards in the computation of values. Notice that the variation in values along the north and east directions is gradual: this supports the scope for generalization between neighboring cells. The values in Figure 1(c) are obtained using dynamic programming. Indeed it is also straightforward under this setup to learn the optimal policy based on experience, for example by using a table of action values updated through Q-learning. However, the objective of our study is to investigate situations in which table-based approaches are not guaranteed to succeed. In the remainder of this section, we specify the aspects of our learning problem that, in ways similar to real-world problems, render table-based approaches infeasible.



(a)

| | | | | | | |
|---|---|---|---|---|---|---|
| .77 | .33 | .59 | .34 | .82 | .39 | |
| .79 | .01 | .26 | .03 | .14 | .55 | .57 |
| .84 | .03 | .44 | .96 | .83 | .53 | .23 |
| .99 | .55 | .75 | .69 | .05 | .23 | .05 |
| .58 | .96 | .36 | .18 | .85 | .92 | .82 |
| .30 | .69 | .84 | .05 | .23 | .95 | .22 |
| | .35 | .28 | .32 | .90 | .45 | .64 |

Example Rewards

(b)

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | |
| 1.2 | 1.3 | 1.1 | 1.1 | 1.1 | 0.6 | 0 |
| 3.1 | 3.2 | 3.0 | 2.2 | 1.5 | 1.0 | 0 |
| 4.8 | 4.4 | 3.7 | 3.1 | 2.3 | 1.4 | 0 |
| 5.9 | 4.9 | 4.4 | 3.7 | 2.5 | 1.5 | 0 |
| 6.5 | 5.8 | 4.7 | 3.8 | 3.3 | 2.2 | 0 |
| 6.8 | 6.4 | 5.5 | 4.4 | 3.5 | 2.9 | 0 |

Values of optimal actions

(c)

Optimal Policy (p = 0.1)

(d)

**Fig. 1** (a) Example of parameterized MDP example with $s = 7$; the number of non-terminal states is 36. (b) Rewards obtained at "next states" of transitions. (c) Optimal action values from each state when $p = 0.1$. (d) Corresponding optimal policy.

## 2.2 Partial Observability

In an MDP, the current system state and action completely determine the dynamics of the ensuing transition. However, in a number of RL applications, perceptual aliasing (Whitehead and Ballard, 1991) and noisy sensors (Stone *et al.*, 2005) deny an agent direct access to the underlying system state. In principle the agent can keep a record of its past observations, and effectively use this memory as a means to reconstruct the system state (Lin and Mitchell, 1993; McCallum, 1996). Indeed the seminal work of Åström (1965) demonstrates that by keeping a “belief state” that is updated based on incoming observations, an agent can eventually disambiguate states perfectly. However, the complexity of doing so is forbidding even in the context of planning (with known transition dynamics) (Cassandra *et al.*, 1994), and is yet to scale to large problems (Pineau *et al.*, 2006). Using experience to disambiguate states in partially observable environments is typically feasible only in very small problems (Chrisman, 1992; McCallum, 1995; Bakker *et al.*, 2003). In effect, learning agents in most RL applications have to treat “observed states” as states, and their performance varies depending on the validity of this assumption (Nevmyvaka *et al.*, 2006).

Each cell in our task MDP corresponds to a state. In order to model partial observability, we constrain the learner to use an observed state $o$, which, in general, can be different from the true state $s$. Our scheme to pick $o$ based on $s$ is depicted in Figure 2. Given $s$, we consider all the cells that lie within $d_x$ from it along the x direction and within $d_y$ along the y direction: from among these cells, we pick one uniformly at random to serve as the corresponding observed state $o$. Controlling $d_x$ and $d_y$ allows us to vary the extent of partial observability.

Before starting a learning run, we fix $d_x$ and $d_y$: each is sampled from a Gaussian distribution with zero mean and a standard deviation equal to $\sigma$, and then rounded to the nearest integer. Note that $d_x$ and $d_y$ can be positive, negative, or zero. Figures 2(b) and 2(c) show an illustrative trajectory of states numbered 1 through 9. Under different settings of $d_x$ and $d_y$, the figures show the set of all possible observed states that could result while the agent traces its trajectory. As is apparent from the figures, by

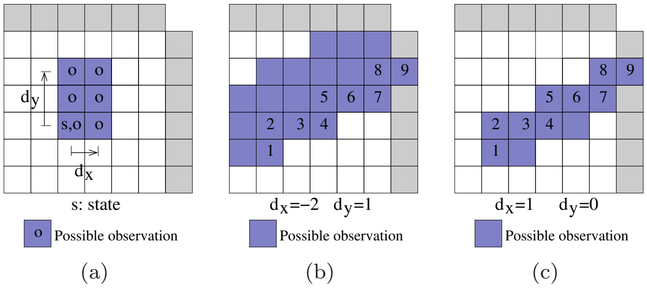


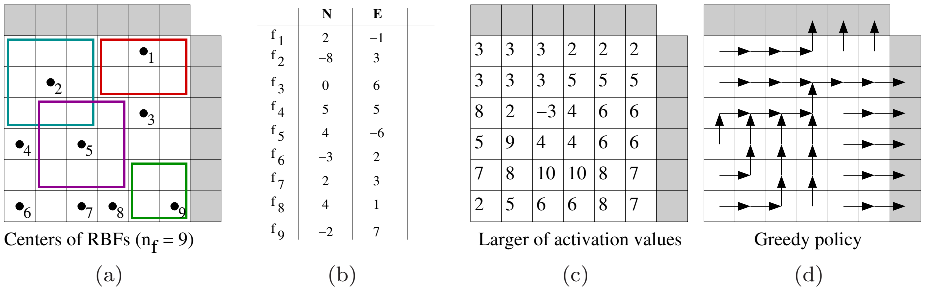
**Fig. 2** An implementation of partial observability in the example MDP from Figure 1. (a) Variables $d_x$ and $d_y$ (themselves generated randomly based on parameter $\sigma$) define a rectangle with the true state at a corner; cells within this rectangle are picked uniformly at random to constitute observed states. (b) A trajectory of true states 1 through 9, and the set of all possible observed states that could be encountered during this trajectory when $d_x = -2$ and $d_y = 1$. (c) For the same trajectory, the set of possible observed states when $d_x = 1$ and $d_y = 0$.

keeping $d_x$ or $d_y$ fixed for the entire course of a learning run (i.e., by not changing them from episode to episode), the state noise encountered by the agent during its lifetime is *systematic* in nature. Informal experimentation with a number of schemes for implementing state noise suggests that biased noise tends to affect learning more severely than zero-mean noise. The magnitude of the noise, implemented through $d_x$ and $d_y$, is controlled by the single free parameter $\sigma$, which we vary in our experiments. Setting $\sigma$ to 0 ensures complete observability of state. Progressively larger values of $\sigma$ lead to observed states that are farther apart from the agent's true state, and render the agent's interaction with the environment non-Markovian.

### 2.3 Function Approximation

The function approximation scheme in our learning problem is motivated by "CMAC" (Albus, 1981), a popular method that is used in a number of RL applications (Singh and Sutton, 1996; Stone *et al.*, 2005; İpek *et al.*, 2008). At each decision making step, we provide the learning agent a vector of $n_f$ features to describe its observed state. Each feature is a square "tile", with a binary activation: 1 within the boundary of the tile and 0 outside. Tiles have a fixed width $w$, which serves as a parameter in our experiments that determines the extent of generalization between states while learning. The centers of the tiles are chosen uniformly at random among non-terminal cells in the MDP. Figure 3 continues the example from Figure 1, describing the architecture used for function approximation. In Figure 3(a), nine tiles (numbered 1 through 9) are used by the function approximator. The tile width $w$ is set to 3; for illustration, four among the nine tiles are shown outlined.

Notice that every non-terminal cell in Figure 3(a) is covered by at least one tile: i.e., every cell has at least one feature that is active. Indeed we ensure that complete coverage is always achieved, in order that non-trivial decisions can be made at every cell. Clearly, not all the cells could be covered if the number of tiles ($n_f$) and the width



Centers of RBFs ($n_f = 9$)

(a)

|  | N | E |
|---|---|---|
| $f_1$ | 2 | −1 |
| $f_2$ | −8 | 3 |
| $f_3$ | 0 | 6 |
| $f_4$ | 5 | 5 |
| $f_5$ | 4 | −6 |
| $f_6$ | −3 | 2 |
| $f_7$ | 2 | 3 |
| $f_8$ | 4 | 1 |
| $f_9$ | −2 | 7 |

(b)

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 3 | 3 | 3 | 2 | 2 | 2 |
| 3 | 3 | 3 | 5 | 5 | 5 |
| 8 | 2 | −3 | 4 | 6 | 6 |
| 5 | 9 | 4 | 4 | 6 | 6 |
| 7 | 8 | 10 | 10 | 8 | 7 |
| 2 | 5 | 6 | 6 | 8 | 7 |

Larger of activation values

(c)

Greedy policy

(d)

**Fig. 3** Function approximation in example MDP from Figure 1. (a) A randomly chosen subset of cells (numbered 1 through 9) are the centers of overlapping tiles (giving $\chi = \frac{9}{36} = 0.25$). The tile width $w$ is set to 3; tiles 1, 2, 5, and 9 are shown outlined (and clipped at the boundaries of the non-terminal region). (b) Table showing coefficients associated with each tile for actions **N** and **E**. (c) The activation value of each cell for an action is the sum of the weights of the tiles to which it belongs. The figure shows the higher activation value (among **N** and **E**) for each cell. (d) Arrows mark a policy that is greedy with respect to the activations: i.e., in each cell, the action with a higher activation value is chosen. In general the agent will take the greedy action from its *observed* state.

of each tile ($w$) are both small; in all our experiments, we set these parameters such that in conjunction they can facilitate complete coverage of all non-terminal cells. The placement of the $n_f$ tiles is performed randomly, but preserving the constraint that all non-terminal cells be covered. In order to implement this constraint, we first place the tiles in regular positions that guarantee complete coverage, and then repeatedly shift tiles, one at a time, to random positions while still preserving complete coverage. Rather than treat $n_f$ directly as a parameter in our experiments, we normalize it by dividing by the number of non-terminal cells: $(s-1)^2$. The resulting quantity, $\chi = \frac{n_f}{(s-1)^2}$, lies in the interval $(0, 1]$, and is more appropriate for comparisons across different problem sizes. In Figure 3(a), $n_f = 9$ and $s = 7$, yielding $\chi = 0.25$. We treat $\chi$ as a parameter in our experiments. As we shortly describe, $\chi$ determines the resolution with which independent actions can be taken from neighboring cells. In this sense, $\chi$ measures the "expressiveness" of the function approximation scheme.

Given the set of features for its observed state, the agent computes a separate linear combination for each action, yielding a scalar "activation" for that action. For illustration, consider Figure 3(b), which shows a set of coefficients for each feature and action. It is these coefficients (or "weights") that the agent updates when it is learning. While learning, the agent may take any action from the states it visits. However, while evaluating learned behavior, we constrain the agent to take the action with the higher activation, breaking ties evenly. Figure 3(c) shows the *higher* of the resulting activations for the two possible actions at each cell in our illustrative example; Figure 3(d) shows the action with the higher activation.

In effect, the only free parameters for the learning agent to update are the sets of coefficients corresponding to each action. By keeping other aspects of the representation — such as the features and policy — fixed, we facilitate a fair comparison between different learning methods. In general, value function-based methods such as Sarsa($\lambda$) seek to learn weights that approximate the action value function. We expect that setting $\sigma = 0$ and $\chi = 1$ would favor them, as the optimal action value function can then be represented. While this is so under any value of $w$, setting $w = 1$ replicates the case of table-based learning with no generalization. Higher settings of $w$ enforce generalization. Increasing $\sigma$ or reducing $\chi$ would likely shift the balance in favor of policy search methods, under which activations of actions are merely treated as action preferences. As Baxter and Bartlett (2001) illustrate, even in simple 2-state MDPs, with function approximation, it is possible that the optimal action value function cannot be represented, even if an optimal policy can be represented.

In summary the design choices listed in this section are the end products of a process of trial and error directed towards constructing a suite of instances that allow us to study trends in learning algorithms, rather than constructing instances that are challenging in themselves. Table 2 summarizes the parameters used in our framework. Parameters $s$, $p$, $\sigma$, $\chi$, and $w$, along with a random seed, fix a learning problem for our experiments. By averaging over multiple runs with different random seeds, we estimate the mean performance achieved by learning methods as a function of $s$, $p$, $\sigma$, $\chi$, and $w$. Note that even if these parameters do not perfectly replicate an instance of any *specific* sequential decision making in practice, they are capable of being *varied* in a controlled manner to measure their effect on learning algorithms.

It must be noted that the parameterized learning problem described above is limited in several respects. While it enables the study of the most central problem parameters — problem size, stochasticity, partial observability and function approximation — it

**Table 2** Summary of learning problem parameters. The last column shows the ranges over which each parameter is valid and meaningful to test.

| Parameter | Property of: | Controls: | Range |
|---|---|---|---|
| $\mathbf{s}$ | Task | Size of state space | $\{2, 3, \ldots, \infty\}$ |
| $\mathbf{p}$ | Task | Stochasticity in transitions | $[0, 0.5)$ |
| $\boldsymbol{\sigma}$ | Agent/task interface | Partial observability | $[0, \infty)$ |
| $\boldsymbol{\chi}$ | Agent | Expressiveness of func. approx. | $(0, 1]$ |
| $\mathbf{w}$ | Agent | Generalization of func. approx. | $\{1, 3, \ldots, 2s - 3\}$ |

does not likewise isolate several other aspects influencing practical implementations of RL. Foremost is the question of exploration, which is not very crucial in our setup due to the occurrence of start states uniformly at random. The learning agent only has two actions; in practice large or continuous action spaces are quite common. Understanding the effect of other aspects, such as computational and memory constraints, the variation among action values from a state, different types of state noise, the sparsity and spread of the rewards, and the average episode length, would also be important for designing better algorithms in practice. We hope that the experimental methodology introduced in this article will aid future investigation on such subjects.

In the next section, we provide brief descriptions of the learning algorithms used in our experiments; in Section 4, the algorithms are compared at a number of different parameter settings drawn from the ranges provided in Table 2. Along with the parameterized learning problem itself, the results of these experiments are an important contribution of our article.

## 3 Methods in Study

As noted earlier, we compare two contrasting classes of learning methods in our study: on-line value function-based (VF) methods, and policy search (PS) methods. With the aim of comparing these classes themselves, we first evaluate various methods *within each class* to pick a representative. In this section, we describe the learning methods thus considered, and describe relevant implementation-specific details. Experiments follow in Section 4.

### 3.1 On-line Value Function-based (VF) Methods

We compare three learning methods from the VF class: Sarsa($\lambda$) (Rummery and Niranjan, 1994; Rummery, 1995), Q-learning($\lambda$) (Watkins, 1989; Watkins and Dayan, 1992; Rummery, 1995; Peng and Williams, 1996; Sutton and Barto, 1998), and Expected Sarsa($\lambda$) (abbreviated "ExpSarsa($\lambda$)") (Rummery, 1995; van Seijen *et al.*, 2009). These methods are closely related: they all continually refine an approximation of the action value function, making a constant-time update every time a new state is encountered. Yet the methods are distinguished by subtle differences in their update rules. We include these methods in our study to examine how their differences affect learning under function approximation and partial observability: settings under which theoretical analysis is limited. We proceed to describe the methods themselves.

Sarsa($\lambda$) is a model-free value function-based method, which makes on-line, on-policy, temporal difference (TD) learning updates. The learning agent maintains an estimate of an action value function, $Q$, which is updated as it encounters sequences of states ($s$), actions ($a$) and rewards ($r$). In particular assume that the agent encounters the following trajectory, in which suffixes index decision steps:

$$s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, a_{t+2}, r_{t+3}, \ldots.$$

The agent updates $Q(s_t, a_t)$ by computing a target, $Q_{Target}(s_t, a_t)$, and taking an incremental step towards it as follows:

$$Q(s_t, a_t) \leftarrow (1 - \alpha_t)Q(s_t, a_t) + \alpha_t Q_{Target}(s_t, a_t),$$

where $\alpha_t \in (0, 1]$ is the learning rate for the update. Recall that in our architecture, $Q$ is represented as a linear function approximator; hence, the learning update is implemented through gradient descent. Under Sarsa(0), the "fully bootstrapping" version of Sarsa, the target is computed as follows:

$$Q_{Target}^{Sarsa(0)}(s_t, a_t) = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}),$$

where $\gamma \in [0, 1)$ is a discount factor.[3] Note that the target does not count the actual rewards accrued beyond time step $t + 1$; rather, the discounted sum of these "future" rewards is substituted with its current estimate: $Q(s_{t+1}, a_{t+1})$. By contrast, a Monte Carlo method, Sarsa(1) computes its estimates wholly from sample returns, as:

$$Q_{Target}^{Sarsa(1)}(s_t, a_t) = r_{t+1} + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+1+k}.$$

This target would not change depending on the actual states that the trajectory visited, but only based on the sequence of rewards obtained. This makes Monte Carlo methods less dependent on the state signal than fully bootstrapping methods. Both methods still try to estimate state-action values, and therefore rely on being able to precisely detect $s_t$ and represent $Q_{Target}^{Sarsa}(s_t, a_t)$. In general, intermediate methods that implement varying extents of bootstrapping can be conceived by varying the "eligibility trace" parameter $\lambda \in [0, 1]$. The estimated target for $Q(s_t, a_t)$ used by Sarsa($\lambda$) is:

$$Q_{Target}^{Sarsa(\lambda)}(s_t, a_t) = r_t + \gamma\{(1 - \lambda)Q(s_{t+1}, a_{t+1}) + \lambda Q_{Target}^{Sarsa(\lambda)}(s_{t+1}, a_{t+1})\}.$$

For the case of discrete MDPs, in which $Q$ can be maintained as a table, Singh *et al.* (2000) show that by following a policy that is "greedy in the limit" with respect to $Q$, and which performs an infinite amount of exploration, Sarsa(0) will ultimately converge to the optimal action value function $Q^*$, from which the optimal policy $\pi^*$ can be derived by acting greedily. For linear function approximation schemes such as in our parameterized learning problem, Perkins and Precup (2003) show that convergence to a fixed point can be achieved by following a method similar to Sarsa(0).

We use a standard implementation of Sarsa($\lambda$) with binary features, a linear representation, and *replacing* eligibility traces (see Sutton and Barto, 1998, p. 212). While learning, the agent follows an $\epsilon$-greedy policy. We treat both the exploration strategy

[3] It is legitimate to use $\gamma = 1$ in episodic tasks. We do so in our experiments (see Section 4).

and the schedule for annealing the learning rate as *parameterizable* processes. We follow an $\epsilon_u$-greedy exploration policy during episode $u$, keeping $\epsilon_0$ as a free parameter, and $\epsilon_U = 0.01$, where $U$ is the total number of training episodes. Intermediate values of $\epsilon_u$ are set based on a harmonic sequence going from $\epsilon_0$ to 0.01. We use such a schedule based on empirical evidence of its effectiveness.

Interestingly, informal experimentation shows us that a similar annealing schedule is also the most effective for the learning rate $\alpha$; i.e., we keep $\alpha_0$ as a free parameter and anneal it harmonically to 0.01 at the end of training. Since features are binary, we divide the mass of each update equally among the features that are active under the state-action being updated. It is worth noting that theoretically-motivated update rules exist for annealing the learning rate. For example, Hutter and Legg (2008) derive a rule based on minimizing the squared loss between estimated and true values. However, their approach is only viable with tabular representations of $Q$, and further, only in continuing (rather than episodic) tasks.

Apart from $\lambda$, $\epsilon_0$, and $\alpha_0$, yet another parameter influencing Sarsa($\lambda$) is the setting of the initial weights (coefficients in the linear representation). In our experiments, we set all the weights initially to $\theta_0$, which is our final method-specific parameter. Table 3 summarizes the parameters defining Sarsa($\lambda$). These parameters also apply to other methods in the VF class, which we now describe.

Whereas Sarsa($\lambda$) computes its target for time $t$ based on the action to be taken at time $t+1$ — $a_{t+1}$ — ExpSarsa($\lambda$) and Q-learning($\lambda$) compute their targets (and make learning updates) *before* $a_{t+1}$ is chosen. Once $s_{t+1}$ is reached, ExpSarsa($\lambda$) computes its target based on an expectation over the possible choices of $a_{t+1}$ while following the current $\epsilon$-greedy policy $\pi_{t+1}$:

$$\begin{aligned} Q_{Target}^{ExpSarsa(\lambda)}(s_t, a_t) = r_t + \gamma\{(1-\lambda)Q(s_{t+1}, a_{t+1}) + \\ \lambda \sum_{a \in A} \mathbb{P}\{a|s_{t+1}, \pi_{t+1}\} Q_{Target}^{ExpSarsa(\lambda)}(s_{t+1}, a)\}. \end{aligned}$$

This alteration leads to a reduced variance in the update, as a sampled action value is now replaced with a smoothed-out estimate. It is shown by van Seijen *et al.* (2009) that like Sarsa(0), ExpSarsa(0) can also be made to converge to the optimal policy in discrete, finite MDPs. Q-learning($\lambda$) differs from Sarsa and ExpSarsa in that it is an *off-policy* method: rather than learning the action value function of the policy being followed, $\pi_t$, Q-learning($\lambda$) seeks to directly learn the action values of the optimal policy $\pi^*$. This objective is achieved by computing the target as follows:

$$Q_{Target}^{Q\text{-}learning(\lambda)}(s_t, a_t) = r_t + \gamma\{(1-\lambda)\max_{a \in A} Q(s_{t+1}, a) + \lambda Q_{Target}^{Q\text{-}learning(\lambda)}(s_{t+1}, a_{t+1})\}.$$

**Table 3** Summary of parameters used by methods within VF. The last column shows the ranges over which we tune each parameter.

| Parameter | Controls: | Range |
|---|---|---|
| $\boldsymbol{\lambda}$ | Eligibility traces | $[0, 1]$ |
| $\boldsymbol{\alpha_0}$ | Initial learning rate | $[0.1, 1]$ |
| $\boldsymbol{\epsilon_0}$ | Initial exploration rate | $[0.1, 1]$ |
| $\boldsymbol{\theta_0}$ | Initial weights | $[-10.0, 10.0]$ |

Sutton and Barto (1998, see p. 184) refer to the update rule above as a “naïve” implementation of Q-learning with eligibility traces, because the rule lacks technical justification as a proper TD learning update. By contrast, there are some sound variations of Q-learning with eligibility traces (Watkins, 1989; Peng and Williams, 1996), under which updates additionally have to account for whether chosen actions were greedy or non-greedy. We refer the reader to the Ph.D. thesis of Rummery (1995, see ch. 2) for an excellent presentation of various TD update rules. Note that Rummery refers to Sarsa as “modified Q-learning”, and to Expected Sarsa as “summation Q-learning”. It would exceed the scope of this article to undertake an extensive study comparing all possible variants of TD update rules. Rather, a novel contribution of our experiments is to consider three among them — Sarsa($\lambda$), ExpSarsa($\lambda$), and (naïve) Q-learning($\lambda$) — in the presence of function approximation and partial observability. Indeed our results show that under this setting, hitherto uncharacterized patterns in performance emerge.

As with Sarsa($\lambda$), we parameterize ExpSarsa($\lambda$) and Q-learning($\lambda$) to control their learning and exploration rates, as well as their initial weights. The corresponding parameters, $\alpha_0$, $\epsilon_0$ and $\theta_0$, are summarized in Table 3. Henceforward we drop the “$\lambda$” from Sarsa($\lambda$), ExpSarsa($\lambda$), and Q-learning($\lambda$), and refer to these methods simply as Sarsa, ExpSarsa, and Q-learning, respectively. We do so to highlight that these methods are no longer only parameterized by $\lambda$ in our experiments — so are they by $\alpha_0$, $\epsilon_0$, and $\theta_0$.

Note that setting $w > 1$ in our parameterized learning problem introduces generalization, and further, setting $\chi < 1$ reduces the expressiveness of the function approximator. Thus, in general, the approximate architectures used are incapable of representing the optimal action value function $Q^*$. Even with full expressiveness ($\chi = 1$), if using generalization ($w > 1$), methods from VF are not guaranteed to converge to the optimal action value function. And even if these methods approximate the action value function well, as defined through the Bellman error, greedy action selection might yet pick suboptimal actions in regions of inaccurate approximation, resulting in low long-term returns (Kalyanakrishnan and Stone, 2007).

A bulk of the research in RL with linear function approximation has been in the context of prediction: estimating the value function of a fixed policy (without policy improvement). An early result due to Sutton (1988) establishes that TD(0) with linear function approximation converges when the features used are linearly independent; Dayan and Sejnowski (1994) extend this result to TD($\lambda$), $\forall \lambda \in [0, 1]$, while Tsitsiklis and Van Roy (1997) show convergence for the more realistic case of infinite state spaces and linearly dependent features. Although most results for the convergence of linear TD learning are for estimating values of the policy that is used to gather experiences, the more general (and potentially useful) case of off-policy learning has also been addressed (Precup *et al.*, 2001; Sutton *et al.*, 2009).

The problems in learning approximate value functions on-line primarily arise due to the nonstationarity and bias *in the targets* provided to the function approximator (Thrun and Schwartz, 1993). The best theoretical guarantees for learning control policies with approximate schemes come with several restrictions. Most results are limited to linear function approximation schemes; in addition some make demands such as Lipschitz continuity of the policy being learned (Perkins and Precup, 2003) and favorable initial conditions (Melo *et al.*, 2008). Results tend to guarantee convergence of certain updating schemes, but invariably lack desirable guarantees about the long-term reward that will be accrued at convergence (Sabes, 1993; Perkins and Pendrith, 2002; Perkins and Precup, 2003).

In recent work, Maei *et al.* (2010) introduce the Greedy-GQ algorithm, which provably converges while making off-policy learning updates to a linear function approximator. Unfortunately, Greedy-GQ requires that the policy followed *while learning* stay fixed, preventing the agent from actively exploring based on the experiences it gathers. Thus, $\epsilon$-greedy exploration, with $\epsilon < 1$, violates the assumptions needed for Greedy-GQ to converge; our informal experiments confirm that such a version of Greedy-GQ does not perform on par with the other methods we consider within the VF class. Thus, we do not include Greedy-GQ in our extensive comparisons.

### 3.2 Policy Search (PS) Methods

We include three methods from the PS class in our study: the Cross-entropy method (CEM) (de Boer *et al.*, 2005), the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen, 2009), and a genetic algorithm (GA). In addition we implement random weight guessing (RWG) to compare as a baseline.

CEM is a general optimization algorithm that has been used effectively as a policy search method on RL problems (Szita and Lőrincz, 2006). In our linear representation, the vector of weights constitute the policy parameters to be adapted. The objective function, or "fitness" function, to be maximized is the expected long-term reward accrued by following the greedy policy that is derived from the weights. An iterative algorithm, CEM maintains and updates a parameterized distribution over the multi-dimensional search space. On each iteration, a population of *#pop* points is sampled from the current distribution. Each point is evaluated, and the $\mu$ points with the highest fitness values are used to determine the distribution parameters for the next iteration. The update rule is such that with time the variance of the distribution shrinks and its mean gravitates towards regions of the parameter space with high fitness values. As is a common choice, in our experiments, we use a Gaussian distribution to generate sample points. We initialize the mean of this distribution to be the zero vector; along each dimension the variance is set to 1 (with no covariance terms). The update rule for Gaussian distributions is such that at every iteration, the updated distribution has as its mean and variance the sample mean and variance of the $\mu$ selected points (independently for each parameter). In general the update can also depend on the current distribution's mean and variance; further, noise can be added to the variance at each iteration to prevent premature convergence (Szita and Lőrincz, 2006). We do not implement these variations in our experiments as they do not appear to have a significant effect in our domain.

Like CEM, the CMA-ES method also employs the principle of updating a distribution at each generation to maximize the likelihood of the $\mu$ points with the highest fitness values being generated. However, unlike CEM, CMA-ES tracks covariances across dimensions and actively monitors the search path in the parameter space leading up to the current generation. Handling several aspects in the search procedure, CMA-ES is a fairly sophisticated optimization technique; we refer the reader to descriptions from Hansen (2009) and Suttorp *et al.* (2009) for details. Nevertheless, we find it surprisingly straightforward to implement the algorithm based on existing code, which automatically sets most of the method-specific parameters.[4] We set the initial distribution identically to the one set under CEM.

We implement GA in a manner akin to CEM and CMA-ES. On each generation, we spawn and evaluate *#pop* policies; of these, the $\mu$ with the highest fitness values are selected to generate the next population. Specifically, pairs are chosen uniformly at random from the selected $\mu$ and crossed over to produce two offspring each. Policies are real-valued vectors over the space of parameters searched. Each parameter, restricted to the interval $[-1, 1]$, is represented using a 32-bit Gray-coded string. To implement crossover between two individuals, the bit strings corresponding to each parameter are cut at a random location and matched across individuals, thereby yielding two offspring. To implement mutation, individuals are picked from the population with a small probability (0.05), and once picked, have each bit flipped with a small probability (0.1). Both under CEM and GA, we set $\mu$, the number of policies selected every generation to seed the next, to 15% of the population size *#pop*. Experiments suggest that these methods are not very sensitive to $\mu$ values in this vicinity. CMA-ES uses a default value for $\mu$ depending on *#pop* and the number of parameters searched.

In general, PS methods can work with a variety of representations. An illustrative example is the PS framework implemented by Kohl and Stone (2004) to optimize the forward walking speed of an Aibo robot. The gait they design has parameters describing trajectory positions and timings, which are combined using manually designed sets of rules. Evolutionary computation has been a particularly popular choice for PS; in particular several neuro-evolutionary techniques have been tested on control tasks (Gomez and Miikkulainen, 1999; Stanley, 2004; Metzen *et al.*, 2008). Typically the policy is represented using a neural network, whose topology and weights are evolved to yield policies with higher values.

In order to maintain a fair comparison with the VF methods in this study, we enforce that the methods chosen from PS employ the same representation, under which real-valued parameters are to be optimized (Section 2.3). In principle numerous evolutionary and optimization techniques apply to this problem: among others, amoeba, particle swarm optimization, hill climbing, and several variants of genetic and "estimation of distribution" algorithms. The reason we choose CEM and CMA-ES in our comparison is due to the several successes they have achieved in recent times (Szita and Lőrincz, 2006, 2007; Hansen *et al.*, 2009), which partly owes to their mathematically-principled derivation. We implement GA on the grounds that although it optimizes exactly the same parameters, it employs a bit string-based *internal* representation during its search, and thus is qualitatively different. Note that all the methods described above only use the ranks among fitness values in a generation to determine the next. In this manner, these methods differ from canonical policy gradient methods for RL (Sutton *et al.*, 2000; Baxter and Bartlett, 2001; Kakade, 2001), which rely on the of gradients with respect to the policy parameters to identify a direction in the parameter space for policy improvement. Our policy is not analytically differentiable since it is deterministic.

The three PS methods described above take two parameters, listed in Table 4. Since fitness is defined as the expected long-term reward accrued by a policy, we estimate it by averaging the returns from $\#trials$ episodes. The other method-specific parameter, $\#gens$, is the number of generations undertaken during the learning period. As a consequence, note that if the total number of training episodes is $U$, the population

[4] See: `http://www.lri.fr/~hansen/cmaes_inmatlab.html`.

**Table 4** Summary of parameters used by methods from PS. The last column shows the ranges over which we tune each parameter. The range shown for $\#trials$ is used when the total number of episodes is 50,000, as in a majority of our experiments (see section 4). The range is scaled proportionately with the total number of training episodes.

| Parameter | Controls: | Range |
|---|---|---|
| $\boldsymbol{\#trials}$ | Samples per fitness evaluation | $[25, 250]$ |
| $\boldsymbol{\#gens}$ | Generations | $[5, 50]$ |

size in each generation is given by $\frac{U}{\#trials \times \#gens}$. Under RWG, we repeatedly generate policies, evaluate each for $\#trials$ episodes, and retain the policy with the highest fitness. Informal experimentation shows that it is more effective to sample policies based on a Gaussian distribution for each parameter, rather than a uniform distribution.

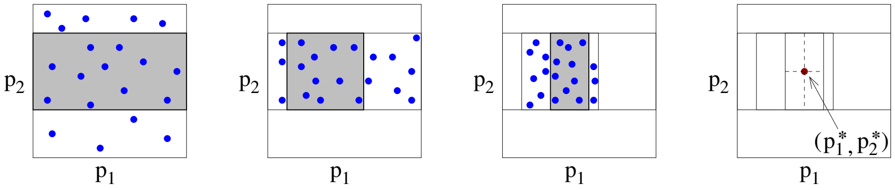
## 4 Experiments and Results

In this section, we present experimental results. First, in Section 4.1, we describe our experimental methodology. In Section 4.2, we perform comparisons *within* the VF and PS classes to pick representative methods from each. These representative methods are further compared across a series of experiments in Sections 4.3 through 4.7 to ascertain their interactions with parameters of the learning problem.

### 4.1 Experimental Methodology

As defined in Section 2, a learning problem is fixed by setting $s$, $p$, $\sigma$, $\chi$, $w$, and a random seed. Additionally, before conducting an experiment, we fix $U$, the total number of learning episodes conducted. Recall from tables 3 and 4 that learning methods themselves have parameters: $\lambda$, $\alpha_0$, $\epsilon_0$, $\theta_0$, $\#trials$, and $\#gens$. In some experiments, we study the learning performance at fixed values of these method-specific parameters. However, note that even for a fixed method (say Sarsa), its best performance at different problem settings will invariably be achieved under different settings of its method-specific parameters ($\lambda$, $\alpha_0$, $\epsilon_0$, $\theta_0$). In response we devise an automatic search procedure over the method-specific parameter space (4-dimensional for Sarsa) to find a configuration yielding the highest learned performance for a given problem instance and number of training episodes.

The search procedure — described schematically in Figure 4 for a 2-dimensional parameter space — involves evaluating a number of randomly generated points in the space and iteratively halving the search volume, always retaining the region with the highest performance density. The procedure is necessarily inexact due to stochasticity in evaluations, and since performance might not be "well-behaved" over the region searched. Yet in practice we find that with sufficient averaging (2000 points per generation) and enough splits (5 times the number of dimensions searched), the procedure yields fairly consistent results. We suffix the method-specific parameter configurations returned by the search "$*$" to indicate that they have been optimized for some task setting and number of training episodes. Thus, Sarsa$^*$ refers to an instance of Sarsa identified through the search procedure, its parameters being $\lambda^*$, $\alpha_0^*$, $\epsilon_0^*$, and $\theta_0^*$. Under Sarsa$(\lambda)^*$, $\lambda$ is fixed, and only $\alpha_0$, $\epsilon_0$, and $\theta_0$ are optimized.



(a) Stage 1 (b) Stage 2 (c) Stage 3 (d) Final solution

**Fig. 4** Illustration of a search over two parameters, $p_1$ and $p_2$. Initial ranges for each parameter are specified as inputs to the search. To begin, points are sampled uniformly from within the specified ranges. At each sampled point, a *single* learning run is conducted and its final performance recorded. Subsequently a split is performed to halve the search volume, retaining an axis-aligned region with the highest density of performance among all such regions. The process is repeated several times: with each split, attention is focused on a smaller part of the search space empirically found to contain the most successful learning runs. Note that at each stage, any parameter could lead to the best split ($p_2$, $p_1$, and $p_1$ at stages 1, 2, 3, respectively, in the illustration). At termination the midpoint of the surviving volume is returned.

For clarity, we enumerate here the sequence of steps undertaken in each of our experiments.

1. We fix learning problem parameters $s$, $p$, $\sigma$, $\chi$, and $w$.
2. We fix the total number of training episodes $U$.
3. Either we manually specify an instance of a learning method, or search for one, as described above, to maximize performance for the problem parameters and training episodes set in steps 1 and 2.
4. With the chosen method instance, we conduct at least 1,000 independent trials of learning runs. Each trial is fixed by setting a different random seed, which can generate additional seeds for the learning problem (to determine features and rewards) and the learning method (to explore, sample, etc.).
5. Each learning trial above results in a fixed policy. We estimate the performance of this policy through 1,000 Monte Carlo samples. (Although sometimes a policy can be evaluated exactly through dynamic programming, the presence of function approximation and partial observability make it necessary to estimate performance through sampling.) Note that methods from VF and PS are both evaluated based on a greedy policy with respect to the learned weights.
6. Since all the rewards in our parameterized learning problem are non-negative, we find that problems with larger state spaces invariably lead to policies with higher absolute rewards. To facilitate meaningful comparison across problems with different parameter settings, we linearly scale the performance of a policy such that 0 corresponds to the value, under the same settings, of a random policy, and 1 to that of an optimal policy. In our graphs, we plot this normalized performance measure. Note that our careful design of the task MDP allows us to compute the performance values of random and optimal policies at each setting, even if the settings themselves preclude the *learning* of optimal behavior by an agent. Policies that are "worse than random" have normalized performance values less than zero.
7. We report the normalized performance achieved (over all trials), along with one standard error (typically these are small and sometimes difficult to distinguish visually in our graphs). Note that standard errors do not apply to the results of

our parameter search, such as to find $\lambda^*$ under some problem instance. For any task instance, the method-specific parameter search is conducted exactly once.

In summary: the steps outlined above aim to provide each method the best chance of success for a given problem instance and training time, and then to fairly evaluate and compare competing methods. Having specified our methodology, we proceed to describe results from our experiments.

## 4.2 Picking Representative Methods and Setting the Training Period

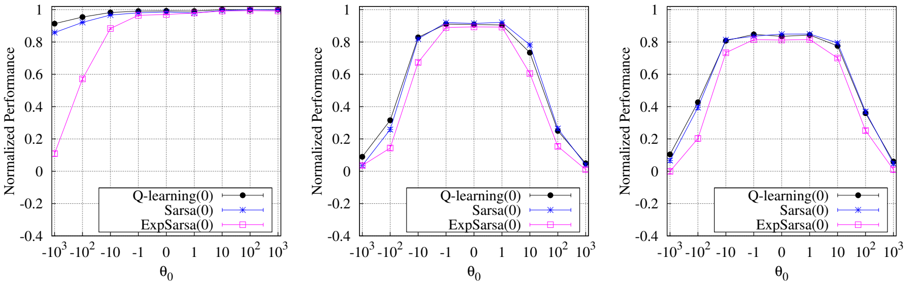
The first phase in our experiments is to pick representative learning methods from the VF and PS classes. We now present comparisons among methods from these classes. We also describe how we set the number of training episodes for learning runs in our study.

### *4.2.1 Picking a Representative Method from VF*

In comparing methods from the VF class, we observe that the method-specific parameter with the most dominant effect on performance is the setting of initial weights, $\theta_0$. For illustration consider Figure 5. In the experiments reported therein, we compare Sarsa(0), ExpSarsa(0), and Q-learning(0). For all these methods, we find that a broad range of the parameters $\alpha_0$ and $\epsilon_0$ yield policies with high performance; we manually pick favorable settings from among these ranges. Q-learning(0) and Sarsa(0) use $\alpha_0 = 0.8$, $\epsilon_0 = 0.8$, while ExpSarsa(0) uses $\alpha_0 = 0.8$, $\epsilon_0 = 0.2$. The total number of training episodes $U$ is set to 50,000.

The three methods show qualitatively similar patterns in performance as $\theta_0$ is varied. In Figure 5(a), we find that all of them achieve near-optimal behavior at large settings of $\theta_0$, directly reflecting the merits of optimistic initialization (Even-Dar and Mansour, 2001). Action values tend to lie in the range $[0, 20]$; correspondingly we notice that "pessimistic" initialization of weights to lower values leads to noticeable degradation in performance. Note that the settings in Figure 5(a) correspond to a fully expressive tabular representation with no generalization. As we introduce generalization by increasing $w$ to 5 (Figure5(b)), we observe a significant change in trend: both very high and very low initial weights lead to a marked decrease in the final performance. This trend persists as the expressiveness $\chi$ is reduced (Figure 5(c)).

In figures 5(b) and 5(c), it is apparent that ExpSarsa(0) falls below Sarsa(0) and Q-learning(0) at most settings of $\theta_0$. We posit that since it performs a weighted average over all next state-action values, updates under ExpSarsa(0) are likely to propagate error from state-actions that are encountered less frequently. For a perfect tabular representation, such as in Figure 5(a), van Seijen *et al.* (2009) prove that ExpSarsa(0) updates have the same bias, but a lower variance, compared to updates under Sarsa(0). However, our results appear to suggest that when generalization is present (as $w$ is increased), and learning starts with a stronger initial bias (by setting $\theta_0$ farther away from the true action values), ExpSarsa(0) suffers more from the error in its updates. Extending this argument, we could expect ExpSarsa to perform worse at high values of $\alpha_0$ and $\epsilon_0$ when generalization is used. Shortly we present the evidence for such a phenomenon.



(a) $\chi = 1$, $w = 1$.

(b) $\chi = 1$, $w = 5$.

(c) $\chi = 0.6$, $w = 5$.

**Fig. 5** [$s = 10$, $p = 0.2$, $\sigma = 0$.] Plots showing the effect of the initial weights $\theta_0$ on the performance of on-line value function-based methods. Note the irregular spacing of points on the x axis. Plot (a) corresponds to an exact tabular representation with no generalization. Generalization is introduced in (b) by increasing $w$; additionally the expressiveness $\chi$ is reduced in (c).

We design three *problem* instances to further investigate differences between Sarsa, ExpSarsa, and Q-learning. Table 5 summarizes these problem instances. Instance $I_1$ corresponds to a fully-expressive tabular representation with no generalization, under which all three methods enjoy provable convergence guarantees. Expressiveness is reduced and generalization introduced in $I_2$. While $I_1$ and $I_2$ are both devoid of state noise, $I_3$ is identical to $I_1$ except for its higher setting of $\sigma$.

Figure 6 plots the performance of Sarsa$^*$, ExpSarsa$^*$ and Q-learning$^*$ on $I_1$, $I_2$, and $I_3$. Notice that under $I_1$, all the methods achieve near-optimal behavior at the end of 50,000 episodes of training. While optimal behavior is not to be expected under $I_2$, it becomes apparent that ExpSarsa$^*$ trails the other methods in this problem (p-value $< 10^{-4}$). This finding parallels the inference we draw from Figure 5: generalization and function approximation adversely affect ExpSarsa, as its learning updates propagate more bias than either Sarsa or Q-learning.

Recall that $I_3$ is identical to $I_1$, except that it introduces state noise. Thus, when compared with $I_1$, we observe that all three methods suffer a significant drop in performance under $I_3$. Yet, the introduction of state noise does not appear to disadvantage any of the methods more than the others. Table 6 reports the optimized method-specific parameters found by our search strategy under the three problem instances. From the table, we see that the values of $\lambda^*$ found for all three methods under $I_3$ are significantly higher than the values found under $I_1$ and $I_2$. We may infer that reducing the reliance on bootstrapped estimates (by setting high values of $\lambda$) counteracts the error introduced in TD updates due to state noise. We also observe from Table 6 that the

**Table 5** Parameter settings for illustrative problem instances $I_1$, $I_2$, and $I_3$.

| Problem instance | $s$ | $p$ | $\chi$ | $w$ | $\sigma$ |
|---|---|---|---|---|---|
| $I_1$ | 10 | 0.2 | 1 | 1 | 0 |
| $I_2$ | 10 | 0.2 | 0.5 | 7 | 0 |
| $I_3$ | 10 | 0.2 | 1 | 1 | 4 |

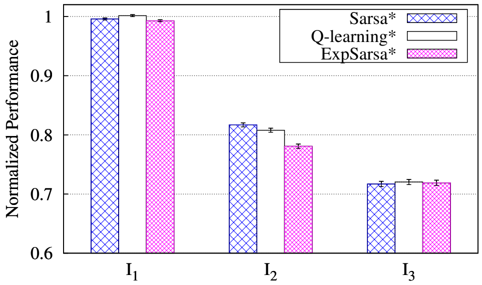


**Fig. 6** Comparison of the performance of different VF methods on the three problem instances from Table 5. Under each instance, and for each of the methods — Sarsa, Q-learning, and ExpSarsa — a systematic search (see Section 4.1) identifies the method-specific parameter settings ($\alpha_0$, $\epsilon_0$, $\theta_0$, and $\lambda$) yielding the highest performance after 50,000 episodes of training. The methods are marked "$*$" as they are run under these optimized parameter settings.

$\theta_0^*$ values found by our search strategy for each method and problem are as one may expect based on Figure 5. These results affirm the reliability of our search strategy.

Predominantly we find that the VF methods compared above are not very sensitive to the learning rate parameter $\alpha_0$ and the exploration parameter $\epsilon_0$ within the ranges in which we optimize them: $[0.1, 1]$ for both parameters. The only significant exception, to which we alluded earlier, is the case of ExpSarsa under $I_2$, which strongly favors lower $\alpha_0$ and $\epsilon_0$ settings. For reference, we provide graphs plotting the performance of VF methods as a function of $\alpha_0$ and $\epsilon_0$ in Appendix A.

In summary: we find that Sarsa and Q-learning (albeit with a "naïve" implementation of eligibility traces) perform equally well on all our experiments; both methods outperform ExpSarsa on problems in which generalization is employed. We pick Sarsa as a representative method from the VF class for our subsequent experiments (Sections 4.3 through 4.7).

### *4.2.2 Picking a Representative Method from PS*

We reuse problem instances $I_1$, $I_2$, and $I_3$ to compare methods from the PS class. As noted in Section 3.2, two parameters have to be set for methods from this class: *#trials* and *#gens*. Optimizing over these parameters, we plot the performance of CEM$^*$, CMA-ES$^*$, GA$^*$ and RWG$^*$ in Figure 7. Unlike with the VF class, the ordering among the methods from PS stays consistent across the problem instances. In all cases, CEM$^*$ and CMA-ES$^*$ outperform GA$^*$ and RWG$^*$ (p-value $< 10^{-4}$). However, CEM$^*$ and CMA-ES$^*$ themselves register virtually identical performance: they cannot

**Table 6** For each of three methods — Sarsa, Q-learning, and ExpSarsa — the method-specific parameters yielding the highest performance (at 50,000 episodes of training), under problem instances $I_1$, $I_2$ and $I_3$. Figures are rounded to one place of decimal.

| Problem | Sarsa$^*$ | | | | Q-learning$^*$ | | | | ExpSarsa$^*$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| instance | $\lambda$ | $\alpha_0$ | $\epsilon_0$ | $\theta_0$ | $\lambda$ | $\alpha_0$ | $\epsilon_0$ | $\theta_0$ | $\lambda$ | $\alpha_0$ | $\epsilon_0$ | $\theta_0$ |
| $I_1$ | 0.3 | 0.6 | 0.6 | 7.0 | 0.2 | 0.3 | 0.7 | 8.7 | 0.2 | 0.5 | 0.7 | 6.7 |
| $I_2$ | 0.1 | 0.7 | 0.7 | -1.0 | 0.2 | 0.7 | 0.8 | 0.8 | 0.1 | 0.9 | 0.2 | 0.8 |
| $I_3$ | 0.9 | 0.5 | 0.5 | 8.9 | 0.9 | 0.2 | 0.9 | 6.4 | 0.8 | 0.6 | 0.8 | 6.7 |

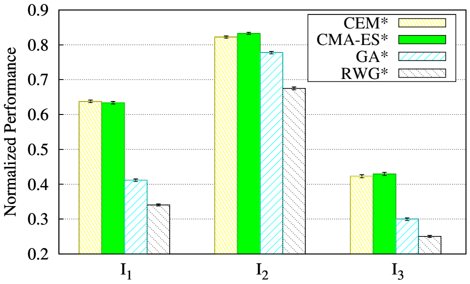


**Fig. 7** Comparison of the performance of different PS methods on the three problem instances from Table 5. Methods are marked "$*$" to denote that method-specific parameters — $\#trials$ and $\#gens$ (except for RWG) — have been optimized for each task instance.

be separated with statistical significance on instances $I_1$ and $I_3$, although in $I_2$, CMA-ES$^*$ emerges the winner (p-value $< 0.02$).

It is worth noting that whereas all the VF methods in our study achieve their highest performance on instance $I_1$, all the methods from PS achieve theirs on $I_2$. 50,000 episodes is a relatively short duration of training for PS methods, which do not make effective use of individual transition samples, but rather, aggregate them in evaluating fitness. Greater generalization across the state space (as in $I_2$, where $w = 5$) enables them to learn more quickly. In Section 4.6, we observe that if optimized for 500,000 episodes, PS methods do perform better at $w = 1$.

The best parameter settings found for each PS method, under the three chosen problem instances, are listed in Table 7. Although we search over $\#trials$ and $\#gens$, note that thereby we implicitly set up a search over the population size $\#pop$ used in every generation. This is a consequence of the relation that $\#trials \times \#gens \times \#pop = U$, the total number of training episodes. From the table, we observe that CMA-ES$^*$ typically employs a smaller population size than CEM$^*$, while GA$^*$ maintains significantly larger population sizes.

Appendix B displays the performance of the various PS methods as a function of their input parameters. We observe a noticeable variance in the performance of all the methods over the parameter ranges considered. While CMA-ES$^*$ and CEM$^*$ have comparable performance on all three problem instances, it is apparent that CMA-ES is more robust to parameter settings; i.e., it registers a higher performance over a wider range of settings. This makes CMA-ES overall a slightly more favorable candidate

**Table 7** For policy search methods, the method-specific parameters yielding the highest performance (at 50,000 episodes of training) for problem instances $I_1$, $I_2$, and $I_3$. Figures are rounded to the nearest integer. Under CEM, CMA-ES, and GA, the parameters searched are $\#gens$ ("$\#g$") and $\#trials$ ("$\#t$"). These parameters automatically fix the population size ("$\#p$"), which is suffixed with "D" to denote that it is implicitly *derived*. Under RWG, only $\#t$ is optimized; $\#g$ is implicitly derived. Derived parameters values are shown for reference.

| Problem instance | CEM$^*$ | | | CMA-ES$^*$ | | | GA$^*$ | | | RWG$^*$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\#g$ | $\#t$ | $\#p_D$ | $\#g$ | $\#t$ | $\#p_D$ | $\#g$ | $\#t$ | $\#p_D$ | $\#g_D$ | $\#t$ |
| $I_1$ | 18 | 44 | 63 | 24 | 61 | 34 | 9 | 49 | 113 | 113 | 444 |
| $I_2$ | 15 | 64 | 52 | 20 | 111 | 23 | 22 | 17 | 134 | 327 | 153 |
| $I_3$ | 10 | 113 | 44 | 17 | 80 | 37 | 15 | 30 | 111 | 125 | 400 |

than CEM from the class of PS methods. Therefore, we select CMA-ES for our further experiments.

### *4.2.3 Setting the Training Period*

Even if the problems in our experiments are themselves reasonably small, the extensive search and evaluation processes incur a significant amount of time during each experiment. One factor that plays a major role in determining the experimental running time is $U$, the total number of training episodes in each run. Setting $U = 50,000$, as we have in the experiments reported thus far, it takes us roughly 1-2 hours to complete a single search and evaluation procedure, such as, for example, identifying Sarsa$^*$ and evaluating it under $I_2$. In this duration, we have roughly 200 processes running in parallel on a computing cluster with 2GHz CPUs. In general we do not find it feasible to conduct extensive experimentation under higher values of $U$ (although we do undertake such investigation under *some* interesting cases, such as in Section 4.6).

To gauge the implications of consistently setting $U = 50,000$ in our subsequent comparisons, we run a single suite of experiments at multiple settings of $U$. Figure 8 shows the performance of Sarsa$^*$, Q-learning$^*$, ExpSarsa$^*$, CEM$^*$, and CMA-ES$^*$; under problem instances $I_1$, $I_2$, and $I_3$; optimized for various settings of $U$. As expected we find that all the methods improve their performance with longer training periods. The gains from a longer training period are more marked among the methods from PS, as in general, methods from VF appear to plateau within a few thousands of episodes.

We observe from Figure 8 that under all problem instances, the trend *within* methods in VF remains roughly the same at all values of $U$: under $I_1$, the methods all achieve comparable performance; under $I_2$, ExpSarsa$^*$ performs poorest; and under $I_3$, Q-learning$^*$. Likewise, no clear winner among CEM$^*$ and CMA-ES$^*$ emerges in any of the instances, for any setting of $U$. Therefore, we may conclude that our choice of picking Sarsa$^*$ and CMA-ES$^*$ for further comparisons is justified. However, the choice of $U$ does affect comparisons between Sarsa$^*$ and CMA-ES$^*$ themselves. Notice that up to 25,000 episodes, Sarsa$^*$ consistently outperforms CMA-ES$^*$. Yet, from 50,000 episodes onward, CMA-ES$^*$ overtakes Sarsa$^*$ on $I_2$ (p-value $< 10^{-3}$). Under $I_1$ and $I_3$, CMA-ES$^*$ narrows the margin with Sarsa$^*$ at $U = 1,000,000$, although it does not reach comparable performance.

The trends in Figure 8 inform our interpretation of the results to follow in the remainder of this section. In general we expect that Sarsa will not significantly improve its performance beyond 50,000 episodes of training, whereas CMA-ES consistently improves at least up to 1000,000 episodes. Even so, in several instances, we find that CMA-ES outperforms Sarsa even at 50,000 episodes, validating this choice of $U$ as a meaningful comparison point between the methods.

In summary: our “within class” comparisons in VF and PS provide convincing evidence that Sarsa and CMA-ES are respectively the best methods to represent these classes in our parameterized learning problem (except that Q-learning performs as well as Sarsa in VF). We now proceed to compare these methods as relevant problem parameters are varied. In each comparison (excepting cases in Section 4.6), the normalized performance of these methods after 50,000 episodes of training is considered while evaluating them.

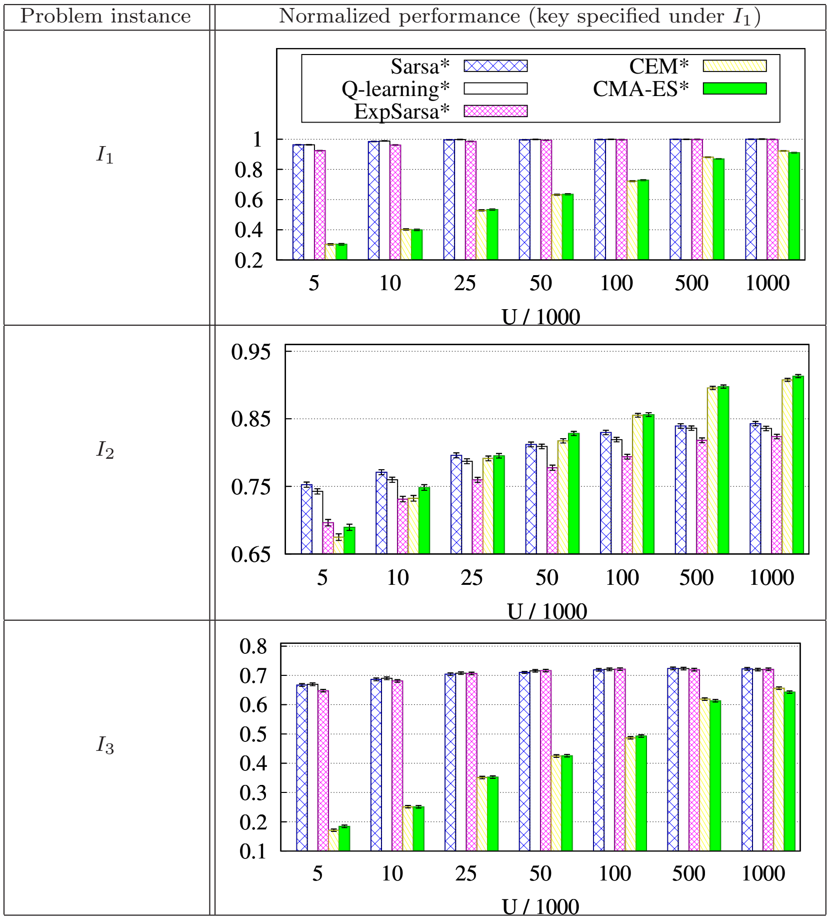


**Fig. 8** Plots showing the performance of different learning methods as the number of training episodes $U$ is varied. Each plot corresponds to a problem instance from Table 5. Note the irregular spacing of points on the x axis. At each point, the best performance achieved by three learning methods from VF (Sarsa*, Q-learning*, and ExpSarsa*) and two from PS (CEM*, CMA-ES*) is shown.

## 4.3 Problem Size and Stochasticity

In our first set of "VF versus PS" experiments, we evaluate our learning methods as the size of the state space and the stochasticity of transitions in the task MDP are varied. Conjunctions of three settings of $s$ (6, 10, 14) and three settings of $p$ (0, 0.2, 0.4) are compared; results are plotted in Figure 9. With complete expressiveness ($\chi = 1$), no generalization ($w = 1$), and full observability ($\sigma = 0$), all nine cases are akin to learning with a classical "tabular" representation.

The most striking observation from the plots in Figure 9 is the disparity in the learning rates of Sarsa* and CMA-ES*. In all nine cases, Sarsa* reaches near-optimal behavior, and typically so within a few thousands of episodes. At 50,000 episodes of training, in none of the problems does CMA-ES* match the performance of Sarsa* (p-

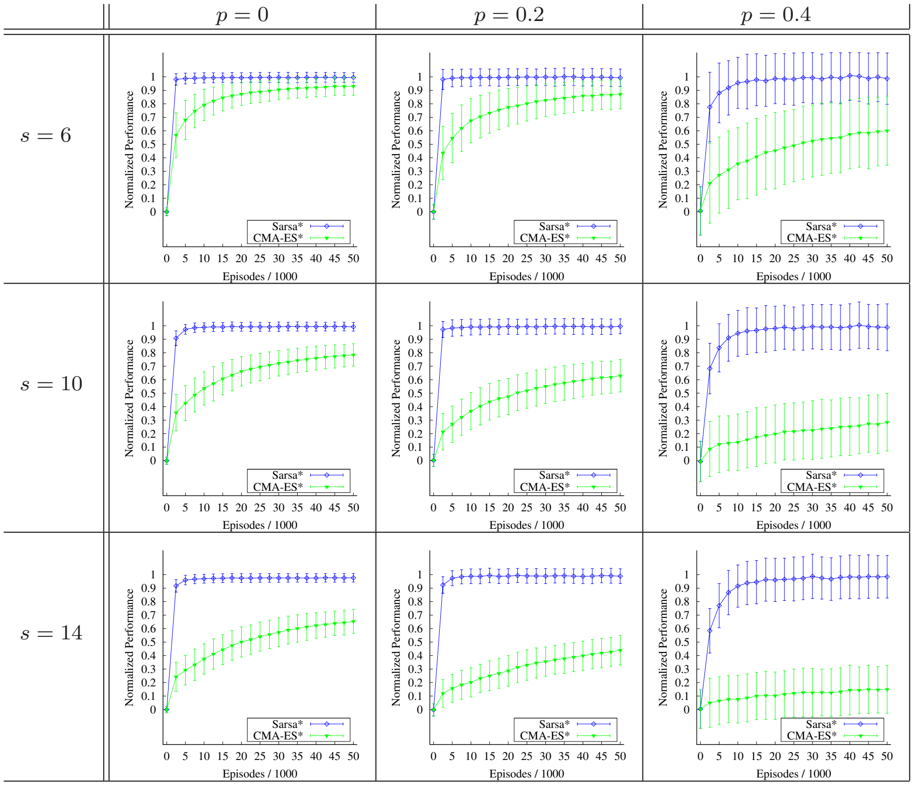


**Fig. 9** [$\chi = 1$, $w = 1$, $\sigma = 0$.] Sarsa* and CMA-ES* (optimized for 50,000 episodes of training) compared at different settings of $s$ and $p$. Unlike other plots in the article, in these learning curves, we plot one *standard deviation* in the performance (instead of one standard error).

value $< 10^{-4}$). The gap between the methods is to be expected, as by making learning updates based on every transition, VF methods make more efficient use of experience for learning than PS methods do. Note that Sarsa is still on-line and model-free; we expect model-based methods (Sutton, 1990) and batch methods (Lin, 1992; Lagoudakis and Parr, 2003) to further improve sample efficiency.

For both Sarsa* and CMA-ES*, we notice a decrease in performance as $s$ is increased. This decrease is more marked for CMA-ES*, as the dimensionality of the parameter space it searches increases quadratically with $s$. The effect of the stochasticity parameter in widening the gap between Sarsa* and CMA-ES* is also significant. The error bars plotted in the graphs show one standard deviation in performance (in all other graphs in the article, one standard error is shown). We observe that for both methods, the variance in performance increases as $p$ is increased, and further, that for any given problem, CMA-ES* displays a slightly higher variance than Sarsa*. As described in Section 2, note that even at $p = 0$, there is stochasticity in the task, as the start state for each episode is picked uniformly at random.

Recall that the method-specific parameters of Sarsa* and CMA-ES* have been optimized for each problem and training period. While we do not note any significant patterns among the method-specific parameters thereby found under Sarsa*, we note that under CMA-ES$^*$, $\#trials^*$ gets consistently higher as $p$ is increased. For example, at $s = 10$, the settings of $\#trials^*$ found by our search procedure are 38, 77, and 152 for $p = 0$, $p = 0.2$, and $p = 0.4$, respectively. In other words, CMA-ES$^*$ benefits from more evaluation trials in evaluating fitness values as the task stochasticity increases. Indeed recent research addresses the problem of tuning $\#trials$ in an adaptive manner (Heidrich-Meisner and Igel, 2009).

The primary inference from the set of experiments above is that Sarsa has significant advantages both in terms of the performance achieved and the variance in performance as problem size and stochasticity are increased. Not only is CMA-ES slower to learn, it demands better tuning of $\#trials$ across different problem instances. To characterize the reasons underlying these observations, we turn to Cobb (1992), who separates the inductive biases in a reinforcement learner into "language" and "procedural" biases. The former corresponds to the representation used by the learner, which in this study, we have fixed to be the same for the methods compared. VF and PS methods are essentially separated by their procedural bias: how they updates weights in the representation. The language bias in the problem instances above — $\chi = 1$, $w = 1$, $\sigma = 0$ — strongly favors the procedural bias of Sarsa. How would the methods fare if the language bias is changed? The experiments to follow examine the effects of state noise, generalization, and expressiveness.

## 4.4 Partial Observability

In our second set of experiments, we study the effect of partial observability by increasing $\sigma$. We notice a conjunctive relationship between $\sigma$ and $w$, the generalization width. In response we conduct experiments with three settings of $\sigma$ (0, 2, 4) and three settings of $w$ (1, 5, 9). Results are plotted in Figure 10: in each graph therein the performance of Sarsa($\lambda$)$^*$ is plotted at six values of $\lambda$ (0, 0.2, 0.4, 0.6, 0.8, 1). The performance of CMA-ES$^*$ (which does not depend on $\lambda$) is also shown.

In general the best memoryless policies for Partially Observable MDPs (POMDPs) can be stochastic (Singh *et al.*, 1994). Perkins and Pendrith (2002) show that in order to converge in POMDPs, it is necessary for methods such as Sarsa and Q-learning to follow policies that are *continuous in the action values*, unlike the $\epsilon$-greedy policies used by the VF methods in our experiments. However, we do not observe any divergent behavior for Sarsa($\lambda$) in the experiments reported here.

We notice that when $\sigma = 0$ and $w = 1$, the effect of $\lambda$ on the performance of Sarsa($\lambda$)$^*$ is not very pronounced. As soon as either $\sigma$ or $w$ is increased, intermediate values of $\lambda$ predominantly yield the highest performance for Sarsa($\lambda$)$^*$. These results echo the findings of Loch and Singh (1998), who demonstrate that deterministic policies learned using Sarsa($\lambda$) with ample exploration perform quite well on a suite of benchmark POMDPs. Key to their success is the high values of $\lambda$ used (between 0.8 and 0.975), which weight true returns from actions much higher than estimated values.

As the generalization width $w$ is increased, notice that there is no longer a single winner between Sarsa$^*$ and CMA-ES$^*$: VF methods no longer dominate PS methods when observability of state is limited. An intriguing trend that becomes apparent from Figure 10 is that the performance of Sarsa($\lambda$)$^*$ is not monotonic with respect to $w$: for most settings of $\sigma$ and $\lambda$, the highest performance is achieved at $w = 1$, followed by $w = 9$, with the lowest performance at $w = 5$. In Section 4.6, we find further evidence of

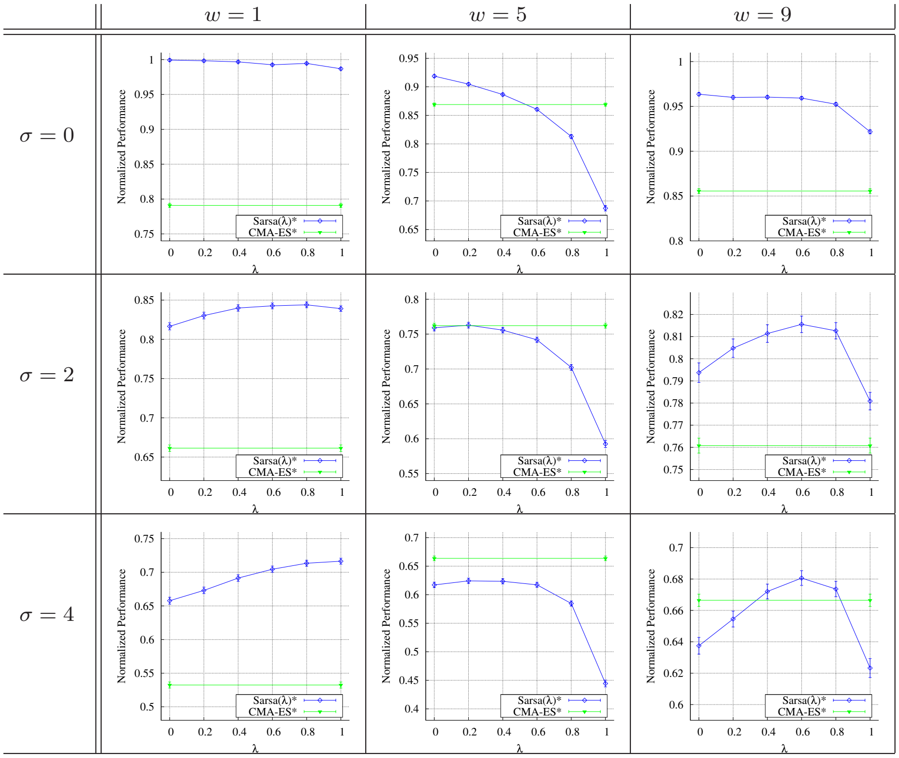


**Fig. 10** [$s = 10$, $p = 0.2$, $\chi = 1$.] Sarsa($\lambda$)* and CMA-ES* compared at different settings of $\sigma$ and $w$. Under each plot, six regularly spaced values of $\lambda$ are chosen and the corresponding Sarsa($\lambda$)* evaluated. CMA-ES* appears as a line, as it does not depend on $\lambda$.

such anomalous patterns in the performance of Sarsa as $w$ is varied. Interestingly CMA-ES* registers its highest performance, for any fixed $\sigma$, at $w = 5$ ($w = 9$ comes a close second). This trend arises as 50,000 episodes is a relatively short training duration for PS methods in this domain; generalization promotes quick initial learning. Experiments in Section 4.6 confirm that with 500,000 training episodes, CMA-ES performs best at $w = 1$.

A recent variant of Sarsa($\lambda$) applied to POMDPs is SarsaLandmark (James and Singh, 2009), in which $\lambda$ is set to 0 (full bootstrapping) when special "landmark" states (which are perfectly observable) are visited, but remains 1 at all other times (Monte Carlo). SarsaLandmark is not directly applicable in our domain as the agent receives no special information about landmark states. In recent work, Downey and Scanner (2010) propose a method to adaptively tune $\lambda$ while learning. Formally derived under a Bayesian framework, their algorithm — Temporal Difference Bayesian Model Averaging (TD-BMA) — is shown to outperform Sarsa($\lambda$) for *any* fixed value of $\lambda$ on illustrative grid-world tasks. Our results highlight that tuning $\lambda$ is of particular relevance in problems with state noise and generalization; our parameterized learning problem therefore becomes an ideal testbed for evaluating adaptive approaches.

In Table 8, we report the best initial weights, $\theta_0^*$, found for Sarsa($\lambda$)*, under various settings of $\lambda$, $\sigma$, and $w$. The most noticeable pattern from the table is the favor for

**Table 8** [$s = 10$, $p = 0.2$, $\chi = 1$.] $\theta_0^*$ (initial weights under Sarsa($\lambda$)$^*$) for different problem instances. Each cell in the table corresponds to a setting of $\sigma$, $w$ (problem parameters), and $\lambda$ (Sarsa parameter); entries correspond to the value of $\theta_0$ found by searching for Sarsa($\lambda$)$^*$. Note that each search is only performed once.

| $w$ | $\lambda = 0$ | | | $\lambda = 0.4$ | | | $\lambda = 1.0$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\sigma = 0$ | $\sigma = 2$ | $\sigma = 4$ | $\sigma = 0$ | $\sigma = 2$ | $\sigma = 4$ | $\sigma = 0$ | $\sigma = 2$ | $\sigma = 4$ |
| 1 | 8.7 | 7.4 | 5.0 | 8.0 | 7.9 | 7.8 | 7.8 | 7.5 | 9.6 |
| 5 | 0.5 | 0.0 | -0.7 | 0.3 | 0.6 | 1.8 | 2.1 | 1.4 | 1.6 |
| 9 | -0.3 | -0.3 | 0.6 | 0.1 | 0.0 | -0.6 | 0.1 | 0.5 | -0.8 |

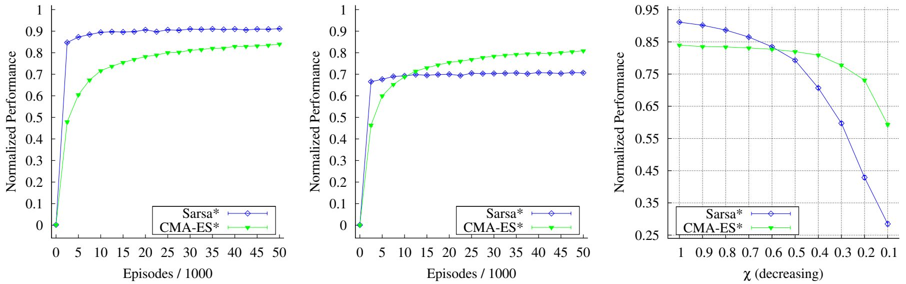
lower settings of $\theta_0$ as $w$ is increased. The best initial weights do not appear to change much as state noise and eligibility traces are varied.

### 4.5 Expressiveness of Function Approximator

Continuing our study, we conduct experiments to gauge the role of the expressiveness parameter $\chi$ in determining the performance of learning methods. Again, we find no single winner among Sarsa$^*$ and CMA-ES$^*$ as $\chi$ is varied. The results shown in Figure 11 are under fixed settings: $\sigma = 0$ and $w = 5$. The qualitative nature of the results does not change as $\sigma$ and $w$ are varied.

In the learning curve in Figure 11(a), under $\chi = 1$ (which allows the optimal action value function to be represented), Sarsa$^*$ displays quick learning to reach a normalized performance close to 0.9, whereas CMA-ES$^*$ fails to achieve comparable performance after 50,000 episodes. By contrast, at $\chi = 0.4$ (Figure 11(b)), we notice that Sarsa$^*$ suffers a dramatic drop in performance, plateauing at a normalized performance value close to 0.7. At the same setting of $\chi$, CMA-ES$^*$ overtakes the learning curve of Sarsa$^*$ and reaches a significantly higher performance at 50,000 episodes (p-value $< 10^{-4}$).

As $\chi$ is decreased, the representation for the value function and policy becomes increasingly handicapped. In Figure 11(c), we observe that both under Sarsa and CMA-ES, performance decreases monotonically as $\chi$ is reduced. However, of the two methods, Sarsa suffers the more significant drop in performance as $\chi$ is reduced. Whereas Sarsa



(a) $\chi = 1$ (b) $\chi = 0.4$ (c) Range of $\chi$

**Fig. 11** [$s = 10$, $p = 0.2$, $w = 5$, $\sigma = 0$.] Plots (a) and (b) show learning curves of Sarsa($\lambda$)$^*$ and CMA-ES$^*$ at different values of $\chi$. Plot (c) shows the performance achieved after 50,000 episodes of training at different values of $\chi$.

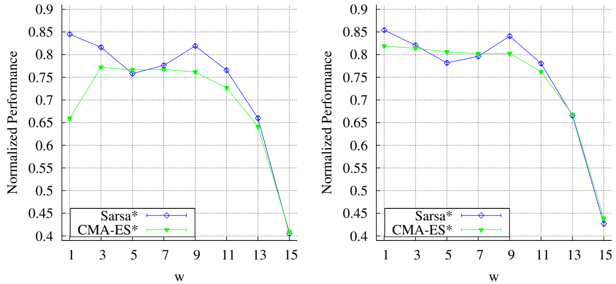
outperforms CMA-ES for $\chi \geq 0.7$ (p-value $< 10^{-4}$), the opposite is true when $\chi \leq 0.5$ (p-value $< 10^{-4}$). We do not observe any striking trends in the method-specific parameters of Sarsa$^*$ and CMA-ES$^*$ as $\chi$ is varied.

To the best of our knowledge, prior literature does not compare methods from VF and PS while constraining them to use the same representation. Our finding that CMA-ES is able to achieve good performance even under a representation that is extremely impoverished for approximating the value function suggests that it is a promising candidate in a large number of real-world domains in which feature engineering and representations are deficient. We posit that like the example constructed by Baxter and Bartlett (2001), many of the cases with $\chi < 1$ allow for the representation of high-reward policies, but only admit poor approximations of the action value function. Notice that we do not have any *irrelevant* features in our learning problem: in the future it would be useful to incorporate such a setting, which is often encountered in practice. Non-linear function approximation would be an equally important avenue to explore.

## 4.6 Generalization of Function Approximator

In Section 4.4, we noted that the generalization parameter $w$ plays a role in determining the relative order between Sarsa and CMA-ES at different values of $\sigma$. In examining the effect of $w$ more closely, we notice that although its effect on CMA-ES is fairly regular, its interaction with Sarsa is less predictable. Figure 12 shows the normalized performance of these methods at settings of $w$ varying from 1 (no generalization) to 15 (very broad generalization). In Figure 12(a), the total number of training episodes is set to 50,000, while in Figure 12(b), it is set to 500,000.

We observe that with 50,000 training episodes, CMA-ES$^*$ performs its best at $w = 3$, but with 500,000 episodes, its best performance is at $w = 1$. We ascribe this effect to the benefit of generalizing early during the search, which quickly identifies the most promising actions in localized regions of the state space. With more time it becomes important to further discriminate among individual states, which it is most possible with $w = 1$. It is not surprising that for both methods, the performance begins to drop sharply for $w > 9$. Since in this problem, $s = 10$, some non-terminal cells in



(a) episodes=50,000

(b) episodes=500,000

**Fig. 12** [$s = 10$, $p = 0.2$, $\chi = 1$, $\sigma = 2$.] Performance of Sarsa$^*$ and CMA-ES$^*$ at different values of $w$, optimized in plot (a) for 50,000 training episodes, and in plot (b) for 500,000 training episodes.

the task MDP necessarily get activated by *all* the tiles present if the tile width exceeds 9. Indeed beyond $w = 19$ (not shown in figure), no two cells in the MDP remain distinguishable.

Interestingly Sarsa$^*$ presents a less regular pattern in performance as $w$ is varied, as evinced by Figure 12(a). We find that Sarsa$^*$ is most effective at $w = 1$, but its performance suffers a dip until $w = 5$; again a rise until $w = 9$; before monotonically decreasing again. 50,000 episodes is a fairly long duration by VF standards, as apparent from several learning curves shown in the article (for instance, see Figure 11(a)). It is clear that the irregular performance pattern of Sarsa$^*$ is not an artefact of training time, as the pattern essentially persists at 500,000 episodes of training (Figure 12(b)). Also, notice the small error bars in both the plots: the pattern is systematic.

We investigate whether the irregular pattern in the performance of Sarsa persists as the problem size is increased beyond $s = 10$. Figure 13 (see top row) indeed affirms that at $s = 14$ and $s = 18$, too, multiple local minima emerge in the performance as $w$ is varied. Curiously, under all three settings of $s$, we observe that as $w$ is varied, a correlation exists — up to $w < s$ — between the performance of Sarsa$^*$ and $\lambda^*$, the eligibility trace parameter optimized for each problem setting. The bottom row in Figure 13 shows the values of $\lambda^*$ under each setting. Observe that for $w < s$ the local maxima and minima in $\lambda^*$ predominantly coincide with those of the normalized performance (recall that for $w \geq s$, states necessarily become aliased).

At present we do not have a conclusive explanation for the phenomenon described above. Since CMA-ES shows predictable variation with $w$, we surmise that the variation shown by Sarsa ultimately arises from its on-line updates to the value function. We speculate that “edge effects” in our tiling scheme, whereby states on the periphery of the grid have fewer neighbors, might induce patterns in the trajectory taken by the

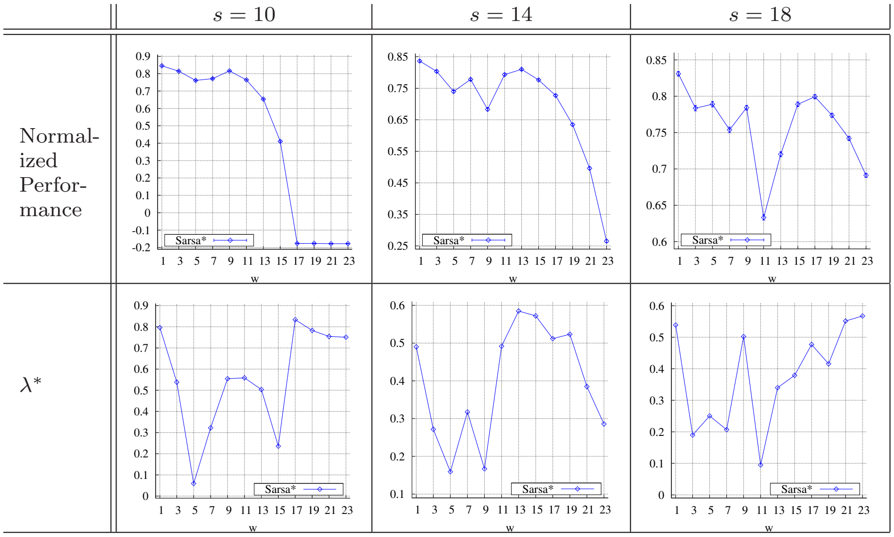


**Fig. 13** [$p = 0.2$, $\chi = 1$, $\sigma = 2$.] Analysis of Sarsa$^*$ as $s$ (columns) and $w$ (x axis in each plot) are varied. The top row shows the normalized performance achieved at each setting; correspondingly the bottom row shows $\lambda^*$ — the value of $\lambda$ found by searching for Sarsa$^*$ — for the same settings.

value function. Nevertheless, closer inspection would be necessary to fully explain such behavior.

In the context of kernel-based methods, Ormoneit and Sen (2002) discuss the "bias-variance tradeoff" induced by generalization widths. Munos and Moore (2002), and Sherstov and Stone (2005), devise schemes for setting different generalization widths in different parts of the state space. Our parameterized learning problem becomes a valuable testbed to evaluate this line of work, which our results hint needs attention.

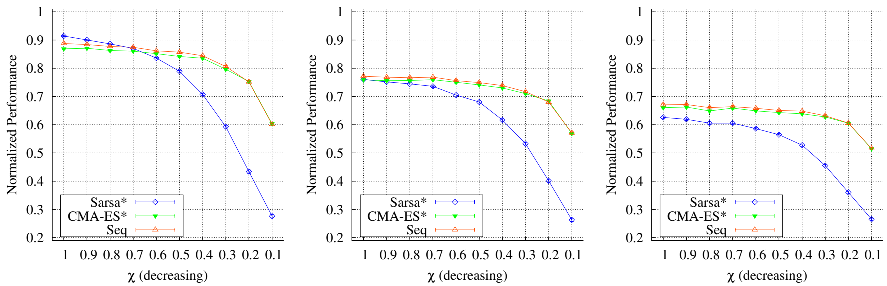
### 4.7 Sequencing Sarsa and CMA-ES

In all our experiments, we find that CMA-ES is significantly slower to learn than Sarsa. In settings with high values of $\sigma$ and low values of $\chi$, CMA-ES outperforms Sarsa mainly because Sarsa reaches a lower asymptote, and not because CMA-ES has any steeper a learning curve. In our last set of experiments, we examine whether CMA-ES can be given a boost by initializing it with a policy learned by Sarsa. We abbreviate the resulting sequencing algorithm "Seq". Since in our experiments, Sarsa and CMA-ES are constrained to share a common representation, a straightforward way to initialize CMA-ES with a policy learned using Sarsa is to set its initial weights to those learned by Sarsa. Although a raw transfer of weights is not always applicable across different representations, we conjecture that the resulting technique can still offer insights about synthesizing the merits of VF and PS methods. Guestrin *et al.* (2002) adopt a similar scheme in a multiagent task to transfer weights learned using LSPI to initialize a policy gradient method. In their experiments, the policy implemented is "softmax" over the learned action values; in our experiments, CMA-ES inherits a greedy policy.

In principle the method-specific parameters of Seq include all of the method-specific parameters of Sarsa ($\lambda$, $\alpha_0$, $\epsilon_0$, $\theta_0$) and CMA-ES ($\#trials$, $\#gens$), along with the number of episodes at which the transfer of weights is effected from Sarsa to CMA-ES. By definition, then, Seq$^*$ would always perform at least as well as the better of Sarsa$^*$ and CMA-ES$^*$ (by running Sarsa$^*$ for 50,000 or for 0 episodes, as appropriate). Rather than search for and evaluate this best instance of Seq, we constrain Seq to (a) use Sarsa$^*$ and CMA-ES$^*$, each optimized independently for 50,000 episodes, as its constituents; and (b) transfer weights from Sarsa$^*$ to CMA-ES$^*$ after 2,500 episodes of training. Thus, Seq essentially amounts to running CMA-ES$^*$, but starting from a potentially useful initialization. The initial variance along each parameter to be optimized by CMA-ES$^*$ after the switch is set to the overall variance of the weights themselves.

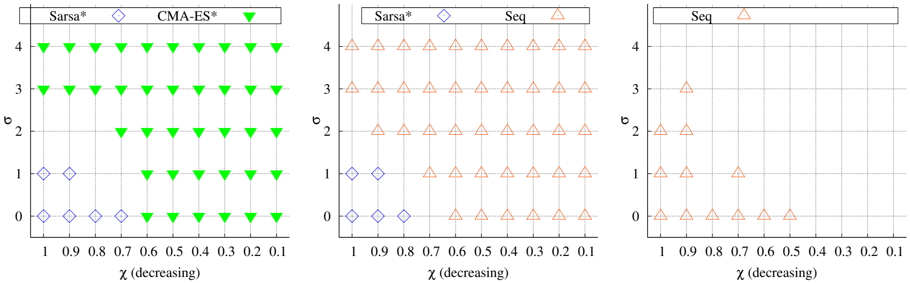
Figure 14 compares Seq with Sarsa$^*$ and CMA-ES$^*$ under problem settings in which $\sigma$ and $\chi$ are varied. Under all settings, we find that Seq performs at least as well as CMA-ES, if not marginally better. Thorough "head-to-head" comparisons between the three methods are plotted in Figure 15. Each plot therein compares two of the methods. For specified settings of $\sigma$ and $\chi$, the method registering higher performance is marked if the evidence is statistically significant (p-value $< 0.01$). Figure 15(a) identifies regions of the problem space suiting Sarsa$^*$ and CMA-ES$^*$. From Figure 15(b), we observe that Seq marginally extends the territory claimed by CMA-ES$^*$. Indeed Seq outperforms CMA-ES$^*$ in regions where $\sigma$ is low and $\chi$ is high, and performs at least as well in the remainder of the problem configurations (Figure 15(c)).

As discussed in Section 4.2, CMA-ES requires several times the training time of Sarsa to achieve comparable performance. The experiments reported here suggest that when time is constrained, and yet CMA-ES can outshine Sarsa, CMA-ES can be further



(a) $\sigma = 0$ (b) $\sigma = 2$ (c) $\sigma = 4$

**Fig. 14** [$s = 10$, $p = 0.2$, $w = 5$.] Sarsa*, CMA-ES*, and Seq compared at different settings of $\sigma$ and $\chi$. Sarsa* and CMA-ES* are optimized independently at each problem setting; Seq combines the methods thus tuned (with no further optimization), transferring weights from Sarsa to CMA-ES after 2,500 training episodes.



(a) Sarsa* vs. CMA-ES* (b) Sarsa* vs. Seq (c) CMA-ES* vs. Seq

**Fig. 15** [$s = 10$, $p = 0.2$, $w = 5$.] Three plots showing pairwise comparisons between Sarsa*, CMA-ES*, and Seq at different settings of $\sigma$ and $\chi$. At each reported setting, one of the methods is indicated if with statistical significance ($p < 0.01$), it achieves a higher performance than the other. At some settings, the methods cannot be thus separated; note that in plot (c), CMA-ES does not outperform Seq at *any* of the reported settings of $\sigma$ and $\chi$.

improved by seeding it with a policy learned by Sarsa. In related work (Kalyanakrishnan and Stone, 2007), we demonstrate the effectiveness of Seq on Keepaway (Stone *et al.*, 2005), a popular RL benchmarking task. In Keepaway, the function approximator used is a neural network, whose weights are initially learned using Sarsa(0), and then transferred to CEM (de Boer *et al.*, 2005). It indeed seems very relevant to extend the entire range of experiments we have presented in this article to a complex domain such as Keepaway. The main challenge in such an exercise would be the sheer time taken for running experiments. Nevertheless, we do hope that future research will extend our current set of experiments into more complex and realistic domains.

## 5 Discussion

The extensive suite of experiments reported in Section 4 uncover several interesting patterns characterizing the interaction between problem parameters and method-specific parameters in the context of sequential decision making from experience. In this section, we highlight some of the main questions thereby brought to relevance.

**Generalization:** Our results consistently indicate that generalization and function approximation significantly alter the landscape while evaluating learning algorithms, in particular those from VF. For instance, Section 4.2 presents conclusive evidence that ExpSarsa suffers more severely due to the bias introduced by generalization than either Sarsa or Q-learning. Section 4.6 brings into focus an irregular — yet systematic — pattern in the performance of Sarsa as the tile width $w$ is increased. Interestingly this pattern is correlated with the best eligibility trace settings. To the best of our knowledge, generalization has not been given explicit attention in the context of PS methods. Our results show that generalization can benefit PS methods, too, when the training duration is short.

As motivated in Section 1, generalization is necessary in nearly every practical application of RL. Thus, the importance of understanding its effects on algorithms cannot be understated. In future work, we hope to use our experimental framework to probe more deeply into this subject.

**Optimistic Initialization:** Of special significance among the ramifications of learning with generalization is its effect on the common practice of optimistic initialization. Optimistic initialization (of action values) has long been employed as a mechanism to promote exploration. In the context of finite MDPs, elegant proofs of convergence of VF methods have been derived on the basis of optimistic initial values (Even-Dar and Mansour, 2001; Szita and Lőrincz, 2008). Grześ and Kudenko (2009) provide experimental justification, again on finite (or suitably discretized) MDPs, for schemes that refine the basic optimistic initialization framework.

Our experiments in Section 4.2 convey that optimistic initialization is only effective in the fully tabular case: for $w > 1$, the error introduced into TD updates by high action values invariably degrades the performance of VF methods. A question that arises in response is how we may initialize action values when learning *with* generalization. Research in this direction appears particularly relevant to algorithms that are guaranteed to reach fixed points under linear function approximation (Perkins and Precup, 2003; Sutton *et al.*, 2009; Maei *et al.*, 2010). Which reasonable strategies for setting initial weights would profit such methods the most?

**Meta-learning and Algorithm Portfolio Design:** The term “meta-learning” (Vilalta and Drissi, 2002) describes the enterprise of (a) characterizing the strengths and weaknesses of learning methods vis-à-vis problem characteristics, with the intent of (b) designing adaptive schemes that, given a problem, apply the method best suited for it (Brodley, 1995; Pfahringer *et al.*, 2000). Similarly, “algorithm portfolio” methods (Gomes and Selman, 2001) rely on applying several candidate algorithms to a problem (either in series or in parallel) before identifying the most effective choice or combination. While existing work in these areas has largely been in the context of supervised learning and search (Leyton-Brown *et al.*, 2003; Xu *et al.*, 2008), our article is motivated by the meta-learning problem within sequential decision making.

Our experiments clearly show that there is a need for meta-learning within RL. For example, Figure 11(c) succinctly conveys that Sarsa outperforms CMA-ES when the expressiveness of function approximation is above a certain threshold, but that the opposite is true below the threshold. Indeed our experiments unearth several other strengths and weaknesses of methods within the VF and PS classes. Additionally we provide a hybrid algorithm, Seq, as a useful step towards combining the strengths of these methods. Our results are validated on a parameterized learning problem that is specifically designed to implement a methodology for meta-learning within RL. We believe that this methodology can support the eventual development of effective algorithm portfolio designs for sequential decision making, which currently appears a rather formidable undertaking.

**Automatic Parameter Tuning:** While meta-learning operates at the *macro* scale of choosing between methods, our experiments also underscore the gains obtained at a *micro* scale by tuning method-specific parameters. In this work, we have employed a search technique to optimize method-specific parameters such as learning rates and population sizes. In practice, an agent would need to automatically tune these parameters *while learning*. In the context of PS methods, it is worth repeating that we find existing code for CMA-ES quite adept in automatically setting and tuning several internal parameters in the algorithm. For VF methods, techniques for tuning learning rates (Sutton and Singh, 1994; George and Powell, 2006; Hutter and Legg, 2008) and eligibility traces (Downey and Scanner, 2010) have predominantly been derived and validated for the case of finite MDPs. Our parameterized learning problem serves as an excellent mechanism for prototyping adaptive schemes at more general settings involving function approximation and partial observability.

**Learning and Representation:** With the purpose of solely comparing the "learning" behavior of VF and PS methods — how they adapt a set of weights — in this study, we have forced them to share a fixed, common representation. In particular we adopt a linear function approximation scheme, whose expressiveness and generalization can be carefully controlled. Our results show that VF and PS methods dominate at different settings of these problem parameters.

While this approach facilitates sound experimental methodology, it must be noted that in general the greatest success can be achieved by adapting the representation itself while learning (Whiteson and Stone, 2006). Indeed Cobb and Bock (1994) argue that representations favoring an expert agent might be unfavorable for an agent beginning to learn. Integrating learning with adaptive representation is yet another area of future work that our parameterized learning problem enables. In pursuing such work, we would treat $\chi$ and $w$ as *internal* to the learning agent, rather than extraneous. The agent could potentially adapt these representational aspects by applying methods from feature selection (Kolter and Ng, 2009; Petrik *et al.*, 2010), structure learning (Degris *et al.*, 2006; Diuk *et al.*, 2009) and manifold learning (Mahadevan, 2009).

In taking steps toward the automated application of RL methods to problems, the issues discussed above are all relevant to consider. We hope that future work will make progress along all these directions by extending the ideas presented in this article.

## 6 Related Work

In this section, we discuss related work in the context of parameterized learning problems and empirical evaluations in RL.

In an early article, Cohen and Howe (1988) consider the strong coupling that exists in many disciplines of AI between problem types and method instances. While formulating guidelines for the evaluation of methodological contributions to the field, they argue the need to precisely characterize the set of problems on which a method is expected to be successful, and symmetrically, the approaches that are likely to succeed on a given class of problems. As mentioned in Section 1, Langley (1988) makes a similar observation in the specific context of machine learning.

Parameterized learning problems have been used in the literature to study the effects of factors such as dimensionality and noise. For example, Spall (2003) extensively uses the "Rosenbrock" function while comparing the performance of optimization algorithms in his textbook. The "Sphere" function discussed by Beyer (2000) has served as a standard benchmark for evolutionary algorithms.

The work within the RL literature that is philosophically closest to the contribution of this article is the notion of "generalized environments" proposed by Whiteson *et al.* (2011). Here, too, the authors argue against "environment overfitting", whereby methods tend to get evaluated on problems that favor them, but the broader scope of their applicability, especially their weaknesses, are not easy to gauge. A generalized environment represents a formally defined distribution of environments: the objective is to develop methods that perform well over the entire distribution. Whereas the motivation for generalized environments comes from realistic tasks such as helicopter control and Tetris, the apparatus developed in this article examines the performance of learning methods in a carefully-designed, controllable, abstract setting. Our results underscore that qualitatively different learning methods excel in different regions of the problem space, while teasing apart effects that method-specific parameters introduce.

The empirical approach we adopt in this article to characterize the interactions between learning problems and methods is complemented by theoretical formulations designed with a similar objective. Littman (1993) characterizes agents and environments based on the amount of memory they can use and the length of the horizon for which they seek to optimize rewards. He then considers several interesting classes of problems — for example, those with nonstationary environments — that fit within this formalization. Ratitch and Precup (2003) define environmental properties such as *state transition entropy* and *forward controllability*, and investigate how these properties bear on the exploration strategy of a learning agent. To the best of our knowledge, generalization and function approximation have not been addressed through similar theoretical formulations.

The main difference between our comparative study and others in the RL literature is that our parameterized learning problem enables us to evaluate the effects of individual parameters while keeping others fixed. For example, in most related studies, methods typically use different function approximation schemes, thereby introducing an additional qualification for comparing them. Also, our formulation allows us to control problem parameters continuously along a scale from "high" to "low"; in the studies we shortly list, comparisons are typically between two or three distinct task settings. In this sense, our article answers the call put forth by Togelius *et al.* (2009) for parameterizable benchmarks, and affirms their basic conjectures on the strengths and weaknesses of "ontogenetic" (similar to VF) and "phylogenetic" (similar to PS) methods. In addition our results shed light on hitherto unexplored questions such as the effects of optimistic initialization when used in conjunction with generalization. We proceed to briefly list a number of studies comparing RL algorithms.

Moriarty *et al.* (1999) apply a suite of "Evolutionary Algorithms for Reinforcement Learning" (EARL) to a simple grid-world MDP, and compare results with Q-learning. Policies are represented using lists of rules or neural networks, which are evolved using standard genetic operators. The main conclusion of their study is that EARL is more suited to tasks with large state spaces (but represented compactly), tasks with incomplete state information, and tasks with nonstationary returns. Whiteson *et al.* (2010) undertake a comparative study between Sarsa(0) and NEAT (Stanley, 2004), a policy search method. These methods are compared on the benchmark tasks of Keepaway soccer (Stone *et al.*, 2005) and Mountain Car (Sutton and Barto, 1998). Their findings are that sensor noise affects the final performance of Sarsa(0) more than NEAT, and indeed that stochasticity has the opposite effect, as policy evaluations under NEAT become more noisy.

Heidrich-Meisner and Igel (2008a) compare the natural actor-critic method with CMA-ES on a pole-balancing task. Both methods are "variable metric"; i.e., they are insensitive to linear transformations of the parameter space. The methods achieve comparable results, but CMA-ES is found to be less sensitive to initial values for the policy, which has a small number of parameters. Similar results are registered in the noisy Mountain Car task (Heidrich-Meisner and Igel, 2008b). A more extensive suite of comparisons on single and double pole-balancing tasks (with hidden state) is carried out by Gomez *et al.* (2008). The methods compared are evolutionary algorithms such as CoSyNE, NEAT, ESP, and SANE, along with Q-learning, Sarsa, recurrent policy gradient, random weight guessing, and "Value and Policy Search" (VAPS) (Baird and Moore, 1999). The findings reinforce the expectation that under partial observability, evolutionary algorithms dominate model-free value function-based methods such as Q-learning and Sarsa. As with our Seq algorithm, a number of approaches, both empirical and theoretical, have been proposed to combine the strengths of qualitatively different learning approaches (Guestrin *et al.*, 2002; Konda and Tsitsiklis, 2003; Whiteson and Stone, 2006).

## 7 Summary

A large number of reinforcement learning (RL) tasks we face in the real world cannot be modeled and solved exactly as finite MDPs, which support theoretical guarantees such as convergence and optimality. The objective of learning in these tasks has to be rescaled to realizing policies with "high" expected long-term reward in a "sample efficient" manner, as determined empirically. Consequently it becomes necessary to characterize the performance of different learning *methods* on different *problems*.

As a framework to conduct empirical studies in RL, we introduce *parameterized learning problems*, in which the factors influencing the performance of a learning algorithm can be controlled systematically through targeted studies. The main merits of our experimental methodology are that (a) the designed task and learning framework are easy to understand; (b) we may examine the effect of subsets of problem parameters while keeping others fixed; (c) we can benchmark learned policies against optimal behavior; and (d) the learning process can be executed in a relatively short duration of time, thereby facilitating extensive experimentation.

In particular we design a parameterized learning problem to evaluate the effects of partial observability and function approximation, which characterize a sizeable fraction of realistic RL tasks. On this problem, we evaluate various methods from the classes of on-line value function-based (VF) methods and policy search (PS) methods. Through a series of carefully-designed experiments, we obtain clear patterns separating the learning methods considered. A novel aspect of our study is a search procedure that enables us to find the best method-specific parameters (such as learning rates and population sizes) for a given problem instance. Largely made possible by the relative simplicity of our simulation, the search procedure uncovers interesting patterns relating problem instances and method-specific parameters.

Within the VF class, we find that Sarsa and Q-learning perform better than Expected Sarsa (ExpSarsa) when learning with generalization and function approximation. Within the PS class, we find that CMA-ES and the cross-entropy method (CEM) achieve significantly better performance than a genetic algorithm (GA); CMA-ES is more robust to its parameter settings than CEM. Comparing Sarsa (from VF) and CMA-ES (from PS), we find that the former enjoys a higher speed of learning, and also better asymptotic performance, when the learner is provided an expressive representation. On the other hand, CMA-ES is significantly more robust to severely deficient representations. Both methods suffer noticeably when state noise is added; their relative performance is additionally determined by the width of generalization in the representation.

Our experiments highlight several promising lines of inquiry involving generalization, representation, meta-learning, initial weight settings, and parameter tuning. By contributing a novel evaluation methodology and a preliminary set of results obtained using it, our article is oriented towards ultimately constructing a field guide for the application of RL methods in practice. The experiments we have reported in this article are part of an ongoing study, which we plan to extend to other classes of methods, including model-based and batch methods, actor-critic architectures, and policy gradient techniques.

**Acknowledgements** We are very grateful to the anonymous reviewers of this article, to the special issue editors, and especially to Michael L. Littman, for raising several important questions concerning our experiments and claims. The article has benefited significantly from their advice. We thank Harm van Seijen for sharing several useful insights about the methods compared in Section 4.2.1, and Juhyun Lee for proofreading the article. This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (CNS-0615104 and IIS-0917122), the Office of Naval Research (N00014-09-1-0658), and the Federal Highway Administration (DTFH61-07-H-00030).

## References

Albus, J. S. (1981). *Brains, Behavior and Robotics*. McGraw-Hill.

Åström, K. J. (1965). Optimal control of Markov Processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, **10**, 174–205.

Baird, L. and Moore, A. (1999). Gradient descent for general reinforcement learning. In M. J. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11 (NIPS 1998)*, pages 968–974. MIT Press.

Bakker, B., Zhumatiy, V., Gruener, G., and Schmidhuber, J. (2003). A robot that reinforcement-learns to identify and memorize important previous observations. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, pages 430–435. IEEE.

Banko, M. and Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, pages 26–33. Association for Computational Linguistics.

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, **36**(1–2), 105–139.

Baxter, J. and Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research (JAIR)*, **15**, 319–350.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, $1^{st}$ edition.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Beyer, H.-G. (2000). Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice. *Computer methods in applied mechanics and engineering*, **186**(2–4), 239–267.

Brafman, R. I. and Tennenholtz, M. (2003). R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, **3**, 213–231.

Breiman, L. (2001). Random forests. *Machine Learning*, **45**(1), 5–32.

Brodley, C. E. (1995). Recursive automatic bias selection for classifier construction. *Machine Learning*, **20**(1–2), 63–94.

Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In W. W. Cohen and A. Moore, editors, *Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006)*, pages 161–168. ACM.

Caruana, R., Karampatziakis, N., and Yessenalina, A. (2008). An empirical evaluation of supervised learning in high dimensions. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, pages 96–103. ACM.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth national conference on Artificial intelligence (AAAI 1994)*, pages 1023–1028. AAAI Press.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In W. R. Swartout, editor, *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI 1992)*, pages 183–188. AAAI Press.

Cobb, H. G. (1992). Inductive biases in a reinforcement learner. Technical Report AIC-92-013, Navy Center for Applied Research in Artificial Intelligence, Washington DC, USA.

Cobb, H. G. and Bock, P. (1994). Using a genetic algorithm to search for the representational bias of a collective reinforcement learner. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of the Third Conference on Parallel Problem Solving from Nature (PPSN III)*, pages 576–587. Springer.

Cohen, P. R. and Howe, A. E. (1988). How evaluation guides AI research: The message still counts more than the medium. *AI Magazine*, **9**(4), 35–43.

Crites, R. H. and Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8 (NIPS 1995)*, pages 1017–1023. MIT Press.

Dayan, P. and Sejnowski, T. J. (1994). TD($\lambda$) converges with probability 1. *Machine Learning*, **14**, 295–301.

de Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. (2005). A tutorial on the Cross-Entropy Method. *Annals of Operations Research*, **134**(1), 19–67.

Degris, T., Sigaud, O., and Wuillemin, P.-H. (2006). Learning the structure of factored Markov Decision Processes in reinforcement learning problems. In W. W. Cohen and A. Moore, editors, *Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006)*, pages 257–264. ACM.

Diuk, C., Li, L., and Leffler, B. R. (2009). The Adaptive $k$-Meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In A. P. Danyluk, L. Bottou, and M. L. Littman, editors, *Proceedings of the Twenty-sixth Annual International Conference on Machine Learning (ICML 2009)*, pages 249–256. ACM.

Downey, C. and Scanner, S. (2010). Temporal difference Bayesian model averaging: A Bayesian perspective on adapting lambda. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010)*, pages 311–318. Omnipress.

Even-Dar, E. and Mansour, Y. (2001). Convergence of optimistic and incremental Q-Learning. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pages 1499–1506. MIT Press.

Finnsson, H. and Björnsson, Y. (2008). Simulation-based approach to General Game Playing. In D. Fox and C. P. Gomes, editors, *Proceedings of the Twenty-third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 259–264. AAAI Press.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In L. Saitta, editor, *Proceedings of the Thirteenth International Conference on Machine Learning (ICML 1996)*, pages 148–156. Morgan Kaufmann.

Gabel, T., Riedmiller, M., and Trost, F. (2009). A case study on improving defense behavior in soccer simulation 2D: The NeuroHassle approach. In L. Iocchi, H. Matsubara, A. Weitzenfeld, and Z. Changjiu, editors, *RoboCup 2008: Robot Soccer World Cup XII*, pages 61–72. Springer.

George, A. P. and Powell, W. B. (2006). Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, **65**(1), 167–198.

Gomes, C. P. and Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, **126**(1–2), 43–62.

Gomez, F., Schmidhuber, J., and Miikkulainen, R. (2008). Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, **9**, 937–965.

Gomez, F. J. and Miikkulainen, R. (1999). Solving non-Markovian control tasks with neuro-evolution. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pages 1356–1362. Morgan Kauffman.

Gomez, F. J. and Miikkulainen, R. (2003). Active guidance for a finless rocket using neuroevolution. In E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. F. Miller, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 2084–2095. Springer.

Grześ, M. and Kudenko, D. (2009). Improving optimistic exploration in model-free reinforcement learning. In M. Kolehmainen, P. J. Toivanen, and B. Beliczynski, editors, *Proceedings of the Ninth International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2009)*, pages 360–369. Springer.

Guestrin, C., Lagoudakis, M. G., and Parr, R. (2002). Coordinated reinforcement learning. In C. Sammut and A. G. Hoffman, editors, *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 227–234. Morgan Kauffman.

Guez, A., Vincent, R. D., Avoli, M., and Pineau, J. (2008). Adaptive treatment of epilepsy via batch-mode reinforcement learning. In D. Fox and C. P. Gomes, editors, *Proceedings of the Twenty-third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 1671–1678. AAAI Press.

Hansen, N. (2009). *The CMA Evolution Strategy: A Tutorial*. `http://www.lri.fr/~hansen/cmatutorial.pdf`.

Hansen, N., Niederberger, A. S., Guzzella, L., and Koumoutsakos, P. (2009). A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, **13**(1), 180–197.

Heidrich-Meisner, V. and Igel, C. (2008a). Similarities and differences between policy gradient methods and evolution strategies. In M. Verleysen, editor, *Proceedings of the Sixteenth European Symposium on Artificial Neural Networks (ESANN 2008)*, pages 149–154. D-side Publication.

Heidrich-Meisner, V. and Igel, C. (2008b). Variable metric reinforcement learning methods applied to the noisy mountain car problem. In S. Girgin, M. Loth, R. Munos, P. Preux, and D. Ryabko, editors, *Recent Advances in Reinforcement Learning: Eighth European Workshop (EWRL 2008)*, pages 136–150. Springer-Verlag.

Heidrich-Meisner, V. and Igel, C. (2009). Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In A. P. Danyluk, L. Bottou, and M. L. Littman, editors, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML 2009)*, pages 401–408. ACM.

Hutter, M. and Legg, S. (2008). Temporal difference updating without a learning rate. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 705–712. MIT Press.

İpek, E., Mutlu, O., Martínez, J., and Caruana, R. (2008). Self-optimizing memory controllers: A reinforcement learning approach. In *Proceedings of the Thirty-fifth International Symposium on Computer Architecture (ISCA 2008)*, pages 39–50. IEEE Press.

James, M. R. and Singh, S. (2009). SarsaLandmark: an algorithm for learning in POMDPs with landmarks. In C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, editors, *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 585–591. IFAAMAS.

Kakade, S. (2001). A natural policy gradient. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pages 1531–1538. MIT Press.

Kalyanakrishnan, S. and Stone, P. (2007). Batch reinforcement learning in a complex domain. In E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, editors, *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, pages 650–657. IFAAMAS.

Kearns, M. and Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, **49**(2–3), 209–232.

Kohl, N. and Stone, P. (2004). Machine learning for fast quadrupedal locomotion. In D. L. McGuinness and G. Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, pages 611–616. AAAI Press.

Kolter, J. Z. and Ng, A. Y. (2009). Regularization and feature selection in least-squares temporal difference learning. In A. P. Danyluk, L. Bottou, and M. L. Littman, editors, *Proceedings of the Twenty-sixth Annual International Conference on Machine Learning (ICML 2009)*, pages 521–528. ACM.

Konda, V. R. and Tsitsiklis, J. N. (2003). On actor-critic algorithms. *SIAM Journal on Control and Optimization*, **42**(4), 1143–1166.

Kwok, C. and Fox, D. (2004). Reinforcement learning for sensing strategies. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (ICRA 2004)*, pages 3158–3163. IEEE Press.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, **4**, 1107–1149.

Langley, P. (1988). Machine learning as an experimental science. *Machine Learning*, **3**(1), 5–8.

Langley, P. and Pendrith, M. (1998). Symposium on applications of reinforcement learning: Final report for NSF Grant IIS-9810208. Technical report, Institute for the Study of Learning and Expertise.

Lee, H., Shen, Y., Yu, C.-H., Singh, G., and Ng, A. Y. (2006). Quadruped robot obstacle negotiation via reinforcement learning. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*, pages 3003–3010. IEEE Press.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., and Shoham, Y. (2003). Boosting as a metaphor for algorithm design. In F. Rossi, editor, *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP 2003)*, pages 899–903. Springer.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, **8**(3–4), 293–321.

Lin, L.-J. and Mitchell, T. M. (1993). Reinforcement learning with hidden states. In J.-A. Meyer, H. L. Roitblat, and S. W. Wilson, editors, *From animals to animats 2*, pages 271–280. MIT Press.

Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, **2**(4), 285–318.

Littman, M. L. (1993). An optimization-based categorization of reinforcement learning environments. In J.-A. Meyer, H. L. Roitblat, and S. W. Wilson, editors, *From Animals to Animats 2*, pages 262–270. MIT Press.

Loch, J. and Singh, S. (1998). Using eligibility traces to find the best memoryless policy in Partially Observable Markov Decision Processes. In J. W. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)*, pages 323–331. Morgan Kauffman.

Maei, H. R., Szepesvári, C., Bhatnagar, S., and Sutton, R. S. (2010). Toward off-policy learning control with function approximation. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010)*, pages 719–726. Omnipress.

Mahadevan, S. (2009). Learning representation and control in Markov Decision Processes: New frontiers. *Foundations and Trends in Machine Learning*, **1**(4), 403–565.

McCallum, A. K. (1996). *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. thesis, Computer Science Department, University of Rochester.

McCallum, R. A. (1995). Instance-based utile distinctions for reinforcement learning with hidden state. In A. Prieditis and S. J. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 387–395. Morgan Kauffman.

Melo, F. S., Meyn, S. P., and Ribeiro, M. I. (2008). An analysis of reinforcement learning with function approximation. In *Proceedings of the Twenty-fifth international conference on Machine learning (ICML 2008)*, pages 664–671. ACM.

Metzen, J. H., Edgington, M., Kassahun, Y., and Kirchner, F. (2008). Analysis of an evolutionary reinforcement learning method in a multiagent domain. In L. Padgham, D. C. Parkes, J. Müller, and S. Parsons, editors, *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 291–298. IFAAMAS.

Moriarty, D. E., Schultz, A. C., and Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artifical Intelligence Research*, **11**, 241–276.

Munos, R. and Moore, A. W. (2002). Variable resolution discretization in optimal control. *Machine Learning*, **49**(2–3), 291–323.

Nevmyvaka, Y., Feng, Y., and Kearns, M. (2006). Reinforcement learning for optimized trade execution. In W. W. Cohen and A. Moore, editors, *Proceedings of the Twenty-third international conference on Machine learning (ICML 2006)*, pages 673–680. ACM.

Ng, A. Y., Kim, H. J., Jordan, M. I., and Sastry, S. (2004). Autonomous helicopter flight via reinforcement learning. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16 (NIPS 2003)*. MIT Press.

Ormoneit, D. and Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning*, **49**(2–3), 161–178.

Peng, J. and Williams, R. J. (1996). Incremental multi-step Q-Learning. *Machine Learning*, **22**(1–3), 283–290.

Perkins, T. J. and Pendrith, M. D. (2002). On the existence of fixed points for Q-Learning and Sarsa in partially observable domains. In C. Sammut and A. Hoffman, editors, *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 490–497. Morgan Kauffman.

Perkins, T. J. and Precup, D. (2003). A convergent form of approximate policy iteration. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pages 1595–1602. MIT Press.

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, **21**(4), 682–697.

Petrik, M., Taylor, G., Parr, R., and Zilberstein, S. (2010). Feature selection using regularization in approximate linear programs for Markov Decision Processes. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010)*, pages 871–878. Omnipress.

Pfahringer, B., Bensusan, H., and Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In P. Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 743–750. Morgan Kaufmann.

Pineau, J., Gordon, G. J., and Thrun, S. (2006). Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, **27**, 335–380.

Precup, D., Sutton, R. S., and Dasgupta, S. (2001). Off-policy temporal difference learning with function approximation. In C. E. Brodley and A. P. Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning (ICML*

*2001)*, pages 417–424. Morgan Kaufmann.

Quinlan, J. R. (1996). Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI 1996)*, pages 725–730. AAAI Press.

Ratitch, B. and Precup, D. (2003). Using MDP characteristics to guide exploration in reinforcement learning. In N. Lavrac, D. Gamberger, L. Todorovski, and H. Blockeel, editors, *Proceedings of the Fourteenth European Conference on Machine Learning (ECML 2003)*, pages 313–324. Springer.

Rottmann, A., Plagemann, C., Hilgers, P., and Burgard, W. (2007). Autonomous blimp control using model-free reinforcement learning in a continuous state and action space. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, pages 1895–1900. IEEE.

Rummery, G. A. (1995). *Problem Solving with Reinforcement Learning*. Ph.D. thesis, Cambridge University Engineering Department, Cambridge, UK.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. CUED/F-INFENG/TR 166, Cambridge University Engineering Department.

Sabes, P. (1993). Approximating Q-values with basis function representations. In M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 264–271. Lawrence Erlbaum.

Sherstov, A. A. and Stone, P. (2005). Function approximation via tile coding: Automating parameter choice. In J.-D. Zucker and L. Saitta, editors, *Proceedings of the Sixth International Symposium on Abstraction, Reformulation and Approximation (SARA 2005)*, pages 194–205. Springer.

Silver, D., Sutton, R. S., and Müller, M. (2007). Reinforcement learning of local shape in the game of Go. In M. M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1053–1058. IJCAI.

Singh, S. and Bertsekas, D. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In M. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9 (NIPS 1996)*, pages 974–980. MIT Press.

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, **38**(3), 287–308.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, **22**(1–3), 123–158.

Singh, S. P., Jaakkola, T., and Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision processes. In W. W. Cohen and H. Hirsch, editors, *Proceedings of the Eleventh International Conference on Machine Learning (ICML 1994)*, pages 284–292. Morgan Kauffman.

Spall, J. C. (2003). *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Hoboken, NJ, USA.

Stanley, K. O. (2004). *Efficient Evolution of Neural Networks Through Complexification*. Ph.D. thesis, Department of Computer Sciences, University of Texas at Austin. Published as technical report AI-TR-04-314.

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

Strehl, A. L. and Littman, M. L. (2005). A theoretical analysis of model-based interval estimation. In L. De Raedt and S. Wrobel, editors, *Proceedings of the Twenty-second*

*International Conference on Machine Learning (ICML 2005)*, pages 856–863. ACM.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., and Littman, M. L. (2006). PAC model-free reinforcement learning. In W. W. Cohen and A. Moore, editors, *Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006)*, pages 881–888. ACM.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, **3**(1), 9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on Approximating Dynamic Programming. In B. W. Porter and R. J. Mooney, editors, *Proceedings of the Seventh International Conference on Machine Learning (ICML 1990)*, pages 216–224. Morgan Kauffman.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8 (NIPS 1995)*, pages 1038–1044. MIT Press.

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

Sutton, R. S. and Singh, S. P. (1994). On bias and step size in temporal-difference learning. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pages 91–96, New Haven, CT, USA. Center for Systems Science, Yale University.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 1057–1063. MIT Press.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In A. P. Danyluk, L. Bottou, and M. L. Littman, editors, *Proceedings of the Twenty-sixth Annual International Conference on Machine Learning (ICML 2009)*, pages 993–1000. ACM.

Suttorp, T., Hansen, N., and Igel, C. (2009). Efficient covariance matrix update for variable metric evolution strategies. *Machine Learning*, **75**(2), 167–197.

Szita, I. and Lőrincz, A. (2006). Learning Tetris using the Noisy Cross-Entropy Method. *Neural Computation*, **18**(12), 2936–2941.

Szita, I. and Lőrincz, A. (2007). Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man. *Journal of Artificial Intelligence Research*, **30**, 659–684.

Szita, I. and Lőrincz, A. (2008). The many faces of optimism: a unifying approach. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML 2008)*, pages 1048–1055. ACM.

Tedrake, R., Zhang, T. W., and Seung, H. S. (2004). Stochastic policy gradient reinforcement learning on a simple 3D biped. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, pages 2849–2854. IEEE.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, **8**(3–4), 257–277.

Tesauro, G., Jong, N. K., Das, R., and Bennani, M. N. (2007). On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, **10**(3), 287–299.

Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 255–263. Lawrence Erlbaum.

Togelius, J., Schaul, T., Wierstra, D., Igel, C., Gomez, F., and Schmidhuber, J. (2009). Ontogenetic and phylogenetic reinforcement learning. *Künstliche Intelligenz*, pages 30–33.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, **42**, 674–690.

van Seijen, H., van Hasselt, H., Whiteson, S., and Wiering, M. (2009). A theoretical and empirical analysis of Expected Sarsa. In *Proceedings of the 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2009)*, pages 177–184. IEEE.

Vilalta, R. and Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial Intelligence Review*, **18**(2), 77–95.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, King's College, Cambridge, UK.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, **8**(3–4), 279–292.

Whitehead, S. D. and Ballard, D. H. (1991). Learning to perceive and act by trial and error. *Machine Learning*, **7**(1), 45–83.

Whiteson, S. and Stone, P. (2004). Adaptive job routing and scheduling. *Engineering Applications of Artificial Intelligence*, **17**(7), 855–869. Special issue on Autonomic Computing and Automation.

Whiteson, S. and Stone, P. (2006). Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, **7**, 877–917.

Whiteson, S., Taylor, M. E., and Stone, P. (2010). Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, **21**(1), 1–35.

Whiteson, S., Tanner, B., Taylor, M. E., and Stone, P. (2011). Protecting against evaluation overfitting in empirical reinforcement learning. In *Proceedings of the 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2011)*, pages 120–127. IEEE.

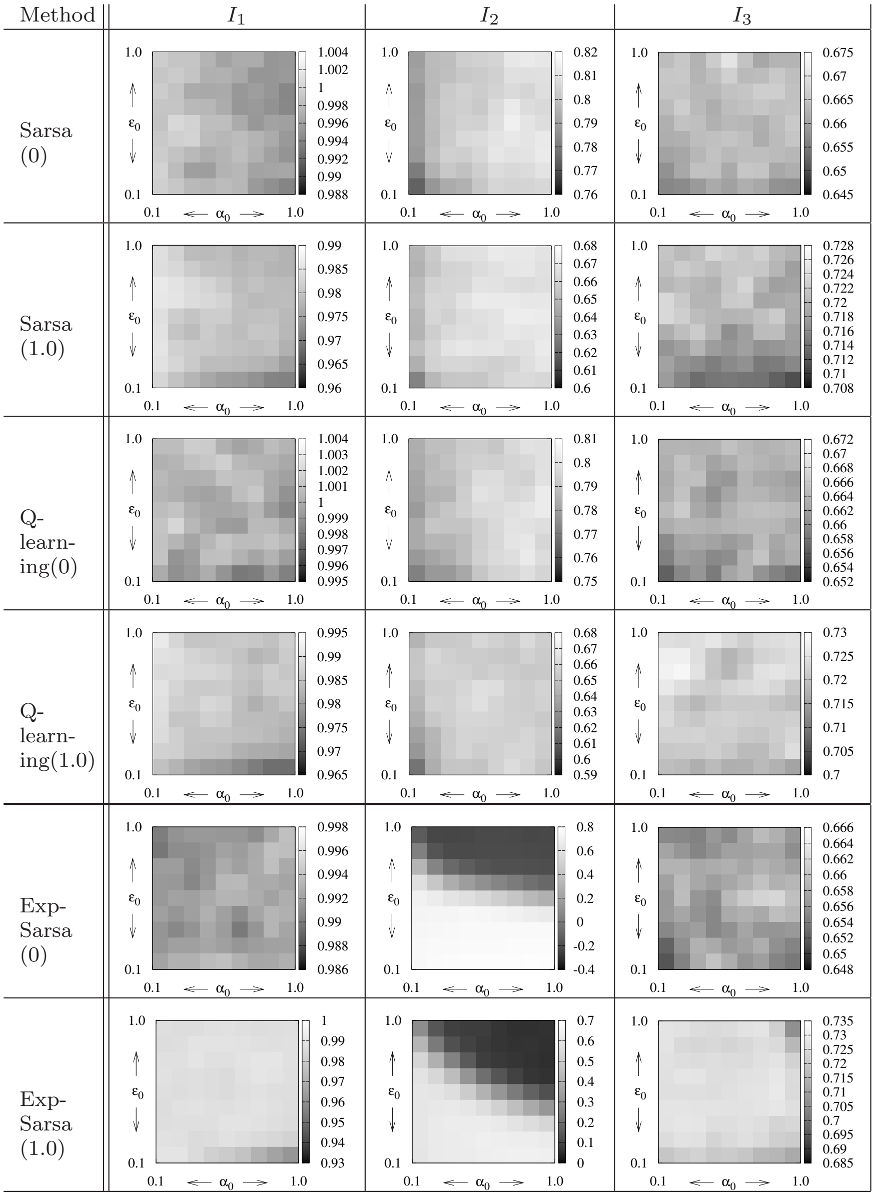
Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, **1**(1), 67–82.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, **32**, 565–606.

Zhang, W. and Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pages 1114–1120. Morgan Kauffman.

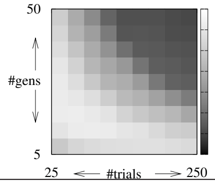
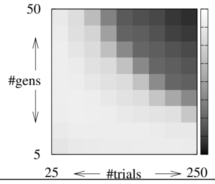
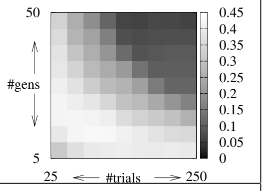
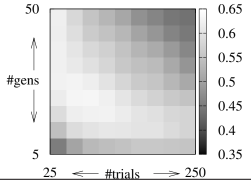
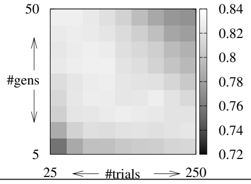
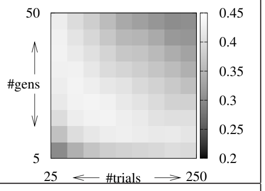
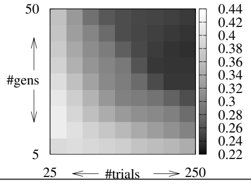
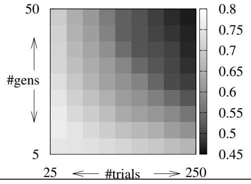
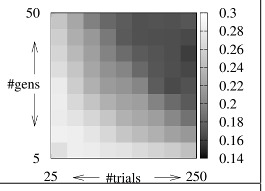
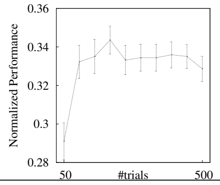
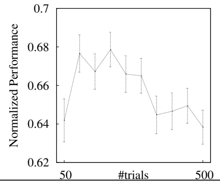
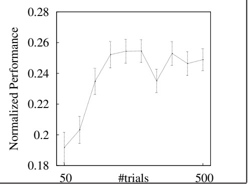
## A Effect of $\alpha_0$ and $\epsilon_0$ on Methods in VF

The plots below show the effect of the initial learning rate $\alpha_0$ and exploration rate $\epsilon_0$ on the learned performance of different methods in VF. Intensity ranges are indicated to the right of each plot. Under $I_1$, all the methods use $\theta_0 = 10$; under $I_2$, they use $\theta_0 = 0.5$; under $I_3$, they use $\theta_0 = 5$. We observe that under instance $I_1$, Sarsa(0), Q-learning(0), and ExpSarsa(0) all achieve normalized performance values close to 1. Under $I_2$, ExpSarsa(0) and ExpSarsa(1) perform best at low values of $\alpha_0$ and $\epsilon_0$.



## B Effect of $\#trials$ and $\#gens$ on Methods in PS

The plots below show the effect of $\#trials$ and $\#gens$ on the performance of different policy search methods. Intensity ranges are indicated to the right of each plot. Under RWG, only $\#trials$ is varied ($\#gens = \frac{50000}{\#trials}$); the mean performance is plotted with one standard error. The methods all show noticeable variance in performance over the ranges plotted, underscoring the need for careful tuning. For all methods, the highest performance is under problem instance $I_2$. We see that CMA-ES performs better on average, over the parameter ranges plotted, than both CEM and GA.

| Method | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|
| CEM |  |  |  |
| CMA-ES |  |  |  |
| GA |  |  |  |
| RWG |  |  |  |