Efficient Exploration In Reinforcement Learning

Sebastian B. Thrun

January 1992
Technical report CMU-CS-92-102
School of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213-3890
e-mail: thrun@cs.cmu.edu

Abstract

Exploration plays a fundamental role in any active learning system. This study evaluates the role of exploration in active learning and describes several local techniques for exploration in finite, discrete domains, embedded in a reinforcement learning framework (delayed reinforcement).

This paper distinguishes between two families of exploration schemes: *undirected* and *directed exploration*. While the former family is closely related to random walk exploration, directed exploration techniques memorize exploration-specific knowledge which is used for guiding the exploration search. In many finite deterministic domains, any learning technique based on *undirected* exploration is inefficient in terms of learning time, i.e. learning time is expected to scale exponentially with the size of the state space (Whitehead, 1991b). We prove that for all these domains, reinforcement learning using a *directed* technique can always be performed in polynomial time, demonstrating the important role of exploration in reinforcement learning. (The proof is given for one specific directed exploration technique named *counter-based exploration.*)

Subsequently, several exploration techniques found in recent reinforcement learning and connectionist adaptive control literature are described. In order to trade off efficiently between exploration and exploitation – a trade-off which characterizes many real-world active learning tasks – combination methods are described which explore and avoid costs simultaneously. This includes a selective attention mechanism, which allows smooth switching between exploration and exploitation.

All techniques are evaluated and compared on a discrete reinforcement learning task (robot navigation). The empirical evaluation is followed by an extensive discussion of benefits and limitations of this work.

Keywords: active learning, reinforcement learning, exploration, Markov decision problems, robot navigation.

1 Introduction

Whenever a learning system learns to control an unknown environment, two opposing objectives have to be combined. On the one hand, in order to identify a (sub-)optimal controller, the environment must be sufficiently explored. For example a robot facing an unknown environment has to spend time moving around and acquire knowledge of its environment. On the other hand, experience made during learning must also be considered for action selection in order to minimize the costs of learning (e.g. in terms of negative reward). E.g. although a robot has to explore its environment, it should avoid collisions with obstacles, once it received some negative reward for collisions. Thus for efficient learning, actions should be generated such that a) the environment is explored and b) pain is avoided. This fundamental trade-off between exploration and exploitation demands efficient exploration capabilities, maximizing the effect of learning while minimizing the costs of exploration and learning time.

This paper addresses the issue of *efficient exploration* in a reinforcement learning framework. Several exploration techniques found in recent literature are investigated and their efficiency is compared in terms of learning time and learning costs (negative reward). We group exploration strategies into two categories: *undirected exploration* and *directed exploration*. While undirected techniques, e.g. random walk exploration and Boltzmann distributed exploration (to be defined later), utilize no exploration-specific knowledge and ensure exploration by merging randomness into action selection, directed techniques rely on knowledge about the learning process itself, allowing for exploring in a more directed manner.

This paper is organized in two parts. It theoretically evaluates the impact of exploration knowledge on the complexity of learning for environments, where no a priori knowledge is provided to guide the exploration (zero-knowledge environments). In particular, the efficiency and thus superiority of one particular directed exploration rule named *counter-based exploration* over all indirect techniques is proved. Subsequently, several undirected and directed exploration techniques, including most of those found in literature, are described and carefully evaluated on a two-dimensional robot navigation task. Finally, based on the view of exploration and exploitation as establishing different and mostly opposing behaviors, a selective attention mechanism for dynamically switching between exploration and exploitation is described and shown to further improve the efficiency of directed exploration in the robot navigation task.

Although the issue of efficient exploration is very general in nature, we relate this paper to one particular learning technique, namely reinforcement learning. Both theoretical and experimental results are inspired by tasks where minimal knowledge is given in advance, since such tasks demonstrate the effects of exploration best. Chapter 8 gives an extensive discussion about limitations of this work and relates exploration to more general active learning tasks.

2 The role of exploration in active learning

Consider the one-dimensional n-state deterministic world depicted in Fig. 1. Each time tick, the robot, starting at the left, has three actions to chose from, two of which carry the robot to the left and one of which carries it to the right (except on the leftmost and rightmost states). For simplification, assume the robot had access to an action model, which is a predictive function from states-action pairs to next states. The goal is to get to the rightmost state, and positive reward is only received upon this goal state. This scenario defines a typical reinforcement learning task: the task is to identify a reactive controller, carrying the robot to its goal with as few steps as possible. Although this task is simple, it serves well for illustrating the fundamental role of the exploration strategy in the complexity of learning.

Let us begin by evaluating the complexity of the most basic and uninformed exploration technique, namely random exploration[1]. The complexity of random exploration is of interest for a much richer class of exploration techniques, the class of undirected techniques. As we will see below, every undirected technique performs random walk in the *first trial* (by first trial we mean the sequence of actions that leads to the goal state, where non-zero reward is received for the first time). In the particular task at hand, the definition of actions makes it more likely for the robot to get away from the goal state than to get closer, given that actions are selected with uniform probability. If random actions are more likely to increase the distance to the goal rather than decreasing it, a fundamental theorem about the complexity of reinforcement learning with undirected exploration by Whitehead (Whitehead, 1991a), (Whitehead, 1991b) becomes applicable:

Theorem (Whitehead): In any finite homogeneous problem solving task[2]*, the expected time using undirected exploration (such as random walk exploration) required to identify an optimal policy is bounded below by an expression exponential in the depth of the state space l, if the number of actions leading away from the goal state is larger than the number of actions leading closer to it.*[3]

The theorem is proved using a standard technique from probability theory, evaluating the expected

[1] Here and in turn we will use the term *random exploration* to refer to exploration where actions are generated randomly with uniform probability distribution.

[2] This condition is used in Whitehead's proof for projecting general state spaces to one-dimensional state spaces – Some of the conditions could easily be extended yielding a more general theorem. According to Whitehead (Whitehead, 1991b), a task is a *homogeneous problem solving task*, if a) each trial begins in a designated start state s_{start}, b) each trail ends when the system gives up or upon reaching a designated goal state s_{goal}, c) the system receives a reward only upon entering the goal state, d) there is no prior knowledge which could be used to guide the search of the goal, e) the state space is 1-step invertible, i.e. each action has its inverse action, and f) the state space is uniformly l-bounded (l is called *depth*), i.e. optimal policies take less than l steps, actions change the *distance* to the goal state only by -1,0, or 1, and for each state (except on the border) there are equally many actions which increase the distance by 1, equally many actions which decrease the distance by 1, and equally many actions which leave the distance unchanged.

[3] This result is especially surprising since one of the preconditions in his proof is that the world is *1-step invertible*, i.e. every action has its inverse. With the knowledge of how to invert actions, efficient search techniques for finding the goal state become applicable, e.g. depth-first search.

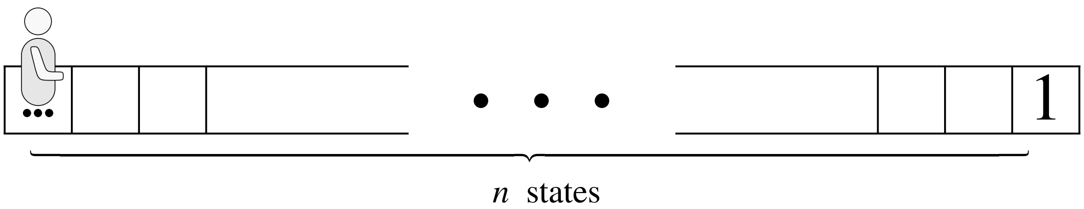

Figure 1: A one-dimensional navigation task. If actions a_1 and a_2 carry the robot to the left and action a_3 carries it to the right, random walk exploration, and as we will see all *undirected* exploration techniques, are expected to take $O(2^n)$ time, *while counter-based* exploration takes $O(n)$ time for finding the goal state.

complexity for finding a goal state by random walk. Any learning system with an exploration technique performing random walk in the first trial is thus expected to be at least exponential in the depth[4] of the state space, e.g. the widely used *Boltzmann-distributed exploration* (Barto *et al.*, 1991), (Watkins, 1989), (Lin, 1991b), (Lin, 1992), (Singh, 1992), (Sutton, 1990) described below. As Whitehead points out, he believes that this theorem holds for many non-homogeneous and non-deterministic cases, and so do we. The crucial condition for exponential complexity is that by taking an action randomly we expect the "distance" to the goal to increase or, more general, to worsen the situation. In this case, a random walk is expected to take exponential time in the depth of the state space. Since the size of state space n is usually polynomial in its depth, the expected time is also exponential in n itself. We believe that many real-world environments fulfill the condition that random actions are expected to worsen the situation in this sense. This is especially the case if the number of possible actions is large. We therefore believe that this theorem covers a rather large class of reinforcement learning tasks, all of which become *intractable* using any kind of undirected exploration.

Whitehead's theorem is applicable to the example demonstrated in Fig. 1. The depth of the state space is n (as is the size of the state space itself) and therefore the expected time to find the goal state with random exploration is exponential in n, and so is reinforcement learning. In this particular example, the expected time for finding the goal is given by $3(2^n - n - 1)$. E.g., with $n = 50$, we expect the agent to take $3.38 \cdot 10^{15}$ steps for finding the goal – and each additional state roughly doubles this number (Howard, 1960), (Nafeh, 1976).

We will now present a *polynomial* worst case bound for any deterministic environment using a *directed exploration* technique, namely *counter-based exploration.* In our example and as we will see in general state-spaces, there is a very simple directed exploration rule which allows for finding the goal state efficiently. Assume we had an action model (we will drop this condition later). The basic principle of counter-based exploration is to count the occurrences of each state. Initially these counters are set to $(1, 0, \ldots, 0)$ for our example, since only the initial, leftmost state has occurred

[4] according to Whitehead, the *depth* of a state space measures the maximal distance to the goal state. See Appendix for a definition

so far. By applying the simple rule "go to the least occurred adjacent state", the agent will explore the state right next to it, and the resulting counter will be $(1, 1, 0, \dots, 0)$. By iteratively applying this rule, the agent will take exactly $n-1$ steps to find the goal, thus the complexity of the first trial is linear in the size of the state space n. E.g. with $n = 50$ as above, the expected time for finding the goal state is 49. In our example, an optimal policy can be identified by browsing the state-space $n-1$ times using standard reinforcement learning methods, and thus the complexity of reinforcement learning is in $O(n^2)$. In conclusion, in this simple example any undirected exploration rule is expected to take exponential number of actions for reinforcement learning, while counter-based exploration and in fact all directed exploration techniques discussed in this paper are always done with $O(n^2)$ actions, both measured in the number of states in the environment, n. This observation can be extended to a general complexity bound, stating that counter-based exploration takes always polynomial exploration time in all ergodic[5] deterministic state spaces.

Theorem 1 (Complexity of the first trial using a complete action model): There is a local counter-based exploration rule, by which the number of actions required for the first trial in any finite ergodic deterministic Markov decision task using a complete action model is bounded above by $O(ln^2)$.

Here l denotes the *depth* of the state space with $l \leq n$. The proof of this and other theorems can be found in the appendix. Note that Theorem 1 covers all environments Whitehead's theorem is valid for. Moreover, this result does not depend on the number of valid actions at each state.

Obviously, the action model plays an important role in this counter-based exploration rule. Unlike there, in many reinforcement learning tasks there is no action model given in advance. It turns out that even in this more general case, the time complexity of counter-based exploration is always polynomial in the size of the state space n:

Theorem 2 (Complexity of the first trial without an action model): There is a local counter-based exploration rule, by which the number of actions required for the first trial in any finite ergodic deterministic Markov decision task is bounded above by $O(dln^2)$.

Here d denotes the maximum number of actions valid at each state. Using this theorem, the worst case complexity of *reinforcement learning* using counter-based exploration is bounded polynomial in the number of states n and the maximum number of actions d:

Theorem 3 (Complexity of finite deterministic, semi-bounded reinforcement learning tasks): There is a local, counter-based exploration rule, by which the number of actions required for finding an optimal policy (controller) in any finite ergodic deterministic, semi-bounded Markov decision task with a goal state is bounded above by $O((d^2l + l^2)n^2)$. If a complete action model is available in advance, the complexity reduces to $O(l^2n^2)$.

This theorem emphasizes the importance of exploration in reinforcement learning tasks: With

[5] A state space is *ergodic*, if for all states s and s' there is a possibility to get from s to s' with non-zero probability. Note that any forward-connected state space is ergodic, if there is a designated *reset*-action.

undirected exploration one expects at least exponential time in many cases, while by using counter-based exploration an optimal controller can be identified always in polynomial time. All theorems claimed in this section (except Whitehead's theorem) are worst case bounds, i.e. the optimal controller is always found within this time. However, in order to derive worst case bounds one has to assume that the environment is deterministic. It is in principle impossible to state a similar result for all non-deterministic environments, since it can be proven that there are stochastic environments at which any exploration and learning technique is at least exponential (see Appendix). But these environments are malicious in the sense that even an optimal controller is expected to take exponential time for entering the goal state. We believe that Theorem 3 bounds the expected learning time in several non-malicious stochastic environments in which no a priori knowledge is available, and the simulation results presented below demonstrate this. However, little is known about the complexity of directed exploration in non-malicious stochastic environments.

3 The exploration scene

So far, we addressed the complexity of reinforcement learning using directed and undirected exploration techniques. In the theoretical results, we neglected a fundamental principle of various reinforcement learning tasks: Usually there is a *trade-off between exploration and exploitation.* Whenever a controller learns to control its environment, two opposing principles have to be combined: exploration (long-term optimization) and avoiding negative rewards (short-term optimization). Let us demonstrate this with two intuitive examples: If a robot is to optimize its behavior, we want it to use its partial knowledge already during learning in order to avoid bad rewards, e.g. for preventing crashes against obstacles. A child, once having burned itself, will avoid doing this again although it continues exploration and learning.

Fig. 2 shows a personal overview over the exploration scene in reinforcement learning and connectionist adaptive control. The interaction of a learning controller and its environment is characterized by the trade-off between exploration and exploitation. There are basically two techniques for exploration: *undirected exploration* and *directed exploration.*

As defined above, **undirected exploration** relys only on knowledge related to optimal control (e.g. utility estimates), but does not utilize any exploration-specific knowledge about the learning process itself. If we think in terms of a learning phase and a performance phase, this knowledge is best characterized by the fact that it is used for control even after learning. The most uninformed undirected exploration technique is the random walk (Nguyen and Widrow, 1989), (Anderson, 1986), (Mozer and Bachrach, 1989), (Bachrach and Mozer, 1991), (Jordan, 1989), (Jordan and Jacobs, 1990), (Mel, 1989), (Munro, 1987), (Thrun *et al.*, 1991) which completely ignores costs and negative rewards from the environment. As a result of Whitehead's Theorem and as we will demonstrate, this exploration technique is even in scenarios where costs do not matter inferior to other exploration techniques. Other undirected exploration techniques rely on optimal control knowledge only – hence the only way to take exploitation into account is by using this knowledge to make "better" actions more likely, while exploring stochastically. The term "undirected" is due to this observation: exploration is ensured only by randomness. Exploration by modifyed probability distributions is typically found in reinforcement learning literature: The probability distribution for action selection is drawn by the utility estimate of each action. We will later present two explicit members of this exploration technique class: Boltzmann distributions (Barto *et al.*, 1991), (Watkins, 1989), (Lin, 1991b), (Lin, 1992), (Singh, 1992), (Sutton, 1990) and semi-uniform distributions (Whitehead and Ballard, 1991), (Mahadevan and Connell, 1990), (Mahadevan and Connell, 1991).

Directed exploration on the other hand does utilize further knowledge of the learning process (Moore, 1990), (Moore, 1991), (Kaelbling, 1990), (Sutton, 1990), (Schmidhuber, 1991), (Thrun and Möller, 1991), (Thrun and Möller, 1992). This exploration-specific knowledge typically cannot be used for controlling the environment and is thus useless after learning. We assume throughout this paper that this knowledge does not exceed the complexity of the knowledge stored for control, and

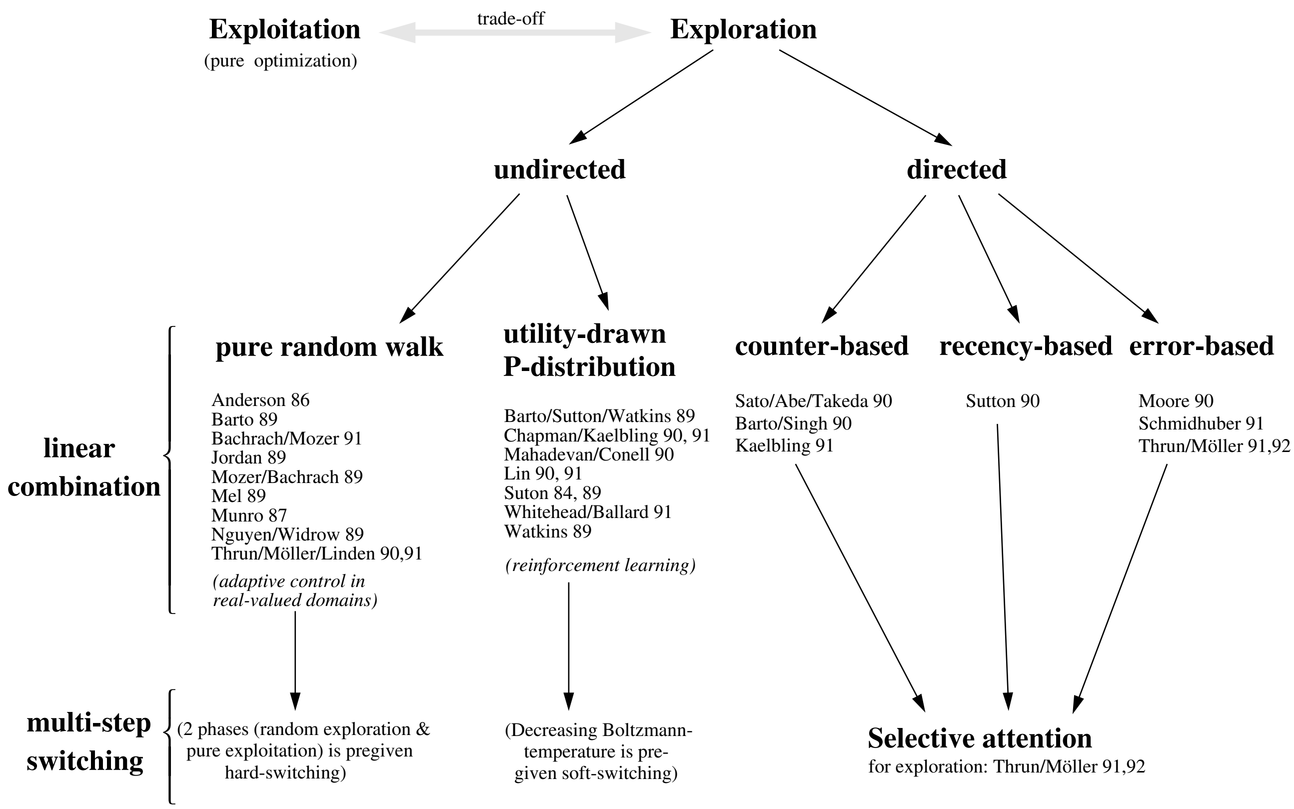

Figure 2: The exploration scene (see text)

that the time for action selection does not suffer from considering this knowledge by more than a constant factor. The term "directed" stems from the fact that – except in some ambiguous cases – the agent directly explores those states which promise to maximize the improvement gain. Unlike undirected exploration, directed techniques can resign the randomness in action selection. We already discussed a basic way of exploring in a directed manner, namely counter-based exploration. Besides this (plus a straightforward extension of counter-based exploration) we will describe two other principles for directed exploration both characterized by the knowledge and the underlying heuristics used for exploration: Error-based exploration bases on estimation of errors of a controller. Such techniques try to provoke states where the controller has previously shown a high prediction error, in order to maximize the knowledge gain. (Schmidhuber, 1991), (Thrun and Möller, 1991), (Thrun and Möller, 1992). Error-based exploration is highly heuristic in nature, since an accurate estimation of the current error is difficult to obtain in most domains. The third directed exploration technique, *recency-based exploration*, is based on the view of the learning system as a dynamical system, which changes its accuracy over time. Recency-based exploration measure the time each state has not occurred, and tries to provoke states which occurred *less recently* (Sutton, 1990). The underlying assumption is that the control knowledge of a state gets worse by the time it is not updated.

Clearly all directed exploration rules are heuristic in nature, but as we saw in the previous section such heuristics may prevent from exponential learning time in many cases. With respect to reinforcement learning it is important to mention that we use a reinforcement learning scheme very similar to Sutton's AHC algorithm (Sutton, 1984). Here each state s is evaluated by a *utility*-value $V(s)$, measuring the future reward one expects if the environment is in state s. However, there is a second important framework for reinforcement learning, namely Q-Learning by Watkins (Watkins, 1989), which differs from AHC in that utility-values are associated with state-action pairs rather than states only. Q-Learning has empirically found to outperform AHC by some authors (Lin, 1992), (Sutton, 1990), although it usually blows up the search space for learning. On the other hand, it is independent of the action selection policy during learning. Given that the state space is sufficiently explored, Q-Learning has proven to always converge to the optimal policy (Watkins, 1989), (Barto and Singh, 1990), but there is no equivalent proof for AHC. However, the extension of the exploration techniques presented here to Q-Learning is not too difficult. Counters or recency-values are then associated to state-action pairs rather than states, as are utility-values. We will address the influence of exploration on AHC- and Q-learning in the discussion.

4 Reinforcement learning

Although above discussion as well as the simulations are very much related to particular reinforcement learning tasks, namely tasks with a designated goal state (absorbing state) such as many adaptive robot navigation tasks studied in reinforcement learning literature, we will embed exploration into a more general framework. Reinforcement learning has often been studied by means of Markov decision problems, where one tries to identify a controller which minimizes costs over time. We will now give a more precise description of what we mean by environments, deterministic environments, policies, utilities, and reinforcement learning. Excellent and more detailed introductions may be found in (Barto *et al.*, 1990), (Barto *et al.*, 1991).

A *Markovian environment* (or simple: *environment*, *world*) is a stochastic mapping from states and actions to subsequent states and rewards. Each time an action is performed on the environment, the environmental state is changed. In addition, an *(immediate) reward* is generated evaluating the current state. The sets of states and actions are finite, but the set of rewards may not. The transition function yielding next states and rewards, subsequently referred to as *state-transition function* and *reward function*, respectively, is probabilistic in nature. Suppose the actual environmental state is x and the action performed by the controller is a. We denote the probability of the next state being y by $P_{xy}(a)$ for any state y and the reward at state y by $r(y)$.[6] A Markovian environment can be completely described by its one-step transition probabilities for states $\{P_{xy}(a)\}_{x,y,a}$, the reward function $\{r(y)\}_y$, and the current state s. We assume that these transition probabilities are fixed throughout time, i.e. the environment is *static*. Furthermore we make the important assumption that the complete state is observable and input to the learning system.[7] A special and important case of a Markovian environment is a *deterministic environment*. Here the state transition function is a deterministic and fully predictable function, as is the reward function. In terms of a Markovian environment, an environment is deterministic, if and only if for all states x and all actions a there is a state y with $P_{xy}(a) = 1$ and for all states $y' \neq y$: $P_{xy'}(a) = 0$. Note that in the complexity theorems we also assume that the environment is *ergodic*, i.e. for any state s, it is possible to change subsequently the environmental state to every arbitrary state s'. This assumption is very

[6] As in (Barto *et al.*, 1991), we assume the reward function to be deterministic, although the framework given here can be easily extended to stochastic rewards, too. We furthermore assume that reward is a function of states rather than state transitions.

[7] In fact, whether an environment is Markovian or not depends often on the notion of a state. In a more general framework one often distinguishes the internal state of the environment and an externally observable state being a projection of the internal state and assumes that the internal state transition function is Markovian and static. E.g. an autonomous robot might not be able to observe the whole environment due to the limited information provided by his sensors. The observed state is often modeled as a probabilistic projection of the internal state, with noise being the probabilistic factor in this function (Kalman, 1960). Although such *partially observable Markovion environments* play an important role in various applications, we will restrict this paper to fully observable Markovian environments. Many techniques for dealing with the perceptual aliasing problem in partially observable environments base on techniques for identifying and distinguishing internal states or state clusters (recent ones are e.g. described in (Bachrach and Mozer, 1991), (Whitehead and Ballard, 1991), (Chapman and Kaelbling, 1991), (Rivest and Schapire, 1987), (Chrisman, 1992)). It seems straightforward to apply the ideas presented in this paper to this more general case.

common in reinforcement learning literature.

A *Markov decision problem* is the problem of identifying a *controller* for a Markovian environment which optimizes rewards over time. This controller is represented as a reactive *policy*, which maps states to actions: For each state of the environment, a policy tells us which action to do next. In general, the policy might be a non-deterministic function, even if the environment is not. If during learning the policy is applied for action selection, actions are usually merged with some non-deterministic function in order to ensure exploration – the exploration techniques evaluated in this paper can be viewed accordingly. The ultimate goal of reinforcement learning is to identify or at least to approximate an *optimal policy*, i.e. a policy which for each state yields an action optimizing future expected cumulated reward. There are several difficulties in finding an optimal policy. Firstly, a learning system receives an immediate reward as response to an action, but never receives the information which action would have been the best, as it happens in supervised learning tasks. Therefore a learning system can figure out optimal actions only by performing repeated trials. But by having performed an action, the environment might be in a different state, hindering to try other actions on the same state. A second problem is called the *temporal credit assignment problem*: Actions might affect the rewards of the environment with some unknown delay. For example, if a battery-powered robot runs out of power, the resulting pain might not only result from the last action. Thus if the learning system receives some reward, it is not clear by which of the actions in the past this particular reward was caused. Reinforcement learning is a learning methodology for these problems. According to dynamic programming, reinforcement learning, as we will describe it here, relys on the notion of *utility* of a state, which evaluates states with respect to cumulated future reward. The major characteristic of reinforcement learning in terms of asynchronous dynamic programming is that actions are generated in *real-time*, thus the time for action selection may not depend on the size of the state space n (usually it is linear in the number of actions d).

We will now define the *utility* of a state. Generally speaking, there are several ways to represent a policy, the simplest of which is a direct mapping from states to actions, yielding one action a time but no further information. In Markovian decision tasks the notion of a policy is often coupled to the notion of *utility*. According to AHC-learning and dynamic programming, the utility $V^\pi(s)$ of a state s measures the *expected cumulated discounted future reward* discounted by a *discount factor* γ ($0 \leq \gamma \leq 1$), given that s is the actual state of the environment and π is the actual policy used:

$$V^\pi(s) \quad = \quad E\left[\sum_{t=t_0}^{\infty} \gamma^t r_t | s\right] \tag{1}$$

r_t denotes the expected reward at time tick t, and t_0 is the *actual* time tick. Clearly, the utility of a state is also a function of the policy π subsequently applied for action selection, by which future reward is also determined. The discount factor γ weights future rewards in the sum of Eq. (1) and basically serves to bound utilities (if $\gamma < 1$). Note, that the utility $V^\pi(s)$ of a state s does not depend on the time tick t, since we assume environments to be static. Thus Eq. (1) can be rewritten as

$$V^{\pi}(s) \quad = \quad E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s\right] \tag{2}$$

The higher the utility of a state, the better the future reward one expects. If a controller has the choice between different actions, thus entering different states, and if it is to optimize future reward, it will prefer those actions yielding the highest utility. This selection process recursively defines an optimal policy π^*: At every state s, select the action a which maximizes:

$$f^*(a) \quad = \quad \sum_{s' \text{ is state}} P_{ss'}(a) \cdot V^{\pi^*}(s') \tag{3}$$

An optimal policy is found by identifying the function V^{π^*}. This policy can be approximated by asynchronously updating the $\hat{V}$-estimates in a recursive manner (*policy iteration*). Let $\hat{V}^t$ be the current estimate for V^{π^*} at time t, and $\hat{\pi}^t$ be the policy obtained by maximizing the *exploitation measure*

$$f(a) \quad = \quad \sum_{s' \text{ is state}} P_{ss'}(a) \cdot \hat{V}^t(s') \tag{4}$$

for action selection. By performing an action a at state s, the current $\hat{V}$-estimate of the actual state s is updated with the immediate reward r^t received and the $\hat{V}$-estimate of the subsequent state s':

$$\hat{V}^{t+1}(s) = \begin{cases} (1-\eta) \cdot \hat{V}^t(s) + \eta \cdot (r_t + \gamma \cdot \hat{V}^t(s')) & \begin{array}{l}\text{if } s \text{ is the state at time } t \\ \text{and } s' \text{ is the state at time } t+1\end{array} \\ \\ \hat{V}^t(s) & \text{otherwise} \end{cases} \tag{5}$$

$\eta > 0$ is a sufficiently small learning rate. Reinforcement learning updates states using this updating rule, but in order to exhaustively explore the whole state space it does not strictly use the policy of maximizing $f(\cdot)$ for action selection. Instead, action selection is merged with an exploration term, which prevents from getting trapped in poor local minima. The update rule (5) is often referred to as *reinforcement comparison*, and the correction term $r_t + \gamma \cdot \hat{V}^t(s') - \hat{V}^t(s)$ taken from Eq. (5) as *temporal difference (TD) error* (Sutton, 1984), (Sutton, 1988).

5 Exploration techniques

5.1 Undirected exploration

Undirected exploration is characterized by generating actions based on randomness. There is no knowledge about the learning process itself available. In order to deal with the trade-off between exploration and exploitation, the current utility estimates can be used to draw the probability distribution actions are selected with. Note that in many reinforcement learning tasks, undirected exploration degrades to a uniform random walk in the first trial, unless there is some a priori information about utilities available. We will discuss three types of undirected exploration often found in literature: (uniform) random exploration and two utility-driven undirected exploration techniques.

5.1.1 Random exploration

The most uninformed and basic way of exploring an unknown environment is to generate actions randomly with uniform probability. This method is often applied if exploration costs do not matter during learning. Sometimes tasks are divided into two phases, namely a learning and a performance phase, and costs are not considered during the learning phase. However, if one wants a learning system to exploit also during learning, this method is most unefficient in terms of costs. Random exploration is also frequently used if it is not feasible to extract exploitation knowledge such as utility-estimates. This is often done in real-valued state spaces (Nguyen and Widrow, 1989), (Anderson, 1986), (Mozer and Bachrach, 1989), (Bachrach and Mozer, 1991), (Jordan, 1989), (Jordan and Jacobs, 1990), (Mel, 1989), (Munro, 1987), (Thrun *et al.*, 1991).

5.1.2 Utility-driven probability distributions

The major charterization of random exploration in our framework is the fixed uniform probability distribution for action selection, which is of course independent of the utility estimates. If costs are relevant during learning, *non-uniform* undirected exploration techniques utilize the current utility estimates to influence action selection. This is typically done by modifying the probability distribution actions are selected with: The higher the expected utility by selecting action a, the more likely a gets selected. This ensures that the learning system explores and exploits simultaneously, and can be viewed as combination of pure random walk and pure exploitation (see Fig. 3).

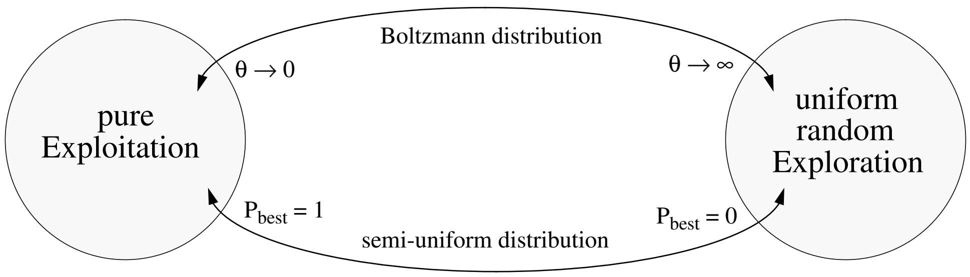

Figure 3: *Undirected exploration techniques:* Boltzmann distributions and semi-uniform distributions are one-dimensional combinations of pure exploitation and random exploration.

5.1.3 Semi-uniform distributions

Semi-uniform distributions establish a simple, undirected exploration rule. Every time tick, each action a is evaluated by $f(\cdot)$ (c.f. Eq. (4)). With some predefined probability P_{best} that action a with the best expected utility is selected: $a^* = \text{argmax}_a f(a)$ (if several best actions exist, one is selected randomly), and with probability $1 - P_{best}$ an action is selected randomly, regardless of its expected utility:

$$P(a) \quad = \quad \begin{cases} P_{best} \; + \; \dfrac{1 - P_{best}}{\#\text{of actions}} & \text{, if } a \text{ maximizes } f \\[2ex] \dfrac{1 - P_{best}}{\#\text{of actions}} & \text{, otherwise} \end{cases} \tag{6}$$

The parameter P_{best} allows for smoothly blending from pure random exploration ($P_{best} = 0$) to pure exploitation ($P_{best} = 1$, see Fig. 3).

5.1.4 Boltzmann distributions

Unline semi-uniform exploration, Boltzmann-distributed exploration takes the f-evaluations of *all* actions into account. The probability for an action to get selected is a non-linear function of f:[8]

$$P(a) \quad = \quad \frac{e^{f(a)\theta^{-1}}}{\sum\limits_{a' \text{ is action}} e^{f(a')\theta^{-1}}} \tag{7}$$

Here $\theta > 0$ is a gain factor, often called "temperature", which determines the amount of randomness in the action selection procedure. With $\theta \longrightarrow 0$ pure exploitation is approached and with $\theta \longrightarrow \infty$ the resulting distribution approaches the uniform distribution, i.e. random exploration (Fig. 3).

[8]The particular probability distribution is closely related to and motivated by the probability distribution in Brown molecular movements in potential fields.

5.2 Directed exploration

Unlike the exploration techniques described so far, *directed exploration* memorizes knowledge about the learning process itself and utilizes this knowledge for directing the exploration. In turn we will describe directed exploration techniques based on a) counters, b) error estimates, and c) recency values.

5.2.1 Counter-based exploration

For each state s, a counter $c(s)$ counts how often this state occurred. Actions are now evaluated by a linear combination of an exploitation term and an exploration term[9]:

$$eval_c(a) \quad = \quad \alpha \cdot f(a) \;+\; \frac{c(s)}{E[c|s,a]} \tag{8}$$

$E[c|s,a] = \sum_{s'} P_{ss'}(a) \cdot c(s')$ denotes the *expected* counter value of the state obtained by applying action a at state s. $\alpha \geq 0$ is a constant gain weighting exploration versus exploitation. As usual, we define $\frac{0}{0} := 0$ and $\frac{z}{0} := \infty$ for all $z > 0$. Eq. (8) evaluates actions with respect to a linear combination of exploitation and exploration. The action selection rule bears the randomness: Always the action maximizing $eval_c$ is chosen deterministically.[10] Related counter-based exploration techniques might be found in (Sato *et al.*, 1990), (Barto and Singh, 1990).

Generally speaking, counter-based exploration evaluates states on the basis of *how often* they occurred. We have seen that this simple rule prevents from exponential exploration time in many cases. But thinking in terms of fast learning, what we really are interested in is provoking those states which yield the best performance improvement. In order to achieve this, we will extend counter-based exploration by two straightforward heuristics, namely *decay* and an *error estimate*.

5.2.2 Counter-based exploration with decay

Plain counter values describe the accuracy of a utility estimate only partially, since they do not contain information *when* a state occurred, although this information is also relevant for efficient exploration. For instance, if two states occurred equally often but one only at the very early beginning of learning, while the other one most recently, we ought to give preference to the first. This observation, which is more radically addressed by recency-based exploration described below,

[9] The counter-based rule described here differs from the previously introduced one by using the quotient of counters rather than the difference. We found this exploration rule more appropriate for combination with an exploitation term, since by using the difference each state will occur equally often in the limit.

[10] If it happens that more than one action have equal best evaluation, one might be picked by an arbitrary rule: e.g. the first action due to any alphabetical order, if any, or one action might be picked randomly. We used the latter technique in the simulations.

is the basis for extending counter-based exploration with *decay*: At each time tick every counter is multiplied with a fixed decay $\lambda \lessapprox 1$.

$$c(s) \quad \longleftarrow \quad \lambda \cdot c(s) \quad \forall s \tag{9}$$

Hence the more recent an occurrence of a state, the more these occurrences contribute to the counter value of this state. With an appropriate choice for λ the resulting decayed counter reflect the accuracy of the utility values better than any undecayed counter, and we will demonstrate this in the simulation results.

5.2.3 Error-/counter-based exploration

Another way to extend counter-based exploration is achieved by directly estimating the *change* of the utility-estimates $\hat{V}$ obtained by updating it (Schmidhuber, 1991), (Thrun and Möller, 1991), (Thrun and Möller, 1992). The idea of the particular error-/counter-based exploration rule described here is to memorize the latest change of the utility estimation of each state denoted by $\Delta\hat{V}_{last}(s)$. The more the utility value of a state changed, the more likely is that neighboring states update its utility value correspondingly. Thus it makes sense to prefer actions leading to states whose utility value has recently changed the most, thus extending the evaluation function of counter-based exploration (8):

$$eval_{c/e}(a) \quad = \quad \alpha \cdot f(a) \quad + \quad \frac{c(s)}{E[c|s,a]} \quad + \quad \beta \cdot E[\Delta\hat{V}_{last}|s,a] \tag{10}$$

$\beta > 0$ is a constant factor determining the portion of the error-heuristic in action selection, and $E[\Delta\hat{V}_{last}|s,a]$ denotes the expected $\Delta\hat{V}_{last}$-value by taking action a at state s and is initialized with a large value. A connectionist architecture for exploration in real-valued domains using counter/error-based directed exploration might be found in (Thrun and Möller, 1992).

5.2.4 Recency-based exploration

A more rigorous approach to overcome the limitations of plain counter-based exploration is called recency-based exploration (Sutton, 1990). The principle of recency-based exploration is to prefer adjacent states which occurred less recently. Basically, for recency-exploration there is a recency value $\rho(s)$ associated with each state s memorizing the number of actions performed since the last occurrence of this state.[11] Based on the recency value, actions are evaluated with respect to the recency value of the corresponding next state:

$$eval_r(a) \quad = \quad \alpha \cdot f(a) \quad + \quad \sqrt{E[\rho|s,a]} \tag{11}$$

[11] For efficiency reason, this is implemented by storing the time tick of the last occurrence. The difference between the current time tick and the last occurrence measures the recency of this state.

Again, $\alpha \geq 0$ is a constant gain, the action with the highest evaluation $eval_r(a)$ is selected deterministically, and $E[\rho|s,a]$ denotes the expected recency-value by selecting action a at state s. Note that even in the first run recency-based exploration behaves different from any counter-based technique. If a state has only occurred once but very recently, actions yielding this state are not likely to get selected.

Sutton (Sutton, 1990) named the exploration term $\sqrt{E[\rho|s,a]}$ in Eq. (11) *exploration bonus*. He uses this bonus differently: Instead of locally evaluating the recency of neighboring states only, the recency-values are subject to an off-line dynamic programming process. A model is used off-line for aligning recency values through the state space, very similar to the alignment of utilities in reinforcement learning (c.f. Eq. (5)). This has the advantage of exploring with an extended horizon: The exploration rule selects actions with respect to *all* states and not only to the neighboring states, which defines a non-local exploration rule. In addition to a model, this requires much off-line computation-time for performing dynamic programming, which clearly violates our real-time constraint. However, by not performing a complete dynamic programming evaluation at each time an action is selected, the off-line computation time can be gradually reduced, and in the limit (i.e. no off-line computation) the exploration rules equals our notion of recency-based exploration.

6 Experimental results

6.1 A two-dimensional robot navigation task

We tested all exploration techniques on the two-dimensional robot navigation task depicted in Fig. 4. The task is to navigate the robot (left circle) to its goal position (right circle) with as few actions as possible. The state space consists of roughly 4 500 states which are represented by its x-y-coordinates in the grid. Each time the robot has 8 valid actions, each of which corresponds to one neighbor in the grid. There are two kinds of non-zero rewards. If the robot collides against a wall, it will not move but receive the negative reward -1. Positive reward 1 is only received upon entering the goal position. Thus, no a priori information is provided to guide the search of the goal, and the $\hat{V}$-values are initialized with 0. We investigated exploration with a deterministic and a non-deterministic state-transition function – in the latter case there was 10% randomness merged in the state transition function, i.e. in 90% of the cases actions were performed correctly, but in 10% actions led to a randomly selected neighboring state. In both cases, an action model was given to the system.

Based on the specific task at hand, we extended our version of reinforcement learning to three update mechanisms:

1. If the agent moves from state s to state s', the value $\hat{V}(s)$ is updated with $\gamma \cdot \hat{V}(s')$, if and only if this term is larger than $\hat{V}(s)$. The restrictive definition of the task – more specifically: $\hat{V}(\cdot)$ is monotonically increasing in time – allows direct alignment of utility values (i.e. the learning rate is 1 in standard reinforcement learning) and overcomes some of the problems of plain AHC-learning.

2. Knowing about the invertibility of this task (i.e. to each action leading from state s to state s', there is an inverse action leading from s' to s), the above alignment is inverted: $\hat{V}(s')$ is updated, if smaller, by $\gamma \cdot \hat{V}(s)$.

3. To make things actually working, the most recent run (at most 30 000 steps) is temporarily stored and once replayed in reverse order after the goal state is reached. This experience replay technique accelerates the backward-alignment of utility values tremendously (Lin, 1991b), (Lin, 1992).

We will now describe the results obtained by applying all above exploration techniques to this task. For each evaluation, parameters have been optimized by hand. Exploration techniques can be assessed by two criteria: the average time for the first trial and the performance during reinforcement learning.

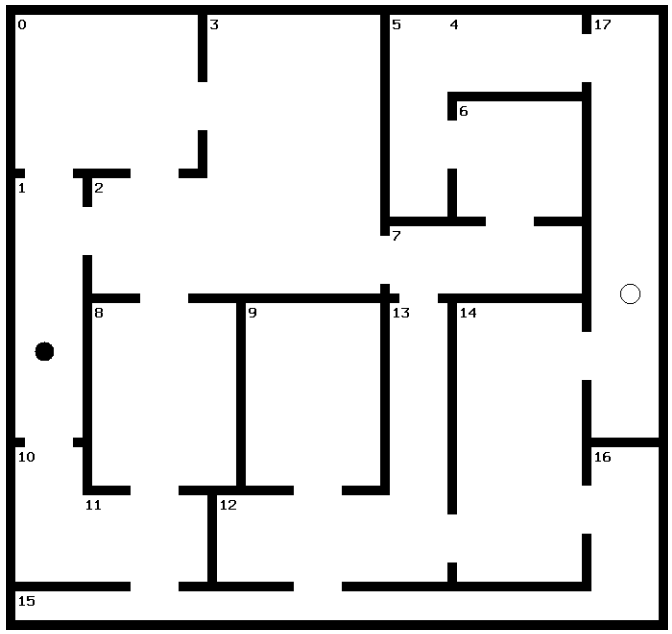

Figure 4: **Task:** The task is to navigate the robot (left circle, room 1) to its destination (right circle, room 17) on the shortest possible path (its length is roughly 94.0, measured with L_2-norm). The shortest possible path leads through the rooms 1-10-11-15-12-14-17; other paths are 1-2-7-13-14-17 and 1-2-7-6-5-4-17 etc. The task is considerably complex – there are about 4 500 states.

6.2 Complexity of the first trial

An important characteristic of the exploration techniques investigated is the number of actions required for the first trial. In this first run there is no knowledge about the environment available, thus the $\hat{V}$-values are initialized with 0. As pointed out above, undirected exploration techniques which rely on an estimation of the utility function cannot do anything but pure random search. Counter-based exploration, error-based exploration and counter-based exploration with selective attention (see Section 7) all follow the same rule "Go to the least visited state". Thus there remain only four different classes of exploration techniques. The following table sketches the average number of steps required for the first run, averaged over 50 experiments each:

exploration-technique	deterministic version average steps	stochastic version average steps
Uniform distribution Boltzmann distribution Semi-uniform distribution	43 000	43 000
Counter-based exploration Counter/error-based exploration Counter-based exploration with selective attention	4 700	4 800
Counter-based exploration with decay ($\lambda = 0.99999$)	5 800	7 300
Recency-based exploration	7 400	8 900

Counter-based techniques appear to be most efficient in terms of expected search steps for finding the goal in the beginning and to be least sensitive to the randomness in the transition function. Random exploration takes in this particular task about nine times more steps. It is somehow surprising that counter-based exploration with decay performs clearly poorer than counter-based exploration without decay, although this decay was close to 1. Another result is that recency-based exploration takes significantly more time for the first trial than counter-based exploration. If one is interested in minimizing this first run, counter-based exploration seems to be the best choice for the first trial. As we will see in the next section, this judgement does not hold if one is interested in cheap and fast learning, i.e. optimal performance over several learning trials using reinforcement learning. These results also clearly demonstrate the weakness of all undirected exploration techniques. Due to above theorems, we expect the gap between directed and undirected exploration techniques to grow with the size of the state space.

6.3 Explanation of the learning curves

We will now compare the effects of the different exploration techniques to the learning process. For each technique, its impact to learning is illustrated by three curves, with the horizontal axis being the number of steps (i.e. actions taken, total of 250 000 steps) in all figures.

1. The first figure shows the average length of each trial, averaged over all trials so far. This curve is intended to reflect the portion of exploitation in the learning process and is the best measure for exploration costs we found for this particular task. Minimizing costs during learning is thus equivalent to minimizing this average length. Therefore one technique outperforms another if, with the same learning result, more runs could be completed with the same number of steps, i.e. the average length of each trial is smaller.
2. The second figure plots the length of the optimal path due to the current utility estimation. Here exploration (and learning) is switched off for a moment, and a single run is performed

to compute the path length which the controller would generate if learning were stopped now (pure exploitation). Shortest path-curves illustrate the state of learning through time. The closer this curve approaches the shortest possible path length (which is in the particular environment at hand approximately 94.0), the better the resulting controller.

3. The third figure visualizes the accuracy of the utility function, i.e. the (negative) deviation of the correct utility function V^{π^*} and the current utility estimation $\hat{V}$ (measured with L_1-norm) scaling from 0% to 100%. In some way, this value measures also the success of the learning process, since the better the utility function, the more likely the optimal solution is found in many tasks.

Due to the trade-off between exploration and exploitation, an exploration technique clearly outperforms another if its average path (figure a) is shorter **and** [the model accuracy (figure b) is higher, or the shortest path found (figure c) is shorter].

6.4 Results obtained for the deterministic environment

In the deterministic version, the state transition function and thus the whole environment is deterministic. For all simulations we carefully optimized parameters by hand. Since the gain factor usually weighs exploitation vs. exploration, gain was chosen such that the results became most comparable.

1. **Random exploration** (Fig. 5 & 6):

   This technique is characterized by its extremely high exploration costs, as shown in Fig. 5. Random exploration succeeds in finding a nearly-optimal path after about 350 000 steps without taking any exploitation into account. Moreover, due to the missing action selection criterion, the number of wall collisions is unreasonable large (not plotted here).

2. **Semi-uniform distribution** (Figs. 6 & 8):

   The diagrams plot the learning curves obtained with $P_{best} = 0.5$. This exploration technique almost always fails in finding the shortest path within the first 250 000 steps. (The same is expected to hold for all $P_{best} > 0.5$.)

3. **Boltzmann distributions** (Fig. 6):

   With $\theta = 0.025$ this technique clearly fails in finding the shortest path within the first 250 000 steps, too (Fig. 6b). Apparently, the task at hand is too difficult using this kind of exploration and only a suboptimal solution is found. It is worth noting that Boltzmann distribution turned out to work poorer than semi-uniform distributions in many cases, although the former ones are more frequently found in literature.

Unlike undirected exploration techniques, which succeeded in identifying an optimal controller within the first 350 000 steps only for random exploration – with extremely high exploration costs –, directed techniques always converged much faster. As quoted above, counter-based techniques are fastest in *finding* the goal (first trial), but the different techniques show interesting different characteristics during learning.

1. **Counter-based exploration** (Fig. 7):

   With $\alpha = 4.0$ we achieved the best results. The learning diagrams show clearly that this directed exploration technique outperforms all undirected ones in all three performance measures: While exploration costs are lower, the result in terms of shortest paths and the accuracy of the utility function are significantly better. However, directed exploration can be done more efficiently due to the above described limitations of plain counter-based exploration.

2. **Counter-based exploration with decay** (Figs. 7 & 8):

   By decaying each counter with $\lambda = 0.99999$ and $\alpha = 25.0$ the results of counter-based exploration were significantly improved. While exploration costs were lower, the shortest path was found much faster. However, the agent focussed its exploration more on states closed to the optimal path while other regions were explored less. Therefore an accurate utility estimate for the whole state space was found slower.

3. **Error-/counter based exploration** (Fig. 7):

   Best results were obtained for $\beta = 1$. Although we tried a couple of different parameter settings, we did not find any significant improvement over plain counter-based exploration. It should be noted that this type of exploration heuristics is assumed to work better in dynamic environments, where changes in the environment cause changes in the utility function. Error estimations might then accelerate learning by attracting states whose utility value has changed due to the dynamics in the environment. However, we restricted the evaluations in this paper to static environments.

4. **Recency-based exploration** (Figs. 7 & 8):

   Best results were achieved using $\alpha = 200.0$. Except for the first trial, this technique worked better than counter-based exploration and showed equally good performance for all $\alpha \in [20, 200]$. The robustness against the choice of α indicates the appropriateness of recency-based exploration for environments where a good estimate for the optimal α is difficult to obtain. It should also be noted that the accuracy of $\hat{V}$-values increases very slowly, indicating that exploration is focussed on regions close to the optimal path – the same important effect was observed for counter-based exploration with decay (Fig. 7c).

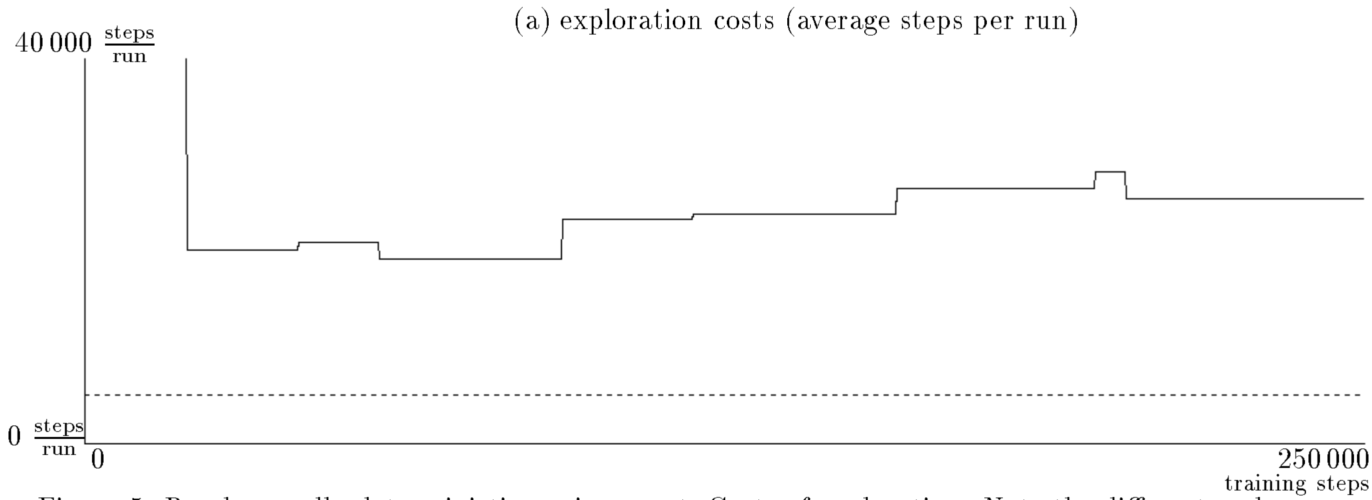

Figure 5: Random walk, deterministic environment: Costs of exploration. Note the different scale: The dashed line indicates the upper border of all other costs diagrams.

6.5 Results obtained for the stochastic environment

In order to test the exploration techniques on stochastic environments we merged the state transition function with 10% randomness, as described above. By that the performance of undirected exploration techniques is not expected to change significantly, since randomness is already their basis of exploration. E.g. random exploration works equally well in both cases, and with semi-uniform distributions the effect of randomness in the state transition function is equivalent to changing the parameter P_{best} (This is a special case, since the particular randomness in our environment happens to be equivalent to the randomness in semi-uniform exploration). Thus the results of undirected exploration did not differ significantly from those shown in Fig. 5 & 6. However, we found that among all directed exploration techniques, none was significantly influenced by this 10%-randomness either, besides the effects listed in Section 6.2. Fig. 8 gives the results for those undirected and directed techniques showing best performance in the deterministic case, namely semi-uniform distributed undirected exploration, counter-based exploration with decay and recency-based exploration, using the same parameter settings as above.

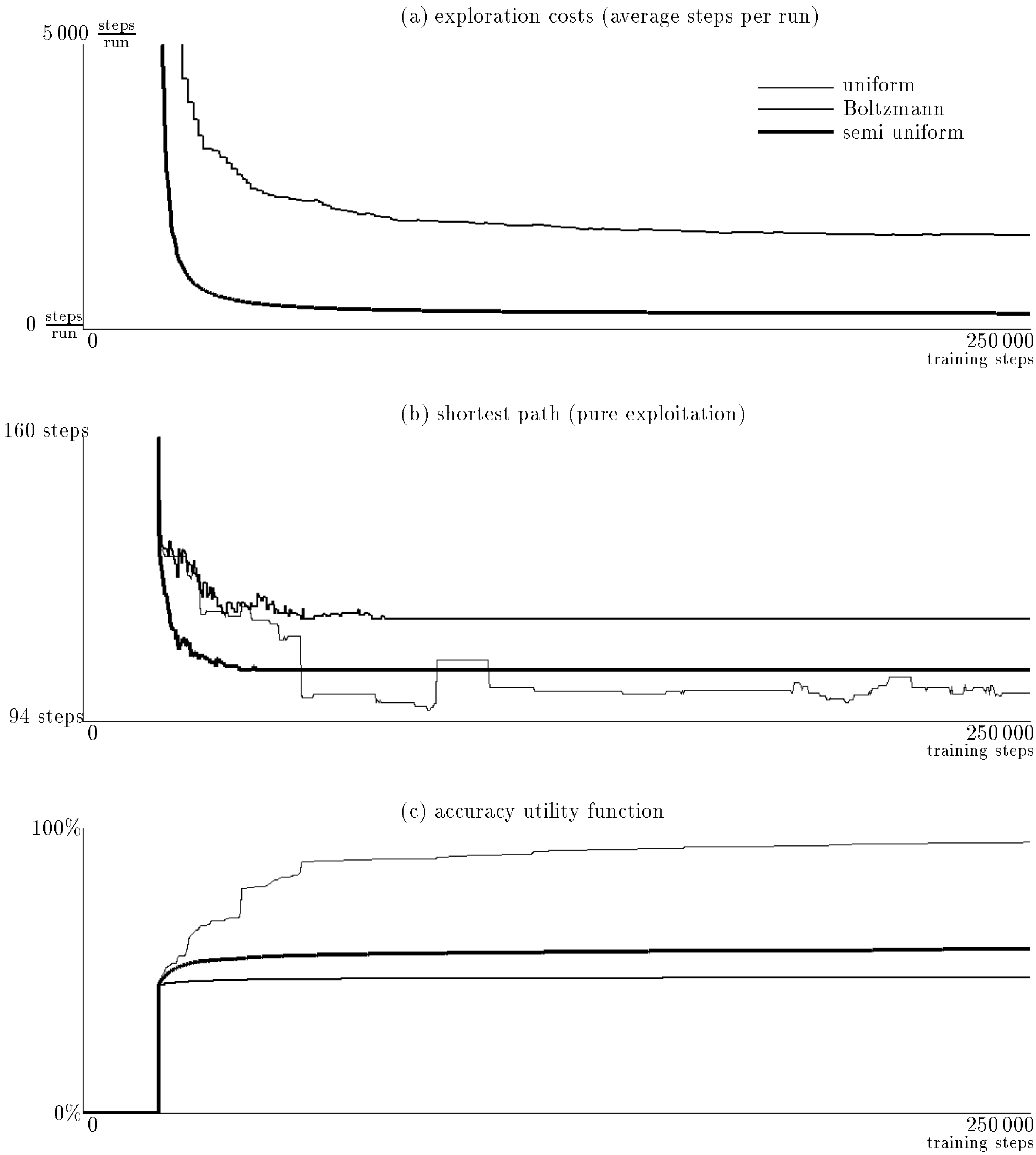

Figure 6: Undirected exploration, deterministic environment: Uniform distribution (thin lines), Boltzmann distribution (medium lines) and semi-uniform distribution (thick lines). The cost function of uniform random exploration does not fit in this diagram and is thus plotted in Fig. 5.

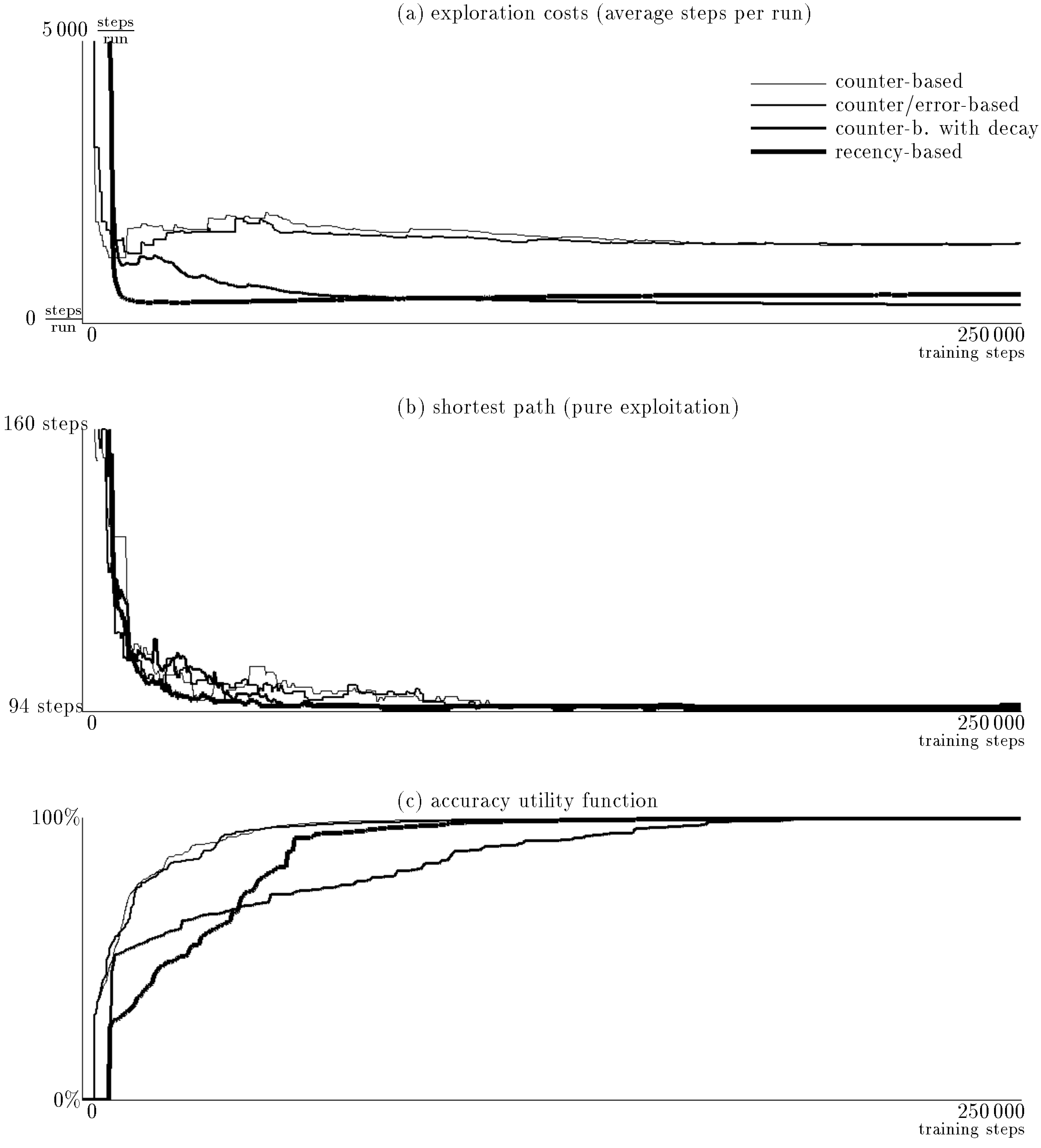

Figure 7: Directed exploration, deterministic environment: Counter-based exploration (thin lines), counter/error-based exploration (medium lines), counter-based exploration with decay (thick lines), and recency-based exploration (very thick lines).

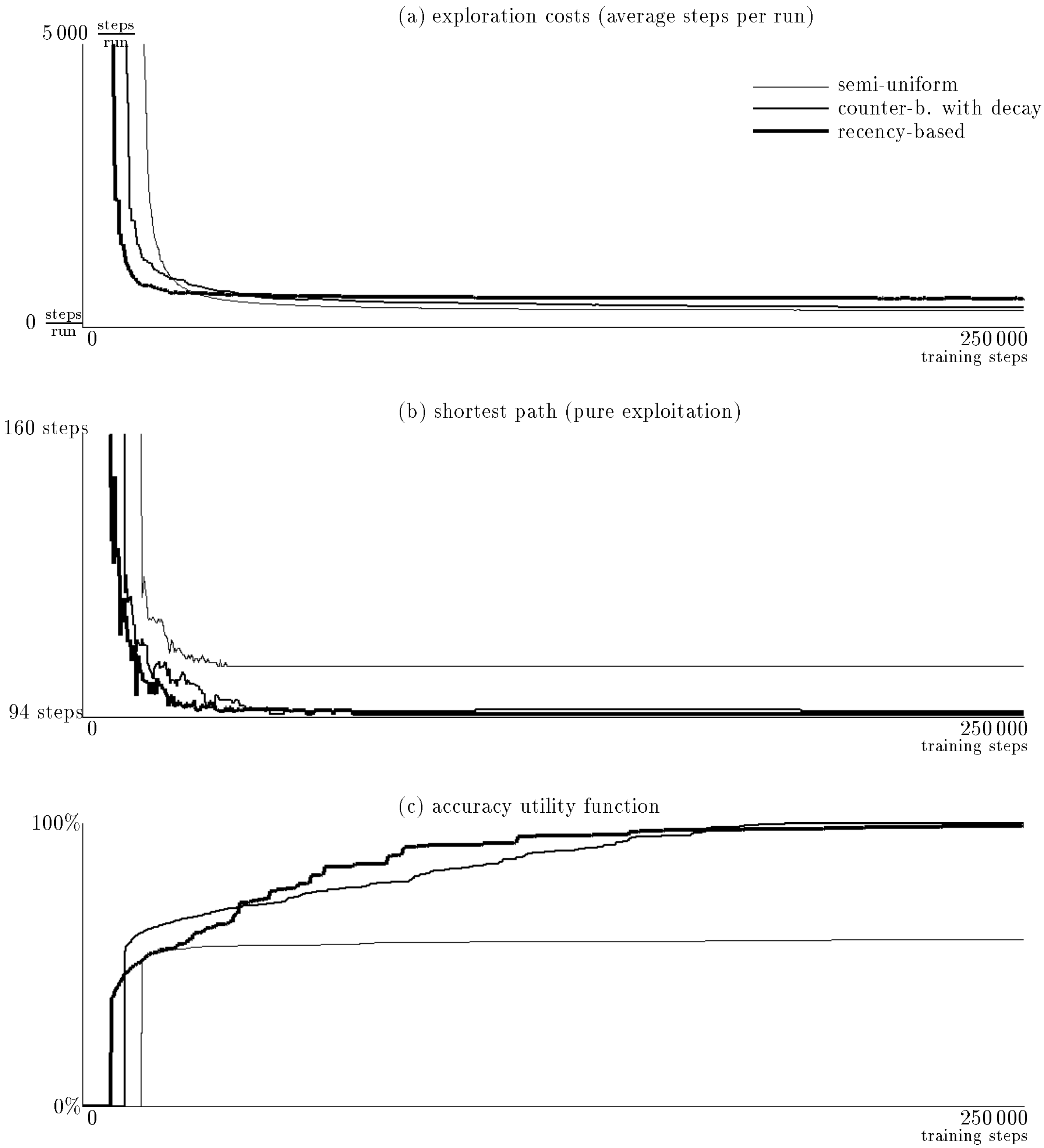

Figure 8: Stochastic environment: Semi-uniform distributed exploration (thin lines), counter-based exploration with decay (medium lines) and recency-based exploration (thick lines). Note that these results differ not significantly from those obtained with the deterministic version.

7 Combining exploration with selective attention

So far we have presented and empirically compared several undirected and directed exploration techniques all dealing with the trade-off between exploration and exploitation. We will now address a principal inefficiency all those techniques have in common: the *fixed, linear combination of exploration and exploitation.* This can yield an undesired effect, namely: If exploration and exploitation point exactly into the opposite direction, the exploration rule might yield an action which neither explores, nor exploits. E.g., suppose an agent facing a lion. Exploitation might make the agent to flee and exploration makes it to explore – but a fixed linear combination of both makes the agent stay where it is, which is clearly the less efficient solution: The agent should either run away, or explore, but not do nothing at all. This clearly undesired effect is inefficient in terms of exploration costs. Since exploration and exploitation usually *are* opposing, fixed linearly combining both seems to be questionable, and some kind of dynamic combination of both criteria might help to decrease the costs. We will describe a selective attention technique based on counter-based exploration, which switches attention dynamically between exploration and exploitation (Thrun and Möller, 1992). The basic idea is to establish a parameter of attention Γ which determines the behavior – exploitation or exploration – playing the major role in action selection. This parameter is updated with a hysteresis property making it considerably stable over several steps. Benefits of switching attention between exploration and exploitation are demonstrated in the simulation results.

Selective attention is established by extending the evaluation function $eval_c(\cdot)$ (c.f. Eq. (8)) of counter-based exploration by an *attention parameter* Γ, $0 < \Gamma < 1$:

$$eval_{c-sa}(action) \quad = \quad \Gamma \cdot \alpha f(a) \quad + \quad (1-\Gamma) \cdot \frac{c(s)}{E[c|s,a]} \tag{12}$$

Γ determines which of the two principles influences the action selection most: If Γ is close to 1, actions are selected such that they optimize the expected future reward (exploitation). With Γ close to 0, exploration is favored almost regardless of the rewards. At each trial Γ is initialized with 0.5; after having selected an *action*, Γ is updated by the bistable equation:

$$\kappa \quad \longleftarrow \quad \Gamma \cdot \frac{f(a)}{V^t(s)} \quad - \quad (1-\Gamma) \cdot \frac{c(s)}{E[c|s,a]} \tag{13}$$

$$\Gamma \quad \longleftarrow \quad \frac{.8}{1+e^{-a\cdot\kappa}} \quad + \quad .1 \tag{14}$$

The value κ measures the trade-off between exploitation and exploration *under the current focus of attention* Γ, and the new Γ is then obtained by squashing κ. Here, $a > 0$ is a factor which determines switching shape. Note that Γ will always be in the interval $(.1, .9)$, hence whenever the system chooses one behavior (exploration or exploitation), the other will contribute at least 10% to the action selection. If one thinks in terms of an exploration behavior and an exploitation behavior of a learning systems, these equations establish a conservative way for smoothly switching between behaviors. Note that these switching equations represent only one choice among many others for selective attention.

7.1 Experimental results with selective attention

We applied counter-based exploration with selective attention to the robot navigation task. Learning curves for both the deterministic and the stochastic version of the environment are shown in Fig. 9. With the switching parameter a set to 30, selective attention could significantly reduce the costs of counter-based exploration while leaving the quality of solutions almost unaffected. The switching behavior established by selective attention can clearly be recognized in (Fig. 9a&b). After a first optimization phase, the controller focuses for some time on pure exploitation, keeping the costs considerably low. After roughly 27 000 steps it changes to exploration and makes one expensive but sufficient exploration walk, which subsequently allows for finding the optimal path while keeping the costs low. In terms of trade-off between exploration and exploitation, selective attention has been empirically shown to improve counter-based exploration.

We also applied selective exploration to recency-based exploration, but did not find this to improve solutions at all.

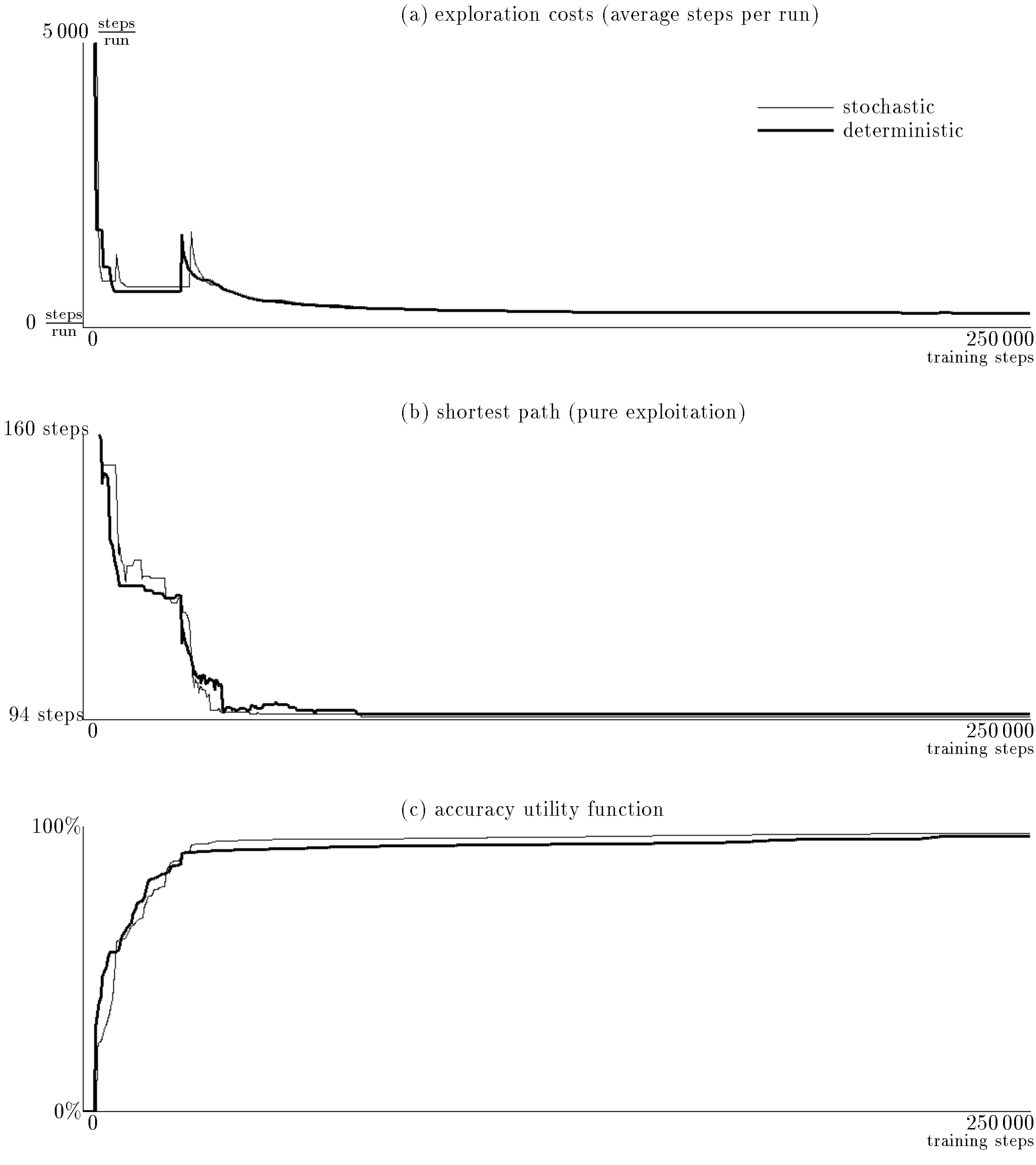

Figure 9: Selective attention: Counter-based exploration with selective attention with the stochastic (thin lines) and the deterministic (thick lines) version of the environment. It is clear to see that after ca. 27 000 steps the agent performs an exhausting directed exploration walk which improves both shortest path and model accuracy significantly, while keeping the average costs on a very low level.

8 Limitations and discussion

In this paper we addressed exploration and reinforcement learning in a very basic form, in order to clearly demonstrate the principal effects of the different exploration techniques. By doing this, we were able to derive the superiority of directed over undirected exploration analytically, and we subsequently demonstrated this on a simple two-dimensional robot navigation task. Although we hope that the different techniques presented in this paper cover a wide variety of exploration mechanisms applicable in more difficult and real-world problems, we neglected important aspects relevant for many learning systems in the area of reinforcement learning and adaptive control, the most important of which will be outlined in turn.

Huge or infinite environments. The theoretical results on the complexity of exploration, as well as all the directed exploration techniques as they are presented here, base on the assumption that it is feasible to exhaustively explore the whole state space. In many domains, including those typically studied in AI, state-spaces or state-action spaces are too large, hence exploring the whole state space is too expensive or simply impossible as it is in real-valued domains (e.g. robot arm control). Generally speaking, the exploration techniques have to be extended or eventually new ones have to be developed which ensure sufficiently much exploration for local optimization, but avoid exhaustive exploration.

An interesting technique to deal with huge state spaces is teaching (Lin, 1991b), (Lin, 1991a). By teaching we mean learning scenarios in which a human teacher provides some potentially suboptimal solutions for the problem, and the aim of reinforcement learning is now to refine these solutions. It is unclear how the directed exploration techniques apply to such tasks. There is evidence that selective attention, as it is presented here, does a poor job, since once exploration is switched on it might spend too much time searching in unrelevant regions of the state space, never switching back to exploitation. Conversely, counter-based exploration with decay and recency-based exploration seems to be most appropriate. This evidence is taken from the accuracy results for these techniques shown in Fig 7c. In our simulations we observed these two techniques to concentrate very much on the current optimal path, while regions in the state space far away from this path were explored only considerably seldom. Usually we are not interested in having the most accurate utility estimate but in the best possible solution, which clearly makes a difference: irrelevant parts of the state space need no exploration. This observation can be utilized for further reducing the complexity of reinforcement learning yielding a different kind of switching attention: If one knows already a solution, exploration should be turned off if it cannot improve this known solution at all. In our example, exploration may be switched off if the length of the exploration walk exceeds $\hat{n}$, given that one found already a path to the goal with length $\hat{n}$.

Incorporation of pre-given knowledge. Another way of efficiently dealing with huge state spaces is to give the learning system some knowledge in advance, represented e.g. as initial estimates of the utilities, logical rules, constraints etc. A priori knowledge is similar to teaching. Although we did not discuss such scenarios in this paper, we believe that incorporating this knowledge in order to diminish exploration costs is feasible within this framework – A first attempt could be to linearly combine this knowledge in the action evaluation functions, as it is done for combining exploration and exploitation (c.f. Eq. (8)). There is evidence that extensions of counter-based exploration with decay as well as recency-based exploration ensure efficient exploration even in large state spaces.

It is important to mention that Whitehead's theorem holds only for tasks where *no* a priori knowledge to guide the search is provided. Indirected exploration techniques are reported to scale much more efficiently even with little such knowledge (Lin, 1991b), (Whitehead, 1991b), and the results of this paper cannot be directly transferred to such scenarios.

Exploration and generalization. We did not address the very important interaction of exploration and generalization throughout this paper. Many learning techniques use a generalization technique for predicting sensations/utilities for not yet explored states. Clearly, if this generalization function is appropriate, exploring the whole state space becomes unnecessary. It is not straightforward to extend the exploration techniques presented here to any learning system which generalizes. One way towards the efficient combination of exploration and generalization it to estimate the accuracy of the generalization and to explore those regions in the state space with lower predicted accuracy (Thrun and Möller, 1992). However, this topic is beyond the scope of the paper.

Q-Learning and Exploration. As mentioned above, most of the exploration techniques presented have their counterpart in Q-learning – in fact, some of the techniques, e.g. all undirected ones, counter-based exploration and recency-based exploration have already been studied with respect to Q-Learning (Barto and Singh, 1990), (Barto *et al.*, 1991), (Watkins, 1989), (Lin, 1991b), (Lin, 1992), (Singh, 1992), (Sutton, 1990), (Whitehead and Ballard, 1991). We strongly believe that the even with Q-Learning the directed exploration techniques presented here outperform the undirected ones, and the theoretical results support this. Counters, error estimates or recency-values may now be associated to state-action pairs rather than states.

The effect of exploration on AHC and Q-Learning. The effect that Q-Learning seems to outperform AHC-Learning in several applications cannot be seen independent from the particular exploration and learning technique at hand, since the result of AHC-Learning depends crucially on the policy and thus the exploration technique actions are generated with during learning, and Q-Learning does not (Barto and Singh, 1990). We will discuss this with an example: If an agent has the choice between ten actions, one of which leading to the goal state and the other nine leading to a highly negative reward, the utility found by AHC-like algorithms would average the cumulated reward observed so far. By using e.g. undirected exploration, the agent will eventually more often select an action which yields negative reward rather than the optimal action, and thus the estimated utility of this particular state, updated by the average observed next utility, would clearly be low. This strikes the notion of optimal utility which bases on the assumption that best actions will be performed always. Q-Learning does a better job, since utilities are assigned to state-action pairs, explaining why Q-Learning is reported to outperform AHC, to the expense of a much larger search space. However, it is not clear that AHC-like representations are still inferior to Q-Learning if exploration is done with a directed technique, and the learning rule takes into account whether an action is selected due to exploration or due to optimization, since the search space of AHC is much smaller than that of Q-Learning. The reader may recognize that the update rule used throughout the simulations established exactly this behavior, but this rule is not applicable in general stochastic domains.

Exploration and system identification. Another point addressed in the Theorems but not in the exploration rules and the simulations is to use directed exploration for system identification. A theoretical result stating the worst-case complexity of system identification can be found in the Appendix. The exploration rules presented in this paper can very easily be extended to do system identification simultaneously. E.g. in the proof of Theorem 3, system identification has always preference to exploration, establishing exploration with greedy system identification. It is straightforward, but beyond the scope of this paper to combine the action evaluation of the directed exploration techniques with a term measuring the *demand* for system identification. For a directed exploration approach to system identification see also (Thrun and Möller, 1992).

Undirected exploration and pseudo random numbers. An important drawback of all undirected exploration techniques completely neglected in recent literature is their dependence of random numbers. While "real" random number generators are only rarely available, computer implementation usually rely on pseudo random number generators. These generators may turn non-exponential problems into exponential, and in the extreme one cannot be sure that with an unknown pseudo random number generator it is possible to reach a goal state at all. Directed techniques do not depend crucially on random numbers, and above theorems hold for *any* choice of random numbers since they state worst-case results.

9 Conclusion

This paper addresses the fundamental role of exploration in the complexity of reinforcement learning. Exploration techniques are grouped into two categories based on the underlying exploration knowledge: *Undirected* exploration techniques do not rely on any exploration-specific knowledge and basically explore unknown environments with randomness. Examples for undirected techniques described in this paper are: Uniform random exploration, semi-uniform distributed exploration, and Boltzmann distributed exploration. Conversely, *directed* exploration techniques utilize knowledge about the learning process itself, represented e.g. in terms of counters (Counter-based exploration with and without decay), error estimates (error/counter-based exploration), or recency-values (recency-based exploration). While undirected techniques are often found in reinforcement learning and related literature, directed techniques are rare.

The significance of this somehow ad-hoc looking distinction is demonstrated by both the theoretical results in Section 2 and the experimental results in Section 6. While for many deterministic environments undirected techniques are proven to be inefficient (Whitehead, 1991b), (Whitehead, 1991a), meaning that their expected learning complexity scales exponentially in the size of the state space, this paper proves as an example the efficiency of a particular directed exploration technique, namely counter-based exploration, by showing that the worst-case complexity of learning is always polynomial in the size of the state space. We furthermore demonstrated the superiority of all directed techniques on a two-dimensional robot navigation task. This task differed from many tasks found in reinforcement literature in that it was too complex for all undirected techniques to succeed, but optimal solutions were always found by all directed techniques.

These results are intriguing. Reinforcement learning is often considered to work well with simple tasks but to fail in complex ones. We argue that exploration plays an important role in reinforcement learning, and that efficient exploration is a precondition for lifting reinforcement learning to more complex tasks. Although this paper is strongly related to reinforcement learning, exploration addresses a much broader field of active learning and adaptive control systems. We expect the results found in this study to be not irrelevant to related fields such as adaptive control, robot navigation etc.

Acknowledgements

The author gratefully acknowledges various fruitful discussions and valuable comments by Andy Barto, Tom Mitchell, Lonnie Chrisman, Long-Ji Lin, Sven König, Ryusuke Masuoka, Jon Slenk, Don Sofge, Alex Waibel, and the reinforcement learning reading group at CMU. This work was supported by a grant from Siemens Corp. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Siemens Corporation.

References

(Anderson, 1986) Charles W. Anderson. Learning and problem solving with multilayer connectionist systems. Technical Report COINS TR 86-50, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA, 1986.

(Bachrach and Mozer, 1991) Jonathan R. Bachrach and Michael C. Mozer. Connectionist modeling and control of finite state systems given partial state information. 1991.

(Barto and Singh, 1990) Andy G. Barto and Satinder P. Singh. On the computational economics of reinforcement learning. In D.S. et al. Touretzky, editor, *Connectionist Models, Proceedings of the 1990 Summer School*, pages 35–44, San Mateo, CA, 1990. Morgan Kaufmann.

(Barto *et al.*, 1990) Andy G. Barto, Rich S. Sutton, and Chris J. C. H. Watkins. Learning and sequential decision making. In M. Gabriel and J.W. Moore, editors, *Learning and Computational Neuroscience*, pages 539–602, Massachusetts, 1990. MIT Press.

(Barto *et al.*, 1991) Andy G. Barto, Steven J. Bradtke, and Satinder P. Singh. Real-time learning and control using asynchronous dynamic programming. Technical Report COINS 91-57, Department of Computer Science, University of Massachusetts, MA, August 1991.

(Chapman and Kaelbling, 1991) David Chapman and Leslie P. Kaelbling. Input generalization in delayed reinforcement learning: an algorithm and performance comparisons. In *Proceedings of IJCAI-91*, 1991.

(Chrisman, 1992) Lonnie Chrisman. Reinforcement learning with perceptual aliasing. Submitted for publication, 1992.

(Howard, 1960) Ronald A. Howard. *Dynamic Programming and Markov Processes.* MIT Press and Wiley, 1960.

(Jordan and Jacobs, 1990) Michael J. Jordan and Robert A. Jacobs. Learning to control an unstable system with forward modeling. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2.* MIT and University of Mass., Amherst, Morgan Kaufmann Publishers, 1990.

(Jordan, 1989) Michael I. Jordan. Generic constraints on underspecified target trajectories. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

(Kaelbling, 1990) Leslie P. Kaelbling. *Learning in Embedded Systems.* PhD thesis, Department of Computer Science, Stanford University, 1990.

(Kalman, 1960) R.E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, Journal of Basic Engeneering*, 82:35–45, 1960.

(Lin, 1991a) Long-Ji Lin. Self-improvement based on reinforcement learning, planning and teaching. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 323–327, 1991.

(Lin, 1991b) Long-Ji Lin. Self-improving reactive agents: Case studies of reinforcement learning frameworks. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 297–305, 1991.

(Lin, 1992) Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning Journal*, 1992. (to appear).

(Mahadevan and Connell, 1990) Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. December 1990.

(Mahadevan and Connell, 1991) Sridhar Mahadevan and Jonathan Connell. Scaling reinforcement learning to robotics by exploiting the subsumption architecture. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 328–332, 1991.

(Mel, 1989) Bartlett W. Mel. Murphy: A neurally-inspired connectionist approach to learning and performance in vision-based robot motion planning. Technical Report CCSR-89-17A, Center for Complex Systems Research Beckman Institute, University of Illinois, 1989.

(Moore, 1990) Andrew W. Moore. *Efficient Memory-based Learning for Robot Control.* PhD thesis, Trinity Hall, University of Cambridge, England, 1990.

(Moore, 1991) Andrew W. Moore. Knowledge of knowledge and intelligent experimentation for learning control. In *Proceedings of IJCNN-91*, 1991.

(Mozer and Bachrach, 1989) Michael C. Mozer and Jonathan R. Bachrach. Discovering the structure of a reactive environment by exploration. Technical Report CU-CS-451-89, Dept. of Computer Science, University of Colorado, Boulder, November 1989.

(Munro, 1987) Paul Munro. A dual backpropagation scheme for scalar-reward learning. In *Ninth Annual Conference of the Cognitive Science Society*, pages 165–176, Hillsdale, NJ, 1987. Cognitive Science Society, Lawrence Erlbaum.

(Nafeh, 1976) John Nafeh. *Markov Decision Process with Policy Constraints.* PhD thesis, Dept. of Engineering-Economic Systems, Stanford University, April 1976.

(Nguyen and Widrow, 1989) D. Nguyen and B. Widrow. The truck backer-upper: An example of self-learning in neural networks. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

(Rivest and Schapire, 1987) Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. In *Proceedings of Foundations of Computer Science*, 1987.

(Sato *et al.*, 1990) M. Sato, K. Abe, and H. Takeda. Learning control of finite markov chains with explicit trade-off between estimation and control. In D.S. et al. Touretzky, editor, *Connectionist Models, Proceedings of the 1990 Summer School*, pages 287–300, San Mateo, CA, 1990. Morgan Kaufmann.

(Schmidhuber, 1991) Jürgen H. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Technische Universität München, 1991.

(Singh, 1992) Satinder P. Singh. Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning Journal*, 1992. (to appear).

(Sofge and White, 1990) A.D. Sofge and D.A. White. Neural network based process optimization and control. In *Proceedings of the 29th IEEE Conference on Decision and Control*, Dezember 1990.

(Sutton, 1984) Rich S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, Department of Computer and Information Science, University of Massachusetts, 1984.

(Sutton, 1988) Rich S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 1988.

(Sutton, 1990) Rich S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning, June 1990*, pages 216–224, 1990.

(Thrun and Möller, 1991) Sebastian B. Thrun and Knut Möller. On planning and exploration in non-discrete environments. Technical Report 528, GMD, Sankt Augustin, FRG, 1991.

(Thrun and Möller, 1992) Sebastian B. Thrun and Knut Möller. Active exploration in dynamic environments. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, San Mateo, CA, 1992. Morgan Kaufmann. (to appear).

(Thrun *et al.*, 1991) Sebastian B. Thrun, Knut Möller, and Alexander Linden. Planning with an adaptive world model. In D. S. Tourtezky and R. Lippmann, editors, *Advances in Neural Information Processing Systems 3*, San Mateo, 1991. Morgan Kaufmann.

(Watkins, 1989) Chris J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, England, 1989.

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

(Whitehead, 1991a) Steven D. Whitehead. Complexity and cooperation in Q-learning. 1991.

(Whitehead, 1991b) Steven D. Whitehead. A study of cooperative mechanisms for faster reinforcement learning. Technical Report 365, University of Rochester, Computer Science Department, Rochester, NY, March 1991.

Appendix: The complexity of reinforcement learning

Recall that an *Markovian environment* is defined as a mapping from states $\times$ actions to states $\times$ rewards, which furthermore is Markovian. The environment may be deterministic (unless otherwise noted), finite (i.e. the number of states n and the number of actions d are finite) and ergodic. By a *Markov decision task* we mean the task of, given an unknown environment, finding an *optimal policy*, a mapping from states to actions which yields for any state an optimal trajectory in terms of cumulated rewards. Let l denote an upper bound of all shortest paths between two arbitrary states s_1 and s_2 (*depth* of the state space). In the worst case $l = n - 1$, but at many tasks l is significantly smaller than n (e.g. for navigation in a d-dimensional grid l might be as small as $O(d\sqrt[d]{n})$). Finally, by a *trial* we mean one trajectory in state space, starting at some initial state, usually ending at the goal state or after each state having occurred at least once. If not otherwise noted, the term *first trial* refers to the first trajectory in state space during a reinforcement learning process. By a Markov decision task *with a goal state* we mean a Markov decision task, for which the reward function is 0 for all states except a designated goal state, s_{goal}, with $r(s_{goal}) = 1$.

Complexity of the first trial using a complete action model

Given that the agent has a *complete action model*, i.e. a model of the state transition function of the environment, an upper bound for the number of actions required for a trial is given by the following Theorem:

Theorem 1 (Complexity of the first trial using a complete action model): There is a local counter-based exploration rule, by which the number of actions required for the first trial in any finite ergodic deterministic Markov decision task using a complete action model is bounded above by $O(ln^2)$. The time for action selection is in $O(d)$, and the additional storage requirement is in $O(n)$.

Before we prove the Theorem, we will define the exploration rule and prove two Lemmata.

For the exploration rule, which indeed is a (slightly modified, see Step 2 below) counter-based exploration rule, we associate to each state s a counter $c(s)$. This counter indicates in essence how often state s has occurred before and is updated only if the state of the environment is s. We use $c^t(s)$ to refer to $c(s)$ at time tick t ($t = 0, 1, \ldots$), meaning $c(s)$ after t actions being executed.
We denote the state of the environment after t actions executed by s^t. Furthermore, let the set of states which can be reached from state s by one single action be denoted by $Succ(s)$. Then we use the following exploration rule for generating actions and updating the look-up table c:

Exploration rule: In the beginning, the time t is set to 0 and for all states s the c-table is initialized by 0: $(\forall s)\ c^0(s) \longleftarrow 0$. At each time tick $t = 0, 1, \ldots$, select an action by the following (local) rule:

1. Let s_{min} be that state in $Succ(s^t)$ that minimizes $c(s)$ over all possible next states $s \in$

$Succ(s^t)$. If several such states exist, choose one randomly. Note that this selection is done in $O(d)$ time on a single processor.

2. If $c(s^t) \leq c(s_{min})$, i.e. the c-value of the current state is not greater than the c-value of s_{min}, set $c(s^t)$ to $c(s_{min}) + 1$.
3. (Now $c(s^t) > c(s_{min})$.) Select and execute the action that leads to s_{min}. By performing this action, the counter of the new actual state $c(s_{min})$ is incremented by 1, and the time t is also incremented by 1.
4. Goto 1.

Note t is the number of actions performed so far, and $\sum_s c^t(s)$ is an upper bound for t. Furthermore, $c^t(s)$ is an upper bound for the number state s occurred so far and c is monotonically increasing in t. The storage requirement for c is in $O(n)$, assuming that the storage required for storing a integer $\in \{0, 1, \ldots, n\}$ is in $O(1)$.

The following Lemma shows that if the agent takes a sequence of actions to get from a state with counter m to a state with counter $\geq m+1$, then Step 2 above is executed at least once on some state $\hat{s}$, whose counter was $c(\hat{s}) \leq m$ formerly and is now $c(\hat{s}) \geq m+1$.

<u>Lemma 1:</u> *Let for some $m \geq 0, t' > t \geq 0$: $c^t(s^t) = m$ and $c^{t'}(s^{t'}) \geq m+1$. Then there is a $\hat{t} \in \{t, t+1, \ldots, t'\}$ with $c^t(s^{\hat{t}}) \leq m$ and $c^{t'}(s^{\hat{t}}) \geq m+1$.*

<u>Proof:</u> There are two ways to increase a counter of a state: Step 2 and Step 3. Whenever Step 3 is performed, the agent gets from a state s_1 to a state s_2 with $c(s_2) < c(s_1)$ and the counter of s_2 is subsequently incremented by one. Therefore, if the counter of the current state is less than or equal to m, Step 3 cannot change this: the counter of the new current state will again be less than or equal to m. The counter of the current state can only be increased by Step 2.

The remainder of this proof is simple. Since $c^t(s^t) = m$, there must be a intermediate state $s^{\hat{t}}$ at which Step 2 is applied, updating the counter $c^{\hat{t}}(s^{\hat{t}})$ from a value $\leq m$ to a value $\geq m+1$. The Lemma follows from the fact that $c(\cdot)$ is monotonically increasing in t. □

The next Lemma shows that there are no arbitrarily large "jumps" in the c-table.

<u>Lemma 2:</u> *At any time t, for all states s and $\hat{s}$ with $s \in Succ(\hat{s})$, the following holds: $c^t(\hat{s}) - c^t(s) \leq n$.*

<u>Proof:</u> Choose s and $\hat{s}$ arbitrarily with $s \in Succ(\hat{s})$. Let $m := c^t(s)$ and $\hat{m} := c^t(\hat{s})$.

Indirect proof: Assume $\hat{m} - m > n$. First we will show that, beginning at a certain time $\hat{t}$, the counter of state $\hat{s}$ was only increased by Step 3 of above exploration rule, which implies that the change of $c(\hat{s})$ in $\{\hat{t}+1, \ldots, t\}$ equals exactly the number occurrences of $\hat{s}$ in that time interval.
More precisely, since $\hat{m} > m$, there was a time $\hat{t} < t$ at which $c(\hat{s})$ became greater than m. This happened either by Step 2 or Step 3 of the exploration rule, but in either cases $c^{\hat{t}}(\hat{s}) = m+1$, since the succeeding state s had a counter $c^{\hat{t}}(s) \leq m$ (thus $c^{\hat{t}}(s_{min}) \leq m$ and Step 2 could not increase $c^{\hat{t}}(\hat{s})$ to a value $\geq m+1$).
The argument for the next $t - \hat{t}$ actions is basically the same: for all $\tau \in \{\hat{t}+1, \ldots, t\}$ holds: $c^{\tau}(\hat{s}) > m$ and $c^{\tau}(s) \leq m$. Since $s \in Succ(\hat{s})$, for all those τ at which state $\hat{s}$ occurred was $c^{\tau}(s_{min}) \leq c^{\tau}(s) \leq m$. This again implies that Step 2 was not executed whenever the agent was in state $\hat{s}$, and $c(\hat{s})$ was only incremented by entering $\hat{s}$, i.e. by Step 3. Therefore $\hat{s}$ occurred exactly $(\hat{m} - m - 1)$ times in $\{\hat{t}+1, \ldots, t\}$.

Since per assumption $\hat{m} - m - 1 \geq n$, the $\hat{s}$ occurred at least n times in $\{\hat{t}+1, \ldots, t\}$. Step 1 always chooses the state with the smallest counter. Therefore the agent left always to states with counter $\leq m$ within this time interval. It also returned at least $n-1$ times to $\hat{s}$, which itself had a counter $\geq m+1$. Lemma 1 implies that for each return at least one state changed its counter from a value $\leq m$ to a value $\geq m+1$. Since c is monotonically increasing, each counter (i.e. each state) can do this only once. Thus, after $n-1$ returns all $n-1$ remaining states (i.e. states $\neq \hat{s}$) have counter $\geq m+1$, which still holds at time t. This contradicts the assumption $c^t(s) = m$. □

The proof of the Theorem is now simple:

Proof of the Theorem: The counter of the last state of a trial when finally finding it is $c(s_{last}) = 1$. By Lemma 2 and induction over all paths to this state it follows that no state has a counter $c(s) > 1 + (n-1) * l = ln - l + 1$. Thus, each state occurred less than ln times, which implies $\sum_s c(s) < ln^2$. As mentioned above, this is an upper bound for the number of actions t, which completes the proof of the Theorem. □

Remark: The proof of Theorem 1 implies that, if a state is not found after ln^2 steps, there is no path from the current state to this state at all. The environment is then not ergodic and *reducible* and the states the agent cycles in can be outruled and the trial restarted.

Complexity of the first trial without a complete action model

A crucial restriction of above Theorem is that it holds only if a complete action model is available, which is used for action selection in Step 1. In adaptive Markov decision tasks in general one seeks an optimal policy without having a complete action model in advance (Barto *et al.*, 1991), (Watkins, 1989). The widely used Q-learning algorithm does not rely on a complete action model either. One straightforward and common way to identify optimal policies in such cases is to do system identification and policy iteration simultaneously: Whenever the agent selects some action a which has not been selected at the current state s before, it keeps track of the resulting subsequent state s'. It incrementally constructs an internal action model, which approximates the transition function of the environment. We will derive a complexity bound for this more general case based on this principle.

Theorem 2 (Complexity of the first trial without an action model): There is a local counter-based exploration rule, by which the number of actions required for the first trial in any finite ergodic deterministic Markov decision task is bounded above by $O(dln^2)$. The time required for action selection is in $O(d)$, and the additional storage requirement is in $O(dn)$ for the model.

For deriving this more general complexity bound, we have to modify the exploration rule. In essence, the extension forces the agent to select new actions, if any, in order to construct an internal action model as fast as possible. This is achieved by extending above exploration rule:

0. If there is any action a which has never been tried before at the current state s, select a (regardless of the counter-table c). Execute a and increment the counter of the new state, as well as t by one. Store the observed state transition in the action model. Goto 0.

   $\vdots$

4. Goto 0.

Note that whenever Step 1, 2 or 3 is executed, the precondition of Step 0 is not fulfilled, meaning that all actions have been tried at the current state s. Then the internal model is complete for state s and allows for a complete 1-ply lookahead for all valid actions at this particular state.

Note also that all of the remarks after the exploration rule still hold true.

> Proof of the Theorem: Lemma 1 is violated by the extended exploration rule, since Step 0 might bring the agent to a state with a counter larger than the current counter. Since the number of state-action pairs is generally bounded by dn, this can only happen at most dn times in the whole trial. Therefore Lemma 1 is extended to:
>
> *Lemma 3: Let for some $m \geq 0, t' > t \geq 0$: $c^t(s^t) = m$ and $c^{t'}(s^{t'}) \geq m+1$. Then there is a $\hat{t} \in \{t, t+1, \ldots, t'\}$ at which Step 0 was used for action selection (i.e. the corresponding action a has not been tried before at $s^{\hat{t}}$), or Lemma 1 holds.*
>
> Following the proof of Lemma 2, Lemma 3 implies that the number of occurrences of a state $\hat{s}$ with $c^t(\hat{s}) = \hat{m}$ in the time interval $\{\hat{t}+1, \ldots, t\}$ is bounded above by $n + nd$.

The additional complexity of nd stems from "new actions" bringing the agent to states with larger counter. Since per definition, $c^\tau(\hat{s}) > m = c^t(s)$ for all $\tau \in \{\hat{t}+1, \ldots, t\}$, $m' - m + 1$ counts how often the agent entered $\hat{s}$ in this interval. For each occurrence there are now two possibilities: a) Lemma 2 holds and some other state changed its counter from a value $\leq m$ to a value $\geq m+1$, or b) there was at least one "new" action tried which led from a state with counter $\leq m$ to a state with counter $\geq m+1$. There are at most $n-2$ states which can change its counter to a value $\geq m+1$ (all states except s and $\hat{s}$), and the number of new actions is bounded by dn, yielding the upper bound $\hat{m} - m - 1 \leq n - 2 + dn < n + dn$. This leads to the extended version of Lemma 2:

Lemma 4: At any time t, for all states s and all states $\hat{s}$ with $s \in Succ(\hat{s})$, the following holds: $c^t(\hat{s}) - c^t(s) \leq n(d+1)$.

The remainder of the proof is equivalent to the proof of Theorem 1, taking the additional complexity factor d into account. This leads to the upper bound $O(dln^2)$ for the number of actions taken in any finite ergodic deterministic Markov decision task using the extended counter-based exploration rule. □

After d occurrences of a state s, all valid actions have been tried at least once and the action model is complete for this state. Therefore a worst-case complexity of system identification is given by the complexity of d exploration walks each with at least one occurrence of each state:

Corollary 2 (Complexity of system identification): At any finite ergodic deterministic Markovian environment the number of actions required for system identification using above exploration technique is bounded by $O(d^2ln^2)$.

Complexity of deterministic reinforcement learning with a goal state

We will now show that ergodic finite deterministic Markov Decision tasks with a goal state s_{goal} can be solved by reinforcement learning in polynomial time using counterbased exploration. This Theorem basically combines the costs of finding a state with the costs of Dynamic Programming.

Theorem 3 (Complexity of finite ergodic deterministic Markov Decision tasks with a goal state): With above exploration rule, the number of actions required for finding an optimal policy in any finite ergodic deterministic Markov Decision task with a goal state is bounded above by $O((d^2l+l^2)n^2)$. If a complete action model is available in advance, the complexity reduces to $O(l^2n^2)$. The reinforcement learning rule is local and takes $O(d)$ time for each iteration[12].

[12] The time requirement for the learning rule influences the number of actions required for learning an optimal policy, because there is a trade-off between off-line processing time and number of actions taken. We chosed $O(d)$ as an upper bound for real-time tasks, since the same time is required for action selection – by this the exploration rule

Let $r(s)$ denote the immediate reward received at state s. Before learning, the utility-values $\hat{V}(s)$ of all states are initialized by $l \cdot \min_s r(s)$, which is a lower bound for $\hat{V}^{\pi^*}$ along the optimal path to the goal, and $\hat{V}(\cdot)$ is subsequently updated by the following rule *update*:

Learning rule *update*:

$$\hat{V}^{t+1}(s) \quad = \quad \begin{cases} \max\{\hat{V}(s), r(s) + \gamma \cdot \max\limits_{a \text{ is action}} \hat{V}^t(Succ(s,a))\} & \text{, if } s = s^t \\ \hat{V}^t(s) & \text{, otherwise} \end{cases}$$

The learning rule establishes a standard policy iteration rule for real-time asynchronous dynamic programming and extends the formerly introduced reinforcement learning rule based on the assumption that the environment is deterministic. At the actual state s^t, *update* updates the utility estimate of state s^t with respect to all neighboring states using the current internal action model (if this is learned on fly, unknown transitions are simply neglected). Note that *update*(s) takes at most $O(d)$ computing time.

Proof of the Theorem: The reinforcement learning rule implies two important observations. Firstly, $\hat{V}(\cdot)$ is monotonically increasing in the learning time. Secondly, if $\langle s = s_1, s_2, \ldots, s_k = s_{goal}\rangle$ is an optimal path for some initial state s, and $\hat{V}(s_i)$ is correct for some state s_i on this path ($0 < i \leq k$), then applying the learning rule at state s_{i-1} will assign the correct utility-value to s_{i-1}, given that the action leading to state s_i is known to the current action model.

System identification and subsequently browsing the whole state space l times will identify the correct utility function and therefore an optimal policy, given that there is a goal state.

- If no complete action model is given in advance, identify the action model. By Corollary 2 this can be done with at most $O(d^2ln^2)$ actions.
- Do l exploration trials and apply the learning rule *update*(s) at each state s. With each exploration run each state occurs – under the preconditions of Corollary 1 – at least once. Thus along each shortest path to the goal the correct utility value is aligned backwards at least one state. Since the length of shortest paths is bounded by l, after $l+1$ iterations the optimal policy is found.

Note that system identification and learning are completely separated. In the learning runs, a complete action model is available. By Theorem 1 the complexity of each trial is then bounded above by $O(ln^2)$. The resulting worst-case complexity for reinforcement learning is then in $O(l^2n^2)$, if a complete action model is available in advance, and $O((d^2l + l^2)n^2)$, if not. $\square$

---

does not increase the processing time at each state by more than a constant factor.

Although this proves that reinforcement learning (under the above preconditions) using counter-based exploration is always polynomial in time, we expect the expected learning time to be much smaller than this worst-case bound. E.g. for this proof we did not utilize the utility-estimations for action selection, although action selection usually relies on the utility estimations, which has heuristically been found to accelerate the convergence of reinforcement learning. Moreover, we found empirically that in many tasks the expected time for finding the goal using counter-based exploration is much smaller.

Complexity of reinforcement learning in stochastic environments

The probably most crucial assumption we made throughout all the theorems is that the environment is deterministic. Reinforcement learning and adaptive control is often applied for non-deterministic Markovian environments, at which state transitions as well as reinforcement are drawn by non-deterministic probability distributions, and the utility-values reflect *expected* utilities. We will show that for above complexity bounds the condition of determinism cannot be dropped.

Theorem 4 (Complexity of some ergodic non-deterministic Markov Decision tasks): For each $n \in \{2, 3, \ldots\}$*, there is a ergodic non-deterministic Markov Decision task, at which the expected time for reaching the goal state is exponential in* n*, even if the optimal policy is given in advance.*

> Proof: We will reduce this Theorem to Whitehead's Theorem. Consider the following one-dimensional environment (similar to that depicted in Fig. 1): There are n states $\{s_1, \ldots, s_n = s_{goal}\}$ and the initial state is s_1. For each state, there is only one action: a, thus the optimal policy is selecting a all the time. The environment is non-deterministic: With probability $P_{-} = \frac{2}{3}$ it brings the agent from the current state s_i ($i \in \{1, \ldots, n\}$) to s_{i-1} (to s_i, if $i = 1$) and with probability $P_{+} = \frac{1}{3}$ to s_{i+1} (to s_i, if $i = n$). For this particular environment, Whiteheads theorem applies, showing that the number of actions required for entering the goal state is exponential in n. □

A direct consequence of this Theorem, as Whitehead points out, is that the expected time for reinforcement learning in such tasks is at least exponential in n. Theorem 4 illustrates the potential effects of non-determinism. However, those environments "untractable", since even an optimal policy cannot prevent from exponential time. Little is known for more realistic non-deterministic tasks, at which the expected time for an agent to get to the goal state is linear (or polynomial with small degree) in n.