KTH Computer Science
and Communication

Embodied Evolution of Learning Ability

STEFAN ELFWING

Doctoral Thesis
Stockholm, Sweden 2007

TRITA-CSC-A 2007:16
ISSN-1653-5723 KTH School of Computer Science and Communication
ISRN-KTH/CSC/A--07/16--SE SE-100 44 Stockholm
ISBN 978-91-7178-787-3 SWEDEN

Akademisk avhandling som med tillstånd av Kungliga Tekniska högskolan framläggs till offentlig granskning för avläggande av teknologie doktorsexamen måndagen den 12 november 2007 kl. 10.00 i sal F3, Lindstedtsvägen 26, Kungliga Tekniska högskolan, Stockholm.

Tryck: Universitetsservice US AB

Abstract

Embodied evolution is a methodology for evolutionary robotics that mimics the distributed, asynchronous, and autonomous properties of biological evolution. The evaluation, selection, and reproduction are carried out by cooperation and competition of the robots, without any need for human intervention. An embodied evolution framework is therefore well suited to study the adaptive learning mechanisms for artificial agents that share the same fundamental constraints as biological agents: self-preservation and self-reproduction.

The main goal of the research in this thesis has been to develop a framework for performing embodied evolution with a limited number of robots, by utilizing time-sharing of subpopulations of virtual agents inside each robot. The framework integrates reproduction as a directed autonomous behavior, and allows for learning of basic behaviors for survival by reinforcement learning. The purpose of the evolution is to evolve the learning ability of the agents, by optimizing meta-properties in reinforcement learning, such as the selection of basic behaviors, meta-parameters that modulate the efficiency of the learning, and additional and richer reward signals that guides the learning in the form of shaping rewards. The realization of the embodied evolution framework has been a cumulative research process in three steps: 1) investigation of the learning of a cooperative mating behavior for directed autonomous reproduction; 2) development of an embodied evolution framework, in which the selection of pre-learned basic behaviors and the optimization of battery recharging are evolved; and 3) development of an embodied evolution framework that includes meta-learning of basic reinforcement learning behaviors for survival, and in which the individuals are evaluated by an implicit and biologically inspired fitness function that promotes reproductive ability. The proposed embodied evolution methods have been validated in a simulation environment of the Cyber Rodent robot, a robotic platform developed for embodied evolution purposes. The evolutionarily obtained solutions have also been transferred to the real robotic platform.

The evolutionary approach to meta-learning has also been applied for automatic design of task hierarchies in hierarchical reinforcement learning, and for co-evolving meta-parameters and potential-based shaping rewards to accelerate reinforcement learning, both in regards to finding initial solutions and in regards to convergence to robust policies.

Keywords: Embodied Evolution, Evolutionary Robotics, Reinforcement Learning, Shaping Rewards, Meta-parameters, Hierarchical Reinforcement Learning, Learning and Evolution. Meta-learning, Baldwin Effect, Lamarckian Evolution.

Contents

Chapter 1

Introduction

The topic of this thesis is meta-learning in an evolutionary context. The research has been conducted within the Cyber Rodent project (Doya and Uchibe, 2005), and the objective of the thesis is well summed up in the goal of the project: to investigate the origins of our reward and affective systems by building artificial agents that share the same intrinsic constraints as biological agents, namely self-preservation and self-reproduction. More specifically, the main objective of thesis is the development of a framework for performing embodied evolution with a limited number of robots, by utilizing time-sharing of subpopulations of virtual agents inside each robot. Embodied evolution (Watson, Ficici, and Pollack, 2002) is a methodology for evolutionary robotics (Nolfi and Floreano, 2000), where a population of robots freely interact with each other, and the evaluation, selection, and reproduction are performed in an autonomous and distributed manner between the robots, without any need of human involvement. A distinctive feature of evolutionary robotics and embodied evolution is the design of the fitness function. Ideally, the fitness function should only consider the behavioral outcome and consists of as few components as possible. The proposed embodied evolution framework applies an implicit and biologically inspired selection scheme, in which there is no explicit representation or communication of the individuals' fitness information. An individual can only reproduce offspring by exchanging genotypes with individuals that control other robots in the environment, and the probability of reproducing offspring is dependent on the individual's "health" at the mating occasion. The standard approach in evolutionary robotics is to directly evolve the behaviors of the individuals, which are typically controlled by a neural network. In this thesis, the evolutionary objective is instead to evolve the learning ability of the individuals, by optimizing parameters that: 1) control the selection of basic learning behaviors according to the current environmental state and the individual's internal energy level; 2) directly influence the learning updates or controls the exploration of the environment, called meta-parameters, and 3) modulate additional and richer reward signals in the form of shaping rewards. The combination of learning and evolution used in this thesis is

related to the work by Floreano and Mondada (1996) and Urzelai and Floreano (2000). Instead of directly evolve the weights of a neural controller, they evolved, for each synapse, the learning rule for updating the strength of the synapse and a meta-parameter that controlled the learning rate of the update.

Reinforcement learning is a computational approach to learning from the interaction between an agent and its environment, to achieve a goal. Reinforcement learning algorithms have been successfully applied to play backgammon at master human level (Tesauro, 1994, 1995), perform space shuttle pre-launch scheduling better than NASA's engineers (Zhang and Dietterich, 1995), and perform autonomous helicopter flight on a real RC helicopter (Abbeel, Coates, Quigley, and Ng, 2007). There is also strong evidence for believing that reinforcement learning is a plausible learning mechanism in animals. For example, Schultz, Dayan, and Montague (1997) showed in monkey experiments that the activation of dopamanergic cells predicts rewards according to reinforcement learning theory. The common characteristic of all successful reinforcement learning applications for robotics and control systems is that they require tuning of meta-properties of the reinforcement learning algorithms, in the form of domain knowledge provided by the human designer. In this thesis, all properties that are not specified by the reinforcement learning algorithms are considered meta-properties. From an engineering perspective, the optimization of meta-properties is a difficult issue, because of the large search space of potential solutions, and because of the perspective of the human designer differs from the perspective of the agent performing the task. The agent's behavior is an emergent property of sensorimotor interactions between the agent and the environment, which are difficult to predict in advance. From our more biological perspective, it is natural to use an evolutionary approach to meta-learning. At least this is true in nature, since learning in animals is the result of an evolutionary process, albeit a very long process. Advantages of an evolutionary approach include that: 1) evolutionary computation methods are global and model-free search techniques; 2) it is possible to co-evolve different meta-properties that may influence each other; and 3) the specifics of the task and environment are naturally considered. In addition to the development of the embodied evolution framework, the evolutionary approach to meta-learning is also applied as a design tool for automatic optimization of meta-properties in reinforcement learning. The considered applications in this thesis are: 1) decomposition of an overall task into suitable subtasks and, thereby, constructing a task hierarchy that is adapted to both the task and the environment, and 2) co-evolution of meta-parameters and shaping rewards to improve the learning both in regards to initial performance and in regards to convergence speed to robust policies.

1.1 Scientific Contributions

This thesis is based on five papers presented at conferences or published in, or submitted to, journals related to machine learning, robotics, adaptive behavior, autonomous artificial systems, and evolutionary computation.

The five appended papers are

I. Stefan Elfwing, Eiji Uchibe, Kenji Doya, and Henrik I. Christensen. (2004). Multi-Agent Reinforcement Learning: Using Macro Actions to Learn a Mating Task. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-2004)*, pages 3164–3169.

II. Stefan Elfwing, Eiji Uchibe, Kenji Doya, and Henrik I. Christensen. (2005). Biologically Inspired Embodied Evolution of Survival. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2005)*, pages 2210–2216.

III. Stefan Elfwing, Eiji Uchibe, Kenji Doya, and Henrik I. Christensen. (2007). Darwinian Embodied Evolution of the Learning Ability for Survival. Submitted to *Adaptive Behavior*.

IV. Stefan Elfwing, Eiji Uchibe, Kenji Doya, and Henrik I. Christensen. (2007). Co-Evolution of Shaping Rewards and Meta-Parameters in Reinforcement Learning. Submitted to *Adaptive Behavior*.

V. Stefan Elfwing, Eiji Uchibe, Kenji Doya, and Henrik I. Christensen. (2007). Evolutionary Development of Hierarchical Learning Structures. *IEEE Transactions on Evolutionary Computations*, 11(2):249–264.

1.2 Outline

This thesis is organized around five appended papers. Before the papers, the background chapters present the technical background, and my perspectives and motivations behind this work.

Chapter 1 is a general introduction. Chapter 2 presents the technical background of standard reinforcement learning, eligibility traces, and value function approximation. Chapter 3 discusses the limitation of the knowledge-free assumption in standard reinforcement learning, and presents and motivates three approaches to introduce domain knowledge in reinforcement learning. Chapter 4 introduces the background of genetic algorithms, genetic programming, evolutionary robotics, and embodied evolution. Chapter 4 also presents and motivates the specifics of the evolutionary meta-learning approaches used in this thesis. Chapter 5 summarizes the appended paper and highlights the scientific contributions. The background concludes with Chapter 6, which also discusses possible future research directions.

1.3 Acknowledgments

My graduate studies have been a joint collaboration between Sweden and Japan. I would first like to thank the persons who made this arrangement possible, my supervisors Henrik I. Christensen and Kenji Doya. I thank Eiji Uchibe for his extensive knowledge of everything related to robotics and the software development of the Cyber Rodents, without which this work could not have been done. I am grateful to all three for inspirational ideas, valuable feedback, and support from the start of my graduate studies to the completion of this thesis.

This work has been supported by a shared grant from the Swedish Foundation for Internationalization of Research and Education (STINT) and the Swedish Foundation for Strategic Research (SSF). The funding is gratefully acknowledged.

This thesis is dedicated to my wife Myriam and our two sons Marcel and Jack.

Chapter 2

Reinforcement Learning

This chapter presents the basic theory of reinforcement learning, eligibility traces, and value function approximation. The focus is on model-free temporal difference learning, which is the basis for all learning in this thesis. Along the way pointers are given to alternative reinforcement learning approaches.

2.1 Introduction

Reinforcement learning (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998) is a computational approach to learning from the interaction between an agent and its environment to achieve a goal. The main difference between reinforcement learning and supervised learning is that there is no teacher telling the agent which actions to take. Instead, the only feedback received by the agent, from the environment, are scalar reward values, indicating the values of the state transitions. The reinforcement learning agent has to learn, by trial-and-error search, which actions yield most rewards over time. The trade-off between exploiting and exploring is a distinguishing feature of reinforcement learning. The *exploitation-exploration dilemma* is also a major challenge for reinforcement learning, because the agent has to exploit its accumulated knowledge to obtain rewards, but at the same time, the agent has to explore the environment to gain new knowledge and, thereby, be able to make better action selections in the future. This is especially true for stochastic tasks, in which each state-action pair has to be tried repeatedly to get a reliable estimate of its expected reward.

In most interesting problems, an action not only affects the immediate reward, but also the next state and, thereby, all subsequent rewards. Delayed rewards are another distinguishing feature of reinforcement learning. The agent may have to select a long sequence of actions, receiving insignificant rewards, to reach a state with large reward. The reward feedback then has to be propagated back along the path of preceding states to give correct estimates of the values in the earlier

states. Consider the example of a reinforcement learning agent playing chess. The agent receives a reward of $+1$ for a won game, -1 for a lost game, and 0 for a draw. If a game ends after 40 moves, how much credit for the final outcome should the agent assign to the action performed at move 20? This problem, the *temporal credit assignment problem*, is another major challenge that has to be addressed for effective learning.

In problems with large or continuous state spaces, i.e., the interesting domain for robotic tasks, the reinforcement learning agent has to address a second assignment problem, the *structural credit assignment problem*. In such problems the agent can not visit all possible states and thereby not estimating the value of each state-action pair directly. Therefore, the agent has to generalize over the state space, by assigning similar estimated values to states with similar structures.

2.2 Basic Concept

This section contains the description of the basic concept of reinforcement learning. The state space, the action space, and the time steps are assumed to be discrete. Each time step the learner, called an *agent*, executes an *action*, which immediately changes the environmental *state*, and the agent receives a *reward* from the environment (Figure 2.1).

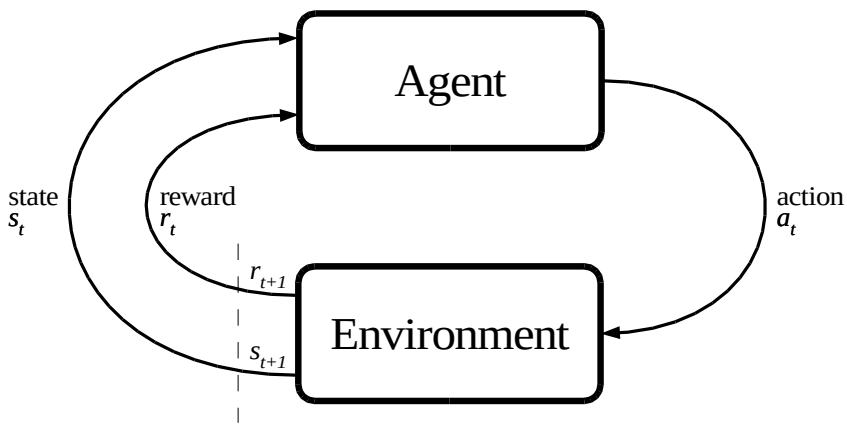

Figure 2.1. The agent-environment interaction in reinforcement learning.

Formally, in each time step, $t = 0, 1, 2, \ldots$, the agent receives a representation of the state of the environment, $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states. On the basis of s_t the agent selects an action, $a_t \in \mathcal{A}(s)$, where $\mathcal{A}(s)$ is the set of all possible actions in state s_t. The environment then makes a transition from s_t to s_{t+1}, and the agent receives a scalar reward, $r_{t+1} \in \Re$. The goal of the

reinforcement learning agent is to learn a state-action mapping, called a *policy*, π_t, that maximizes the reward over time. $\pi_t(s,a) = P(a_t|s_t)$ is the probability of that action $a_t = a$, given state $s_t = s$. The most common goal used in reinforcement learning, and the goal used in this thesis, is to maximize the expected accumulated discounted reward over time, called the *discounted return*, R_t, defined as

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.1}$$

where the parameter $0 \leq \gamma \leq 1$ is called the *discount rate*. γ determines the value of future rewards at the current state, and thereby how farsighted the agent is. If γ is zero, then the agent only tries to maximize the immediate rewards. As γ approaches one, future rewards are more strongly weighted in the calculation of R_t.

An alternative to discounting is to maximize the average reward per time step (Mahadevan, 1996), which, in theory, is more appropriate for cyclic tasks.

2.3 Markov Decision Process

Almost all reinforcement learning algorithms realize the goal of maximizing the expected discounted return by estimating *value functions*. Value functions are functions of states (or of state-action pairs) that estimate the value, with respect to a policy, of a given state (or a given action in a given state). The estimation of value functions is based on the assumption that reinforcement learning problems satisfy the Markov property and can be considered a finite *Markov decision process* (MDP). A reinforcement learning problem is a finite MDP if the state space and the action space are finite, and if the state at time $t+1$, s_{t+1}, only depends on the previous state, s_t, and action, a_t. A finite MDP is defined by the state and action spaces, and the one-step dynamics of the environment. Given a state, s, and an action, a, the *transition probability* of the each possible next state, s', is

$$P(s'|s,a) = P(s_{t+1} = s'|s_t = s, a_t = a), \tag{2.2}$$

and the expected value of the next reward, $R(s'|s,a)$, is

$$R(s'|s,a) = E\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\}. \tag{2.3}$$

For a MDP, the value of state s under a policy π, the *state-value function* V^π, is the expected return when starting in state s, and following the policy π:

$$V^\pi(s) = E_\pi\{R_t|s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\right\}. \tag{2.4}$$

Similarly, the value of selecting action a in state s under a policy π, the *action-value function* Q^π, is defined as

$$Q^\pi(s,a) = E_\pi\{R_t|s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\right\}. \tag{2.5}$$

The state-value for any state s and policy π, $V^\pi(s)$, can be recursively defined, using dynamic programming techniques (Bellman, 1957), by the *Bellman equation* for V^π, as

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P(s'|s,a) \left[R(s'|s,a) + \gamma V^\pi(s') \right]. \tag{2.6}$$

The Bellman equation for Q^π is defined as

$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a) \left[R(s'|s,a) + \gamma \sum_{a'} \pi(s',a') Q^\pi(s',a') \right]. \tag{2.7}$$

For a finite MDP the optimal policy, π^*, i.e., the objective of the learning, can be defined exactly. A policy is optimal, $\pi^* \geq \pi$, if and only if $V^{\pi^*}(s) \geq V^\pi(s)$ for all $s \in \mathcal{S}$ and all policies π . The optimal policies (there is at least one policy that is better or equal to all other policies) share the same state-value function, the *optimal state-value function* V^*. The Bellman equation for V^* is defined as

$$V^*(s) = \max_\pi V^\pi(s) = \max_a \sum_{s'} P(s'|s,a) \left[R(s'|s,a) + \gamma V^*(s') \right]. \tag{2.8}$$

The optimal polices also share the same *optimal action-value function*, Q^*, defined as

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) = \sum_{s'} P(s'|s,a) \left[R(s'|s,a) + \gamma \max_{a'} Q^*(s',a') \right]. \tag{2.9}$$

Given the optimal action-value function, the computation of the optimal policy is straightforward. For all states, the optimal policy is equal to the greedy policy, defined as

$$\pi^*(s) = \arg\max_a Q^*(s,a). \tag{2.10}$$

An alternative to value function-based reinforcement learning is *policy gradient reinforcement learning* (Williams, 1992; Sutton, McAllester, Singh, and Mansour, 2000; Baxter and Bartlett, 2001), which has attracted attention in recent years. Instead of estimating the value function and using that estimate to derive the policy, policy gradient reinforcement learning algorithms estimate the parameterized policy directly, using gradient-descent techniques.

2.4 Temporal-Difference Learning

The probably most central and novel idea proposed in reinforcement learning theory is *temporal-difference* (*TD*) learning. TD-learning combines ideas from Monte Carlo methods, i.e., learning from experience without a model of the environmental dynamics, and dynamic programming, i.e., updating the estimates of the value function based on other learned estimates, without the need to wait for the final

result, called *bootstrapping*. Bootstrapping is a key feature of TD algorithms, because it makes it natural to implement TD algorithms on-line in a fully incremental fashion. The central part of TD-learning is the *TD-error*, δ_t, defined as

$$\delta_t = \overbrace{r_{t+1} + \gamma V(s_{t+1})}^{\text{value prediction}} - V(s_t). \tag{2.11}$$

In TD-learning the agent learns the value function by the difference between temporally successive value predictions. For the simplest TD method, TD(0), the value function is updated as

$$V(s_t) \leftarrow V(s_t) + \alpha\big[\overbrace{r_{t+1} + \gamma V(s_{t+1}) - V(s_t)}^{\delta_t}\big], \tag{2.12}$$

where $0 \leq \alpha \leq 1$ is a step-size parameter called *learning rate*. For any fixed policy π, TD(0) is guaranteed to converge to V^π, given sufficient decay of α and that there is sufficient exploration of the environment.

The extension of TD-learning to action-value functions are straightforward. The two probably most popular and well studied TD-learning algorithms are Sarsa (Rummery and Niranjan, 1994; Sutton, 1996) and Q-learning (Watkins, 1989), and they are the basis for all learning in this thesis. Sarsa is an *on-policy* algorithm, learning an estimate of the action-value function while the agent follows a particular policy. For simplest version of Sarsa, Sarsa(0), the Q-values are updated according to

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[\overbrace{r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)}^{\delta_t}\big]. \tag{2.13}$$

Q-learning is an *off-policy* algorithm, learning an estimate of the action-value function independently of the policy followed by the agent. For the simplest version of Q-learning, one-step Q-learning, the Q-values are updated according to

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[\overbrace{r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)}^{\delta_t}\big]. \tag{2.14}$$

Both Sarsa and Q-learning are proved to converge with a probability of 1 to the optimal policy and action-value function. However, the convergence proof for Q-learning is stronger, enabled by the fact that Q-learning estimates the optimal value function directly, without reference to the agent's exploratory policy.

Both Q-learning and Sarsa are model-free algorithms. The alternative is to learn an optimal policy by learning a model of the problem, i.e., the transition probability $P(s'|s,a)$ and the expected reward function $R(s'|s,a)$. Examples of this approach are the Dyna algorithm by Sutton (1990), and Prioritized Sweeping by Moore and Atkeson (1993)

2.5 Eligibility Traces

Eligibility traces is a basic mechanism for temporal credit assignment, and, thereby, to increase the efficiency of reinforcement learning algorithms. For action-value-based algorithms, each state-action pair is associated with a memory, the eligibility trace, $e_t(s, a)$. The TD-error, δ_t, is propagated back along the trajectory of state-action pairs leading to state s_t, decaying by $\gamma\lambda$ per time step. The eligibility traces versions of Sarsa(0) and one-step Q-learning are called Sarsa(λ) and Q(λ), and both update the Q-values according to

$$Q(s, a) \leftarrow Q(s, a) + \alpha\delta_t e_t(s, a) \qquad \text{for all } s,\ a \tag{2.15}$$

where δ_t is defined as in Equation 2.13 for Sarsa(λ), and as in Equation 2.14 for Q(λ). There are two basic implementations of traces, called *accumulating traces* and *replacing traces*. Accumulating traces for Sarsa(λ) are defined as

$$e_t(s, a) = \begin{cases} 1 + \gamma\lambda e_{t-1}(s, a) & \text{if } s = s_t \text{ and } a = a_t; \\ \gamma\lambda e_{t-1}(s, a) & \text{otherwise.} \end{cases} \quad \text{for all } s,\ a, \tag{2.16}$$

where the trace-decay rate, $0 \leq \lambda \leq 1$, controls the exponential decay of the traces. Q-learning is more difficult than Sarsa to combine with eligibility traces, because the learned policy, the greedy policy, is different than the policy used for selecting actions. The straightforward combination of accumulating traces and Q-learning, called Watkins's Q(λ) (Watkins, 1989), works like Sarsa(λ), except that all traces are set to 0 whenever an exploratory, non-greedy action, is selected. However, this approach loses much of the advantage of using traces, especially in the beginning of the learning when the agent usually needs to explore the environment frequently.

A potential problem with accumulating traces is that the trace for a state-action pair can incrementally become greater than 1, if the agent revisits a state and selects the same action. Replacing traces avoids this by setting the trace for the current state-action-pair to 1, instead of $1 + \gamma\lambda e_{t-1}(s, a)$. For some tasks replacing traces can significantly increase the learning performance compared with accumulating traces (Sutton, 1984). The implementation of eligibility traces for Sarsa(λ) used in this thesis, as recommended by Singh and Sutton (1996), is called replacing traces with optional clearing, and is defined as

$$e_t(s, a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t; \\ 0 & \text{if } s = s_t \text{ and } a \neq a_t; \\ \gamma\lambda e_{t-1}(s, a) & \text{if } s \neq s_t. \end{cases} \quad \text{for all } s,\ a \tag{2.17}$$

The optional clearing (the second line) sets the traces for all non-selected actions from the revisited state to 0.

In a naive implementation the computational cost of eligibility traces becomes very expensive, since it requires that the action values and the traces are updated for all state-action pairs in every time step. In practice this is not a major problem, because most state-action pairs have zero or very small traces. A simple solution is

therefore to truncate the traces and only store the traces for a limited number, m, of the most recent state-action pairs, i.e., the state-action pairs for which $e_t(s,a) \geq (\gamma\lambda)^m$.

2.6 Action Selection

In contrast to supervised learning, reinforcement learning agents have to explicitly explore the environment. The trade-off between exploitation of the current policy and exploration to gain new information is a fundamental dilemma. Unfortunately, there exist no formal justified techniques for exploring that scale to multi-state delayed reward tasks (Kaelbling, Littman, and Moore, 1996). Instead, two simple ad-hoc strategies are widely used in most reinforcement learning algorithms. The simplest and probably most popular exploration strategy is *ϵ-greedy*. The agent selects the greedy action most of the time, but with a probability of ϵ a random action is selected. A variation of this strategy is to start with a large ϵ value, to increase the initial exploration, and then slowly decrease ϵ as the learning converges. The obvious objection to ϵ-greedy is that it does not discriminate between non-greedy actions when exploring. A more advance alternative is *softmax* action selection with a Boltzmann distribution. The selection probability of action a in state s is defined as

$$P(a|s) = \frac{e^{Q(s,a)/\tau}}{e^{\sum_b Q(s,b)/\tau}}, \tag{2.18}$$

where the parameter $\tau > 0$, controlling the exploration rate, is called the *temperature*. Higher temperatures decrease the differences between the action selection probabilities, and lower temperatures increase the differences between the action selection probabilities. For the same reason as for the ϵ-greedy strategy, it is common to decay τ over time, either linearly, exponentially ($\tau_t = k \cdot \tau_{t-1}$, where k is a constant), or hyperbolically ($\tau_t = \tau_0/(1 + k \cdot t)$, where τ_0 is the initial temperature, and k is a constant).

A simple alternative to use an explicit stochastic policy is, e.g., to use greedy action selection together with *optimistic initial values*. Optimistic initialization means that all initial Q-values are uniformly assigned to a value that is larger than the optimal values. In the beginning of learning, whichever action the agent selects the reward will be less than the current estimated value. The agent is therefore forced to explore all actions several times before the value estimates converge. For example, this selection strategy is normally used in the mountain-car task (Moore, 1990), a standard benchmark task for continuous reinforcement learning. The mountain-car task is described in detail in Paper IV, where it is used as at testbed for validation of our evolutionary approach of co-evolving meta-parameters (see section 3.3), and potential-based shaping rewards (see section 3.2).

2.7 Generalization by Value Function Approximation

To calculate the optimal value functions we have assumed that the values for all states, or state-action pairs, can be stored explicitly in a lookup table. In practice this is not a feasible solution, except for small discrete problems, due to computational time and memory requirements. In general, the learning time grows exponentially with the size of the state space, often called the "*curse of dimensionality*". This means that for most problems the optimal solution is intractable, and we have to settle for near-optimality, i.e., an approximation of the optimal value function and the optimal policy. The key to learning in high-dimensional or in continuous state spaces is *generalization*, i.e., to generalize knowledge acquired in small subset of the state space to a larger subset. The type of generalization required for reinforcement learning is called *function approximation* and is, fortunately, a very well studied topic in supervised learning. In principal, any standard supervised technique that support noisy training examples can be applied to reinforcement learning, such as neural networks, pattern recognition, and statistical curve fitting. However, in practice the techniques often have to be adjusted to the specifics of the reinforcement learning algorithm and the task.

In this thesis two popular linear gradient-descent methods have been used: normalized radial basis function networks (Broomhead and Lowe, 1988) and tile coding (Sutton, 1996), also called CMAC (Albus, 1971, 1981). For linear gradient-descent methods, the approximated value function, $V_t(s) \approx V^\pi(s)$, is a linear function of a column parameter vector, $\boldsymbol{\theta}_t = (\theta_t(1), \theta_t(2), \ldots, \theta_t(n))^T$ (T denotes transpose), which gives

$$V_t(s) = \boldsymbol{\theta}_t{}^T \boldsymbol{\phi}_s = \sum_{i=1}^{n} \theta_t(i)\phi_s(i), \tag{2.19}$$

where $\boldsymbol{\phi}_s$ is a column vector of features with the same length, n, as $\boldsymbol{\theta}_t$. The gradient of the approximated value function with respect to the parameters, $\boldsymbol{\theta}_t$, is easily computed as

$$\nabla_{\boldsymbol{\theta}_t} V_t(s) = \boldsymbol{\phi}_s. \tag{2.20}$$

It is therefore natural for linear methods to use gradient-descent to update the parameter vector as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\delta_t \nabla_{\boldsymbol{\theta}_t} V_t(s) = \boldsymbol{\theta}_t + \alpha\delta_t \boldsymbol{\phi}_s, \tag{2.21}$$

where

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t). \tag{2.22}$$

This is the linear gradient-descent version of TD(0) (Equation 2.4).

It is straightforward to apply linear gradient-descent function approximation to action-value-based algorithms, such as Sarsa and Q-learning. The approximated

action value function, $Q_t(s,a) \approx Q^\pi(s,a)$, is parameterized by the matrix $\boldsymbol{\Theta}_t$ of size $n \times |\mathcal{A}|$ (assuming an equal number of actions for all states), and defined as

$$Q_t(s,a) = \sum_{i=1}^{n} \Theta_t(i,a)\phi_s(i), \text{ for all } a \in \mathcal{A}. \tag{2.23}$$

The column parameter vector $\boldsymbol{\theta}_t^a = (\Theta_t(1,a), \Theta_t(2,a), \ldots, \Theta_t(n,a))^T$, for action $a = a_t$, is updated as

$$\boldsymbol{\theta}_{t+1}^a = \boldsymbol{\theta}_t^a + \alpha\delta_t \nabla_{\boldsymbol{\theta}_t^a} Q_t(s,a) = \boldsymbol{\theta}_t^a + \alpha\delta_t \boldsymbol{\phi}_s, \tag{2.24}$$

where, for Sarsa, the TD-error, δ_t, is defined as

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t), \tag{2.25}$$

and for Q-learning δ_t is defined as

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t), \tag{2.26}$$

A natural function approximation for continuous state input is normalized *radial basis function* (*RBF*) networks. Typically, a normalized RBF network has normalized Gaussian features, ϕ_s, defined as

$$\phi_s(i) = \frac{e^{-\frac{||s-c_i||^2}{2\sigma_i^2}}}{\sum_{j=1}^{n} e^{-\frac{||s-c_j||^2}{2\sigma_j^2}}}. \tag{2.27}$$

The response of a feature i, $\phi_s(i)$, depends only on the distance between the

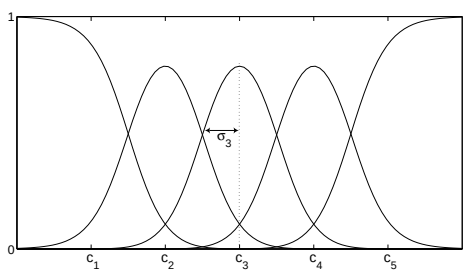

Figure 2.2. Five one-dimensional normalized radial basis functions

current state, s, and the center of the feature, c_i, and the width of the feature, σ_i. Figure 2.2 shows an example of five one-dimensional normalized RBFs using

Algorithm 2.1: Watkins's $Q(\lambda)$ with normalized RBF network function approximation

Variables: n, number of features (RBFs)
$\boldsymbol{\Theta}$, parameter matrix ($n \times |\mathcal{A}|$)
$\boldsymbol{\phi}_s$, RBF features for state s ($n \times 1$)
$\mathbf{e}$, eligibility trace for all features and actions ($n \times |\mathcal{A}|$)
$\boldsymbol{Q}_s$, Q-values for all actions $a \in \mathcal{A}$, $\boldsymbol{a}$, in state s ($1 \times |\mathcal{A}|$)

Initialize the parameter matrix $\boldsymbol{\Theta}$ arbitrarily
foreach *episode* **do**
  $\mathbf{e} \leftarrow \mathbf{0}$
  $s \leftarrow$ initial state of episode
  for $i \leftarrow 1$ **to** n **do**
    $\phi_s(i) \leftarrow \exp\left(-\frac{||s-c_i||^2}{2\sigma_i^2}\right) \Big/ \sum_{j=1}^{n} \exp\left(-\frac{||s-c_j||^2}{2\sigma_j^2}\right)$
  $\boldsymbol{Q}_s \leftarrow \boldsymbol{\phi_s}^T \boldsymbol{\Theta}$
  Select initial action a using policy derived from $\boldsymbol{Q}_s$
  repeat
    for $i \leftarrow 1$ **to** n **do**
      $e(i,a) \leftarrow e(i,a) + \phi_s(i)$
    Execute action a, observe reward r, and next state s'
    $\delta \leftarrow r - \boldsymbol{Q}_s(a)$
    for $i \leftarrow 1$ **to** n **do**
      $\phi_{s'}(i) \leftarrow \exp\left(-\frac{||s'-c_i||^2}{2\sigma_i^2}\right) \Big/ \sum_{j=1}^{n} \exp\left(-\frac{||s'-c_j||^2}{2\sigma_j^2}\right)$
    $\boldsymbol{Q}_{s'} \leftarrow \boldsymbol{\phi}_{s'}^T \boldsymbol{\Theta}$
    $\delta \leftarrow \delta + \gamma \max_a \boldsymbol{Q}_{s'}$
    $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} + \alpha\delta\mathbf{e}$
    Select next action a using policy derived from $\boldsymbol{Q}_{s'}$
    if $a = \arg\max_{a'} \boldsymbol{Q}_{s'}(a')$ **then**
      $\mathbf{e} \leftarrow \gamma\lambda\ \mathbf{e}$
    else
      $\mathbf{e} \leftarrow \mathbf{0}$
    $s \leftarrow s'$
    $\boldsymbol{Q}_s \leftarrow \boldsymbol{Q}_{s'}$
    $\boldsymbol{\phi}_s \leftarrow \boldsymbol{\phi}_{s'}$
  until *s is terminal*

Euclidean distance metric. The advantage of RBF networks is they produce smooth and differentiable approximated functions. The major drawbacks of RBF networks are that they add computational complexity and that it can be difficult to design the features. This is especially the case for high-dimensional state spaces, which require a very large number of features to cover the state space. It is also problematic to combine RBF networks and eligibility traces, because there is no discrimination

between different states. Algorithm 2.1 shows the pseudo-code for Watkins's Q(λ) with normalized RBF network function approximation. This method was used to approximate the value functions in the MAXQ framework (see section 3.4.3) in Paper V, and for learning with options (see section 3.4.2) in Paper I.

In contrast to RBF networks, tile coding is a weak generalization method that is very effective for on-line learning. Tile coding performs only local generalization, i.e., a change in a feature only affects a limited part of the approximated function. Tile coding represents the value of a continuous variable as a large binary feature vector with many 0s and a few, m, 1s. The idea is to partition the state-space multiple times, where the partitions are called *tilings* and the elements of the tilings are called *tiles* (or receptive fields). For each state exactly one tile is active in each tiling, corresponding to the 1s in the binary feature vector. The computation of the value function is therefore very simple, by summation of the $m << n$ components of the parameters corresponding to the non-zero features. The simplest version of

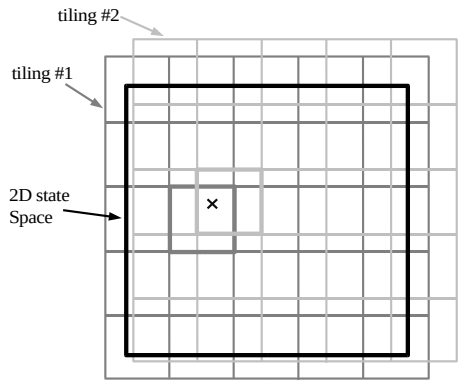

Figure 2.3. Two grid-like tilings in a 2D state space.

tile coding is when grid-like tilings are used, as illustrated for the two-dimensional state space in Figure 2.3. Each tiling is represented by an uniform grid, offset by a different amount, either randomly or by predefined design. The active tile for each tiling is easily computed given the coordinates (marked by $\times$ in the figure) for the current state. The generalization achieved by tile coding is determined by the number of tilings, and the shape of the tiles. A large number of tilings increases the accuracy of the function approximation, but also increases the computational cost. Broader features give smoother function approximation, but, in general, the design of the shape of the tiles has little effect on the final learning performance. Algorithm 2.2 shows the pseudo-code for Sarsa(λ) with tile coding function approximation and replacing traces with the optional clearing. This method is used for

approximating the action-value function in Papers II, III, and IV.

Algorithm 2.2: Sarsa(λ) with tile coding and replacing traces with the optional clearing

Variables: n, number of features (tiles)
N, number of tilings
$\boldsymbol{\Theta}$, parameter matrix ($n \times |\mathcal{A}|$)
$\mathbf{e}$, eligibility trace for all features and actions ($n \times |\mathcal{A}|$)
$\boldsymbol{I}_s$, Indexes of non-zeros features in state s ($N \times 1$)
$\boldsymbol{Q}_s$, Q-values for all actions $a \in \mathcal{A}$, $\boldsymbol{a}$, in state s ($|\mathcal{A}| \times 1$)

Initialize the parameter matrix $\boldsymbol{\Theta}$ arbitrarily
foreach *episode* **do**
  $\mathbf{e} \leftarrow \mathbf{0}$
  $s \leftarrow$ initial state of episode
  $\boldsymbol{I}_s \leftarrow$ set of non-zero features present in s
  $\boldsymbol{Q}_s \leftarrow \sum_{i \in \boldsymbol{I}_s} \Theta(i, \boldsymbol{a})$
  Select initial action a using policy derived from $\boldsymbol{Q}_s$
  repeat
    $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e}$
    foreach $i \in \boldsymbol{I}_s$ **do**
      $e(i, a) \leftarrow 1$ `/* replacing traces */`
      forall $a' \in \mathcal{A}$ **do**
        if $a' \neq a$ **then**
          $e(i, a') \leftarrow 0$ `/* optional clearing */`
    Execute action a, observe reward r, and next state s
    $\delta \leftarrow r - \sum_{i \in \boldsymbol{I}_s} \Theta(i, a)$
    $\boldsymbol{I}_s \leftarrow$ set of non-zero features present in s
    $\boldsymbol{Q}_s \leftarrow \sum_{i \in \boldsymbol{I}_s} \Theta(i, \boldsymbol{a})$
    Select next action a using policy derived from $\boldsymbol{Q}_s$
    $\delta \leftarrow \delta + \gamma \sum_{i \in \boldsymbol{I}_s} \Theta(i, a)$
    for $i \leftarrow 1$ **to** n **do**
      forall $a' \in \mathcal{A}$ **do**
        if $e(i, a') > 0$ *($> (\gamma\lambda)^m$ for truncated traces)* **then**
          $\Theta(i, a') \leftarrow \Theta(i, a') + \alpha\delta e(i, a')/N$
  until s *is terminal*

Value function approximation is, in general, an open question in reinforcement learning, due to the lack of mathematical proofs for convergence of the learning. There are several studies (Bradtke, 1993; Baird, 1995; Boyan and Moore, 1995) showing, usually by simple counter-examples, that off-policy reinforcement learning, such as Q-learning, with value function approximation can diverge to infinity. Fortunately, on-policy bootstrapping methods, such as Sarsa(λ), with linear value function approximation can be shown to converge, under standard assumptions,

such as sufficient exploration and decreasing α. The asymptotic mean square error, $MSE(\boldsymbol{\theta}_\infty)$ is bounded according to

$$MSE(\boldsymbol{\theta}_\infty) \leq \frac{1-\gamma\lambda}{1-\gamma} MSE(\boldsymbol{\theta}^*), \tag{2.28}$$

where $\boldsymbol{\theta}^*$ is the minimum-error parameter vector.

Chapter 3

Meta-Properties in Reinforcement Learning

This chapter discusses the limitations of the knowledge-free assumption of the basic TD-learning framework presented in the previous chapter. Three approaches used to introduce domain knowledge into the reinforcement learning framework are presented and motivated. A common characteristic of the approaches is that they require that the knowledge is provided in advance by the designer. In this thesis all properties of reinforcement learning that are not specified by the algorithms themselves are called meta-properties. The next chapter presents our evolutionary computation approach to meta-learning of the meta-properties discussed in this chapter.

3.1 Introduction

One of the main attraction with reinforcement learning, and especially with TD-learning algorithms, is that is a *tabula rasa*, i.e., knowledge-free, approach to learning. In this naive view the designer does not have to worry about the specifics of the task or the environment. The designer only has design a reward function specifying the goal of the task and then, almost magically, the agent will learn an optimal, or near optimal, behavior. Unfortunately, this approach only works for small discrete simulation problems. In real-world applications, the specifics of the task and the environment plays a major role for successful learning. This insight is far from new. For example, in the 1996 reinforcement learning survey by Kaelbling, Littman, and Moore (1996), they observed that the most striking common feature of all the five successful robotics and control applications they studied in detail, was that they required domain knowledge provided by the designer. For these tasks a knowledge-free approach could, maybe, find a good solution in theory, but in practice it would not have accomplished acceptable performance within the finite lifetime of the robots.

A good example of the limits of the knowledge-free approach, and the im-

portance of the specifics of the problem is the backgammon player TD-Gammon developed by Tesauro (1992, 1994, 1995), usually considered the most successful reinforcement learning application to date. Tesauro used TD-learning with little pre-programmed backgammon knowledge, and trained the system by letting it play against itself hundreds of thousands of times. However, the success of the of TD-Gammon was dependent on Tesauro's skillful design of a non-linear multilayered neural network, used for value function approximation in the Backgammon domain consisting of approximately 10^{20} states. The results of TD-gammon are very impressive (Sutton and Barto, 1998). The final version of TD-Gammon, TD-Gammon 3.0, plays at the level of the world's best professional players. However, this required that domain knowledge was provided to the learning system in the form of a selective three-ply search for selecting moves. The basic version, TD-Gammon 0.0, without any domain knowledge, played at par with the best computer player at the time, Neurogammon (Tesauro, 1989), but not at professional human level. The success of the TD-gammon seems to an exception in field of two-player games. Although, reinforcement learning has been applied to other major board games, such as checkers (Samuel, 1959), chess (Thrun, 1995), and go (Schraudolph, Dayan, and Sejnowski, 1994), no other application has been close to top human-level performance. The reason for this has probably to do with nature of backgammon (Kaelbling, Littman, and Moore, 1996). In backgammon, independent of the agent's action selection strategy: 1) a game ends within finite time, and 2) relevant states are continually visited, because the state transitions are inherently stochastic, determined by the rolls of the two dices.

The previous chapter presented the basic reinforcement learning techniques for temporal credit assignment, i.e., eligibility traces, and structural credit assignment, i.e., generalization by value function approximation. This chapter presents and motivates the three additional methods used in this thesis to increase the efficiency of reinforcement learning: 1) introduction of hierarchies structures in hierarchical reinforcement learning methods; 2) introduction of an additional and richer reward signal by potential-based shaping rewards, and 3) optimization of meta-parameters. The common characteristic of these methods is that they require the designer to provide domain knowledge in advance to agent, typically by a time consuming trial-error process. The evolutionary methods used for meta-learning in this thesis are presented in the next chapter.

3.2 Shaping Rewards

Shaping rewards (Dorigo and Colombetti, 1998) is a popular technique for improving the efficiency of reinforcement learning for delayed reward problems. Delayed reward problems are characterized by that the agent receives very sparse reward signals from the environment. Typically, the agent receives zero reward for all state transactions, except for the transition to the absorbing goal state, for which the agent receives a positive reward. Delayed reward problems are difficult for rein-

forcement learning in two regards: 1) finding initial solutions, because the agent has to rely on a more of less random search to find the rewarding states in the initial learning phase, and 2) convergence to an optimal solution, because the delayed rewards have to be propagated back along, potentially very long, sequences of state-action pairs. This is especially problematic in stochastic large-scale tasks, in which the likelihood for a successful random search is very low, and each path to a rewarding state has to be tested several times to get a reliable value estimate.

The idea of shaping rewards is to provide the agent with knowledge in the form of an additional more informative reward signal that guides the learning towards highly rewarded states. The concept of shaping in reinforcement learning is borrowed from the psychology literature, where the term was originally coined by the famous behavioral psychologist Burrhus Frederic Skinner (1938). In psychology, shaping is a training technique for guiding the learning of animals in complex tasks. The animal is rewarded for the completion of subgoals that progresses the behavior towards the overall goal. Shaping has been successfully used for learning animals complex behaviors such as pigeons pecking at a selected spot (Skinner, 1953, p. 93), and pigs eating at a table and vacuum cleaning a floor (Atkinson, Atkinson, Smith, Bem, and Nolen-Hoeksema, 1996, p. 242). One of the first successful real robot experiments in reinforcement learning was conducted by Mataric (1994, 1997). She used shaping rewards, called *progress estimators*, to learn a high-dimensional multi-robot foraging task. The shaping rewards were used to decompose the overall goal into subgoals.

A potential problem with shaping rewards is that, if they are not carefully designed, the agent can be trapped in a suboptimal behavior, i.e., the learning converges to a solution that is optimal in the presence of the shaping rewards, but suboptimal for the original problem. For example, Randløv and Alstrøm (1998) used shaping rewards to speed up the learning for a simulated bicycle agent. The task was to ride the bicycle to predefined location, and the agent was given an additional reward for each action moving the bicycle closer to target location. Initially, the resulting learned behavior was to ride in tiny circles close to the starting position, because there was no penalty for moving away from the goal. As suggested by the bicycle example, the shaping rewards must obey certain conditions to be theoretically justified, i.e., conditions for optimal policy invariance under reward function transformations. Ng, Harada, and Russell (1999) proved, under standard assumptions, that for potential-based shaping rewards (the shaping rewards depends only on the difference of a function of successive states), an optimal policy for a MDP augmented with shaping rewards will also be optimal for the original MDP. They define a potential function $\Phi(\cdot)$ over states, where the shaping reward for the transition from state s_t to s_{t+1}, $F(s_t, s_{t+1})$, is defined as the discounted change in potential:

$$F(s_t, s_{t+1}) = \gamma\Phi(s_{t+1}) - \Phi(s_t), \tag{3.1}$$

where γ is the discount factor of future rewards. The shaping reward is added to original reward function in each learning update, which, e.g., changes the updates

of the Q-values for Sarsa(0) to

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma(Q(s_{t+1}, a_{t+1}) + \Phi(s_{t+1})) - Q(s_t, a_t) - \Phi(s_t) \right]. \tag{3.2}$$

Wiewiora (2003) complimented the this work, by showing that using potential-based shaping rewards are equivalent to using the same potential function as a non-uniform initial action-value function, $Q(s, a) = \Phi(s)$. If the agent uses an advantage-based policy, which is defined as a policy that selects an action in a given state with a probability that is determined by differences in the Q-values, not their absolute magnitude, then the updates of the Q-values are equivalent in both cases. Given the fact that almost all reinforcement learning polices are advantage-based, e.g., greedy, ϵ-greedy, and softmax, this result is applicable for most reinforcement learning applications. The result by Ng, Harada, and Russell (1999) is important because it gives the necessary and sufficient conditions for shaping reward construction. However, it gives no guide to the designer how to construct the shaping rewards, i.e., how to assign the "correct" potential for each state in the state space, for a specific task and environment. From our evolutionary robotics perspective the potential-based shaping rewards formulation is attractive, because 1) it does not add any additional parameters to the system, and 2) it is only dependent on the states of the agent, which makes it easy to approximate the potential function for the shaping rewards with standard function approximation methods used in reinforcement learning. The parameters of the function approximation can be optimized in a straightforward manner by evolutionary computation methods.

In this thesis, the evolutionary optimization of potential-shaping rewards is a major factor in increasing the learning efficiency in our embodied evolution framework (see Paper III). Another interesting result is that proper shaping rewards guiding the learning can reduce the amount exploration required for efficient learning. In both Papers III and IV the evolutionary obtained action selection strategy is equal to the greedy policy. This result is consistent with the theoretical analyzes conducted by Laud and DeJong (2003). They analyzed shaping in terms of a reward horizon, which is a measure of the number of decisions the agent must take before experiencing accurate reward feedback. For example, if the agent only receives a delayed reward at the goal state, then the reward horizon is equal to the task length, and if the potential function of the shaping rewards is equal to the optimal value function, V^*, then the reward horizon is equal to 1.

3.3 Meta-Parameters

The performance of reinforcement learning algorithms depend critically on few *meta-parameters* that directly influence the updates in the learning algorithms or the exploration of the environment (Doya, 2002). The relevant meta-parameters in this thesis are the learning rate, α, the discount factor of future rewards, γ, the decay rate of the eligibility traces, λ, and the parameters controlling the exploration rate, ϵ or τ. In general, to find the optimal settings of the meta-parameters re-

quires an extensive and time-consuming search of the multi-dimensional parameter space. In addition to the difficulty to set the values of meta-parameters, it is often desirable to decay or vary some of the meta-parameters over the learning time. As described in the previous chapter (see section 2.6), it is common to decrease the exploration over time as the learning converges, but it is difficult to decide in advance which decaying scheme to use, and what the initial exploration rate should be. A standard assumption of convergence proofs in reinforcement learning is that the learning rate decays over time, but, once more, that does not specify which decaying scheme that is desirable for a specific problem. It has also been suggested that it may be preferable to vary λ according to the certainty of the value estimate of each state (Sutton and Singh, 1994).

In general, the setting of the meta-parameters is an overlooked issue in reinforcement learning studies. Normally, the authors simply state the values of the meta-parameters, but do not motivate the specific values used in their experiments. The evolutionary approach to meta-learning used in this thesis makes the optimization of meta-parameters straightforward (see Papers III and IV). Each meta-parameter, or each constant in the functional expression of the meta-parameter, is coded as a real-valued gene and optimized by the evolutionary search process. There are several earlier studies that use an evolutionary approach to optimize meta-parameters in reinforcement learning (Unemi, Nagaoyoshi, Hirayama, Nade, Yano, and Masujima, 1994; Augustsson, Wolff, and Nordin, 2002b; Eriksson, Capi, and Doya, 2003). The probably most studied meta-parameter is the learning rate, α. Sutton (1992a,b) proposed three algorithms for dynamic setting of the learning rate based on previous learning experience and statistical methods, such as Kalman filtering and least squares. Bowling and Veloso (2002) introduced the WoLF ("Win or Learn Fast") principle for varying the learning rate in multi-agent reinforcement learning. The basic idea is that an agent should adapt quickly (large α) if it performs worse than expected, and adapt more slowly (smaller α) if it performs better than expected.

3.4 Hierarchical Reinforcement Learning

Hierarchical reinforcement learning algorithms (Dayan and Hinton, 1993; Parr and Russell, 1997; Sutton, Precup, and Singh, 1999; Dietterich, 2000) are methods for introducing temporal abstractions and domain knowledge into the reinforcement learning framework. The idea is to capture hierarchical structures in complex MDPs, by breaking down the original problem into smaller set of suitable subproblems. This gives a task hierarchy of the problem where the actions of a subtask is other subtasks or primitive actions. There are three main approaches to defining the subtasks. The first approach, used in the option formalism by Sutton, Precup, and Singh (1999), defines subtasks (called options) in terms of a fixed policies provided by the designer. The second approach, used in the hierarchy of abstract machines (HAM) method by Parr and Russell (1997), defines subtasks in terms

of non-deterministic finite-state controllers. The designer provides partial policies for the subtasks that limits the set of actions that can be executed. The third approach, used in the MAXQ framework by Dietterich (2000), defines the subtasks in terms of termination predicates and local reward functions.

The learning performance for all hierarchical reinforcement learning methods depends on the provided task hierarchy. The hierarchy limits the actions that can be executed and thereby, also, limits the number of policies that can be considered. Reinforcement learning with options, HAM, and MAXQ are guaranteed, under standard assumptions, to converge to the best policy given the provided definition of the subtasks. However there is no guarantee that this policy is equal to the optimal policy for the original MDP.

3.4.1 Semi-Markov Decision Process

The mathematical foundation of hierarchical reinforcement learning is the theory of *semi-Markov Decision Processes* (semi-MDPs). A discrete-time semi-MDP is a generalization of the MDP, in which the actions are extended in time and can take a variable number of time steps to complete. For a semi-MDP the transition probability (see Equation 2.2) is extended to be a joint distribution of the result state, s', and the number of time steps, N, to complete action a in state s at time t:

$$P(s', N|s, a) = P(s_{t+N} = s', N|s_t = s, a_t = a). \tag{3.3}$$

Similarly, the expected reward (see Equation 2.3) can be defined as

$$R(s', N|s, a) = E\{r_{t+1}, \gamma r_{t+2}, \ldots, \gamma^{N-1} r_{t+N}|s_t = s, a_t = a, s_{t+N} = s'\}. \tag{3.4}$$

The modification of the Bellman equations for semi-MDPs are straightforward. The state-value function for a policy π, V^π, is defined as

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s', N} P(s', N|s, a) \left[R(s', N|s, a) + \gamma^N V^\pi(s')\right], \tag{3.5}$$

and the action-value function, Q^π, is defined as

$$Q^\pi(s, a) = \sum_{s', N} P(s', N|s, a) \left[R(s', N|s, a) + \gamma^N \sum_{a'} \pi(s', a') Q^\pi(s', a')\right]. \tag{3.6}$$

The only differences compared with the Bellman equations for MDPs, are that that expectation is computed with respect to both s' and N, and that γ is decayed exponentially by the number of time steps, N, it takes takes to complete action a.

3.4.2 Options

The option formalism (Sutton, Precup, and Singh, 1999) was developed to, by minimal extension of the reinforcement learning framework, include a general treatment

of temporally abstract knowledge and actions. Options are generalizations of primitive actions (normal one-step actions) to include actions that are extended in time.

Definition 3.1 *An option o is a three-tuple, $\langle \mathcal{I}, \pi, \beta \rangle$, defined as*

- *$\mathcal{I}$, the initiation set, is a subset of the state space, $\mathcal{S}$. An option o can only be selected in state s if and only if $s \in \mathcal{I}$.*
- *π is a predefined policy.*
- *β is a stochastic termination condition. If an option o is selected, then actions are selected according to the option's policy, π, until the option terminates according to β. In some cases it is useful to introduce a timeout to an option, in which case the option terminates after a fixed number of time steps, even if the termination condition is not satisfied.*

A primitive action, a, is a special case of an option that is available in all states that the primitive action is available ($\mathcal{I} = \{s \text{ such as } a \in \mathcal{A}(s)\}$), always terminates after one time step ($\beta(s) = 1$ for all states $s \in \mathcal{S}$), and selects the primitive action a with a probability of 1 ($\pi(s, a) = 1$, for all states $s \in \mathcal{I}$).

For example, consider an option called `capture-battery` for a mobile robot, which consists of a policy for moving to and, thereafter, docking with external batteries to be able to recharge the robot's internal battery. The option might only be available in states, $\mathcal{I}$, in which an external battery is visible. Then, the termination condition β could be defined to be 1 when no battery is visible, $s \notin \mathcal{I}$, and when the robot has successfully docked with a battery.

The semi-MDP version of Sarsa(0) updates, after termination, the Q-value of an option $o_t \in \mathcal{O}(s_t)$, executed in state s_t, and terminated in state s_{t+N}, as

$$Q(s_t, o_t) = Q(s_t, o_t) + \alpha \left[\sum_{i=1}^{N} \gamma^{i-1} r_{t+i} + \gamma^N Q(s_{t+N}, o_{t+N}) - Q(s_t, o_t) \right], \quad (3.7)$$

and the updates for the semi-MDP version of one-step Q-learning are defined as

$$Q(s_t, o_t) = Q(s_t, o_t) + \alpha \left[\sum_{i=1}^{N} \gamma^{i-1} r_{t+i} + \gamma^N \max_{o'} Q(s_{t+N}, o') - Q(s_t, o_t) \right]. \quad (3.8)$$

The learning in Paper I combines a simple version of options and the Watkins's Q(λ) algorithm (see Algorithm 2.1) to learn a mating behavior between two Cyber Rodents (see section 4.6). All options in the study are designed to force the agent to repeat a primitive action for several time steps, which makes the learning more stable and, thereby, the behavior more predictable for the mating partner.

3.4.3 The MAXQ Framework

The heart of the MAXQ method (Dietterich, 2000) is the MAXQ value function decomposition, which decomposes the value function for a MDP and a policy into a collection of value functions for suitable subtasks, and sub-subtasks, recursively. The decomposition gives a tree graph representation, called MAXQ graph, of the problem. The MAXQ graph, which has to be provided in advance by the designer, consists of two types of subtasks: 1) primitive subtasks/actions (the leaf nodes in the graph) that executes commands to the agent, and 2) composite subtasks (inner nodes in the graph) that selects other subtasks, primitive or composite, to solve their tasks. An important feature of the MAXQ framework is that it allows state abstraction within the subtasks.

Formally, the MAXQ decomposition takes a given MDP M and decomposes it into a finite set of subtasks $\{M_0, M_1, \ldots, M_n\}$, where M_0 is the root subtask.

Definition 3.2 *A subtask M_i is a three-tuple, $\langle T_i, A_i, \tilde{R}_i \rangle$, defined as*

- *T_i is a termination predicate that partitions $\mathcal{S}$ into a set of active states, S_i, and a set of terminal states, T_i. The subtask M_i can only be selected if and only if $s \in S_i$, and it terminates immediately if the MDP enter a state $s' \in T_i$, during its execution.*

- *A_i is the set of actions that can be selected to solve subtask M_i. These actions can either be primitive actions, $a \in \mathcal{A}$, for the MDP M, or composite subtasks, denoted by their index i. The MAXQ graph may not include any cycles, i.e., no subtask can invoke itself recursively, neither directly nor indirectly.*

- *$\tilde{R}_i(s')$ is the pseudo-reward function, which specifies a deterministic pseudo-reward for each transition to a terminal state $s' \in T_i$. The pseudo-reward function is only used for learning within the composite subtask, and tells how desirable each of the terminal states is for the subtask.*

Each primitive action, $a \in \mathcal{A}$, for the MDP M is a primitive subtask in the MAXQ decomposition such that a is always executable, it always terminates immediately after execution, and its pseudo-reward function is uniformly zero.

In the MAXQ method, each subtask M_i has its own policy, π_i. In the option terminology a subtask's policy is a deterministic option, i.e., $\beta(s) = 0$ if $s \in S_i$, and $\beta(s) = 1$ if $s \in T_i$. The set of policies for all subtasks is called a hierarchical policy, $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_n\}$. The natural objective in hierarchical reinforcement learning is to learn a *hierarchical optimal policy*, i.e., a policy that achieves the highest cumulative reward among all policies consistent with the given task hierarchy. For example, Parr and Russell (1997) showed that HAMQ learning converges to a hierarchical optimal policy, and Sutton, Precup, and Singh (1999) showed that semi-MDP learning with options converges to a hierarchical optimal value function. However, the objective of the MAXQ method is to learn a *recursively optimal policy*, which is a weaker form of optimality than hierarchical optimality. Recursively

optimality makes it possible to learn the policy of each subtask without reference to the policy of the parent subtask. This makes it easier to share and re-use subtasks, and it is also a key to successful state abstraction within the subtasks.

Formally, a recursively optimal policy for MDP M with MAXQ decomposition $\{M_0, M_1, \ldots, M_n\}$ is a hierarchical policy, $\boldsymbol{\pi}$, such that for each subtask M_i, the corresponding policy π_i is optimal for the semi-MDP defined by the set of states S_i, the set of actions A_i, the state transition probability function $P_i^{\boldsymbol{\pi}}(s', N|s, a)$, and the reward function given by the sum of the original reward function $R(s'|s, a)$ and the pseudo-reward function $\tilde{R}_i(s')$. The major drawback of seeking recursively optimality in the MAXQ method is that it requires careful design of the pseudo-reward function, $\tilde{R}$. In the worst case scenario, the design of the pseudo-reward can make the recursively optimal policy arbitrarily worse than the hierarchically optimal policy. A practical solution to this problem is to specify that goal terminal states are always preferable over non-goal terminal states, by assigning a pseudo-reward of 0 to goal terminal states and an uniform negative pseudo-reward to all non-goal terminal states.

The value functions learned by the MAXQ method is called the *projected value functions*. The projected value function of hierarchical policy $\boldsymbol{\pi}$ on subtask M_i, $V^{\boldsymbol{\pi}}(i, s)$, is the expected cumulative discounted reward of executing π_i (and the policies of all children of M_i) starting in state s until M_i terminates. The projected value function for subtask i in state s is recursively defined as

$$V^{\boldsymbol{\pi}}(i, s) = \begin{cases} Q^{\boldsymbol{\pi}}(i, s, \pi_i(s)) & \text{if } i \text{ is a composite subtask} \\ \sum_{s'} P(s'|s, i)R(s'|s, i) & \text{if } i \text{ is a primitive subtask,} \end{cases} \tag{3.9}$$

and $Q^{\boldsymbol{\pi}}(i, s, a)$ is recursively defined as

$$Q^{\boldsymbol{\pi}}(i, s, a) = V^{\boldsymbol{\pi}}(a, s) + C^{\boldsymbol{\pi}}(i, s, a). \tag{3.10}$$

The completion function, $C^{\boldsymbol{\pi}}(i, s, a)$, is the expected discounted cumulative reward of completing subtask M_i after invoking the subtask M_a in state s, defined as

$$C^{\boldsymbol{\pi}}(i, s, a) = \sum_{s', N} P_i^{\boldsymbol{\pi}}(s', N|s, a)\gamma^N Q^{\boldsymbol{\pi}}(i, s', \boldsymbol{\pi}(s')) \tag{3.11}$$

The Equations 3.9, 3.10, and 3.11 are called the *decomposition equations*, and tell how to decompose the projected value for the root, $V^{\boldsymbol{\pi}}(0, s)$, into projected value functions for the individual subtasks $\{M_1, M_2, \ldots, M_n\}$, and the individual completion functions $C^{\boldsymbol{\pi}}(j, s, a)$ for $j = 1, \ldots, n$. The projected value functions are stored explicitly as V values for all primitive actions and implicitly as C values for all composite subtasks. In general, the decomposition of the projected value function can be expressed as

$$V^{\boldsymbol{\pi}}(0, s) = V^{\boldsymbol{\pi}}(a_m, s) + C^{\boldsymbol{\pi}}(a_{m-1}, s, a_m) + \cdots + C^{\boldsymbol{\pi}}(a_1, s, a_2) + C^{\boldsymbol{\pi}}(0, s, a_1), \tag{3.12}$$

where $a_0, a_1, \ldots, a_m$ is the path of subtasks in the MAXQ graph, from the root node to a leaf node, selected by the hierarchical policy $\boldsymbol{\pi}$.

The learning algorithm in the MAXQ framework is called MAXQ-Q. To be able to learn recursively optimal policies MAXQ-Q uses two completion functions, C and $\tilde{C}$. C is the completion discussed so far and it is used by the parent task to compute $V(i,s)$, the expected cumulative reward for performing action i starting in state s. The second completion function, $\tilde{C}$, is only used inside subtask i to discover the local optimal policy for M_i. $\tilde{C}$ uses both the original reward function for the MDP, $R(s'|s,a)$, and the pseudo-reward function, $\tilde{R}_i(s')$. The MAXQ-Q algorithm uses batch updating of the completion functions. If subtask i invokes action a which terminates after N time steps in state s', then $\tilde{C}$ and C are updated for the sequence of visited state while invoking a, $\{s_n\}$ for $n = 0, \ldots, N-1$, as

$$\begin{aligned}\tilde{C}_{t+1}(i,s_n,a) \quad \leftarrow \quad & \tilde{C}_t(i,s_n,a) + \alpha_t(i)\big[\gamma^{n+1}\big(\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s_n)\big)\\ & - \tilde{C}_t(i,s_n,a)\big],\end{aligned} \tag{3.13}$$

$$\begin{aligned}C_{t+1}(i,s_n,a) \quad \leftarrow \quad & C_t(i,s_n,a) + \alpha_t(i)\big[\gamma^{n+1}\big(C_t(i,s',a^*) + V_t(a^*,s')\big)\\ & - C_t(i,s_n,a)\big],\end{aligned} \tag{3.14}$$

where $a^* = \arg\max_{a'}(\tilde{C}_t(i,s',a') + V_t(a',s'))$ is the "recursively" greedy action.

The update of the projected value function for the primitive subtasks are very simple, i.e., Sarsa or Q-learning with $\gamma = 0$. If the primitive subtask i is executed in state s, receiving reward r, then $V(i,s)$ is updated as

$$V_{t+1}(i,s) \leftarrow V_t(i,s) + \alpha_t(i)[r - V_t(i,s)] \tag{3.15}$$

Algorithm 3.1 shows the pseudo-code for the recursive MAXQ-Q algorithm, where the projected value function for state s and subtask i is computed recursively according to Equation 3.9. After convergence the recursively optimal policy is computed for each subtask i and state s, as the action a that maximizes $\tilde{C}_t(i,s,a) + V_t(a,s)$.

3.4.4 MAXQ Example

To illustrate the MAXQ method consider the simple taxi problem suggested by Dietterich in his original paper (Dietterich, 2000). Figure 3.1 shows the taxi domain, a 5×5 grid-world with four taxi stands marked with the letters R,G,B, and Y. In each episode the taxi is placed at a random location. The task for the taxi agent is to pickup a passenger at one randomly selected taxi stand (the "source") and then put down the passenger at another randomly selected taxi stand (the "destination"), which ends the episode. There are six primitive subtasks: four navigation actions that moves the taxi one square in the directions North, South, East, or West, a Pickup action, and a Putdown action. There is a negative reward for each action of -1. Additionally, there is a reward of $+20$ for putting down the passenger at the "destination", and a reward of -10 for illegal execution of Pickup or Putdown. If

Algorithm 3.1: The MAXQ-Q learning algorithm.

```
Main Program
  Initialize V, C, and C̃ arbitrarily
  MAXQ-Q(root node 0, initial state s_0)

Function MAXQ-Q(subtask i, state s)
  seq ← ()                                  /* sequence of visited states */
  if i is a primitive subtask then
      Execute i, receive r, and observe next state s'
      V_{t+1}(i,s) ← V_t(i,s) + α_t(i)[r − V_t(i,s)]
      Push s onto the beginning of seq
  else
      count ← 0
      while T_i(s) is false do
          Select action a according to the subtask's policy π_i
          childSeq ← MAXQ-Q(a, s)
          Observe result state s'
          a* ← arg max_{a'}[C̃_t(i,s',a') + V_t(a',s')]
          N ← 1
          foreach s ∈ childSeq do
              C̃_{t+1}(i,s,a) ← C̃_t(i,s,a) + α_t(i)[γ^N(R̃_i(s') + C̃_t(i,s',a*)
                                  + V_t(a*,s)) − C̃_t(i,s,a)]
              C_{t+1}(i,s,a) ← C_t(i,s,a) + α_t(i)[γ^N(C_t(i,s',a*) + V_t(a*,s'))
                                  − C_t(i,s,a)]
              N ← N + 1
          Append childSeq onto the front of seq
          s ← s'
  return seq
```

the taxi is moved to hit a wall, then the taxi remains at the previous location, but there is no extra punishment.

Dietterich defines four composite subtasks for solving the overall task:

- Navigate(t) to move the taxi from its current location to one of the four taxi stands, indicated by the parameter t.
- Get to move the taxi from its current location to the "source" and pick up the passenger.
- Put to move the taxi from its current location to the "destination" and put down the passenger.
- Root is the overall taxi task.

The suggested MAXQ graph for the taxi problem is shown in Figure 3.2. The graph contains two types of nodes. The triangular nodes, called Max nodes, represent the

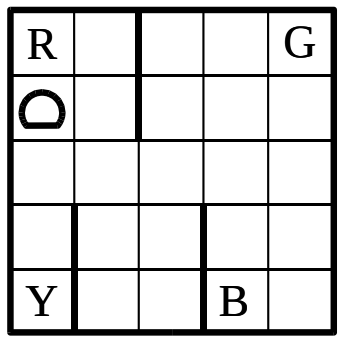

Figure 3.1. The taxi domain: a 5×5 grid-world with four taxi stands, marked with letters R,G,B, and Y in the figure.

subtasks. Each Max node representing a primitive subtask, i, stores the values of $V^{\boldsymbol{\pi}}(i, s)$, for all states $s \in S_i$. The rounded rectangular nodes, called Q nodes, represent the actions of the subtasks. Each Q node representing parent subtask i and child subtask a stores the values of $C^{\boldsymbol{\pi}}(i, s, a)$, for all states $s \in S_i$.

Suppose that in the current state, s_1, the taxi is located as shown in Figure 3.1, and the passenger is located at taxi stand R and wants to go to taxi stand B. If the agent follows a hierarchical policy that is equal to the optimal policy, π^*, then the value of state s_1 is 10: a reward of -1 to move north to taxi stand R, a reward of -1 to pick up the passenger, a accumulated reward of -7 to move to taxi stand B, a reward of -1 to put down the passenger, and, finally, a reward of $+20$ for completing the task. Figure 3.2 shows how the MAXQ method computes this value as the projected value function for the Root node, $V^{\pi^*}(\mathsf{Root}, s_1)$ (see Equation 3.12) as

$$\begin{aligned} V^{\pi^*}(\mathsf{Root}, s_1) &= V^{\pi^*}(\mathsf{North}, s_1) + C^{\pi^*}(\mathsf{Navigate}(R), s_1, \mathsf{North}) + \\ &\quad C^{\pi^*}(\mathsf{Get}, s_1, \mathsf{Navigate}(R)) + C^{\pi^*}(\mathsf{Root}, s_1, \mathsf{Get}) \\ &= (-1) + 0 + (-1) + 12 \\ &= 10 \end{aligned} \tag{3.16}$$

The taxi problem highlights both the strengths and weaknesses of the MAXQ method. Obviously, the MAXQ method supports temporal abstraction in the form of actions that are extended in time. The MAXQ method also support state abstraction within subtasks, by the definition of S_i. Dietterich introduces five relatively complicated conditions under which state abstractions are theoretically justified. For the MAXQ graph of the taxi problem, the size of the state space can thereby be reduced from 3000 distinct values (state-action pairs) for flat Q-learning to 632 distinct values. Finally, the MAXQ method supports sharing of subtasks. For example, in the MAXQ graph for the taxi problem, both Get and Put use $\mathsf{Navigate}$

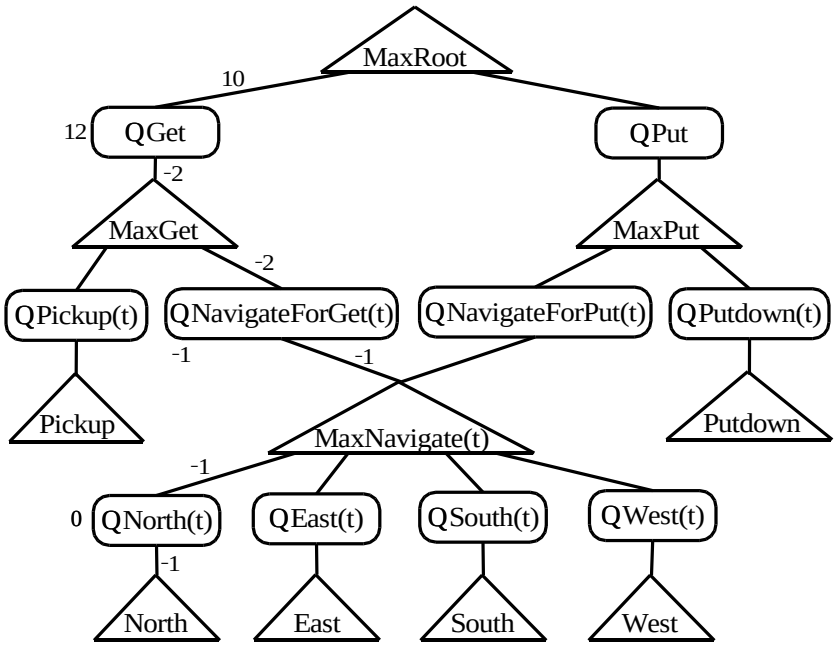

Figure 3.2. Suggested MAXQ graph of the taxi problem. The values in the figure show the computation of $V^{\pi^*}(\mathsf{Root}, s_1)$, when the taxi is located as shown in Figure 3.1, and the passenger is located at taxi stand R and wants to go to taxi stand B. For the path of subtasks given by π^*, the C values are shown next to the Q nodes, and the V values are shown next to the arcs.

to move the taxi to the specified location. The drawback of the MAXQ method is that the designer not only have to provide the complete decomposition of the overall task, the MAXQ graph, but also the state abstractions within the subtasks and the meta-parameters for the subtasks. For example, in the reported experiments for the taxi problem, Dietterich used different values of the temperature τ for softmax action selection for all four composite subtasks, and the values varied for different settings of the MAXQ-Q algorithm (e.g., with or without state abstraction). The learning rate, α, was also varied depending on the algorithm setting: 0.5, without state abstraction, and 0.25 with state abstraction. These issues are duly noted in the introduction of Dietterich's paper, and he suggested that MAXQ framework should be viewed as a computer program, where the designer has to fill in the "implementation" of each of the subtasks and how the subtasks invoke each other. Dietterich also expressed the hope that subsequent research would be able to automate most of the work that is required of the designer to do. An important part of this thesis (see Paper V) is to realize a part of this goal, by automation of the construction of the task hierarchy in the MAXQ method by genetic programming

techniques (see section 4.2).

Chapter 4

Artificial Evolution

Artificial evolutionary methods are global and model-free search techniques inspired by biological evolution. This chapter introduces four evolutionary computation methods: 1) standard genetic algorithms; 2) genetic programming, i.e., evolution of computer programs; 3) evolutionary robotics, i.e., automatic creation of control systems of autonomous robots; and 4) embodied evolution, i.e., applying evolutionary robotics to a colony of autonomous robots. This chapter also presents and motivates the specifics of the evolutionary approaches to meta-learning used in this thesis.

4.1 Genetic Algorithms

Genetic algorithms (Holland, 1975; Goldberg, 1989) is a computational methodology for optimization that is inspired by the Darwinian principle of selective reproduction of the fittest (Darwin, 1859). In a genetic algorithm the potential solutions are represented by a population of competing individuals. Each individual is represented by an artificial chromosome, the *genotype* (or genome), which encodes the characteristics of the individual, the *phenotype*. In the simplest case, the genotype is represented by a fixed length bit-string that, e.g., encodes the value(s) of the variable(s) of a mathematical function. Initially all genotypes are randomly created. The *fitness*, f_i, of the phenotype, x_i, of each individual, i, is evaluated by a fitness function, Φ: $f_i = \Phi(x_i)$, where higher fitness values are better. The fitness function is the designer's tool for defining the goal of the genetic algorithm. Unfortunately, there are no general principles for fitness function design. Often a suitable fitness function for a particular task has to be searched for by a potentially time consuming trial-and-error process. This is a major difference between artificial evolution and biological evolution. In nature, there is no explicit fitness function. Instead, the reproductive success of the genes determine the frequencies of the genes in the gene pool of the individuals in the population.

After evaluation, a new generation of individuals are created by applying selective reproduction and genetic operations. Selective reproduction means that individuals with higher fitness values have higher probability of reproduce their genotypes to the next generation. A standard implementation of selective reproduction is *roulette wheel selection*. The selection probability, $P(i|f_i)$, of an individual, i, is proportional to its fitness value, f_i, normalized by the total fitness of the population of N individuals:

$$P(i|f_i) = \frac{f_i}{\sum_{j=1}^{N} f_j}, \ f_i \geq 0. \tag{4.1}$$

Another widely used selection method is *tournament selection*. In tournament selection with a tournament size of k, k individuals are chosen randomly with uniform probability and the individual with the highest fitness value, among the randomly chosen, is selected. After the reproductive selection of N individuals (assuming a

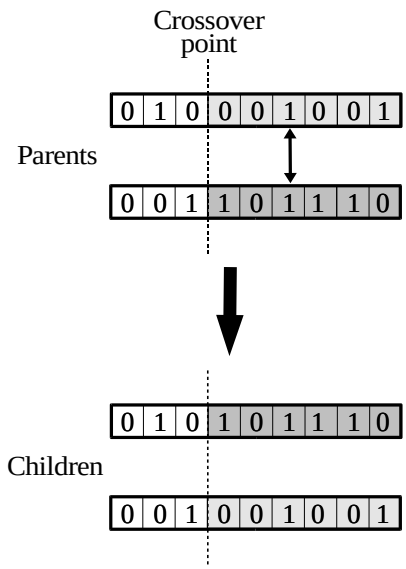

Figure 4.1. One-point crossover between two bit-string genotypes.

constant population size of N), the new population is randomly paired and crossover is applied to each pair according to a given probability. The simplest version of crossover, called *one-point crossover* (shown in Figure 4.1), switches substrings between two parent chromosomes, around a randomly selected crossover point. After crossover, mutation is applied to each gene of the chromosomes in the new population, with a given mutation probability. In the case of binary representation,

mutation normally means that the binary value of a gene is flipped, from 0 to 1, or 1 to 0.

This process of evaluation, reproductive selection, crossover and mutation is continued until a predefined termination condition is fulfilled, e.g., that a desired fitness value has been reached by the best individual, or that a maximum number of generations has been produced. The general scheme of a genetic algorithm is shown in Algorithm 4.1

Algorithm 4.1: General genetic algorithm scheme

```
Initialize the population, P, of N chromosomes of length n randomly
for i ← 1 to N do
    x_i ← Evaluate (P_i)
    f_i ← Φ(x_i)
repeat
    for i ← 1 to N do
        P_i ← ReproductiveSelection (P,f)
    P ← random permutation of P
    for i ← 1 to N/2 do
        r ← random number [0, 1]
        if r < crossover probability then
            [P_i, P_{i+N/2}] ← Crossover (P_i, P_{i+n/2})
    for i ← 1 to N do
        for j ← 1 to n do
            r ← random number [0, 1]
            if r < mutation probability then
                P_i(j) ← Mutate (P_i(j))
    for i ← 1 to N do
        x_i ← Evaluate (P_i)
        f_i ← Φ(x_i)
until termination condition is fulfilled
```

4.2 Genetic Programming

Genetic programming (Koza, 1992, 1994) is relatively new evolutionary methodology for evolving computer programs. Instead of evolving a solution to a problem, genetic programming evolves a program to solve the problem. The overall evolutionary scheme is the same as for standard genetic algorithms (see Algorithm 4.1), but the representation of the genotypes and the implementation of the genetic operations for mutation and crossover are different. Genetic programming operates on a tree representation, normally implemented as Lisp-like expressions. The genetic programming tree representation consists of two types of nodes: 1) the function set, the inner nodes in the tree representation, typically consists of mathematical

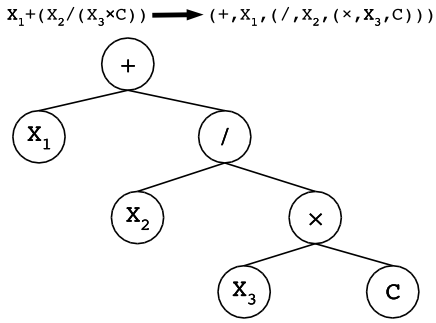

Figure 4.2. Tree representation of computer programs in genetic programming.

expressions, or if-else statements, and 2) the terminal set, the leaf nodes in the tree representation, consists of variables and constants. Figure 4.2 shows the parse tree representation of the expression $X_1 + (X_2/(X_3 \times C))$, which is read from left to right as $(+, X_1, (/, X_2, (\times, X3, C)))$.

The crossover operation between two parent tree representations is performed by switching subtrees at randomly selected crossover nodes, which is shown in Figure 4.3. Mutation is accomplished by either deleting a randomly selected node or creating new random subtrees.

In this thesis genetic programming is applied in an unconventional manner (see Paper V). Instead of evolving computer programs, we apply the genetic programming crossover operator to construct task hierarchies, MAXQ graphs, in the MAXQ framework (see section 3.4.3). We have also used a form of strongly typed genetic programming (Montana, 1995). Strongly typed genetic programming allows the designer to assign a type to the arguments and the return value of each function, and, thereby, limits the number of tree representations that can be constructed. In our case this means that we restrict which subtasks that can be parent nodes to other subtasks, based on the design of state spaces and termination conditions. Related work that combines reinforcement learning and genetic programming techniques has been performed by Iba (1998), Downing (2001), and Kamio and Iba (2005).

4.3 Learning and Evolution

Learning and Evolution are two types of biological adaptations that occur on different time-scale and operate on different parts of an organism. Learning is lifetime adaptations occurring within a single agent. Learning operates only on the phenotype. The resulting modifications cannot directly affect the genotype and are, therefore, not inherited by the offspring. Evolution operates on a macro-scale over

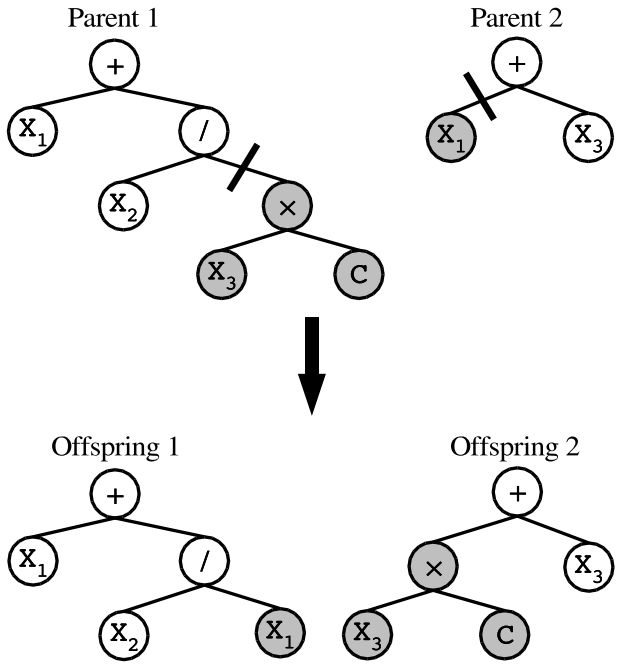

Figure 4.3. Crossover operation in genetic programming, by switching subtrees between the parents.

generations, by slowly changing the frequencies of the genes in the gene pool of the individuals in the population. Evolution can only capture slow changes in the environment, while learning makes it possible to adapt to faster environmental changes that occurs during the agent's life. In short, evolution determines the innate abilities of an agent, and learning represents the abilities acquired during an agent's lifetime.

In addition to giving the agent abilities to adapt to changes that occurs during the agent's lifetime, learning can also help and guide the evolutionary process. In 1896, Baldwin (Baldwin, 1896) proposed that behaviors that were learned during an organism's lifetime could accelerate the rate of evolution, by indirect assimilation of learned traits. The reasoning behind Baldwin's theory, called the *Baldwin effect*, is that an animal discovers and learns a new useful skill. If this new behavior is imitated by the other individuals in the population, then the selection pressure is changed. Individuals who learn the new skill reproduce more offspring. However, as learning has a cost, i.e., the behavior will be suboptimal during the learning phase, evolution will tend to select individuals who have have innate abilities that otherwise would have to be learned. The Baldwin effect has been supported by

scientific evidence (Waddington, 1942) and also been proved to real in computer simulations of artificial evolution (Hinton and Nowlan, 1987; Turney, Whitley, and Anderson, 1996). Mayley (1996) showed two conditions for genetic assimilation by the Baldwin effect: that the search space for the learning is correlated with evolutionary search space and that the evolutionary cost of learning is not too high.

Our evolutionary approach to meta-learning in reinforcement learning, can be considered as a Baldwinian evolution process. The Baldwin effect is a form of knowledge transfer. Features that have to be learned in early generations become innate abilities in later generations. In our case this means the innate learning abilities of the agents are increased over evolutionary time as the meta-parameters, shaping rewards, or tasks hierarchies are being optimized

Superficially, the Baldwin effect sounds like the biologically disproved theory of Lamarckian evolution (Lamarck, 1809). Lamarck's theory of evolution was proposed 50 years before Darwin's and suggested that acquired characteristics by learning were imprinted back into the genotype and inherited to the offspring. Although Lamarckian mechanisms are not to be found in nature, they can be more effective than Darwinian evolution in artificial evolution. In general, Ackley and Littman (1991) showed that Lamarckian evolution is more effective in a stationary environment, while Sasaki and Tokoro (1997) showed that Darwinian evolution outperforms Lamarckian evolution in a non-stationary environment or when different individuals encounter different learning experiences. In Paper V Lamarckian mechanisms are explored, as a method for accelerate the learning of the policies of the subtasks in the MAXQ method. Because the MAXQ method seeks recursively optimal policies, it is possible to continue the learning over generation for the subtrees in the MAXQ graphs not affected by the genetic programming crossover operations. This minimizes the need for re-learning the policies of the subtasks, and, thereby, increases the learning performance compared with Darwinian evolution.

4.4 Evolutionary Robotics

Evolutionary robotics (Husbands, Harvey, and Cliff, 1993; Nolfi and Floreano, 2000; Walker, Garrett, and Wilson, 2003) is a methodology for automatic creation of control systems of autonomous robots, using evolutionary computation techniques. The main attraction of evolutionary robotics and other artificial intelligence approaches is that the design of control systems for autonomous robots are difficult, even for simple problems. This is clearly shown by fact that there exists almost no autonomous robots assisting humans in their daily life. The main reason why robotic control systems are difficult to design is that the robot's behavior is an emergent property of the dynamic interaction between the robot and its environment, which makes it difficult to predict the behavior in advance for a designer.

An important difference between evolutionary robotics and the "normal" engineering approach to evolutionary computation, is the objective of the evolution,

i.e., the design of the fitness function. From an engineering perspective the objective of the evolution is functional, i.e., to optimize a number of parameters for a well defined control problem in a predictable environment with known properties. In evolutionary robotics the objective is behavioral, i.e., to evolve the behavior of autonomous robots in unpredictable and partially known environments. Nolfi and Floreano (2000) define a 3D space for the classification of fitness functions for different evolutionary objectives:

- The dimension *functional-behavioral* indicates whether the fitness function considers the behavioral outcome or the specific function of the controller.
- The dimension *explicit-implicit* specifies the number of components, variables and constraints in the fitness function. An explicit fitness function has many components and an implicit fitness function has few components.
- The dimension *external-internal* indicates whether the agent relies on global or local state information. An external fitness function includes global state information that cannot be accessed by an autonomous robot. A "pure" internal fitness function includes only local state information that are available through the robot's sensors.

In general, fitness functions for engineering purposes are located in the functional-explicit-external part of the 3D space, and require human expert knowledge. In contrast, fitness functions for evolutionary robotics are, ideally, located in the behavioral-implicit-internal part of the space, and require little domain knowledge.

Evolutionary robotics and reinforcement learning are applicable to the same types of problems, but solve them in different manners. Both methodologies search the space of policies, and often use the same type of representation of the policies, in the form of a neural network. The basic difference is that evolutionary methods change the policies of all individuals after the final outcome of the task, while reinforcement learning methods change the policy of an agent continuously during the execution of the task. This is also the basic difference between learning and evolution in nature. Learning is lifetime adaptations of the phenotype, which do not, directly, affect the genotype, and evolution is adaptations of the genotypes over generations of individuals in the population.

Another difference between reinforcement learning and evolutionary robotics is that evolutionary methods do not specify a specific type of controller. In principle, any control system that posses *evolvability*, i.e., the ability of random variations to sometimes produce improvements (Wagner and Altenberg, 1996), can be used in evolutionary robotics. The three main approaches (Nolfi and Floreano, 2000) in evolutionary robotics are to: 1) evolve neural network controllers; 2) optimize parameters of predefined controllers, and 3) evolve the computer programs themselves using genetic programming. Of these three, the most common approach is to evolve some type of neural controller. The evolution of neural controllers is well illustrated by a series of experiments by Floreano and Mondada (Mondada and

Floreano, 1995; Floreano and Mondada, 1996, 1998). Their task was to evolve a neural controller for navigation in a small looping maze. The goal of the robot was to move as fast as possible forward without colliding with the walls, and the fitness function was designed to encourage straight motion and obstacle avoidance, defined as

$$\begin{aligned} \Phi &= V\left(1-\sqrt{\Delta v}\right)(1-i) \\ & 0 \leq V \leq 1 \\ & 0 \leq \Delta v \leq 1 \\ & 0 \leq i \leq 1, \end{aligned} \tag{4.2}$$

where V is sum of wheel speeds of the two wheels, Δv is absolute value of the difference in velocity between the two wheels, and i is the normalized activity of the infrared distance sensor with the highest activity. The fitness of an individual was evaluated for each sensory-motor loop and the total fitness of an individual was computed as the sum of the fitness values for all evaluations, divided by the number of evaluations.

In the first experiment (Mondada and Floreano, 1995), they used the small round Khepera robot, and the genotype consisted of the real-valued weights of a recurrent Elman neural network (Elman, 1990) controller. The network had eight input units, corresponding to the Khepera's eight infrared proximity sensors, and two sigmoid output units, corresponding to the wheel velocities of the two wheels. After 50 generation, the best individuals, in a population of 100 individuals, had obtained a smooth navigation behavior, where the robot did not collide with the walls. In the second experiment (Floreano and Mondada, 1996), they tested re-adaptation of the neural controllers to a new morphology and sensorimotor system. The neural controllers was first evolved for 106 generations on the Khepera for the navigation task described above. The controllers were then moved to the much larger Koala robot and evolved for 44 more generations (totally 150 generations) in a scaled-up looping maze. In addition to the difference in size, the Koala robot has a different shape and proximity sensor layout compared with the Khepera robot. After an initial drop in fitness, it took about 30 generations of evolutionary re-adaptation to reach similar performance level as the individuals in the last generation on the Khepera robot. In the third experiment (Floreano and Mondada, 1996), they used the Khepera robot and the looping maze task to investigate the evolution of learning mechanisms. Instead of evolving the weights of neural controller directly, they evolved the learning rules and learning rates of the synaptic weights of controller. Each weight of the neural network was changed continuously during the individual's lifetime according to one of four genetically determined Hebbian learning rules (Hebb, 1949):

$$w_t = w_{t-1} + \eta \Delta w_t, \tag{4.3}$$

where η was the genetically determined learning rate of the synapse, and the change of the weight, Δw_t, was computed as a function of the activations of the presynaptic

and postsynaptic units, according to the specific Hebbian learning rule. The resulting behaviors and obtained fitness levels were similar to the results in previous experiment without learning, described above. The initial learning phase was very short. The best obtained individuals needed only approximately 10 times steps, i.e., sensory-motor loops, of initial adaptations to be able move forward without colliding with the walls.

A good example of the second evolutionary robotics approach of optimizing parameters of a predefined controller is the experiment conducted by Hornby, Fujita, Takamura, Yamamoto, and Hanagata (1999). They evolved two types of dynamic locomotion gaits for the quadruped AIBO robot. The predefined locomotion controller had 20 real-valued parameters, specifying the position and orientation of the body, the swing trajectory and speed of the legs, the oscillation of the body location and orientation, and three additional gain parameters. Artificial evolution was used to optimize the values of the parameters on the physical robot, using a fitness function that promoted fast straight-forward motion. The fitness of an individual was computed as a product of the average velocity and the straightness of the locomotion. The experiment had mixed results. One of the evolved gaits was not truly dynamic, dragging one of the front legs along the ground, and could, therefore, not achieve high speeds. On the other hand, the result for the other locomotion gait was impressive. The obtained gait of the best individual was almost twice as fast as the best handcoded controller.

There are few studies using genetic programming to evolve control systems on physical robots. An exception is the work by Nordin and Banzhaf (1997). They applied a specialized variant of genetic programming to evolve an obstacle avoidance behavior on a Khepera robot, using a similar fitness function and task setting as in the study by Mondada and Floreano (1995). Their genetic programming method operates directly on the machine code of the onboard computer of the robot, and they developed a special crossover operator that can be applied on linear chromosomes of variable length consisting of machine instructions. The used a steady-state genetic programming population, where only a few individuals in the population were tested in each evaluation loop. The results of the experiment are impressive. It took only 40-60 minutes in real time to evolve a good program for obstacle avoidance. Augustsson, Wolff, and Nordin (2002a) applied the same type of genetic programming to evolve a feasible flying (flapping) behavior on a physical robot.

4.5 Embodied Evolution

Watson, Ficici, and Pollack (2002) introduced the embodied evolution methodology for evolutionary robotics. Embodied evolution was inspired for the case where a large number of robots freely interact with each other while performing some task in a shared environment. The robots produce offspring by mating, and, naturally, the probability for a robot to produce offspring is regulated by the robot's per-

formance of the task. Embodied evolution mimics the distributed, asynchronous and autonomous properties of biological evolution. The main difference between embodied evolution and other evolutionary robotics approaches is that reproduction is integrated with other autonomous behaviors. The evaluation, selection and reproduction are carried out by cooperation and competition of the robots, without any need for human intervention. There is no centralized mechanism performing selection and applying genetic operation objectively from a "God's" point of view. Watson, Ficici, and Pollack (2002) validated their approach by successfully apply embodied evolution to a simple phototaxis task. They used 8 small mobile robots to evolve a very simple neural network controller. The controller had two input nodes. One binary input, indicating which of the two light sensors receiving more light, and one bias node. The input nodes were fully-connected to two output motor neurons, controlling the speed of the wheels, giving totally four integer weights. In their experiments mating was not a directed behavior. Instead, an agent broadcasted its genotype according to a predefined scheme and the other robots within communication range could then pick up the genotype. Differential selection was achieved by that more successful agents broadcasted their genotypes more often and were less inclined to accept genotypes broadcasted by other robots. A similar approach was used by Nehmzow (2002). He used embodied evolution to evolve three different sensory-motor behaviors, using two small mobile robots. In the experiments, the two robots first evaluated their current behavioral strategies, and after a fixed amount of time the robots initiated a robot seeking behavior. The robots then performed an exchange of genetic material via IR-communication, and genetic operations were applied according to the fitness values. Each robot stored two genotype strings: the currently active string and the best solution so far. If the genetic algorithm did not produce an improved result, then the best solution was used in the next generation.

Our general objective is to investigate the adaptive mechanisms of artificial agents under the same fundamental constraints as biological agents, namely self-preservation and self-reproduction (see the description of Cyber Rodent project in the next section 4.6). This gives some specific requirements on the embodied evolution:

- Reproduction should be truly integrated as a directed autonomous behavior. The individuals have to find mating partners and physically exchange genotypes with the mating partners.
- Maintaining the internal energy levels of the individuals is a natural constraint of the embodied evolution. Battery power is in general considered a limitation of evolutionary robotics (Mataric and Cliff, 1996), because the robots have to interrupt their activity for a considerable amount of time to recharge their batteries. In evolutionary robotics studies using the Khepera robot the battery supply issue is often solved by connecting the Khepera to an external battery source through a thin cable and rotating contacts. Another popular solution used in the embodied evolution study by Watson, Ficici, and Pollack

(2002) is to use an electrified floor that provides power to the robots. From our biological perspective, the recharging of the individuals internal batteries is a biological constraint that is necessary for survival. The individuals have to find and recharge from external battery sources to be able to stay alive.

- The individuals should not be evaluated for their performance on some arbitrary task. Instead the individuals should be evaluated for their survival ability, i.e., their ability to forage food, in the form of external batteries, and to perform mating with selected mating partners. Ideally, there should be no explicit fitness function promoting a certain behavior. Instead, like in nature, the performance of an agent should be determined by its reproductive ability.

Another difference between our approach and standard evolutionary robotics and the embodied evolution studies described above is the purpose of the evolution. The role of the evolution, in this thesis, is not to evolve correct low-level mappings between sensory input and motor output. Instead the genotype encodes the meta-properties of learning by reinforcement learning, such as meta-parameters, potential-based shaping rewards, and switching of primitive behaviors. The role of the evolution is to accomplish meta-learning by optimizing these meta-properties. Our approach is clearly related to the evolution of learning rules in neural controllers (Floreano and Mondada, 1996; Urzelai and Floreano, 2000). An obvious difference between these approaches is algorithmical. In the evolution of learning rules, the weights of the neural controller is modified by Hebbian learning and the learning has no explicit goal of its own. In our approach the learning is accomplished by TD-learning and each learning behavior has its own goal defined by a reward function. The similarity of these two approaches was demonstrated in the work by Niv, Joel, Meilijson, and Ruppin (2002) and Ruppin (2002), where they showed that optimal learning rules for a reinforcement learning agent could be evolved in a general Hebbian learning framework. Unfortunately, it seems very difficult to scale up their method to more complex robotic applications.

An important issue that has be considered in embodied evolution for more complex robotic tasks is the limited number of available physical robots. In the original embodied evolution framework each physical robot equals one individual in the population. Although this may be the ideal case, it makes the methodology inapplicable for more complex evolutionary robotics tasks, because of the large number of robots required for an appropriate population size. To overcome this limitation, we have embedded a subpopulation of virtual agents inside each robot. The virtual agents are evaluated in the survival task by time-sharing, i.e., taking control over the robot for a limited period of time.

4.6 The Cyber Rodent Robot

The research presented in this thesis has been conducted within the Cyber Rodent project (Doya and Uchibe, 2005). The goal of the Cyber Rodent project is to

understand the origins of our reward and affective systems by building artificial agents that share the same intrinsic constraints as natural agents: self-preservation and self-reproduction. The robotic platform, the Cyber Rodent, was developed with the embodied evolution objectives outlined in the previous section 4.5 in mind. The Cyber Rodent (see Figure 4.4) is a rat-like mobile robot, 22 cm in length and

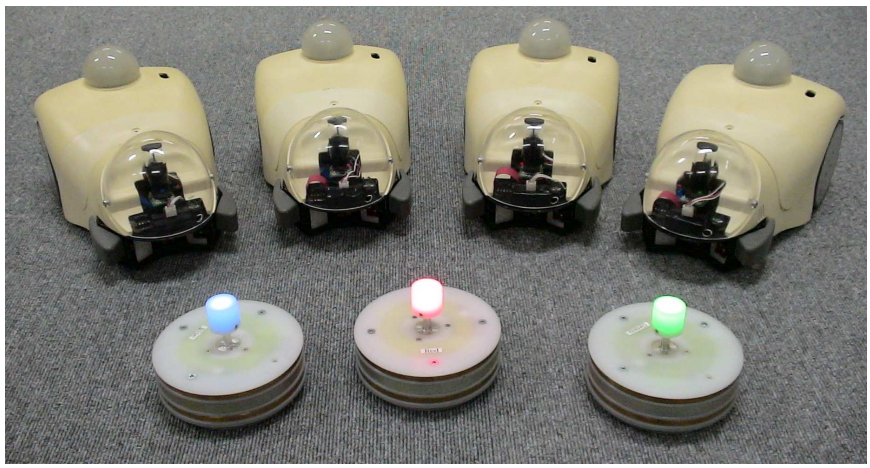

Figure 4.4. Four Cyber rodents and three battery packs with different colored (Red, Green, and Blue) LEDs mounted on top.

1.75 kg in weight. The robot has a variety of sensors, including a wide-angle C-MOS camera, an infrared range sensor, seven infrared proximity sensors, gyros, and accelerometers (see Figure 4.5). It has two wheels and a maximum speed of 1.3 ms^{-1}, and a magnetic jaw that latches onto battery packs. It also has a speaker, two microphones, a three-color LED for audio-visual communication, and an infrared communication port. It is further equipped with USB and wireless communication ports for connection with a host computer.

The "brain" of the Cyber Rodent CR is a Hitachi SuperH-4 CPU chip, which allows fully on-board, real-time learning and control. It also has a FPGA chip for real-time visual processing, such as color blob detection. The programs for the Cyber Rodent are coded in C on top of the real-time operation system eCos. The hardware was manufactured by Robos (Nagoya, Japan) and its basic software was developed by R-Lab (Tokyo, Japan). The two main features of the Cyber Rodent robot is the ability to recharge from external battery packs in the environment, for self-preservation, and the ability to exchange genetic information by infrared communication, for self-reproduction. To our knowledge the Cyber Rodents are the first autonomous robots that realize survival by recharging and evolution by local communication. The project currently has four Cyber Rodents. Most experiments in this thesis have been performed in a Matlab simulator, mimicking the real Cyber

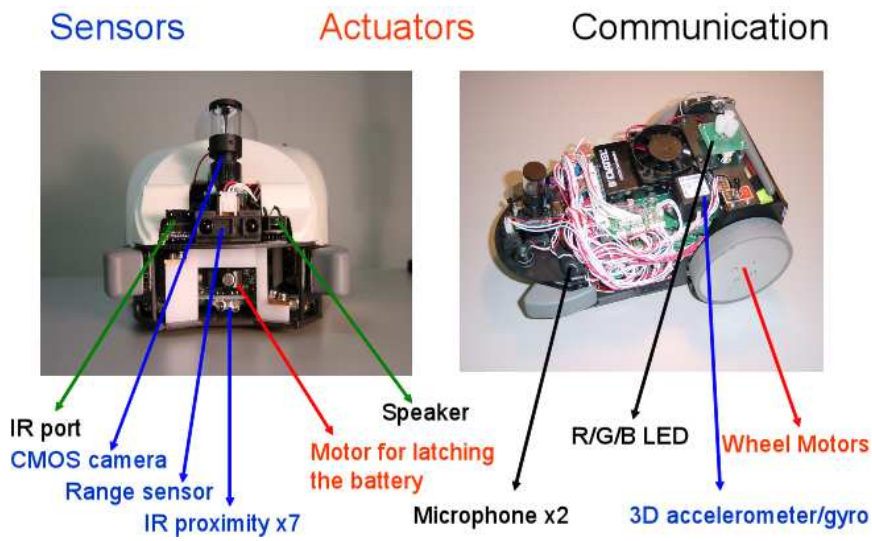

Figure 4.5. The sensors, actuators and communication devices of the Cyber Rodent.

Rodents and their environment.

Research in Cyber Rodent project includes: 1) the evolution of neural controllers for sequential navigation (Capi and Doya, 2003, 2004); 2) evolution of the meta-parameters α and τ for a foraging task where the number of battery packs change over the learning time (Eriksson, Capi, and Doya, 2003); 3) development of a novel reinforcement learning method, where an appropriate policy is selected from a set of heterogeneous reinforcement learning modules and the policies of all modules, including those not selected, are improved, using importance sampling (Uchibe and Doya, 2004), and 4) the demonstration of a sustaining colony of four Cyber Rodents for hours of operation, where the supplementary reward signals of three basic reinforcement learning behaviors were evolutionary optimized (Doya and Uchibe, 2005).

Chapter 5

Summary of Included Papers

This chapter summarizes the included papers and highlights the scientific contributions. The first three papers show the cumulative research process of the development of the embodied evolution framework. The fourth paper investigates the co-evolution of meta-parameters and potential-based shaping rewards. The fifth paper proposes an evolutionary approach to the automatic construction of task hierarchies in the hierarchical reinforcement learning method MAXQ.

5.1 Paper I: Multi-Agent Reinforcement Learning: Using Macro Actions to Learn a Mating Task

This paper is the first step towards the development of an embodied evolution framework, by studying the learning of mating between two Cyber Rodent robots in isolation. Standard reinforcement learning is not well suited for learning cooperative multi-agent tasks, because the success of an agent's behavior depends strongly on the dynamic interaction with other agents, and not only with the interaction with a static environment, which is the standard assumption in reinforcement learning. For mating, the main challenge relates to the fact that it is difficult to predict the actions of the other, autonomous, agent, which increases the uncertainty in the outcome of the agent's actions, i.e., the state transitions and the reward signals. We introduce a few simple macro actions, i.e., actions that are extended in time for more than one time step, to make the agent's behavior more predictable for the mating partner. The macro actions are implemented as options and force the agent to repeat the same action for several time steps. The simulation results show a significant increase in performance for learning with macro actions, both in regards to the initial learning performance and in regards to the obtained learned policy after learning.

In hindsight, the conclusion that macro-actions are required for learning the mating behavior was premature. In the subsequent studies (see Papers II and III),

learning of mating without macro-actions works satisfactory, after changing the algorithm from Q-learning to Sarsa, and, most importantly, changing the function approximation from a normalized radial basis function network to tile coding with finer resolution. However, this confirms one major argument in this thesis, that the design of meta-properties in reinforcement learning is difficult from an engineering perspective, and it shows the need for automatic meta-learning methods.

5.2 Paper II: Biologically Inspired Embodied Evolution of Survival

In this paper we propose a framework for performing embodied evolution with a limited number of robots, by utilizing time-sharing for evaluation of subpopulations of virtual agents inside each robot. We apply an autonomous selection scheme, where each virtual agent stores the genotype of the best estimated mating partner. The genotypes and the estimated fitness of the other agents are received via mating, a pair-wise exchange of information with virtual agents controlling other robots. In this study, there is no within-generation learning of the basic behaviors. Instead, the policies of basic behaviors are learned in advance, in isolation, by reinforcement learning. The behavior of a virtual agent is genetically determined by a linear feedforward neural network that selects basic behaviors according to the current environmental state and the virtual agent's internal battery level, and computes the recharge time when the virtual agent has captured an external battery pack. We define the fitness as the number of batteries captured during the lifetime of a virtual agent. To validate the proposed method, we compared the evolutionary performance with evolution with standard centralized selection. The simulation results show that the individuals in evolution with centralized selection captures more batteries, which is explicitly promoted by the fitness function. However, the individuals in our proposed method perform more matings, which is only implicitly promoted by fact that virtual agents that mates more frequently have higher probability of spreading their genotypes in the population.

We also present experimental results for the implementation of our embodied evolution framework in hardware. We, first, evolved the virtual agents in simulation for 40 generations and, then, transferred the obtained genotypes to the real Cyber Rodent robots. In hardware, we evolved the virtual agents for approximately 10 additional generations, and for all three robots used in the experiments the average fitness values increased significantly.

5.3 Paper III: Darwinian Embodied Evolution of the Learning Ability for Survival

In this paper we propose an improved embodied evolution framework, compared with Paper II, for for performing embodied evolution with a limited number of robots. Within this framework we combine within-generation learning of basic be-

haviors by reinforcement learning, and evolutionary adaptation of parameters that modulate the learning ability of the basic behaviors. A top-layer neural network selects learning modules, corresponding to the basic behaviors, according to the current state. The learning modules learn their behaviors by the reinforcement learning algorithm Sarsa(λ), and the learning is modulated by additional reward signals, implemented as potential-based shaping rewards, and the global meta-parameters. We introduce an implicit and biologically inspired selection scheme, in which there is no explicit representation or communication of the virtual agents' fitness information. A virtual agent can only reproduce offspring by mating with virtual agents controlling other robots, and the probability that the virtual agent will reproduce offspring in its own subpopulation is dependent on the virtual agent's energy level at the mating occasion. Differential selection is achieved by that virtual agents that perform more reproductive matings have higher probability of transferring its offspring to the next generation.

The experimental results in simulation show that the proposed method has similar evolutionary performance compared to evolution with standard centralized selection. We also transfer the best obtained solutions in simulation to the hardware using two real Cyber Rodent robots. Very encouragingly, the learning performance in hardware of the two basic behaviors, mating and foraging, are similar to the simulation results under the same environmental conditions. An interesting result is that meta-parameter τ, which controls the trade-off between exploration and exploitation in softmax action selection, quickly evolves to become 0, and thereby makes the action selection greedy for both learning modules. There are also a highly significant correlation between the number of matings the virtual agents performed in the last generation in all 20 simulation runs (used as estimate of the efficiency of the shaping rewards of the mating behavior) and the values of the meta-parameter λ, which controls the decay of the eligibility traces. For virtual agents that perform more matings, the obtained λ-values are smaller, which suggests that more optimized shaping rewards reduce the need to propagate the reward information to preceding state-action pairs.

5.4 Paper IV: Co-Evolution of Shaping Rewards and Meta-Parameters in Reinforcement Learning

In this paper we explore the co-evolution of shaping rewards and meta-parameters, using the mountain-car task and a foraging task for the Cyber Rodent robot, as testbeds. The simulation results for the mountain-car task show that the best obtained solution improves the initial performance by factor larger than 10 and improves the convergence speed to robust policies by a factor larger than 60. We also transferred the best obtained solution for the foraging task to the real Cyber Rodent. Similar to the results in Paper III, there was no significant difference between the learning performance in hardware and in simulation. This suggest that the co-evolution of shaping rewards and meta-parameters could be used as an

effective method for bridging the difference between idealized computer simulation and real hardware experiments. Also similar to the results in Paper III, the action selection tends to be become greedy very early on in the evolutionary process and the obtained trace-decay meta-parameter, λ, for the optimized shaping rewards is small.

5.5 Paper V: Evolutionary Development of Hierarchical Learning Structures

Hierarchical reinforcement learning algorithms can learn a policy faster than standard reinforcement learning algorithms. However, the applicability of hierarchical reinforcement learning algorithms depends critically on the decomposition of the task into suitable subtasks, which has to be performed in advance by the human designer. In this paper we propose a Lamarckian evolutionary approach for automatic construction of task hierarchies in the hierarchical reinforcement learning method MAXQ. In the proposed method the MAXQ method learns the policy based on the task hierarchies obtained by genetic programming, while the genetic programming explores the appropriate hierarchies using the results of the MAXQ method. In the MAXQ method the learning of a subtask's policy is independently of the policy of its parent subtask's policy, which makes it possible to reuse learned policies of the subtasks over the generations in a Lamarckian fashion.

We validate the proposed method in simulation, using a foraging task for the Cyber Rodent robot. The task for the Cyber Rodent is to find and capture a battery pack, and then return the battery pack to the robot's nest. The main results from the experiments are that: 1) it is a strong interconnection between the obtained task hierarchy and the given task environment; 2) the genetic programming tries to find a minimal solution, which minimizes the number of primitive subtasks that can be executed in each type of situation; 3) Lamarckian evolution can have an advantage over standard Darwinian evolution, given that the environmental setting is difficult enough to require a mix of primitive subtasks to accomplish the subgoals, and 4) it is important to take both the task and the environment in consideration when performing task decomposition.

Chapter 6

Concluding Remarks

This thesis has explored meta-learning in reinforcement learning from an evolutionary perspective in two contexts: 1) as a method for equipping agents with meta-learning capabilities for survival in an embodied evolution framework (Paper III), and 2) as a design tool for automatic optimization of meta-properties in reinforcement learning (Papers IV and V).

This thesis proposed a framework for realizing embodied evolution with a limited number of robots, by utilizing time-sharing of subpopulation of virtual agents (Paper III). The framework integrates meta-learning of basic reinforcement learning behaviors for survival, i.e., capturing and recharging from external batteries, and pair-wise mating between robots to exchange genotypes for reproduction. The framework applies an implicit and biologically inspired selection scheme, whereas in nature an agent can only reproduce offspring by mating with agents that control other robots in the environment. In addition, the probability of reproducing offspring is dependent on the agents' "health" at the mating occasions. Simulation results show that our proposed method has similar performance to evolution with standard centralized selection. The only notable difference is that the proposed method has higher initial performance. We also transferred the best obtained solution to the real Cyber Rodent robot, and the learning performance was similar in both hardware and simulation. This result was confirmed in Paper IV, where we transferred, for a foraging task, shaping rewards and meta-parameters obtained in simulation to a real Cyber Rodent robot. In both Paper III and IV, the experimental results show that shaping rewards can drastically reduce the amount of exploration required for efficient learning. In both studies the action selection tends to be greedy very early on in the evolutionary process. Also, both studies show that more optimized shaping rewards, by giving more accurate reward feedback for the state transitions, can reduce the need to propagate received rewards back to preceding states. Simulation results for the mountain-car task (Paper IV) show that co-evolution of shaping rewards and meta-parameters can greatly improve both the initial performance and the convergence speed to robust policies.

This thesis proposed a Lamarckian evolutionary method for automatic construction of task hierarchies in hierarchical reinforcement learning method MAXQ, using genetic programming techniques (Paper V). The results of the simulation experiments for different environmental settings of a foraging and homing task showed that there is strong relation between the obtained task hierarchy and the environmental setting. This demonstrates the importance of taking the environment into consideration when performing task decomposition in hierarchical reinforcement learning. The results also showed that Lamarckian evolution can be more effective than standard Darwinian evolution, because Lamarckian evolution allows the learning of the policies of the subtasks to be continued over the generations.

6.1 Future Research Directions

The main unrealized goal of this thesis is the validation of the proposed methods in hardware experiments on the Cyber Rodent robots. The evolutionary method for automatic construction of task hierarchies in the MAXQ method (Paper V) is relatively difficult to transfer directly to the physical robots. The main obstacle is related to computational capacity of the Cyber Rodents and the high computational cost, both for the computation of the projected value functions needed for action selection and for the computation of the batch updating of the completion functions for learning in the subtasks. It would probably also be wise to change function approximation method from normalized radial basis functions networks to tile coding, to reduce the computational requirements.

The two biggest issues that have be addressed to implement the embodied evolution framework (Paper III) on the Cyber Rodents are how to sustain the power level of the robot's internal batteries for an extended period of time, and the time required for evolution and learning. The simplest solution to the first problem would be to pre-program the recharging scheme of the external battery packs (Doya and Uchibe, 2005). This is, of course, not an ideal solution. In an embodied evolution framework intended for the investigation of learning mechanisms under biological constraints, the agent should learn or evolve the ability to sustain the internal energy level of the robot. The Cyber Rodent project is now developing the second generation hardware platform. One of the design objectives of the new robot is that it should be easier to monitoring the robot's internal battery level, and also to improve the ability to control the recharging from external batteries. The project will also produce considerably more second generation robots than the four Cyber Rodents the project currently has in its possession. This will most likely reduce the required number of virtual agents inside each robot, and thereby reduce the total evolution time. To accelerate the evolutionary process, a possible source of inspiration is the evaluation approach proposed by Whiteson and Stone (2006). They introduced explicit selection within the generations. Instead of selecting individuals for evaluation randomly, they selected individuals according to standard reinforcement learning action selection strategies, e.g., ϵ-greedy or softmax. They showed

that explicit selection within the generations can improve the efficiency of the evolution, for evolutionary meta-learning of function approximation in Q-learning.

An interesting result in the embodied evolution study is that there are variations in the obtained shaping rewards for the mating behavior in the different simulation runs. Combined with the fact that the shaping rewards are gradually improved throughout the evolution, this suggests a feedback dynamics between the within-generation learning of the mating behavior and the evolutionary adaptation of the learning ability for mating. If this result is related to Baldwinian evolution, then the within-generation learning of the mating should influence the shaping rewards in subsequent generations.

The distributed and multi-agent nature of embodied evolution makes it particular suited for the study of several interesting issues. In the current implementation, the agents are completely honest about their health, i.e., the implicit expression of the fitness. It would be interesting to explore settings where the agents can deceive the mating partner, and also have the ability to resist deception. Another mating related issue is, under which circumstances different mating strategies are more appropriate, e.g., whether to use a liberal mating approach or to use a more conservative approach, by searching for high-performance mating partners. Embodied evolution is also a natural framework for studying the evolution of multi-agent behaviors. Suitable tasks for the Cyber Rodents could be cooperative foraging of batteries, or competition for limited sources of energy. In multi-agent tasks it also seems natural to explore the communication between the agents. In the colony of Cyber Rodents, communication in the form of visual or audio signaling, displaying the state or intention of the agents, could be used as a means to aid the cooperation between the robots. Some form of communication for the display of the agents' fitness would also be required to study different mating strategies.

Bibliography

P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. 2007. An Application of Reinforcement Learning to Aerobatic Helicopter Flight. To appear in *Proceedings of Advances in Neural Information Processing Systems (NIPS-2007)*.

D. H. Ackley and M. L. Littman. 1991. Interactions between learning and evolution. In *Proceedings of the Conference on Artificial Life*, volume 2, pages 478–507.

J. S. Albus. 1971. A theory of cerebellar function. *Mathematical Biosciences*, 10: 25–61.

J. S. Albus. 1981. *Brain, Behavior, and Robotics*. Byte Books.

R. L. Atkinson, R. C. Atkinson, E. E. Smith, D. J. Bem, and S. Nolen-Hoeksema. 1996. *Hilgard's Introduction to Psychology*, 12 edition. Harcourt Brace College Publishers.

P. Augustsson, K. Wolff, and P. Nordin. 2002a. Creation of A learning, flying robot by means of evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 1279–1285.

P. Augustsson, K. Wolff, and P. Nordin. 2002b. The evolution of variable learning rates. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 52–59.

L. C. Baird. 1995. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the International Conference on Machine Learning (ICML-1995)*, pages 30–37.

J. Baldwin. 1896. A new factor in evolution. *American Naturalist*, 30:441–451.

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

R. E. Bellman. 1957. *Dynamic Programming*. Princeton University Press.

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

M. Bowling and M. Veloso. 2002. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250.

J. A. Boyan and A. W. Moore. 1995. Generalization in reinforcement learning: Safely approximating the value function. In *Proceedings of Advances in Neural Information Processing Systems (NIPS-1995)*, volume 7, pages 369–376.

S. J. Bradtke. 1993. Reinforcement learning applied to linear quadratic regulation. In *Proceedings of Advances in Neural Information Processing Systems (NIPS-1993)*, volume 5, pages 295–302.

D. S. Broomhead and D. Lowe. 1988. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355.

G. Capi and K. Doya. 2003. Evolving recurrent neural controllers for sequential tasks: A parallel implementation. In *Proceedings of the Congress on Evolutionary Computation (CEC-2003)*, volume 1, pages 514–519.

G. Capi and K. Doya. 2004. Evolution of neural architecture fitting environmental dynamics. *Adaptive Behavior*, 13:53–66.

C. Darwin. 1859. *On The Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* John Murray.

P. Dayan and G. E. Hinton. 1993. Feudal reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems (NIPS-1993)*, volume 5, pages 271–278.

T. G. Dietterich. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

M. Dorigo and M. Colombetti. 1998. *Robot Shaping: An Experiment in Behavior Engineering.* MIT Press/Bradford Books.

K. L. Downing. 2001. Adaptive genetic programs via reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001).*

K. Doya. 2002. Metalearning and neuromodulation. *Neural Networks*, 15(4).

K. Doya and E. Uchibe. 2005. The cyber rodent project: Exploration of adaptive mechanisms for self-preservation and self-reproduction. *Adaptive Behavior*, 13 (2):149–160.

J. L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14(2):179–211.

A. Eriksson, G. Capi, and K. Doya. 2003. Evolution of meta-parameters in reinforcement learning algorithm. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS-2003)*, pages 412–417.

D. Floreano and F. Mondada. 1996. Evolution of plastic neurocontrollers for situated agents. In *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB-1996)*, pages 401–410.

D. Floreano and F. Mondada. 1998. Evolutionary neurocontrollers for autonomous mobile robots. *Neural Networks*, 11(7-8):1461–1478.

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

D. O. Hebb. 1949. *The Organization of Behavior.* Wiley and Sons.

G. E. Hinton and S. J. Nowlan. 1987. How learning can guide evolution. *Complex Systems*, 1:495–502.

J. H. Holland. 1975. *Adaptation in natural and artificial systems.* Ann Arbor, MI: The University of Michigan Press.

G. S. Hornby, M. Fujita, S. Takamura, T. Yamamoto, and O. Hanagata. 1999. Autonomous evolution of gaits with the sony quadruped robot. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, volume 2, pages 1297–1304.

P. Husbands, I. Harvey, and D. Cliff. 1993. An evolutionary approach to situated AI. In *Proceedings of the conference of the Society for the Study of Artificial Intelligence and the Simulation of Behavior (AISB-1993).*

H. Iba. 1998. Multi-agent reinforcement learning with genetic programming. In *Proceedings of the Annual Conference on Genetic Programming (GP-1998).*

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

S. Kamio and H. Iba. 2005. Adaptation technique for integrating genetic programming and reinforcement learning for real robots. *IEEE Transactions on Evolutionary Computation*, 9(3):318–333.

J. R. Koza. 1992. *Genetic Programming: On the programming of Computers by Means of Natural Selection.* MIT Press.

J. R. Koza. 1994. *Genetic programming II: automatic discovery of reusable programs.* MIT Press.

J. B. Lamarck. 1809. *Philosophie Zoologique.* Chez Dentu.

A. Laud and G. DeJong. 2003. The influence of reward on the speed of reinforcement learning: An analysis of shaping. In *Proceedings of the International Conference on Machine learning (ICML-2003)*, pages 440–447.

S. Mahadevan. 1996. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1-3):159–195.

M. Mataric. 1994. Reward functions for accelerated learning. In *Proceedings of the international conference on Machine learning (ICML-1994)*.

M. Mataric. 1997. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83.

M. Mataric and D. Cliff. 1996. Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19:67–83.

G. Mayley. 1996. Landscapes, learning costs, and genetic assimilation: Modeling the evolution of motivation. *Evolutionary Computation*, 4(3):213–234.

F. Mondada and D. Floreano. 1995. Evolution of neural control structures: Some experiments on mobile robots. *Robotics and Autonomous Systems*, 16(2–4):183–195.

D. J. Montana. 1995. Strongly typed genetic programming. *Evolutionary Computation*, 3(2–4):199–230.

A. W. Moore. 1990. *Efficient Memory-based Learning for Robot Control*. PhD thesis, University of Cambridge.

A. W. Moore and C. G. Atkeson. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130.

U. Nehmzow. 2002. Physically embedded genetic algorithm learning in multi-robot scenarios: The PEGA algorithm. In *Proceedings of the international Workshop on Epigenetic Robotics and Robotics (EPIROB-2002)*.

A. Ng, D. Harada, and S. Russell. 1999. Policy invariance under reward transformations: theory and application to reward shaping. In *Proceedings of the international conference on Machine learning (ICML-1999)*, pages 278–287.

Y. Niv, D. Joel, I. Meilijson, and E. Ruppin. 2002. Evolution of reinforcement learning in uncertain environments: a simple explanation for complex foraging behaviors. *Adaptive Behavior*, 10(1):5–24.

S. Nolfi and D. Floreano. 2000. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines*. MIT Press.

P. Nordin and W. Banzhaf. 1997. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5(2):107– 140.

R. Parr and S. Russell. 1997. Reinforcement learning with hierarchies of machines. In *Proceedings of Advances in Neural Information Processing Systems (NIPS-1997)*, volume 10, pages 1043–1049.

J. Randløv and P. Alstrøm. 1998. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the international conference on Machine learning (ICML-1998)*.

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

E. Ruppin. 2002. Evolutionary autonomous agents: A neuroscience perspective. *Nature Review Neuroscience*, 3:132–141.

A. Samuel. 1959. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:210–229.

T. Sasaki and M. Tokoro. 1997. Adaptation toward changing environments: Why darwinian in nature? In *Proceedings of the European Conference on Artificial Life (ECAL-1997)*, volume 4, pages 145–153.

N. N. Schraudolph, Peter Dayan, and Terrence J. Sejnowski. 1994. Temporal difference learning of position evaluation in the game of go. In *In Proceedings of Advances in Neural Information Processing (NIPS-1994)*, pages 817–824.

Wolfram Schultz, Peter Dayan, and P. Read Montague. 1997. A neural substrate of prediction and reward. *Science*, 275:1593–1599.

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

B. F. Skinner. 1938. *The Behavior of Organisms: An Experimental Analysis*. Prentice Hall.

B. F. Skinner. 1953. *Science and Human Behavior*. Collier-Macmillian.

R. S. Sutton. 1984. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts.

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

R. S. Sutton. 1992a. Adapting bias by gradient descent: an incremental version of the delta-bar-delta. In *Proceedings of the National Conference on Artificial Intelligence*.

R. S. Sutton. 1992b. Gain adaptation beats least squares? In *Proceedings of the Yale Workshop on Adaptive and Learning Systems*, pages 161–166.

R. S. Sutton. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Proceedings of Advances in Neural Information Processing Systems (NIPS-1996)*, volume 8, pages 1038–1044.

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

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of Advances in Neural Information Processing Systems (NIPS-2000)*, volume 12, pages 1057–1063.

R. S. Sutton, D. Precup, and S. P. Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211.

R. S. Sutton and S. P. Singh. 1994. On step-size and bias in temporal difference learning. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pages 91–96.

G. Tesauro. 1989. Neurogammon wins computer olympiad. *Neural Computation*, 1:321–323.

G. Tesauro. 1992. Practical issues in temporal difference learning. In *Proceedings of Advances in Neural Information Processing Systems (NIPS-1992)*, volume 4, pages 259–266.

G. Tesauro. 1994. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219.

G. Tesauro. 1995. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

S. Thrun. 1995. Learning to play the game of chess. In *Proceedings of Advances in Neural Information Processing Systems (NIPS-1995)*, pages 1069–1076.

P. Turney, D. Whitley, and R. Anderson. 1996. Introduction to the special issue: Evolution, learning, and instinct: 100 years of the baldwin effect. *Evolutionary Computation*, 4(3):iv–viii.

E. Uchibe and K. Doya. 2004. Competitive-cooperative-concurrent reinforcement learning with importance sampling. In *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB-2004)*, pages 287–296.

T. Unemi, M. Nagaoyoshi, N. Hirayama, T. Nade, K. Yano, and Y. Masujima. 1994. Differentiation of learning abilities – a case study on optimizing parameter values in q-learning by a genetic algorithm. In *Proceedings of the International Workshop on the Synthesis and Simulation of Living Systems*, pages 331–336.

J. Urzelai and D. Floreano. 2000. Evolutionary robotics: Coping with environmental change. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 941–948.

C. H. Waddington. 1942. Canalization of development and the inheritance of acquired characters. *Nature*, 150:563–565.

G. P. Wagner and L. Altenberg. 1996. Complex adaptations and the evolution of evolvability. *Evolution*, 50:967–976.

J. Walker, S. Garrett, and M. Wilson. 2003. Evolving controllers for real robots: A survey of the literature. *Adaptive Behavior*, 11(3):179–203.

C. J. Watkins. 1989. *Learning from delayed rewards*. PhD thesis, Cambridge University.

R. Watson, S. Ficici, and J. Pollack. 2002. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18.

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

E. Wiewiora. 2003. Potential-based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208.

R. J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

W. Zhang and T. G. Dietterich. 1995. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intellience*.