Evolutionary Computation versus Reinforcement Learning

Jürgen Schmidhuber
IDSIA, Galleria 2, 6928 Manno (Lugano), Switzerland

Abstract

Many applications of Reinforcement Learning (RL) and Evolutionary Computation (EC) are addressing the same problem, namely, to maximize some agent's fitness function in a potentially unknown environment. The most challenging open issues in such applications include partial observability of the agent's environment, hierarchical and other types of abstract credit assignment, and the learning of credit assignment algorithms. I will summarize why EC provides a more natural framework for addressing these issues than RL based on value functions and dynamic programming. Then I will point out fundamental drawbacks of traditional EC methods in case of stochastic environments, stochastic policies, and unknown temporal delays between actions and observable effects. I will discuss a remedy called the success-story algorithm which combines aspects of RL and EC.

1 Introduction

A policy is the set of modifiable, behavior-defining parameters of a learning agent. A learning algorithm is an algorithm that modifies the policy. In the context of sequential decision making there are two broad classes of widely used learning algorithms: (1) methods based on dynamic programming (DP) [2], and (2) evolutionary computation (EC). DP-based RL (DPRL) learns a value function mapping input/action pairs to expected discounted future reward and uses online variants of DP for constructing rewarding policies [28, 43, 49]. EC runs and evaluates policies directly, building new policy candidates from those with the highest evaluations observed so far. EC methods include evolutionary strategies [23, 38], genetic algorithms (GAs) [9], genetic programming (GP) [4, 1], and adaptive extensions of Levin Search [41, 37]. EC offers several advantages over DPRL, but also has some drawbacks. I will list advantages first, then point out a major problem of EC, and offer a remedy.

EC Advantage 1: No States. Finite time convergence proofs for DPRL [13] require (among other things) that the environment can be quantized into

a finite number of discrete states, and that the topology describing possible transitions from one state to the next, given a particular action, is known in advance. Even if the real world was quantizable into a discrete state space, however, for all practical purposes this space will be inaccessible and remain unknown. Current proofs do not cover apparently minor deviations from the basic principle, such as the world-class RL backgammon player [47], which uses a nonlinear function approximator to deal with a large but finite number of discrete states and, for the moment at least, seems a bit like a miracle without full theoretical foundation. Prior knowledge about the topology of a network connecting discrete states is also required by algorithms for partially observable Markov decisicion processes (POMDPs), although they are more powerful than standard DPRL, e.g., [12, 19]. In general, however, we do not know *a priori* how to quantize a given environment into meaningful states.

EC, however, completely avoids the issues of value functions and state identification — it just cares for testing policies and keeping those that work best.

EC Advantage 2: No Markovian Restrictions. Convergence proofs for DPRL also require that the learner's current input conveys all the information about the current state (or at least about the optimal next action). In the real world, however, the current sensory input typically tells next to nothing about the "current state of the world," if there is such a thing at all. Typically, memory of previous events is required to disambiguate inputs. For instance, as your eyes are sequentially scanning the visual scene dominated by this text you continually decide which parts (or possibly compressed descriptions thereof) deserve to be represented in short-term memory. And you have presumably *learned* to do this, apparently by some unknown, sophisticated RL method fundamentally different from DPRL. Some DPRL variants such as $Q(\lambda)$ are limited to a very special kind of exponentially decaying short-term memory. Others simply ignore memory issues by focusing on suboptimal, memory-free solutions to problems whose optimal solutions do require some form of short-term memory [11]. Again others can in principle find optimal solutions even in partially observable environments [12, 19], but they (a) are practically limited to very small problems [18], and (b) do require knowledge of a discrete state space model of the environment. To various degrees, problem (b) also holds for certain hierarchical RL approaches to memory-based input disambiguation [24, 25, 26, 20, 50]. Although no discrete models are necessary for DPRL systems with function approximators based on recurrent neural networks [31, 17], the latter do suffer from a lack of theoretical foundation, perhaps even more so than the backgammon player.

EC, however, does not care at all for Markovian conditions and full observability of the environment. While DPRL is essentially limited to learning reactive policies mapping current inputs to output actions, EC in principle can be applied to search spaces whose elements are general algorithms or programs with time-varying variables that can be used for memory purposes [52, 46, 32, 51, 27].

EC Advantage 3: Straight-forward Hierarchical Credit Assignment. There has been a lot of recent work on hierarchical DPRL. Some

researchers address the case where an external teacher provides intermediate subgoals and/or prewired macro actions consisting of sequences of lower-level actions [21, 48, 44, 39, 10, 7, 45]. Others focus on the more ambitious goal of automatically learning useful subgoals and macros [30, 8, 24, 26, 5, 50, 42].

Most current work in hierarchical DPRL aims at speeding up credit assignment in fully observable environments. Approaches like HQ-learning [50], however, additionally achieve a qualitative (as opposed to just quantitative) decomposition by learning to decompose problems that cannot be solved at all by standard DPRL into several DPRL-solvable subproblems and the corresponding macro-actions. Generally speaking, non-trivial forms of hierarchical RL almost automatically run into problems of partial observability, even those with origins in the MDP framework. Feudal RL [5], for instance, is subject to such problems (Ron Williams, personal communication). As Peter Dayan himself puts it (personal communication): *"Higher level experts are intended to be explicitly ignorant of the details of the state of the agent at any resolution more detailed than their action choice. Therefore, the problem is really a POMDP from their perspective. It's easy to design unfriendly state decompositions that make this disastrous. The key point is that it is highly desirable to deny them information – the chief executive of [a major bank] doesn't really want to know how many paper clips his most junior bank clerk has – but arranging for this to be benign in general is difficult."*

In the EC framework, however, hierarchical credit assignment via frequently used, automatically generated subprograms becomes trivial in principle. For instance, suppose policies are programs built from a general programming language that permits parameterized conditional jumps to arbitrary code addresses [6, 22, 51, 37, 35]. EC will simply keep successful hierarchical policies that partially reuse code (subprograms) via appropriate jumps. Again, partial observability is not an issue.

EC Advantage 4: Non-Hierarchical Abstract Credit Assignment. Hierarchical learning of macros and reusable subprograms is of interest but limited. Often there are *non*-hierarchical (nevertheless exploitable) regularities in solution space. For instance, suppose we can obtain solution B by replacing every action "*turn(right)*" in solution A by "*turn(left).*" B will then be regular in the sense that it conveys little additional conditional algorithmic information, given A [40, 14, 3, 16], that is, there is a short algorithm computing B from A. Hence B should not be hard to learn by a smart RL system that already found A. While DPRL cannot exploit such regularities in any obvious manner, EC in general algorithm spaces does not encounter any fundamental problems in this context. For instance, all that is necessary to find B may be a modification of the parameter *"right"* of a single instruction *"turn(right)"* in a repetitive loop computing A [37].

EC Advantage 5: Metalearning Potential. In a given environment, which is the best way of collecting reward? Hierarchical RL? Some sort of POMDP-RL, or perhaps analogy-based RL? Combinations thereof? Or other

nameless approaches to exploiting algorithmic regularities in solution space? A smart learner should find out by itself, using experience to improve its own credit assignment strategy (metalearning or "learning to learn") [15, 29]. In principle, such a learner should be able to run *arbitrary* credit assignment strategies, and discover and use "good" ones, without wasting too much of its limited life-time [35]. It seems obvious that DPRL does not provide a useful basis for achieving this goal, while EC seems more promising as it does allow for searching spaces populated with arbitrary algorithms, including metalearning algorithms.

Summary. Given the potential EC advantages listed above (most of them related to partial observability), it may seem that the more ambitious the goals of some RL researcher, the more he/she will get drawn towards methods for EC in spaces of fairly general algorithms, as opposed to the more limited DPRL-based approaches. Standard EC does suffer from major disadvantages, though, as I will point out next for the case of realistic, stochastic worlds.

2 Problems: Unknown Delays, Stochasticity

As pointed out above, EC does not require certain assumptions about the environment necessary for traditional RL. In particular, while the latter is essentially limited to memory-free, reactive mappings from inputs to actions, EC can be used to search among fairly arbitrary, complex, event-memorizing programs (using memory to deal with partial observability), at least in simulated, deterministic worlds. In realistic settings, however, reliable policy evaluations are complicated by (a) unknown delays between action sequences and observable effects, and (b) stochasticity in policy and environment. Given a limited life-time, how much time should an EC-based search algorithm spend on policy evaluations to obtain reliable statistics? Despite the fundamental nature of this question it has not received much attention yet. Here I motivate an efficient approach based on the *success-story algorithm* (SSA). It provides a radical answer prepared for the worst case: it *never* stops evaluating any previous policy modification except those it undoes for lack of empirical evidence that they have contributed to lifelong reward accelerations.

The problem. In realistic situations EC exhibits several fundamental drawbacks: **(1)** Often there are unknown temporal delays between causes and effects. In general we cannot be sure that a given trial was long enough to observe all long-term rewards/punishments caused by actions executed during the trial. We do not know in advance how long a trial should take. **(2)** The policy may be stochastic, i.e., the learner's actions are selected nondeterministically according to probability distributions conditioned on the policy. Stochastic policies are widely used to prevent learners from getting stuck. Results of policy evaluations, however, will then *vary from trial to trial.* **(3)** Environment and reward may be stochastic, too. And even if the environment is deterministic it may appear stochastic from an individual learner's perspective, due to partial observability.

Time is a scarce resource. Hence all EC-based methods face a central question: to determine whether some policy is really useful in the long run or just appears to be (e.g., because the current trial was too short to encounter a punishing state, or because it was a lucky trial), how much time should the learner spend on its evaluation? In particular, how much time should a single trial with a given policy take, and how many trials are necessary to obtain statistically significant results without wasting too much time?

Basic idea. There is a radical answer to such questions: Evaluate a previous policy change at any stage of the search process by looking at the entire time interval that has gone by since the change occurred — at any given time aim to use *all* the available empirical data concerning long-term policy-dependent rewards! A change is considered "good" as long as the average reward per time since its creation exceeds the corresponding ratios for previous "good" changes. Changes that eventually turn out to be "bad" get undone by an efficient backtracking scheme called the *success-story algorithm* (SSA). SSA always takes into account the latest information available about long-term effects of changes that have appeared "good" so far ("bad" changes, however, are not considered again). Effectively SSA adjusts trial lengths retrospectively: at any given time, trial starts are determined by the occurrences of the remaining "good" changes representing a success story. The longer the time interval that went by since some "good" change, the more reliable the evaluation of its true long-term benefits (in general, SSA's own computation time has to be taken into account). No trial of a "good" change ever ends unless it turns out to be "bad" at some point. Thus SSA is prepared for the worst case. For instance, no matter what's the maximal time lag between actions and consequences in a given domain, eventually SSA's effective trials will encompass it. Details of SSA are described elswhere [37, 35, 33].

Experiments. Previous experimental applications of SSA include simple tasks with unknown delays and stochasticity designed to show how SSA can outperform traditional EC [34], as well as challenging tasks in partially observable environments, which represent a major motivation of EC-based methods because most traditional RL methods are not applicable here [35]. For instance, a previous paper [35] describes two agents A and B living in a partially observable 600×500 pixel environment with obstacles. They learn to solve a complex task that could not be solved by various TD(λ) Q-learning variants [17]. The task requires (1) agent A to find and take a key "key A"; (2) agent A go to a door "door A" and open it for agent B; (3) agent B to enter through "door A", find and take another key "key B"; (4) agent B to go to another door "door B" to open it (to free the way to the goal); (5) one of the agents to reach the goal. Both agents share the same design. Each is equipped with limited "active" sight: by executing certain instructions, it can sense obstacles, its own key, the corresponding door, or the goal, within up to 50 pixels in front of it. The agent can also move forward, turn around, turn relative to its key or its door or the goal. It can use short-term memory to disambiguate inputs — unlike

Jaakkola et al.'s method [11], ours is not limited to finding suboptimal stochastic policies for partially observable environments with an optimal solution. The agents are *metalearners:* Each agent can explicitly modify its own stochastic policy via special actions that can address and modify the probability distributions according to which action sequences (or "subprograms") are generated (this also contributes to making the set-up highly non-Markovian). SSA is used to identify those modifications that have led to long-term reward accelerations. Reward is provided only if one of the agents touches the goal. This agent's reward is 5.0; the other's is 3.0 (for its cooperation — note that asymmetric reward also introduces competition). Due to the complexity of the task, in the beginning the goal is found only every 300,000 actions on average (including actions called "primitive learning algorithms" that modify the policy itself). No prior information about good initial trial lengths is given to the system. Within 130,000 goal hits (10^9 actions) the average trial length decreases by a factor of 60 (mean of 4 simulations). Both agents learn to cooperate to accelerate reward intake, by retrospectively adjusting their effective trial lengths using SSA.

3 Summary

Evolutionary computation (EC) applied to general policy spaces offers several advantages over traditional reinforcement learning (RL). For instance, EC does not need *a priori* information about world states and the topology of their interactions. It does not care whether the environment is fully observable. It makes hierarchical credit assignment conceptually trivial, and also allows for many alternative, non-hierarchical types of abstract credit assignment.

Existing EC methods, however, do suffer from fundamental problems in presence of environmental stochasticity and/or unknown temporal delays between actions and observable effects. In particular, they do not have a principled way of deciding when to stop policy evaluations.

Stochastic policy evaluation by the success-story algorithm (SSA) differs from traditional EC. SSA never quits evaluating any previous policy change that has not yet been undone for lack of empirical evidence that it has contributed to a lifelong reward acceleration. Each invocation of SSA retrospectively establishes a success history of surviving policy modifications: only policy changes that have empirically proven their long-term usefulness so far get another chance to justify themselves. This stabilizes the "truly useful" policy changes in the long run.

Unlike many traditional value function-based RL methods, SSA is not limited to fully observable worlds, and does not require discounting of future rewards. It shares these advantages with traditional EC algorithms. Unlike stochastic hill-climbing and other EC methods such as genetic algorithms, however, SSA does not heavily depend on *a priori* knowledge about reasonable trial lengths necessary to collect sufficient statistics for estimating long-term conse-

quences and true values of tested policies.

On the other hand, many EC methods can be augmented by SSA in a straight-forward way: just measure the time used up by all actions, policy modifications, and policy tests, and occasionally invoke SSA. In this sense SSA's basic concepts are not algorithm-specific — instead they reflect a novel, general way of thinking about how "true" performance should be measured in systems using EC for search in policy space.

Since SSA automatically collects statistics about long-term effects of earlier policy changes on later ones, it is of interest for improving the credit assignment method itself [29, 36, 35, 37, 33].

References

[1] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction.* Morgan Kaufmann Publishers, San Francisco, CA, USA, 1998.

[2] R. Bellman. *Adaptive Control Processes.* Princeton University Press, 1961.

[3] G.J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *Journal of the ACM*, 16:145–159, 1969.

[4] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In J.J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Hillsdale NJ, 1985. Lawrence Erlbaum Associates.

[5] P. Dayan and G. Hinton. Feudal reinforcement learning. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 5*, pages 271–278. San Mateo, CA: Morgan Kaufmann, 1993.

[6] D. Dickmanns, J. Schmidhuber, and A. Winklhofer. Der genetische Algorithmus: Eine Implementierung in Prolog. Fortgeschrittenenpraktikum, Institut für Informatik, Lehrstuhl Prof. Radig, Technische Universität München, 1987.

[7] B.L. Digney. Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. In Pattie Maes, Maja Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, Cambridge, MA*, pages 363–372. MIT Press, Bradford Books, 1996.

[8] M. Eldracher and B. Baginski. Neural subgoal generation using backpropagation. In George G. Lendaris, Stephen Grossberg, and Bart Kosko, editors, *World Congress on Neural Networks*, pages III–145–III–148. Lawrence Erlbaum Associates, Inc., Publishers, Hillsdale, July 1993.

[9] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[10] M. Humphrys. Action selection methods using reinforcement learning. In Pattie Maes, Maja Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors, *From Animals to Animats 4: Proceedings of the Fourth International*

Conference on Simulation of Adaptive Behavior, Cambridge, MA, pages 135–144. MIT Press, Bradford Books, 1996.

[11] T. Jaakkola, S. P. Singh, and M. I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 345–352. MIT Press, Cambridge MA, 1995.

[12] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. Technical report, Brown University, Providence RI, 1995.

[13] M. Kearns and S. Singh. Finite-sample convergence rates for Q-learning and indirect algorithms. In M. Kearns, S. A. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems 12*. MIT Press, Cambridge MA, 1999.

[14] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–11, 1965.

[15] D. Lenat. Theory formation by heuristic search. *Machine Learning*, 21, 1983.

[16] M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.

[17] L.J. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, Pittsburgh, January 1993.

[18] M.L. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, March 1996.

[19] M.L. Littman, A.R. Cassandra, and L.P. Kaelbling. Learning policies for partially observable environments: Scaling up. In A. Prieditis and S. Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 362–370. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[20] R. A. McCallum. Learning to use selective attention and short-term memory in sequential tasks. In Pattie Maes, Maja Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, Cambridge, MA*, pages 315–324. MIT Press, Bradford Books, 1996.

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

[22] T. S. Ray. An approach to the synthesis of life. In C.G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 371–408. Addison Wesley Publishing Company, 1992.

[23] I. Rechenberg. Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Dissertation, 1971. Published 1973 by Fromman-Holzboog.

[24] M. B. Ring. Incremental development of complex behaviors through automatic construction of sensory-motor hierarchies. In L. Birnbaum and G. Collins, editors, *Machine Learning: Proceedings of the Eighth International Workshop*, pages 343–347. Morgan Kaufmann, 1991.

[25] M. B. Ring. Learning sequential tasks by incrementally adding higher orders. In J. D. Cowan S. J. Hanson and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 115–122. Morgan Kaufmann, 1993.

[26] M. B. Ring. *Continual Learning in Reinforcement Environments.* PhD thesis, University of Texas at Austin, Austin, Texas 78712, August 1994.

[27] R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.

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

[29] J. Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. Institut für Informatik, Technische Universität München, 1987.

[30] J. Schmidhuber. Learning to generate sub-goals for action sequences. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 967–972. Elsevier Science Publishers B.V., North-Holland, 1991.

[31] J. Schmidhuber. Reinforcement learning in Markovian and non-Markovian environments. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 500–506. San Mateo, CA: Morgan Kaufmann, 1991.

[32] J. Schmidhuber. Discovering solutions with low Kolmogorov complexity and high generalization capability. In A. Prieditis and S. Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 488–496. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[33] J. Schmidhuber. A general method for incremental self-improvement and multi-agent learning. In X. Yao, editor, *Evolutionary Computation: Theory and Applications*, pages 81–123. Scientific Publ. Co., Singapore, 1999.

[34] J. Schmidhuber and J. Zhao. Direct policy search and uncertain policy evaluation. In *AAAI Spring Symposium on Search under Uncertain and Incomplete Information, Stanford Univ.*, pages 119–124. American Association for Artificial Intelligence, Menlo Park, Calif., 1999.

[35] J. Schmidhuber, J. Zhao, and N. Schraudolph. Reinforcement learning with self-modifying policies. In S. Thrun and L. Pratt, editors, *Learning to learn*, pages 293–309. Kluwer, 1997.

[36] J. Schmidhuber, J. Zhao, and M. Wiering. Simple principles of metalearning. Technical Report IDSIA-69-96, IDSIA, 1996.

[37] J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning*, 28:105–130, 1997.

[38] H. P. Schwefel. Numerische Optimierung von Computer-Modellen. Dissertation, 1974. Published 1977 by Birkhäuser, Basel.

[39] S.P. Singh. The efficient learning of multiple task sequences. In J.E. Moody, S.J. Hanson, and R.P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 251–258, San Mateo, CA, 1992. Morgan Kaufmann.

[40] R.J. Solomonoff. A formal theory of inductive inference. Part I. *Information and Control*, 7:1–22, 1964.

[41] R.J. Solomonoff. An application of algorithmic probability to problems in artificial intelligence. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 473–491. Elsevier Science Publishers, 1986.

[42] R. Sun and C. Sessions. Self-segmentation of sequences: automatic formation of hierarchies of sequential behaviors. *IEEE Transactions on Systems, Man, and Cybernetics: Part B Cybernetics*, 30(3), 2000.

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

[44] R. S. Sutton. TD models: Modeling the world at a mixture of time scales. In A. Prieditis and S. Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 531–539. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[45] R. S. Sutton, S. Singh, D. Precup, and B. Ravindran. Improved switching among temporally abstract actions. In *Advances in Neural Information Processing Systems 11*. MIT Press, 1999. To appear.

[46] A. Teller. The evolution of mental models. In Jr. Kenneth E. Kinnear, editor, *Advances in Genetic Programming*, pages 199–219. MIT Press, 1994.

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

[48] C.K. Tham. Reinforcement learning of multiple tasks using a hierarchical CMAC architecture. *Robotics and Autonomous Systems*, 15(4):247–274, 1995.

[49] C.J.C.H Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[50] M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1998.

[51] M.A. Wiering and J. Schmidhuber. Solving POMDPs with Levin search and EIRA. In L. Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 534–542. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

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