Solving POMDP by On Policy Linear Approximate Learning Algorithm

by Qiming He, Mark A. Shayman

T.R. 99-68

Solving POMDP by On-Policy Linear Approximate Learning Algorithm

Qiming He, Mark Shayman

Dept. of Electrical and Computer Engineering,
University of Maryland, College Park, MD 20742
qiminghe,shayman@eng.umd.edu

Abstract

This paper presents a fast Reinforcement Learning (RL) algorithm to solve Partially Observable Markov Decision Processes (POMDP) problem. The proposed algorithm is devised to provide a policy-making framework for Network Management Systems (NMS) which is in essence an engineering application without an exact model. The algorithm consists of two phases. Firstly, the model is estimated and policy is learned in a completely observable simulator. Secondly, the estimated model is brought into the partially observed real-world where the learned policy is then fine-tuned. The learning algorithm is based on the on-policy linear gradient-descent learning algorithm with eligibility traces. This implies that the Q-value on belief space is linearly approximated by the Q-value at vertex over the belief space where on-line TD method will be applied. The proposed algorithm is tested against the exact solutions to extensive small/middle-size benchmark examples from POMDP literature and found near optimal in terms of average-discounted-reward and step-to-goal. The proposed algorithm significantly reduces the convergence time and can easily be adapted to large state-number problems.

1 Introduction

This paper is motivated by the problem of designing an intelligent Network Fault Management (NFM) system. In such a system, agents at network elements (NE) render network statistics when being polled. While polling information arrives sequentially, the Network Management System (NMS) manager is responsible for deciding which element is in fault. However, polling introduces additional overhead [10] that may itself degrade network performance especially when the network is in a stressed state. Thus, a tradeoff must be made between the amount of data collected and transferred on one hand, and the speed and accuracy of fault detection and diagnosis on the other hand.

Several factors make such sequential decision-making non-trivial. Firstly, each agent sends symptoms (i.e., observations) of the faults rather than the root-cause (i.e., completely-observed states) of faults. Secondly, a single root-cause may be related to different symptoms in many agents and vice versa. Thirdly, symptoms may arrive to NMS manager misordered, corrupted, or may fail to arrive at all (i.e., noisy sensor).

The NMS manager will take management actions (e.g., to poll or repair a network element) based on sequential observations. Each state and action pair has an associated cost which reflects, e.g., communication bandwidth used by polling, network downtime, or the cost to replace a device. An optimal policy is such an action sequence determined by incoming observations to minimize long-term operating cost. For example, polling devices randomly (or in round-robin) may waste bandwidth and delay fault diagnosis, thereby increasing network downtime. On the other hand, choosing a smart polling sequence that focuses on network elements that are most likely to be fault-related will isolate the problem more quickly and with less communication overhead. An optimal policy will represent a tradeoff between the amount of monitoring on one hand, and the speed and accuracy with which potential faults can be detected and diagnosed.

The problem of choosing an optimal policy for monitoring, diagnosis, and mitigation can be formulated as a Partially Observable Markov Decision Process (POMDP) ([2],[11],[21]) . POMDP may serve as the theoretical basis of general diagnosis without exact knowledge of states and has found many applications [8] [23]. Recent years have seen various exact solution algorithms for POMDP's ([12],[20], [4],[26]). But they all have two major drawbacks that prevent them from being widely deployed in many engineering applications. First, the majority of them require accurate a priori system models. Unfortunately, such models are not realistic for complex systems. This gives rises to demand for model-free RL approaches which learn the control policy in a trial-and-error way. Second, most of exact solution algorithms run into computational intractability when the number of states becomes fairly large as is the case in network management.

In this paper, we develop a fast RL algorithm to obtain approximate solutions to POMDP's with large numbers of states. The idea is that we assign Q-values to belief-action pairs instead of state-action pairs as is done in RL algorithms for MDP, and use linear function to approximate it when rewards (or costs) are received. We build a simulator of the real-world (network system) in which we have complete observability. State-transition probabilities and observation-function are estimated on-line during such simulation phase. Then, we have not only identified an explicit simulator model (if it is not given) but also the policy for the simulator which can be used as jump-starter to supply the next phase. In the execution phase where the state cannot be observed completely, the agent uses the estimated model to update its belief state and fine-tune its policy.

The proposed algorithm is tested against extensive benchmark examples in the literature. Average-discounted-reward and steps-to-goal are recorded as algorithm performance metrics to be compared with one of the most effective exact POMDP solution ([26]). For the small and middle size problems for which the policies can be obtained by the exact solutions within moderate time, values of above two metrics are found to be close to those of the exact solution. Our algorithm can easily be adapted to larger problems by changing iterative parameters. Our experiments show that in the case of model discrepancy between simulator and real-world, learning-based approximate algorithm outperforms exact solutions by virtue of its inherent learning capability. Our experiments also support the observation that, within the spectrum of learning algorithms, on-policy learning is preferable to off-policy learning for control when Q-value on belief-state is approximated linearly.

This paper is organized in four sections. Section 2 briefly covers various solutions to POMDP. Section 3 outlines our fast algorithm and its performances. Section 4 contains conclusions.

2 Solutions to POMDP

POMDP is a framework to handle observation uncertainty in the MDP. A model-based POMDP has 6-tuple $< \mathcal{S}, \mathcal{T}, \mathcal{A}, \mathcal{O}, \mathit{\Pi}, \mathcal{R} >$ where $\mathcal{S}$ is a finite set of state, $\mathcal{A}$ is a finite set of actions, $\mathcal{T}$ is a mapping $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to Pr(\mathcal{S})$. $\mathcal{R}$ is a reward function. $\mathcal{O}$ is the set of observations and $\mathit{\Pi}$ is an observation function $\mathit{\Pi} : \mathcal{S} \times \mathcal{A} \to Pr(\mathcal{O})$. Solution to POMDP refers to a control policy that will yield greatest reward over some finite or infinite horizons ([12]).

In MDP (or Completely Observable MDP), current action only depends on the current state; previous states and actions are irrelevant. In the POMDP, the current action may depend on the entire sequence of past actions and resulting observations. However, it is equivalent to have the current action depend on a sufficient statistic called *belief state*, which gives the probability for each possible value for the current state given all the past actions and observations. By doing such a conversion, a POMDP problem over state space can be treated as an MDP problem over belief space and Markov property holds.

2.1 Exact solutions to POMDP

In principle, solutions to MDP can be used directly to solve POMDP assuming POMDP is an MDP over belief states. Unfortunately, the infinite number of belief states makes the direct use impractical. However, due to Sondik's work ([20]), the optimal value function for any POMDP can be approximated arbitrarily well by piecewise-linear convex function (PWLC), thus the optimal action is

$$\pi(b) = argmax_{\{a \in \mathcal{A} | l_a \in L\}} l_a \cdot b \tag{1}$$

where b is a belief state, $\mathcal{L}$ is a collection of solution vectors of $\mid \mathcal{S} \mid$-dimension.

The discussion of exact solution is beyond the scope of this paper. Basically, numerous exact algorithms to POMDP have their different ways of finding solution vectors set in (1) ([26], [4]). Most of them start by constructing a finite representation of PWLC function over belief state, then iteratively updating this representation, expanding horizon and truncating it at some desired depth.

Generally, such truncated exact solution works well for small problems with less than 100 states. For larger problem, it becomes computationally intractable when the horizon or $|\mathcal{S}|$, $|\mathcal{A}|$, $|\mathcal{O}|$ increases.

2.2 Approximate Solutions to POMDP

In order to overcome to the computing difficulty, quite a few approximate approaches are proposed to solve POMDP quickly. Some of them are based on model-based heuristic search while others utilize model-free learning techniques.

[7] represents policy as a finite-state controller and iteratively improves controller by heuristic search in a policy space. [6] extends Real-Time-Dynamic-Programming (RTDP) algorithm for MDP to POMDP by combining exact value updating and heuristic search and produces near optimal solution especially for large state number grid problem.

[13] and [15] use approximate approach to learn memoryless policy in the case of POMDP. [13] uses a function approximator to represent a stochastic policy which controls a robot moving in a quantized continuous state space. The learning process is based on stochastic gradient ascent algorithm which is performed by a neural-network-like agent with eligibility trace involved. Though the learned policy is memoryless, there is still good chance to combine it with state estimate to yield better results. In [15], memoryless near-optimal policy of POMDP is obtained by learning Q-value of observation-action pair (o, a) with $Sarsa(\lambda)$. Since no explicit state estimation is needed, it relieves the learning algorithm from doing function approximation. However, our specific application shows that immediate observation to action mapping is error-prone especially when observation is noisy or observation space is much smaller than that of action space . Though a couple of benchmark examples are tested in [15], no direct performance comparison is made with exact solutions.

2.3 Solving POMDP by Simulation and Function Approximation

In order to use POMDP as a framework for constructing intelligent decision system, we need to solve POMDP problem without the exact model of the real world. The dynamics of our network are hard to obtain explicitly. A practical approach is to build a simulator that closely resembles the real-world model. The model of the simulator is easy to identify. Then we pretend the real-world model is the same as that of the simulator. Agent brings the optimal policy from the simulator to the real-world and keeps learning to resolve the model inconsistency, if any. Furthermore, the real-life problems often end up with huge number of states, actions and observations. Therefore, it is crucial to use function approximation to parameterize value functions.

Our on-policy linear approximation algorithm is derived from on-policy control algorithm $Sarsa(\lambda)$ ([22], [17]) which is originally applied in MDP. The idea is that we assign Q-value to belief-action pair instead of state-action pair and use linear function to approximate it. It is in essence like the *hybrid solution*

[14] in the sense that both use linear function to approximate Q-value based on gradient descent method. But it differs in that it uses on-policy learning with eligibility traces instead of off-policy Q-learning. On-policy algorithm, backing up via on-line state sampling, is superior to off-policy algorithm in the case of linear function approximation ([24]). In fact, some counterexamples have been found for the off-policy case even though basis function (i.e. features) and learning-step are favorably defined([1] [3]). On the other hand, on-policy TD learning is more elegant, flexible and robust. In particular, on-policy TD(λ), approximated by gradient-descent linear function, is proved to converge to near global optimum with bounded error on condition that features, parameters, cost- function and learning-step-size are chosen properly ([24]). Nevertheless, such analysis only holds for prediction problem where the action is greedily chosen, though it empirically extends to control problem in [3]. Moreover, ([19]) proves the convergence result for $Sarsa(0)$ for specific learning policy families. However, such result has not been extended to multi-step and function approximation yet. The contribution of this paper is to explore experimentally the possibility of approximating Q-value over an infinite belief space by finite state Q-value which is updated in the manner of on-policy TD learning algorithm, e.g. $Sarsa(\lambda)$.

3 Implementation and Performance

3.1 Implementation

As discussed in the previous section, Q-value on belief-action pair will be approximated in the linear form of parameters which can be simply chosen as the vertex Q-values over the belief space

$$Q_t(b_t, a) = \sum_{s \in \mathcal{S}} Q_t(s, a) b_t(s) \tag{2}$$

Using belief as features and vertex Q-value as parameters in POMDP is intuitive and definitely efficient in practice. The updating rule for parameters is ([22])

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha \delta_t e_t(s, a) \qquad \forall s \tag{3}$$

where

$$\delta_t = r_{t+1} + \gamma Q_t(b_{t+1}, a) - Q_t(b_t, a) \tag{4}$$

$$e_t(s, a) = \gamma \lambda e_{t-1}(s, a) + \nabla_{Q_t(s,a)} Q_t(b_t, a) = \gamma \lambda e_{t-1}(s, a) + b_t(s) \qquad \forall s \tag{5}$$

in which $e_0(s, t) = 0, \forall s$. $0 < \lambda < 1$ is the trace-decay parameter. $0 < \alpha < 1$ is the learning step. $0 < \gamma < 1$ is the discount factor. Theoretically, in (5) $b_t(s)$ can be regarded as coming from derivative of approximated value function on parameters. A mechanistic view is that (5) is a direct application of $Sarsa(\lambda)$ to the belief space with Q-value approximated by (2). All states here are eligible for credit (or blame) which is assigned proportional to the belief on corresponding states.

Our algorithm runs at simulation and execution phase sequentially. In simulation phase, we have complete observability and inexact model of the real-world. The belief-state is the unit vector corresponding to the known current state. State-transition and observation-function are estimated on-line during simulation. After the MDP-like simulation phase, we have not only identified an explicit model (for the simulator) but also obtained the policy for the simulator which can be used as a jump-starter to supply the next phase. In the execution phase where the state cannot be observed completely, the belief is updated by previously estimated model and policy is thus fine-tuned (as shown in Figure 1). A pseudo-code procedure is reported in [9].

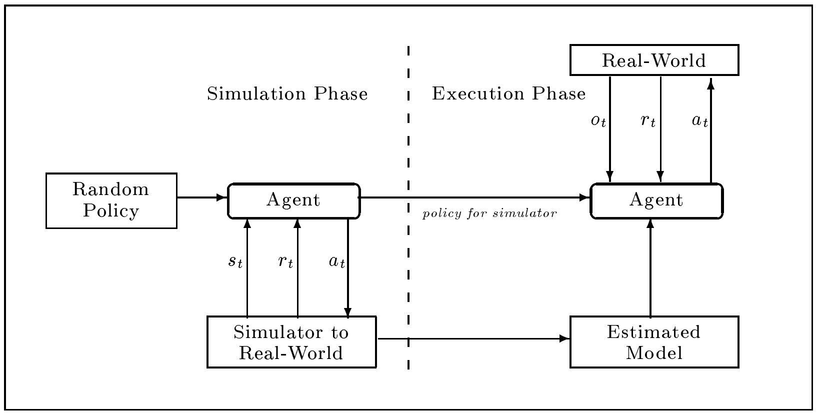

Figure 1: Two-phase Learning Algorithm

3.2 Performance

In order to measure the performances, we build a tester which can execute greedy policies derived from both exact algorithm and our algorithm referred as “Fast-RL”. Average-discounted-reward (ADR) and steps-to-goal (STG) are two metrics to measure Fast-RL vs. the exact algorithm. The policy from exact algorithm is fixed while Fast-RL policies are learned before each test.

To calculate ADR, successive policies are tested and rewards are discounted, added and averaged accordingly. Each test starts from an arbitrary belief state with a given policy, discounted reward is added for each step until the goal-state or the maximum steps limit is reached. Test repeats itself for the number of trials. Steps are added among all the trials. The ADR and STG are represented in the form of (mean $\pm$ sample-standard-deviation) among all tested policies.

3.2.1 Benchmark Examples

Following examples are selected from POMDP literature as test benchmarks, e.g., 4x3 ([21]), Cheese ([16]), Milos ([8]), Two-layer, Hallway-1-2 ([4]) (as shown in Table 1). We choose currently the most effective exact algorithm ([26], [5]). Policy is optimized for the given horizon and epsilon which controls the convergence rate. Other parameters are chosen as default. The number of solution vectors is obtained after time given at its right column (in seconds).

Benchmarks	$\mid \mathcal{S} \mid$	$\mid \mathcal{A} \mid$	$\mid \mathcal{O} \mid$	Horizon	Epsilon	Vector	Time
4x3	11	4	6	7	10^{-9}	436	29.9
Cheese	11	4	7	100	10^{-9}	14	5.6
Milos	20	6	8	5	0.1	157	11157
Two-layer	39	4	4	10	10^{-9}	1414	358
Hallway-1	60	5	21	5	0.004	1944	3010
Hallway-2	92	5	17	4	0.01	1897	5211

Table 1: Benchmark examples from POMDP literature

For Fast-RL, although learning parameters can be optimized for each individual case, they are intentionally not chosen to do so. In the following test, goal-state is absorbing which means test will terminate either after the goal-state or the "test-step" is reached. For the sake of simplification, the max-step in execution phase is chosen to be the same as max-step in simulation phase, thus denoted as "RL-step". Time until the approximate policy obtained is denoted as "RL-Time", which covers both simulation and execution phase. Trial of execution phase is much smaller than the episode of simulation phases. Learning rate α=0.01, exploration rate ϵ=0.1 and is decayed by the factor of 0.99 each step. Eligibility trace decay factor λ=0.9. ADT and STG are also obtained for random policies for the purpose of providing the performance baselines.

4x3	ADR	STG
Exact	365.2±8.1	4952±64
Fast-RL	364.6±15.0	5004±76
Random	153.5±97.9	8543±3895

Table 2: RL-step=100 Episode=100, Test-step=100, RL-Time=17 Sec.

Cheese	ADR	STG
Exact	78.3±0.0	580±0
Fast-RL	76.3±3.4	642±121
Random	11.0±6.8	8828±725

Table 3: RL-step=100 Episode=100, Test-step=100, RL-Time=18 Sec.

Milos	ADR	STG
Exact	2359±15	27900±881
Fast-RL	4041±623	34564±1201
Random	2037±1752	34685±2249

Table 4: RL-step=200 Episode=100, Test-step=300, RL-Time=313 Sec.

Two-layer	ADR	STG
Exact	255.4±0	3510±0
Fast-RL	221.7±16.7	4803±766
Random	125±57.9	35691±1793

Table 5: RL-step=100 Episode=100, Test-step=100, RL-Time=182 Sec.

Hallway-1	ADR	STG
Exact	205.8±5.2	19648±830
Fast-RL	201.1±45.9	17355±8118
Random	13.1±6.8	25937±1111

Table 6: RL-step=100 Episode=100, Test-step=100, RL-Time=303 Sec.

Hallway-2	ADR	STG
Exact	172±3.4	52674±458
Fast-RL	252±10.1	32980±2950
Random	70.9±27.9	288119±42884

Table 7: RL-step=200 Episode=100, Test-step=500, RL-Time=1620 Sec.

As shown in the above tables, the performance metrics of fast-RL is close to or in some case even superior to that of exact algorithm. The later case is probably due to early termination of exact algorithm when no progress is made within 3 hours. In general, the standard deviation of policies from fast-RL is larger than that of exact algorithm. This implies that the exact algorithm is preferred in the sense of stability. The deterministic model (e.g. Cheese) has no uncertainty in action and sensor, therefore policy from exact algorithm performs consistently well while learning algorithm does not.

3.2.2 Model Inaccuracy

In the previous section, we consider the case where the model of the simulator, i.e. state-transition matrix, observation-function etc., is exactly the same as that of the real-world. In our algorithm, policies learned in the simulator will be brought into the real-world and will be fine-tuned during the policy is executed. However, the policy from the exact solutions will not tolerate model inaccuracy. On the other hand, RL algorithm during execution phase will adjust the policy while making observations and penalizing "bad" actions that incur large costs. Following is an example from previous section to show how Fast-RL outperforms exact solution when the model is wrong. Suppose state-5 in 4x3-example in simulator is actually blocked in the real-world which means actions leading to state-5 will be bouncing back to the original state, e.g., in state-9 taking action-up will stay at the same place with probability 1.

4x3	ADR	STG
Exact	109±5.87	11383±203
Fast-RL	192.5±43	8248±2690

Table 8: Performance metrics when state-5 in 4x3-examples is blocked

What makes RL algorithm more significant is that a real-world problem, like network management, is too complex to have pre-defined models. The tolerance of model inaccuracy is an important aspect to measure the robustness of control algorithm.

3.2.3 On-Policy vs. Off-Policy Algorithm

In this section, we compare briefly the performance issue of Fast-RL vs. off-policy linear Q-learning in ([14]). Using linear Q-learning, two navigation problems with 57 and 89 states respectively only have 8.4% and 5.2% trials reach goals within 251 steps. Since these two problems in ([14]) are model-based, we skip the simulation phase and assume the model is known to make a fair comparison. These two cases are tried by Fast-RL and found 99.6% and 99.1% trials reach the goals within 251 steps.

Hallway-1	ADR	STG
Hallway-1	199.27±48.0	16611±7651
Hallway-2	219.38±23.7	46146±13654

Table 9: Performances metrics when model is known

4 Conlusion

Some experimental result ([15], [13]) shed the light that on-policy TD learning (e.g. $Sarsa(\lambda)$) works better when eligibility trace is employed. However, the general convergence property has not been proved even for the simplest table-lookup cases. This paper does not focus on the convergence property, in stead it finds a quick solution to POMDP with satisfactory performances. Our features and parameters selection is practically most efficient and archiving near-optimality quickly in all benchmark examples. Although replacing trace outperform accumulating trace in many table-lookup cases ([18]), the later one is more conceptually natural in our case. This observation complements the claims in ([22]) which primarily consider the binary features.

5 Acknowledgement

We would like to thank for Anthony Cassandra for helping us port pomdp-solver. Our benchmark examples are available from his POMDP site ([5]).

References

[1] Baird, Leemon, (1995), Residual Algorithms: Reinforcement Learning with Function Approximation. *Machine Learning*: Proceedings of the Twelfth International Conference, 9-12 July, Morgan Kaufman Publishers, San Francisco, CA.

[2] Bertsekas, D. P., (1995), *Dynamic Programming and Optimal Control.* Athena Scientific.

[3] Boyan, J. A. and Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value functions. *Advances in Neural Information Processing Systems: Proceedings of the 1994 Conference*, pages 369-376.

[4] Cassandra, A. R., (1998), *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* Ph.D. Thesis. Brown University, Department of Computer Science.

[5] Cassandra A. R (1999), *pomdp-solver 4.0* http://www.cs.brown.edu /research/ai/pomdp/index.html

[6] Geffner H., Bonet B. (1998), Solving Large POMDPs by Real Time Dynamic Programming, *Working Notes Fall AAAI Symposium on POMDPS.*

[7] Hansen E. A. (1998), Solving POMDPs by Searching in Policy Space, *Proceedings of the 14th International Conference on Uncertainty in Artificial Intelligence.*

[8] Hauskrecht Milos, Fraser H. (1998), Modeling Treatment of Ischemic Heart Disease with Partially Observable Markov Decision Processes, In *Proceedings of American Medical Informatics Association annual symposium on Computer Applications in Health Care*, Orlando, Florida, pp.538-542.

[9] He Qiming, Shayman M. A., (1999), Using Reinforcement Learning for Proactive Network Fault Management, *ISR Technical Report*, UMCP.

[10] HP Solution Note (1997), Testing Operation and Maintenance Implementation for ATM *1997 ATM/Broadband Testing seminar*, Hewlett-Packard Company.

[11] Kaelbling L. P., Littman M. L. Moore, A. W., (1996), Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research*, Volume 4.

[12] Kaelbling L. P., Littman M. L., Cassandra, A. R. (1998), Planning and Acting in Partially Observable Stochastic Domains, *Artificial Intelligence*, Vol. 101.

[13] Kimura, H., Miyazaki, K., and Kobayashi, S. (1997), Reinforcement Learning in POMDPs with Function Approximation, *Proceedings of the 14th International Conference on Machine Learning*, pp.152–160.

[14] Littman, M. L., Cassandra, A. R., Kaelbling L. P. (1995), Learning Policies for Partially Observable Environments: Scaling Up, *Proceedings of the Twelfth International Conference on Machine Learning*, p 362–370.

[15] Loch J, Singh S. (1998), Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes To appear in *ICML98.*

[16] McCallum A. K. (1996) *Reinforcement Learning with selective Percepttion and Hidden State*, Ph.D. thesis, University of Rochester.

[17] Rummery, G. A. and Niranjan, M. (1994), On-line Q-learning Using Connectionist Systems, *Technical Report CUED/F-INFENG/TR 166.* Cambridge University Engineering Department.

[18] Singh S., Sutton R. (1996), Reinforcement Learning with Replacing Eligibility Traces, *Machine Learning*, vol. 22, pp.123-158.

[19] Singh S., Jaakkola T., Littman M. L. and Szepesvri C. (1998), Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Machine Learning*, to appear.

[20] Sondik E. J. (1971), *The Optimal Control of Partially Observable Markov Processes*, Ph.D. Thesis. Stanford University.

[21] Stuart J. Russell , Peter Norvig. (1994), *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, New Jersey.

[22] Sutton R. S., Barto A. G. (1998), *Reinforcement Learning: An introduction* MIT Press, 1998.

[23] Sylvie Thibaux, et al. (1996), Supply Restoration in Power Distribution Systems: A Case Study in Integrating Model-Based Diagnosis and Repair Planning, *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, p525-532.

[24] Tsitsiklis J. N. and Roy B. V., (1997), An Analysis of Temporal-Difference Learning with Function Approximation, *IEEE Transactions on Automatic Control*, Vol. 42, No. 5, May 1997, pp. 674-690.

[25] Watkins C. J. C. H., Dayan P. (1992), Technical note: Q-learning, *Machine Learning*, 8(3/4) p.272-292.

[26] Zhang N., Liu W., (1997), A Model Approximate Scheme for Planning in Partially Observable Stochastic Domains, *Journal of Artificial Intelligence*, 7, p.199-230.