Diffusion Gradient Temporal Difference for Cooperative Reinforcement Learning with Linear Function Approximation

Sergio Valcarcel Macua, Pavle Belanovic, Santiago Zazo

***Abstract*—We introduce a diffusion-based algorithm in which multiple agents cooperate to predict a common and global state-value function by sharing local estimates and local gradient information among neighbors. Our algorithm is a fully distributed implementation of the gradient temporal difference with linear function approximation, to make it applicable to multi-agent settings. Simulations illustrate the benefit of cooperation in learning, as made possible by the proposed algorithm.**

***Index Terms*—TD, distributed reinforcement learning, distributed control, cooperative learning, multi-agent, distributed decision making, distributed temporal difference**

I. Introduction

Diffusion stochastic gradient descent algorithms have had much success implementing distributed estimation and learning problems [1], [2], in particular in simulated bio-inspired agents [3], [4]. Although many variations of these diffusion adaptation algorithms exist, very recently they have been unified as a general tool for distributed optimization of a sum of local convex cost functions [5] by Chen and Sayed.

A common approach to achieve multi-agent systems that exhibit self organized patterns (like those in [3], [4]) is to give them some pre-designed behavior. Although these hard-wire strategies are reasonable (e.g. behavioral genetics), we believe that much improvement would be achievable if some optimal policy learning mechanism is embedded in the agents as well. Thus multi-agent adaptive systems must learn to make decisions taking into account accumulated future expected rewards that will depend, not only on their individual actions, but also on the interactions and decisions made by other agents of the system.

There are many domains of application for multi-agent adaptive systems, ranging from sensor management to teams of robots (e.g. control the power grid or any other distribution network, vehicular networks, cognitive radio, network routing, collaborative SLAM...). In many real applications, even with the assumption that there is a Markov decision process (MDP) underlying the system, usually the agents do not know a dynamic model of the environment a priory and have to deal with large number of states. Moreover they typically require low complexity in order to save battery and be embedded in myriads of low cost devices. Reinforcement learning [6] is a tool well suited for this purpose.

Gradient temporal difference (GTD) algorithms are a breakthrough in reinforcement learning showing convergence for off-policy learning with linear [7] and non-linear [8] function approximation, with performance comparable to temporal difference (TD) methods [9], and with eligibility traces [10]. In particular we are going to focus on the so-called GTD version 2 algorithm (or simply GTD2) [9], though extension to other variations should be straightforward. This algorithm combines off-policy updates, TD learning, linear function approximation, linear complexity in memory and per-time-step computations, speed of convergence similar to standard TD, and guaranteed stability.

Although this family of GTD algorithms has also been extended to finding the optimal policy (*control problem*) [11], in this paper we will only consider the problem of estimating the state value function underlying some MDP (*prediction problem*), leaving the cooperative control extension for future work.

A. Related work

Distributed reinforcement learning is an important topic and some solutions have been proposed. However they lack for a combination of features (namely low complexity, off-policy learning, function approximation and stability) that could make them suitable for many real applications. Comprehensive surveys on the topic are [12], [13]. Here we just mention two approaches related to the one we propose.

Authors in [14] introduced the concept of distributed value functions for Dynamic Programming (DP), in which every node exchanges their local estimate so they can be weighted in the update. Here we move from DP to TD, in particular to GTD2. Although we similarly propose to combine some intermediate estimates of the nodes in the neighborhood for the GTD2, our approach comes from the distributed optimization literature. Indeed we include a double combination of gradient information and estimated value function. This way we can benefit from all the advantages that GTD2 offer over DP and,

This work was supported in part by the Spanish Ministry of Science and Innovation under the grant TEC2009-14219-C03-01; the Spanish Ministry of Science and Innovation in the program CONSOLIDER-INGENIO 2010 under the grant CSD2008-00010 COMONSENS; the European Commission under the grant FP7-ICT-2009-4-248894-WHERE-2; the Spanish Ministry of Science and Innovation under the complementary action grant TEC 2008-04644-E; and the Spanish Ministry of Science and Innovation under the grant TEC2010-21217-C02-02-CR4HFDVL.

at the same time, get improvement over the non-cooperative solution. In addition, the combination coefficients that we use here come from gossip and consensus averaging algorithms, thus they are well grounded (see e.g. [15]–[17]).

In [18] consensus averaging was introduced, for averaging experience and reward, into the learning process of a gradient search in the policy space. This approach is somehow related to ours since this kind of agreement can be seen as a particular case of the more general framework of distributed-gradient optimization [19]. Nevertheless solving the optimization problem in a really distributed manner has some advantages. For instance, it does not require perfect agreement over the entire network of agents, in each iteration, being more natural for many real models, such as the bio-inspired ones. Moreover, that algorithm was developed for problems with small state space (or at least small enough to be represented by a look-up table), which might narrow its applicability in some real environments.

B. Contributions

The central contribution presented here is a novel, fully distributed reinforcement learning algorithm that allows multiple agents to cooperate with each other in order to find the global optimal representation of a common state value function. No fusion-center is required, just local computation and one-hop communication among neighbors are enough for asymptotic convergence to the same optimal solution that a fusion-center would achieve.

It is based on the well-known (but centralized) GTD2 algorithm, which we distribute using the powerful diffusion adaptation methods. We illustrate the performance gain that individual agents achieve through coordination via simulations of several classical reinforcement learning toy problems.

Doing so we move from estimation to control settings, getting one step closer to a fully autonomous and self-organized multi-agent system. In particular we study a prediction problem which is an important part of the more general control problem.

C. Outline

This paper is structured as follows. In Section II we define the problem, summarize the diffusion adaptation scheme for optimization and present the centralized GTD2 algorithm. Then, in Section III, we present our Cooperative GTD2 (C-GTD2) algorithm. It is followed by three sets of simulation results in Section IV showing the benefit of cooperative learning. Finally, we draw some conclusions in Section V.

II. Background

A. Distributed optimization problem

Consider a set of N agents with some communication constraints (e.g. distance, congestion, environment, etc.). Their interconnection defines a graph with links that can vary with time (e.g. because of mobility, link failures, etc.). We define the neighborhood of a node k as the set of nodes that can communicate with node k, included the node k itself, and denote it $\mathcal{N}_k$.

Now suppose that every agent k seeks to estimate a *global* minimizer θ^o of some *global* cost function $J^{\text{glob}}(\theta)$, which is defined as the sum of all the *local* cost functions, $J_k(\theta)$, as

$$J^{\text{glob}}(\theta) = \sum_{k=1}^{N} J_k(\theta) \tag{1}$$

In other words, the agents are trying to learn a common and optimal parameter vector θ^o, such that

$$\theta^o = \arg\min_{\theta} J^{\text{glob}}(\theta) \tag{2}$$

B. Diffusion adaptation for distributed optimization

As explained in [5], the global cost function $J^{\text{glob}}(\theta)$ can be efficiently optimized in a distributed fashion making some approximations in the local cost functions. In particular, every node k in the network aims to minimize its local estimate of the global cost function, $J_k^{\text{glob}}(\theta)$, defined by

$$J_k^{\text{glob}}(\theta) = \sum_{l\in\mathcal{N}_k} c_{l,k} J_l(\theta) + \sum_{l\in\mathcal{N}_k, l\neq k} b_{l,k}\|\theta - \theta^o\|^2 \tag{3}$$

where $c_{l,k}$ and $b_{l,k}$ are weight coefficients used by node k to combine information diffused by its neighbors. Note that θ^o is unknown to any node so it will be approximated by its best local estimate (i.e. the one obtained in the last iteration).

Every node can cooperatively minimize (3) applying a steepest-descent approach, which leads to a 2-steps iterative algorithm named Adapt-Then-Combine (ATC).

In the first step (4) of ATC every node k shares gradient information with its neighbors. This incoming information is applied to its own local estimate, $\theta_{k,i}$, in order to generate an updated intermediate estimate, $\psi_{k,i+1}$, as

$$\psi_{k,i+1} = \theta_{k,i} + \alpha_k \sum_{l\in\mathcal{N}_k} c_{l,k} \nabla J_l(\theta_{k,i}) \tag{4}$$

where α_k is a sufficiently small constant step size that assures convergence to a fixed value in the mean square error (MSE) sense.

In the second step (5) every node k updates its own estimate, $\theta_{k,i+1}$, as a convex combination of the intermediate estimates in its neighborhood

$$\theta_{k,i+1} = \sum_{l\in\mathcal{N}_k} a_{l,k} \psi_{l,i+1} \tag{5}$$

Note that we do not have to worry about $b_{l,k}$ anymore since they have been mixed into a new set of weights $a_{l,k}$.

The combination coefficients for every pair of nodes in the network, $a_{l,k}$ and $c_{l,k}$, form two matrices A and C, being stochastic and doubly stochastic respectively, and with zero entries wherever nodes k and l are not neighbors. These stochastic matrices can be computed locally in different ways, leading to different performance in the speed of how information diffuses through the network (see e.g. [15]–[17]).

For a thorough explanation of diffusion adaptive algorithms see [1], [5], [20], [21] and the references therein.

C. Value prediction problem

We assume the agents live in an environment which can be modeled as a MDP defined by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, the transition probabilities of going from one state s_t to another s_{t+1} given an action a_t, and the rewards r_{t+1} associated to those transitions.

The state value function $V^\pi : \mathcal{S} \to \mathbb{R}$ gives the expected cumulative discounted return of following a policy π, when the agent is in state s. It is defined by

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s\right], \; s \in \mathcal{S} \tag{6}$$

where the rewards r_{t+1} come from the state transitions, from s_t to s_{t+1}, induced by actions taken according to policy π. And $\gamma \in [0, 1]$ is a discount factor meaning that, though future expected rewards are taken into account, the farther they are in the future, the less they weight in the value function.

In order to choose among different policies the agents should be able to evaluate how good those policies are, i.e. to predict their value.

D. Linear approximation of value functions

In many problems of interest the number of states is too big for us to estimate each one individually. One possibility is to learn an approximate state value function which depends on many fewer parameters than the number of states. In particular we consider linear approximations of the form

$$V(s) \approx V_\theta(\phi(s)) = \theta^\mathsf{T} \phi(s) \tag{7}$$

where $V(s)$ is the real state value function for the state $s \in \mathcal{S}$, $V_\theta(\phi(s))$ is its linear approximation, $\phi(s) \in \mathbb{R}^n$ is the feature vector that represents the state, with $n \ll |\mathcal{S}|$, and θ is the vector of parameters to be learned. Since we are dealing just with prediction, instead of control, we have omitted any specific policy in the notation. Moreover let $\Phi \in \mathbb{R}^{|\mathcal{S}| \times n}$ be the matrix with rows $\phi(s)^\mathsf{T}$, then we denote $V_\theta \in \mathbb{R}^{|\mathcal{S}|}$ the vector of approximated values for each state, such that $V_\theta = \Phi\theta$.

TD(λ) is a powerful family of algorithms that iteratively refine the estimate of the value function while the agent is interacting with the environment. The tunable parameter $\lambda \in [0, 1]$ trades bias and variance of the predicted value.

Although TD(λ) is one of the most celebrated ideas in reinforcement learning, when it is combined with linear function approximation (named linear-TD(λ)) and off-policy learning, stability of the solution is not guaranteed. Nevertheless, if linear-TD(0) converges, it is shown in [22] that the solution satisfies $V_\theta = \Pi T V_\theta$, where T is a contraction mapping (the Bellman operator), and Π is a projection operator that takes any value function and projects it to the nearest value function representable by the function approximation.

Therefore a natural objective function that can be optimized is the mean-square projected Bellman-error (MSPBE) [9]

$$J(\theta) = \|V_\theta - \Pi T V_\theta\|_\mu^2 \tag{8}$$

where μ is the state-visitation probability-distribution vector whose components represent the probability of visiting each state, and being $\|v\|_\mu^2 = \sum_s \mu(s) v^2(s)$.

E. Gradient Temporal Difference learning: GTD2

The GTD2 algorithm aims to minimize the MSPBE. It uses the set of triples (ϕ, r, ϕ') (i.e. feature vector of the current state, the transition reward and the feature vector of next state respectively) as data, which we assume drawn i.i.d. from μ.

The MSPBE can be also expressed as

$$J(\theta) = \mathbb{E}[\delta(\theta)\phi]^\mathsf{T} \mathbb{E}[\phi\phi^\mathsf{T}]^{-1} \mathbb{E}[\delta(\theta)\phi] \tag{9}$$

where $\delta(\theta) = r + \gamma\theta^\mathsf{T}\phi' - \theta^\mathsf{T}\phi$ (i.e. the standard TD(0) iteration update). Hence the direction opposite to the gradient of the MSPBE is

$$-\frac{1}{2}\nabla J(\theta) = \mathbb{E}\left[(\phi - \gamma\phi')\phi^\mathsf{T}\right] \mathbb{E}\left[\phi\phi^\mathsf{T}\right]^{-1} \mathbb{E}\left[\delta(\theta)\phi\right] \tag{10}$$

One problem that arises when trying to apply gradient descent directly on (10) is that we can not sample each of the three expected values, since the samples would be correlated and their product would be biased. The solution proposed in [7] is to sample only one of the expectations while tracking a long-term, quasi-stationary estimate of the others. In particular [9] noticed that $w(\theta) = \mathbb{E}[\phi\phi^\mathsf{T}]^{-1}\mathbb{E}[\delta(\theta)\phi]$ is the linear predictor obtained with the Least Mean Square (LMS) algorithm [23]. Therefore (10) can be rewritten as

$$-\frac{1}{2}\nabla J(\theta) = \mathbb{E}\left[(\phi - \gamma\phi')\phi^\mathsf{T}\right] w(\theta) \tag{11}$$

which directly leads to the two iterations of the GTD2 algorithm

$$\theta_{i+1} = \theta_i + \alpha_i(\phi_i - \gamma\phi_i')(\phi_i^\mathsf{T} w_i) \tag{12}$$

$$w_{i+1} = w_i + \beta_i(\delta_i(\theta_i) - \phi_i^\mathsf{T} w_i)\phi_i \tag{13}$$

Following [9], we can aggregate (12) and (13) in a single iteration

$$\rho_{i+1} = \rho_i + \alpha_i(G_{i+1}\rho_i + g_{i+1}) \tag{14}$$

where $\rho_i^\mathsf{T} = [w_i^\mathsf{T},\ \theta_i^\mathsf{T}]$ and $g_{i+1}^\mathsf{T} = [\eta r_i \phi_i^\mathsf{T},\ 0^\mathsf{T}]$ are the aggregated parameter and aggregated reward vector, respectively, the coefficient matrix[1] is

$$G_{i+1} = \begin{pmatrix} -\eta\,\phi_i\phi_i^\mathsf{T} & \eta\phi_i(\gamma\phi_i' - \phi_i)^\mathsf{T} \\ -(\gamma\phi_i' - \phi_i)\phi_i^\mathsf{T} & 0 \end{pmatrix} \tag{15}$$

and $\eta = \beta_i/\alpha_i > 0$ is the ratio between step-sizes.

For a complete presentation of gradient temporal difference learning see [24].

[1]Note that we are slightly abusing the notation since 0 denotes both a column vector and block matrix of zeros.

III. COOPERATIVE GTD2

Consider a system where each agent collects its own experience and reward. In this section we derive an algorithm that minimizes (1), with common minimizer θ, and with local cost functions being the local MSPBE of each node

$$J_k(\theta) = \|V_{\theta,k} - \Pi T V_{\theta,k}\|^2_{\mu_k} \tag{16}$$

where $V_{\theta,k}$ is the local estimate (at node k) of V_θ.

The algorithm we propose consists of substituting (14) by a distributed implementation using ATC, which leads to

$$\begin{aligned} \psi_{k,i+1} &= \rho_{k,i} + \alpha_{k,i} \sum_{l \in \mathcal{N}_k} c_{l,k}(G_{l,i+1}\rho_{k,i} + g_{l,i+1}) \\ \rho_{k,i+1} &= \sum_{l \in \mathcal{N}_k} a_{l,k}\psi_{l,i+1} \end{aligned} \tag{17}$$

Note that, though using some information from its neighbors, every node computes the update over its local estimate.

In order to obtain a more natural and easier to implement form, let's disaggregate (17) into the two iterations of GTD2. To do so we first dissagregate the iterative update as

$$G_{l,i+1}\rho_{k,i} + g_{l,i+1} = \begin{pmatrix} \eta(\delta_{l,i}(\theta_{k,i}) - \phi_{l,i}^\mathsf{T} w_{k,i})\phi_{l,i} \\ \phi_{l,i} - \gamma\phi'_{l,i}\phi_{l,i}^\mathsf{T} w_{k,i} \end{pmatrix} \tag{18}$$

where $\delta_{l,i}(\theta_{k,i}) = r_{l,i} + \gamma\theta_{k,i}^\mathsf{T}\phi'_{l,i} - \theta_{k,i}^\mathsf{T}\phi_{l,i}$ could be understood as the local TD error of node l, evaluated with the last estimate of node k, at iteration i.

Therefore the locally sampled expected value of the gradient of the MSPBE is cooperatively estimated as

$$\begin{aligned} \sigma_{k,i+1} &= \theta_{k,i} + \alpha_{k,i} \sum_{l \in \mathcal{N}_k} c_{l,k}(\phi_{l,i} - \gamma\phi'_{l,i})(\phi_{l,i}^\mathsf{T} w_{k,i}) \\ \theta_{k,i+1} &= \sum_{l \in \mathcal{N}_k} a_{l,k}\sigma_{l,i+1} \end{aligned} \tag{19}$$

And the local long-term estimate of the global LMS solution results in

$$\begin{aligned} \nu_{k,i+1} &= w_{k,i} + \beta_{k,i} \sum_{l \in \mathcal{N}_k} c_{l,k}(\delta_{l,i}(\theta_{k,i}) - \phi_{l,i}^\mathsf{T} w_{k,i})\phi_{l,i} \\ w_{k,i+1} &= \sum_{l \in \mathcal{N}_k} a_{l,k}\nu_{l,i+1} \end{aligned} \tag{20}$$

Equations (19) and (20) can be seen as the equivalent distributed implementation of (12) and (13). Thus they constitute each iteration of the C-GTD2 algorithm.

IV. SIMULATIONS

In this section we show some experiments with the proposed C-CTD2 algorithm in order to compare its performance with the (non-cooperative) GTD2 in terms of both MSPBE and stability.

A. Problems description

We take some problems from the benchmark presented in [9], namely three versions of a random-walk problem (with different features each), the Boyan-chain problem and the Baird's counterexample.

The random-walk is a standard Markov chain. It includes 5 states, plus 2 more absorbing terminal states, one at each end. The reward is zero in every transition except when ending in the terminal state at the right end. The initial state is in the middle of the walk. We try three different representations of the problem. The first two sets of features, namely *tabular* and *inverted* are represented with a feature vector with same number of entries as the number of states; while the third set of features, named *dependent*, uses only 3 features for the five states.

The Boyan chain has 14 states. There is only one absorbing terminal state, numbered 0 at the right end. State 13, at the left end, is the initial one. States 13 to 1 can take two actions: either go to the next in the right, or jump one position moving two states to the right. State 1 can take only one action which is to go to the terminal state.

The last problem is the Baird's counterexample, a "star" of 7 states. The reward is zero for every transition. Every state can take two actions, one that leads to any other state, with equal probability, or another that goes to state 7. The goal is to learn a target policy different from the behaviour policy. In particular the behaviour policy makes every state to evolve to state 7.

Although different for each problem, we have used the same parameters α and η for both the non-cooperative and cooperative cases (i.e. $\alpha_k = \alpha$ and $\eta_k = \eta$). The distributed setting consists of a network of 10 nodes, with average degree 5 and random topology. Finally, the MSPBE is computed exactly using the optimal parameter vector learned by the algorithms, and averaged over 10 independent runs.

B. Results

Our algorithm shows a clear improvement in all versions of the random walk, as well as in the Boyan chain. All nodes improve results with respect to the non-cooperative case. Diffusion alleviates the gradient noise, leading to a lower minimum of the MSPBE and to much less noisy and with less variance estimates (see Figures 1 and 2, note that we plot the root MSPBE).

The Baird's counterexample is a classical test that shows how TD(0) can diverge when it is combined with linear approximation. In this case we compare C-GTD2 with a diffusion-based distributed implementation of the linear-TD(0) (which we call C-LTD(0)), and show similar results as when comparing with their non-cooperative versions: while C-LTD(0) diverges, C-GTD2 converges to zero (see Figure 3).

V. CONCLUSIONS

In this paper we presented a cooperative version of the GTD2 algorithm for coordinated learning in a multi-agent setting. Our algorithm conserves all the desirable properties of

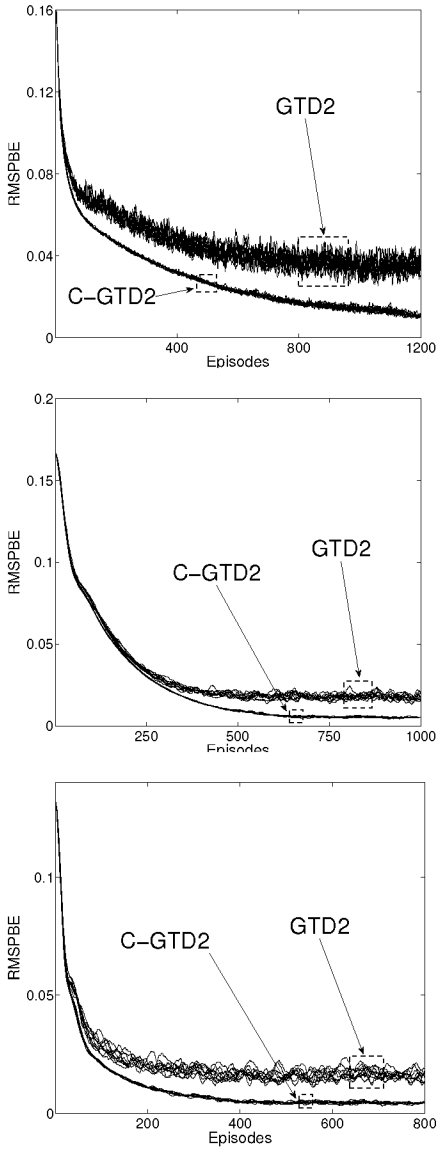

Fig. 1. Random walk for (left) *tabular*, (middle) *inverted*, and (right) *dependent* features. We use the same standard Markov chain with the linear arrangement shown in [9]. Tabular features are similar to a look-up table, in which every state is represented by a vector with every entry equal zero except the one of the state (e.g. $\phi_2 = [0, 1, 0, 0, 0]^\top$). Inverted features are still represented with the same number of features as number of states, but opposite to the tabular case; only the entry corresponding with the actual state is zero, while the rest are equal and normalized to make the vector unitary (e.g. the second state is represented as $\phi_2 = [\frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}]^\top$). Finally, dependent features represent the state with a vector of only 3 entries, namely $\phi_1 = [1, 0, 0]^\top$, $\phi_2 = [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0]^\top$, $\phi_3 = [\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{2}}]^\top$, $\phi_4 = [0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]^\top$ and $\phi_5 = [0, 0, 1]^\top$. In every case, the terminal states are represented by all zeros, as usual. We choose constant step-sizes $\alpha = 0.06$ and $\eta = 2$. It is clear that the curves corresponding to the local estimates obtained with C-GTD2 show considerable less MSPBE, are much smoother and have much less variance than those obtained with the non-cooperative version (GTD2).

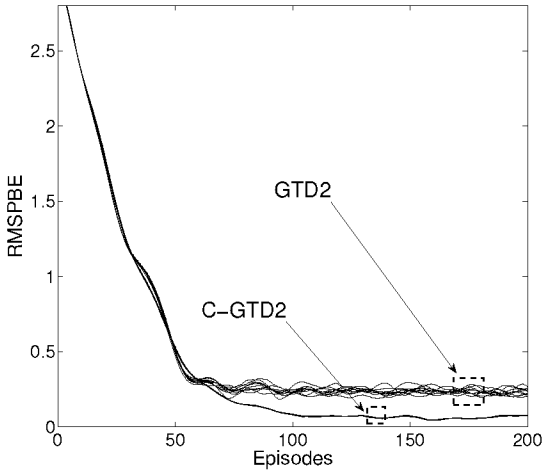

Fig. 2. Boyan chain. The Boyan chain is a Markov chain. We choose the same version as [9], with 14 states. States are represented with only 4 features, named *interpolated*, defined as follows: states 13, 9, 5, and 1 are defined as $\phi_{13} = [1, 0, 0, 0]^\top$, $\phi_9 = [0, 1, 0, 0]^\top$, $\phi_5 = [0, 0, 1, 0]^\top$ and $\phi_1 = [0, 0, 0, 1]^\top$ respectively; and the others are obtained linearly interpolating between these (i.e. $\phi_2 = [0, 0, 1/4, 3/4]^\top$, $\phi_3 = [0, 0, 1/2, 1/2]^\top$, $\phi_4 = [0, 0, 3/4, 1/4]^\top$, and so on). We choose constant step-sizes $\alpha = 0.5$ and $\eta = \frac{1}{8}$. Similar to the results of the Random Walk (see Figure 1), C-GTD2 achieves less error and the estimates are smoother and with less variance than those of the non-cooperative version (GTD2).

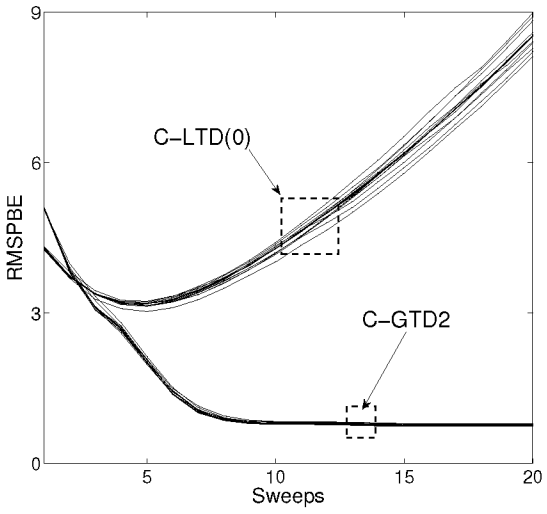

Fig. 3. Baird's counterexample. We use the same 7-state version of the problem shown in [9]. The target policy sets a probability of taking action of going to any other state is $6/7$, and of going to state 7 is $1/7$. While the behaviour policy is going to state 7 with probability 0.97. Every state is represented with 8 features as follows: for states 1 to 6 there is 2 in the same component as state, and 1 in component 8 (e.g. $\phi_3 = [0, 0, 2, 0, 0, 0, 0, 1]^\top$), but for state 7 we have $\phi_7 = [0, 0, 0, 0, 0, 0, 1, 2]^\top$. Estimates are updated in sweeps through the state-space, very much like in dynamic programming. Parameters are $\gamma = 0.99, \alpha = 0.05$ and $\eta = 5$. For C-LTD(0) convergence is not guaranteed, while C-GTD2 shows to be stable.

its well-known centralized counterpart, but affords the agents an added benefit in performance derived through cooperation. This has been demonstrated practically on three classical reinforcement learning problems.

Future work includes investigating the possibility to extend this approach to other gradient based TD algorithms like GQ(λ) [10]. Also, the extension to the policy improvement (control) problem (such as greedy-GQ [11]) is also possible. Moreover this same methodology of combining diffusion strategies with TD learning can be applied to Least Squares TD (LSTD) and its variations [25], [26].

Finally it is worth to mention that performance and convergence analysis are still ongoing work. This is because the MSE analysis derived for diffusion optimization does not directly apply here. In fact the aggregated equation of C-GTD2 (17) is not truly a gradient, but rather a diffusion stochastic approximation. Nevertheless results with the Baird's counter example seem to be very promising (see Section IV).

ACKNOWLEDGMENT

The authors would like to thank to Hamid Reza Maei for his kind support in reproducing the centralized results showed in [24], including explanations and snippets of code to understand how the MSPBE was computed. We would also like to thank Jianshu Chen and Ali H. Sayed for their kind help with the details and analysis of the ATC algorithm. Finally we want to thank Jose Ignacio Ronda for his insightful discussions.

REFERENCES

[1] F. S. Cattivelli and A. H. Sayed, "Diffusion LMS Strategies for Distributed Estimation," *IEEE Transactions on Signal Processing*, vol. 58, no. 3, pp. 1035–1048, 2010.

[2] Z. J. Towfic, Jianshu Chen, and A. H. Sayed, "Collaborative Learning of Mixture Models Using Diffusion Adaptation," in *International Workshop on Machine Learning for Signal Processing (MLSP)*, 2011.

[3] F. S. Cattivelli and A. H. Sayed, "Self-Organization in Bird Flight Formations Using Diffusion Adaptation," in *International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, 2009, pp. 49–52.

[4] Sheng-Yuan Tu and A. H. Sayed, "Cooperative Prey Herding Based on Diffusion Adaptation," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 3752–3755.

[5] Jianshu Chen, Sheng-Yuan Tu, and A.H. Sayed, "Distributed optimization via diffusion adaptation," in *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2011 4th IEEE International Workshop on*, dec. 2011, pp. 281 –284.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Adaptive computation and machine learning. MIT Press, 1998.

[7] R. S. Sutton, C. Szepesvari, and H. R. Maei, "A Convergent O(n) Algorithm for Off-policy Temporal-difference Learning with Linear Function Approximation," in *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[8] H. R. Maei, D. Silver, and R. S. Sutton, "Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[9] R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvari, and E. Wiewiora, "Fast Gradient-Descent Methods for Temporal-Difference Learning with Linear Function Approximation," in *International Conference on Machine Learning (ICML)*, 2009.

[10] H. R. Maei and R. S. Sutton, "GQ(lambda): A General Gradient Algorithm for Temporal-Difference Prediction Learning with Eligibility Traces," in *Conference on Artificial General Intelligence (AGI)*, 2010.

[11] H. R. Maei, C. Szepesvari, S. Bhatnagar, and R. S. Sutton, "Toward Off-Policy Learning Control with Function Approximation," in *International Conference on Machine Learning (ICML)*, 2010.

[12] Liviu Panait and Sean Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, pp. 387–434, 2005, 10.1007/s10458-005-2631-2.

[13] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 2, pp. 156 –172, march 2008.

[14] Jeff Schneider, Weng-Keen Wong, Andrew Moore, and Martin Riedmiller, "Distributed value functions," in *In Proceedings of the Sixteenth International Conference on Machine Learning*. 1999, pp. 371–378, Morgan Kaufmann.

[15] R. Olfati-Saber and R.M. Murray, "Consensus Problems in Networks of Agents with Switching Topology and Time-delays," *IEEE Transactions on Automatic Control*, vol. 49, pp. 1520–1533, Sep 2004.

[16] Lin Xiao and Stephen Boyd, "Fast Linear Iterations for Distributed Averaging," *Systems and Control Letters*, vol. 53, pp. 65–78, 2004.

[17] A. Olshevsky and J. N. Tsitsiklis, "Convergence Speed in Distributed Consensus and Averaging," *SIAM Journal on Control and Optimization*, vol. 48, pp. 33–55, Jan 2009.

[18] Paulina Varshavskaya, Leslie Pack Kaelbling, and Daniela Rus, "Efficient distributed reinforcement learning through agreement," in *Proceedings of the 9th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, Tsukuba, Japan, November 2008.

[19] K. Srivastava and A. Nedic, "Distributed asynchronous constrained stochastic optimization," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 5, no. 4, pp. 772 –790, aug. 2011.

[20] J. Chen and A. H. Sayed, "Diffusion Adaptation Strategies for Distributed Optimization and Learning over Networks," *ArXiv*, , no. 1111.0034v1, 2011.

[21] Jianshu Chen and A.H. Sayed, "Performance of diffusion adaptation for collaborative optimization," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, march 2012.

[22] A. Antos, C. Szepesvari, and R. Munos, "Learning Near-Optimal Policies with Bellman-Residual Minimization Based Fitted Policy Iteration and a Single Sample Path," *Machine Learning*, vol. 71, pp. 89–129, 2008, 10.1007/s10994-007-5038-2.

[23] A. H. Sayed, *Adaptive Filters*, John Wiley & Sons, 2008.

[24] H. R. Maei, *Gradient Temporal-Difference Learning Algorithms*, Ph.D. thesis, University of Alberta, 2011.

[25] C. Szepesvari, "Algorithms for Reinforcement Learning," in *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool Publishers, 2009.

[26] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 2, ch. 6 -updated online- of *Athena Scientific Optimization and Computation Series*, Athena Scientific, 2005.