Reduced Variance Deep Reinforcement Learning with Temporal Logic Specifications

Qitong Gao
Duke University
Durham, NC
qitong.gao@duke.edu

Davood Hajinezhad
Duke University
Durham, NC
davood.hajinezhad@duke.edu

Yan Zhang
Duke University
Durham, NC
yan.zhang2@duke.edu

Yiannis Kantaros
Duke University
Durham, NC
yiannis.kantaros@duke.edu

Michael M. Zavlanos
Duke University
Durham, NC
michael.zavlanos@duke.edu

ABSTRACT

In this paper, we propose a model-free reinforcement learning method to synthesize control policies for mobile robots modeled as Markov Decision Process (MDP) with unknown transition probabilities that satisfy Linear Temporal Logic (LTL) specifications. Specifically, we develop a reduced variance deep Q-Learning technique that relies on Neural Networks (NN) to approximate the state-action values of the MDP and employs a reward function that depends on the accepting condition of the Deterministic Rabin Automaton (DRA) that captures the LTL specification. The key idea is to convert the deep Q-Learning problem into a nonconvex max-min optimization problem with a finite-sum structure, and develop an Arrow-Hurwicz-Uzawa type stochastic reduced variance algorithm with constant stepsize to solve it. Unlike Stochastic Gradient Descent (SGD) methods that are often used in deep reinforcement learning, our method can estimate the gradients of an unknown loss function more accurately, improving the stability of the training process. Moreover, our method does not require learning the transition probabilities in the MDP, constructing a product MDP, or computing Accepting Maximal End Components (AMECs). This allows the robot to learn an optimal policy even if the environment cannot be modeled accurately or if AMECs do not exist. In the latter case, the resulting control policies minimize the frequency with which the system enters bad states in the DRA that violate the task specifications. To the best of our knowledge, this is the first model-free deep reinforcement learning algorithm that can synthesize policies that maximize the probability of satisfying an LTL specification even if AMECs do not exist. Rigorous convergence analysis and rate of convergence are provided for the proposed algorithm as well as numerical experiments that validate our method.

ICCPS '19, April 16–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6285-6/19/04…$15.00
https://doi.org/10.1145/3302509.3311053

CCS CONCEPTS

• **Theory of computation → Formal languages and automata theory**; **Design and analysis of algorithms**; • **Computing methodologies → Artificial intelligence**; **Machine learning**; **Semi-supervised learning settings**;

KEYWORDS

Linear Temporal Logic, Reinforcement Learning, Reduced Variance Stochastic Optimization

ACM Reference Format:
Qitong Gao, Davood Hajinezhad, Yan Zhang, Yiannis Kantaros, and Michael M. Zavlanos. 2019. Reduced Variance Deep Reinforcement Learning with Temporal Logic Specifications. In *10th ACM/IEEE International Conference on Cyber-Physical Systems (with CPS-IoT Week 2019) (ICCPS '19), April 16–18, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3302509.3311053

1 INTRODUCTION

Traditionally, the robot motion planning problem consists of generating robot trajectories that reach a desired goal region starting from an initial configuration while avoiding obstacles [21]. More recently, a new class of planning approaches have been developed that can handle a richer class of tasks than the classical point-to-point navigation, and can capture temporal and boolean specifications. Such approaches typically rely on formal languages, such as Linear Temporal logic (LTL), to represent complex tasks and on discrete models, such as transition systems [14, 20] or Markov Decision Processes (MDPs) [10, 15, 36], to capture the robot dynamics and the uncertainty in the workspace.

Control of MDPs under LTL specifications has been extensively studied recently [10, 15, 36]. Often, the common assumption is that the transition probabilities in the MDPs are known. In this case, tools from probabilistic model checking [2] can be used to design policies that maximize the probability of satisfying the assigned LTL task. For example, in [10, 15, 36], first a product MDP is constructed by combining the MDP that captures robot mobility and the Deterministic Rabin Automaton (DRA) that represents the LTL specification and then, by computing the Accepting Maximum End Components (AMECs) of the product MDP, control policies that

maximize the probability of satisfying the LTL formula are synthesized. However, construction of the product MDP and computation of AMECs is computationally expensive.

In this paper, we consider robots modeled as MDPs with unknown transition probabilities that are responsible for accomplishing complex tasks captured by LTL formulas. Our goal is to design policies that maximize the probability of satisfying the LTL specification without the construction of a product MDP and AMECs. This signficantly decreases the computational cost and also renders the proposed method applicable to planning problems where AMECs do not exist. To achieve that, we first assign rewards to the transitions of the product MDP so that high rewards correspond to transitions that lead to accepting states that need to be visited infinitely often, as in [30]. Transitions in the product MDP are essentially determined by transitions in the MDP given the current action and the next MDP state. Since the DRA is a deterministic automaton, given the current DRA states and the next MDP states, DRA transitions can be uniquely determined and combined with MDP transitions to form the transitions in the product MDP. This allows us to design a policy for the product MDP that maximizes the collection of the rewards. By construction of the rewards and by the accepting condition of the DRA, maximizing the collection of the rewards implies that the probability of satisfying the LTL formula is maximized.

Since the state-space of the proposed control problem under LTL specifications is typically very large, tabular Reinforcement Learning (RL) methods become intractable since they need to perform value estimation for every single state separately [8, 35]. To overcome this difficulty, we employ function approximation, which assigns a parametrized mapping to the value function and the goal is to learn the mapping parameters such that an approximation error is minimized. Specifically, we employ deep Q-networks [27, 37]. Compared to linear approximation methods that have been extensively studied in RL, deep Q-networks avoid the need for an *a priori* selection of appropriate basis functions that are needed to accurately approximate the value function. Then, we convert the deep Q-learning problem into a non-convex min-max optimization problem with finite-sum structure, and develop an Arrow-Hurwicz-Uzawa [19] type stochastic reduced variance algorithm with constant step size to solve it. Unlike Stochastic Gradient Descent (SGD) methods that are often used in deep RL and are known to suffer from instabilities that affect convergence and the quality of the resulting policies [8], our method can estimate the gradients of an unknown loss function more accurately, improving the stability of the training process. Despite the popularity of deep RL methods due to their appealing numerical performance, the theoretical understanding of these methods is still quite limited. In this paper, we analytically prove that the proposed algorithm converges to the first-order stationary solution of this problem with a sub-linear rate.

To the best of our knowledge, the most relevant works on learning for control under LTL specifications are [10, 12, 25, 26, 30, 38]. Specifically, [30] converts the LTL specification into a DRA and takes the product between the MDP and the DRA to form a Product MDP. Then it employs a model-based learning method to learn the transition probabilities and optimize the current policy using Temporal Difference (TD) learning [34]. However, since this method stores all transition probabilities and state values, it can typically be used to solve problems with small and low dimensional state spaces. Furthermore, in this model-based approach, if the model of the environment and robot dynamics are not learned accurately, then the learned policy is not optimal. In our work, the state-action values of each state in the product MDP are approximated by Neural Networks which require much fewer resources to store and are also more expressive. To avoid the dependence of model-based methods on learning an accurate enough model of the environment, model-free learning algorithms can be used instead. Specifically, [12, 38] find AMECs in the learned product MDP to determine the accepting states and then learn the policy through value iteration [12] or actor-critic type learning [38]. Nevertheless, the quadratic complexity of finding AMECs prohibits these algorithms from handling MDPs with large state spaces and complex LTL specifications, i.e., DRAs with large numbers of states. This is not the case with our method that does not construct a product MDP nor does it require the computation of AMECs. Moreover, note that [12, 38] return no policies if there do not exist AMECs in the product MDP. To the contrary, in such cases, our proposed control synthesis algorithm can compute a policy that maximizes the probability of satisfying the acceptance condition of the DRA. In practice, this means that the policy returned by our method minimizes the frequency with which the system enters bad states in the DRA that violate the task specifications. This is relevant to applications where the LTL specifications can be thought of as soft constraints whose violation is not destructive for the system, as in our recent work [15]. In [10, 25, 26], control of MDPs with unknown transition probabilities under Signal Temporal Logic (STL) and Truncated Linear Temporal Logic (TLTL) specifications is considered. However, STL and TLTL specifications are satisfied by finite paths. To the contrary, here we consider LTL tasks that are satisfied by infinite paths.

On the other hand, the most relevant works on RL using function approximation are presented in [7, 11]. In [11] the authors generalize the popular reduced variance algorithms named SAGA [9] and SVRG [18] to solve the policy evaluation problem using linear function approximation. They show that the proposed reduced variance algorithms converge significantly faster than stochastic gradient based algorithms such as the GTD2 [33] algorithm. However, this analysis is based on linear function approximation, which makes the problem convex, and cannot be easily extended to the case of nonlinear function approximation. In [7] the authors generalize the GTD2 algorithm for nonlinear smooth function approximation. However, the algorithm in [7] is incremental and thus sample-inefficient in practice. Furthermore, our algorithm uses constant step-size instead of the non-increasing step-size proposed in [7]. To the best of our knowledge, our proposed algorithm is the first RL algorithm utilizing nonlinear function approximation of the value function with provable convergence rate.

This paper is organized as follows: In Section 2, we review necessary background on LTL, labeled MDPs, deep Q-Learning, and we formulate the proposed learning problem. In Section 3, we develop the proposed Arrow-Hurwicz-Uzawa type reduced variance algorithm to solve the LTL planning problem under consideration. Simulation studies are provided in Section 4.

2 PRELIMINARIES AND PROBLEM DEFINITION

In this section, we briefly review preliminaries on LTL, MDPs, deep Q-Learning, and we formally define the proposed LTL planning problem.

2.1 LTL Specifications

The basic ingredients of Linear Temporal Logic are a set of atomic propositions $\mathcal{AP}$, the boolean operators, i.e., conjunction $\wedge$, and negation $\neg$, and two temporal operators, next $\bigcirc$ and until $\mathcal{U}$. LTL formulas over a set $\mathcal{AP}$ can be constructed based on the following grammar: $\phi ::= \text{true} \mid \xi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2$, where $\xi \in \mathcal{AP}$. For the sake of brevity we abstain from presenting the derivations of other Boolean and temporal operators, e.g., *always* $\Box$, *eventually* $\Diamond$, *implication* $\Rightarrow$, which can be found in [2]. Any LTL formula can be translated into a Deterministic Rabin Automaton (DRA) defined as follows [2].

Definition 2.1 (DRA). *A DRA over $2^{\mathcal{AP}}$ is a tuple $\mathcal{R}_\phi = (Q, q^0, \Sigma, \delta, \mathcal{F})$ where Q is a finite set of states; $q^0 \subseteq Q$ is the set of initial states; $\Sigma = 2^{\mathcal{AP}}$ is the input alphabet; $\delta : Q \times \Sigma \to Q$ is the transition function and $\mathcal{F} = \{(\mathcal{G}_1, \mathcal{B}_1), \ldots, (\mathcal{G}_n, \mathcal{B}_n)\}$ is a set of accepting pairs where $\mathcal{G}_i, \mathcal{B}_i \subseteq Q, i \in \{1, \ldots, n\}$.*

A run of $\mathcal{R}_\phi$ over a infinite word $\omega_\Sigma = \omega_\Sigma(1)\omega_\Sigma(2)\omega_\Sigma(3)\cdots \in \Sigma^\omega$ is a infinite sequence $\omega_Q = \omega_Q(1)\omega_Q(2)\omega_Q(3)\cdots$, where $\omega_Q(1) \in q_0$ and $\omega_Q(k+1) \in \delta(\omega_Q(k), \omega_\Sigma(k))$ for all $k \geq 1$. Let $\text{Inf}(\omega_Q)$ denote the set of states that appear infinitely often in ω_Q. Then, a run ω_Q is accepted by $\mathcal{R}_\phi$ if $\text{Inf}(\omega_Q) \cap \mathcal{G}_i \neq \emptyset$ and $\text{Inf}(\omega_Q) \cap \mathcal{B}_i = \emptyset$ for at least one pair $i \in \{1, \ldots, n\}$ of accepting states $i = 1, \ldots, n$ [3].

2.2 MDP Robot Model

Robot mobility in the workspace can be represented by a product Markov Decision Process (MDP) defined as follows.

Definition 2.2 (MDP). *A Markov Decision Process (MDP) is a tuple $M = (S, s_0, A, P, R, \gamma)$, where S is a finite set of states; s_0 is the initial state; A is a finite set of actions; P is the transition probability function defined as $P : S \times A \times S \to [0, 1]$; $R : S \times A \times S \to \mathbb{R}$ is the reward function; $\gamma \in [0, 1]$ is the discount factor.*

In order to model uncertainty in both the robot motion and the workspace properties, we extend Definition 2.2 to include probabilistic labels giving rise to a labeled MDP defined as follows [30].

Definition 2.3 (Labeled MDP). *A labeled MDP is a tuple $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{P}, R, \gamma, \mathcal{AP}, L)$, where $\mathcal{S}$ is a finite set of states; s^0 is the initial state; $\mathcal{A}$ is a finite set of actions; $\mathcal{P}$ is the transition probability function defined as $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$; $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function; $\gamma \in [0, 1]$ is the discount factor; $\mathcal{AP}$ is the set of atomic propositions; $L : \mathcal{S} \to 2^{\mathcal{AP}}$ is the labeling function that returns the atomic propositions that are satisfied at a state $s \in \mathcal{S}$.*

Definition 2.4 (Policy of MDP). *A deterministic policy π of a labeled MDP $\mathcal{M}$ is a function, $\pi : \mathcal{S} \to \mathcal{A}$, that maps each state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$.*

Observe that given an initial state s^0 and a policy π, an *infinite path* u_π of the MDP is generated under the policy π, defined as an infinite sequence $u_\pi = s^0, s^1, \ldots, s^t, \ldots$, where $s^t \in \mathcal{S}$ is the state of the MDP at the stage t. Note that, given an action a^t due to the policy π, a transition from $s^t \in \mathcal{S}$ to $s^{t+1} \in \mathcal{S}$ in the labeled MDP occurs with probability $\mathcal{P}(s^t, a^t, s^{t+1})$, and a scalar reward r^t is generated.

Moreover, given a policy π we can define the accumulated reward over a finite horizon starting from stage t as follows.

Definition 2.5 (Accumulated Return). *Given a labeled MDP $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{P}, R, \gamma, \mathcal{AP}, L)$ and a policy π, the accumulated return over an finite horizon starting from the stage t and ending at stage $t + T$, $T > 0$, is defined as $G_t = \sum_{k=t}^{T} \gamma^k r_{t+k}$, where r_{t+k} is the return at the stage $t + k$.*

2.3 Deep Q-Learning

Consider the labeled MDP defined in Definition 2.3 with unknown transition probabilities $\mathcal{P}$. Our goal in this section is to compute a policy π^* that maximizes the expected accumulated return from the initial stage, i.e.,

$$\pi^* = \text{argmax}_\pi \, \mathbb{E}_{s^{t\geq 1} \sim \mathcal{S}, r^{t\geq 1} \sim R, a^{t\geq 1} \sim \pi}[G_1], \tag{1}$$

where $s^{t\geq 1} \sim \mathcal{S}$ means that the states $s^1, s^2, \ldots, s^T$ are selected from the finite state set $\mathcal{S}$, $r^{t\geq 1} \sim R$ means that the rewards are determined by the reward function R of the MDP $\mathcal{M}$, and $a^{t\geq 1} \sim \pi$ means that the actions are determined by the policy π.

To solve the optimization problem (1) we can use Q-Learning, which relies on the state-action value function defined bellow.

Definition 2.6 (State-Action Value Function). *Given an MDP $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{P}, R, \gamma, \mathcal{AP}, \mathcal{L})$ and policy π, the state-action value function $Q^\pi(s, a)$ is defined as the expected return for taking action a when at state s following policy π, i.e., $Q^\pi(s, a) = \mathbb{E}[G_t | s^t = s, a^t = a]$.*

Definition 2.7 (Optimal State-Action Value Function). *The optimal state-action value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ is the maximum state-action value for state s and action a for some policy π.*

Q-Learning learns a greedy deterministic policy $\pi(s) = \text{argmax}_a Q^\pi(s, a)$. In what follows, we present a Deep Q-Learning approach that relies on Neural Networks (NNs) to represent the state-action value function that depends on an unknown parameter θ. The parameter θ and the respective state-action value function, denoted by $Q(s, a; \theta)$, can be learned iteratively by minimizing the following loss function:

$$J(\theta) = \mathbb{E}_{s, s' \sim \rho^\beta, r \sim R, a \sim \beta} \left[(Q(s, a; \theta) - y)^2 \right], \tag{2}$$

where y is the training target defined as

$$y = R(s, a, s') + \gamma Q(s', \pi(s'); \theta). \tag{3}$$

In (2), β is an exploration policy, such as ϵ-greedy, ρ^β is the state visitation distribution over policy β, and θ are the weights of the NN. Note that the training target y depends on the current network weights θ, that change with every iteration causing overestimation problems [37]. In supervised learning, the training target should be independent of the training weight. In [27], this issue is addressed

using a separate network, with weights θ', to generate the Q values needed to form the new target $y_{DoubleQ}$:

$$y_{DoubleQ} = R(s, a, s') + \gamma Q(s', \pi(s'); \theta'). \quad (4)$$

Note that deep Q-Learning can be implemented in a batch setting to accelerate the training process and reduce the variance of the computed gradients [13, 27, 37, 39]. To do so, we define an empirical data buffer for batch training as follows:

Definition 2.8 (Empirical Data Buffer). *Let $e^t = (s^n, a^n, s^{n+1}, r^n)$ denote an agent's experience at time step n. Then, the empirical data buffer is defined as the set $\mathcal{E}^n = \{e^1, \ldots, e^n\}$ that collects all prior experiences up to time n.*

2.4 Problem Definition

Consider a robot operating in a workspace $\mathcal{W} \subseteq \mathbb{R}^d$, $d = 2, 3$ that is responsible for accomplishing a high level complex task captured by an LTL formula ϕ. Robot mobility in $\mathcal{W}$ is captured by a labeled MDP $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{P}, R, \gamma, \mathcal{AP}, L)$, as defined in Definition 2.3. We can compose the labeled MDP $\mathcal{M}$ and the DRA defined in Definition 2.1 to obtain a product MDP.

Definition 2.9. *A product MDP between the labeled MDP $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{P}, R, \gamma, \mathcal{AP}, L)$ and the DRA $\mathcal{R}_\phi = (Q, q^0, \Sigma, \delta, \mathcal{F})$ is a tuple $\mathcal{P} = (\mathcal{S}_p, s_p^0, \mathcal{A}_p, \mathcal{P}_p, \mathcal{F}_p, R_p, \gamma_p)$, where $\mathcal{S}_p = \mathcal{S} \times Q$ is the set of states; $s_{0_p} = (s_0, q_0)$ is the initial state; $\mathcal{A}_p$ is the set of actions inherited from MDP, so that $\mathcal{A}_p((s, q)) = \mathcal{A}(s)$; $\mathcal{P}_p$ is the set of transition probabilities, so that for $s_p \in \mathcal{S}_p, a_p \in \mathcal{A}_p$ and $s_p' \in \mathcal{S}_p$, if $q' = \delta(q, L(s))$, then $\mathcal{P}_p(s_p, a_p, s_p') = \mathcal{P}(s, a, s')$; $\mathcal{F}_p = \{(\mathcal{G}_{p_1}, \mathcal{B}_{p_1}), \ldots, (\mathcal{G}_{p_n}, \mathcal{B}_{p_n})\}$ is the set of accepting states, where $\mathcal{G}_{p_i} = \mathcal{S} \times \mathcal{G}_{p_i}$ and $\mathcal{B}_{p_i} = \mathcal{S} \times \mathcal{B}_i$; $R_p : \mathcal{S}_p \times \mathcal{A}_p \times \mathcal{S}_p \to \mathbb{R}$ is the reward function; and γ_p is the discounting rate inherited from the labeled MDP.*

Our goal is to design a policy that maximizes the probability of satisfying the assigned LTL specification ϕ. The probability $\mathbb{P}_\pi^{\mathcal{P}}(\phi)$ that a policy π of the product MDP $\mathcal{P}$ satisfies an LTL formula ϕ is defined as

$$\mathbb{P}_\pi^{\mathcal{P}}(\phi) = \mathbb{P}(\{u_\pi^{\mathcal{P}} \in \mathcal{P}_\pi \mid L(\Pi|_{\mathcal{M}} r_\pi^{\mathcal{P}}) \models \phi\}), \quad (5)$$

where $\mathcal{P}_\pi$ is a set that collects all infinite paths $u_\pi^{\mathcal{P}}$ of the product MDP $\mathcal{P}$ (see also Definition 2.4) that are induced under the policy π. Also in (5), $\Pi|_{\mathcal{M}} u_\pi^{\mathcal{P}}$ stands for the projection of the infinite run $u_\pi^{\mathcal{P}}$ onto the state-space of $\mathcal{M}$, i.e., $\Pi|_{\mathcal{M}} u_\pi^{\mathcal{P}}$ is an infinite sequence of states in $\mathcal{M}$, and $L(\Pi|_{\mathcal{M}} r_\pi^{\mathcal{P}})$ denotes the word that corresponds to $\Pi|_{\mathcal{M}} u_\pi^{\mathcal{P}}$. Then the problem that we address in this paper can be summarized as follows.

Problem 1. *Given a labeled MDP $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{P}, R, \gamma, \mathcal{AP}, L)$ with unknown transition probabilities and an LTL specification ϕ, find a policy π such that the probability $\mathbb{P}_\pi^{\mathcal{P}}(\phi)$, defined in* (5)*, of satisfying the ϕ is maximized.*

3 PROPOSED ALGORITHM

In this section, we first formulate (5) as a model-free reinforcement learning problem that allows us to maximize the probability of satisfying an LTL specification ϕ without constructing the product MDP $\mathcal{P}$ and AMECs, see the definition of AMECs in [15]. Then, we propose an Arrow-Hurwicz-Uzawa type stochastic reduced variance algorithm to solve this reinforcement learning problem.

3.1 Model-Free RL Problem

In this section we discuss how to transform Problem 1 into a model-free reinforcement learning problem. Define the reward function $R_{\mathcal{P}} : \mathcal{S}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{P}} \times \mathcal{S}_{\mathcal{P}} \to \mathbb{R}$ in Definition 2.9 as follows

$$R_{\mathcal{P}}(s_{\mathcal{P}}, a_{\mathcal{P}}, s_{\mathcal{P}}') = \begin{cases} r_{\mathcal{G}} & \text{if } s_{\mathcal{P}}' \in \mathcal{G}_{p_i}, \\ r_{\mathcal{B}} & \text{if } s_{\mathcal{P}}' \in \mathcal{B}_{p_i}, \\ r_d & \text{if } q' \text{ is deadlock state}, \\ r_o & \text{otherwise} \end{cases} \quad (6)$$

for all $i \in \{1, \ldots, n\}$. Specifically, we assign high rewards $r_{\mathcal{G}}$ to transitions that lead to states in the sets $\mathcal{G}_i$ and assign low rewards $r_{\mathcal{B}}$ to transitions that end in states in the sets $\mathcal{B}_i$. This selection of rewards can guide the robot to maximize the probability of satisfying a given LTL formula ϕ with minimum number of movements. A negative reward with large absolute value, r_d, is assigned to transitions that lead to deadlock states in the DRA. This prevents violation of the safety constraints in the LTL formula (e.g., $\square\neg\xi$). A negative reward with small absolute value, r_o, is assigned to all other transitions so that the robot reaches the final states with the minimum number of movements. This reward should be smaller than $r_{\mathcal{G}}$ because $\mathcal{G}_i$ is the set of states that will be visited infinitely often and should be larger than $r_{\mathcal{B}}$ because $\mathcal{B}_i$ is the set of states that are not supposed to be visited infinitely often. Overall, these four rewards are ranking in the following monotone increasing order:

$$r_d << r_{\mathcal{B}} < r_o < r_{\mathcal{G}} \quad (7)$$

Then, to solve Problem 1, we design a policy π^* for the product MDP $\mathcal{P}$ that maximizes the accumulated rewards determined in (6), i.e.,

$$\pi^* = \text{argmax}_\pi \mathbb{E}_{s^{t\geq 1}, r^{t\geq 1} \sim R_{\mathcal{P}}, a^{t\geq 1} \sim \pi}[G_1], \quad (8)$$

where $s^t \in \mathcal{S} \in \mathcal{P}$, $r^{t\geq 1} \sim R_{\mathcal{P}}$ means that the reward r^t at time step t is determined by the product MDP reward function $\mathcal{R}_{\mathcal{P}}$, and $a^{t\geq 1} \sim \pi$ means that the action a^t taken at time step t is determined by the policy π.

Although the reward function is typically defined over the product MDP, in practice, we can obtain the reward of taking action $a_{\mathcal{P}}$ given $s_{\mathcal{P}}$ directly from the transitions in the associated DRA. This way we can avoid constructing the product MDP and this approach is more suitable for model-free learning. To obtain the reward $R_{\mathcal{P}}$ of the product MDP $\mathcal{P}$ directly from the associated DRA $\mathcal{R}_\phi$, we introduce the DRA reward function $R_{\mathcal{R}_\phi} : Q \times \mathcal{A}_{\mathcal{P}} \times Q \to \mathbb{R}$ as

$$R_{\mathcal{R}_\phi}(q, a, q') = \begin{cases} r_{\mathcal{G}} & \text{if } q' \in \mathcal{G}_i, \\ r_{\mathcal{B}} & \text{if } q' \in \mathcal{B}_i, \\ r_d & \text{if } q' \text{ is deadlock state}, \\ r_o & \text{otherwise}, \end{cases} \quad (9)$$

for all $i \in \{1, \ldots, n\}$. Then, replacing the product MDP reward function $R_{\mathcal{P}}$ with the DRA reward function $R_{\mathcal{R}_\phi}$ in (8), the learning

Algorithm 1 Policy generation using the proposed reduced variance deep Q-Learning algorithm

Input: $\theta, \theta', \alpha_\theta, \alpha_\lambda, \beta, \tau, \gamma, max\_epi, max\_step, max\_epoch, s_0, q_0, \mathcal{M}, \mathcal{E}^n = [], G_\theta, G_\lambda$

Begin:

for $epi = 0$ to $max\_epi - 1$ **do**
  $s \leftarrow s_0, q \leftarrow q_0, m \leftarrow (s, q)$
  for $step = 0$ to $max\_step - 1$ **do**
    Sample $a \sim \beta$
    Get s' following the dynamics in $\mathcal{M}$ after performing a
    $q', r \leftarrow$ `CheckRabin`(s, q, s')
    $m' \leftarrow (s', q')$
    Append tuple (m, a, r, m') to $\mathcal{E}^n$
    $s \leftarrow s', q \leftarrow q', m \leftarrow m'$
    if CheckTerminal(m) **then**
      break
    end if
  end for
end for
Initialize G_θ and G_λ
for $r = 0$ to $max\_epoch - 1$ **do**
  Sample data $(m_{i_r}, a_{i_r}, r_{i_r}, m'_{i_r})$ from $\mathcal{E}^n$
  if CheckTeminal(m'_{i_r}) **then**
    $y_{i_r} \leftarrow r_{i_r}$
  else
    $y_{i_r} \leftarrow r_{i_r} + \gamma \max_{a'} Q(m'_{i_r}, a'; \theta')$
    ▷ $Q(\cdot, \cdot; \theta')$ is the output from target network with input $(\cdot, \cdot)$
  end if
  $\delta_{i_r} \leftarrow y_{i_r} - Q(m_{i_r}, a_{i_r}; \theta)$
  Set $(\tilde{\theta}_i^r, \tilde{\lambda}_i^r) = (\theta^r, \lambda^r)$ if $i = i_r$, and $(\tilde{\theta}_i^r, \tilde{\lambda}_i^r) = (\tilde{\theta}_i^{r-1}, \tilde{\lambda}_i^{r-1})$ otherwise.
  $G_\theta \leftarrow \frac{1}{n}\sum_{i=1}^n \nabla_\theta f_i(\tilde{\theta}_i^{r-1}, \tilde{\lambda}_i^{r-1}) + \frac{1}{np_{i_r}}[\nabla_\theta f_{i_r}(\theta^r, \lambda^r) - \nabla_\theta f_{i_r}(\tilde{\theta}_{i_r}^{r-1}, \tilde{\lambda}_{i_r}^{r-1})]$
  $G_\lambda \leftarrow \frac{1}{n}\sum_{i=1}^n \nabla_\lambda f_i(\tilde{\theta}_i^{r-1}, \tilde{\lambda}_i^{r-1}) + \frac{1}{np_{i_r}}[\nabla_\lambda f_{i_r}(\theta^r, \lambda^r) - \nabla_\lambda f_{i_r}(\tilde{\theta}_{i_r}^{r-1}, \tilde{\lambda}_{i_r}^{r-1})]$
  $\theta \leftarrow \theta - \alpha_\theta G_\theta$
  $\lambda \leftarrow \lambda + \alpha_\lambda G_\lambda$
  Set $\theta' \leftarrow \theta$ after every fixed amount of epochs
end for
$\pi(\cdot) \leftarrow \text{argmax}_{a'} Q(\cdot, a'; \theta)$
return π

objective becomes:

$$\pi^* = \text{argmax}_\pi \mathbb{E}_{s^{t\geq 1}\sim\mathcal{P}, r^{t\geq 1}\sim\mathcal{R}_\phi, a^{t\geq 1}\sim\pi}[G_1]. \tag{10}$$

The only difference between (8) and (10) is that in (10) the reward r^t, is determined by the DRA reward function (9), while in (8), r^t is determined by the product MDP reward function (6).

3.2 Reduced Variance RL Algorithm

Reduced variance optimization algorithms have been widely used for solving convex finite-sum problems in recent years; see, e.g., SAG/SAGA [9, 31], the SDCA [32], the SVRG [18], and the RPDG

Algorithm 2 `CheckRabin`(s, q, s')

Input:
Current MDP state s; Current Rabin state q; Next MDP state s'; MDP $\mathcal{M}$; DRA $\mathcal{R}_\phi$;

Begin:

$q' \leftarrow \delta(q, L(s'))$
if q' is a deadlock state **then**
  $q' \leftarrow q$
end if
$r \leftarrow R_\mathcal{R}(q, q')$ ▷ $R_\mathcal{R}$ is the reward function in DRA $\mathcal{R}_\phi$
return q', r

[23]. However, for nonconvex problems as the one considered here the analysis is significantly more difficult. Very recently, a few works focused on reduced variance algorithms for non-convex finite-sum optimization problems. For example, in [16] a stochastic algorithm named NESTT is proposed for finite-sum non-convex problems over distributed networks. Similar algorithms have been proposed and analyzed in [29, 40]. Nevertheless, [16, 29, 40] are not fit to the problem formulation considered here or they are not numerically efficient. Specifically, [16] is a distributed algorithm, [29] is not exactly SAGA because it needs to sample twice in each iteration, and [40] is SVRG which means that it needs the full gradient in each inner loop. To address these limitations, we reformulate the deep Q-Learning problem (2) into a finite-sum non-convex optimization problem and propose a new efficient Arrow-Hurwicz-Uzawa type reduced variance algorithm to solve it. Compared to the methods in [16, 29, 40], the proposed algorithm computes the gradient at one data point at each iteration, similar to SAGA in [9]. Furthermore, we show that the proposed algorithm converges with sublinear rate for the nonlinear problem considered in this paper.

To develop the proposed algorithm, first define the first order optimality condition associated with the minimization of the deep Q-learning loss function in (2) using the training target defined in (4) as

$$\mathop{\mathbb{E}}_{s,s'\sim\rho^\beta, a\sim\beta, r\sim R_\mathcal{P}}[\delta_\theta \nabla_\theta Q(s, a; \theta)] = 0, \tag{11}$$

where $\delta_\theta = y_{DoubleQ} - Q(s, a; \theta)$ is the TD-error of Q-Learning and $\theta \in \mathbb{R}^d$ is the parameter in the function $Q(s, a)$. Using (11), we can define the error function

$$\hat{J}(\theta) = \frac{1}{2}||\mathbb{E}[\delta_\theta \nabla_\theta Q(s, a; \theta)]||^2, \tag{12}$$

and formulate the optimization problem

$$\min_\theta \hat{J}(\theta) \ [= \frac{1}{2}||\mathbb{E}[\delta_\theta \nabla_\theta Q(s, a; \theta)]||^2]. \tag{13}$$

For simplicity, we denote the expectation $\mathbb{E}_{s,s'\sim\rho^\beta, a\sim\beta, r\sim R_\mathcal{P}}[\delta_\theta \nabla_\theta Q(s, a; \theta)]$ by $\mathbb{E}[\delta_\theta \nabla_\theta Q(s, a; \theta)]$ in (12), (13) and in the following analysis. Note that $Q(\cdot, \cdot; \theta)$ is a nonlinear function, due to the use of NNs to approximate the true Q value, therefore, (13) becomes a non-convex optimization problem that is more challenging to solve.

To convert (13) to a finite sum form, define a new variable $w \in \mathbb{R}^d$ to denote the expectation in (13) and replace this expectation by its empirical estimate, using the data stored in the empirical buffer $\mathcal{E}^n$

defined in Definition 2.8. Then, problem (13) can be rewritten as

$$\min_{\theta,w} J(\theta,w) = \frac{1}{2}w^T w \tag{14}$$
$$\text{s.t.} \quad w = \frac{1}{n}\sum_{i=1}^{n}\delta_\theta^i \nabla_\theta Q(s^i,a^i;\theta),$$

The Lagrangian corresponding to problem (14) can be defined as

$$\mathcal{L}(\theta,\lambda,w) = \frac{1}{2}w^T w + \lambda^T\left(\frac{1}{n}\sum_{i=1}^{n}\delta_\theta^i\nabla_\theta Q(s^i,a^i;\theta) - w\right), \tag{15}$$

where $\lambda \in \mathbb{R}^d$ is the dual variable associated with the constraint. Applying the Karush-Kuhn-Tucker (KKT) condition $\nabla_w \mathcal{L}(\theta,\lambda,w) = 0$, (15) becomes

$$\mathcal{L}(\theta,\lambda) = -\frac{1}{2}\lambda^T\lambda + \lambda^T\frac{1}{n}\sum_{i=1}^{n}\delta_\theta^i\nabla_\theta Q(s^i,a^i;\theta), \tag{16}$$

and the corresponding dual problem is

$$\max_{\lambda}\min_{\theta}\frac{1}{n}\sum_{i=1}^{n} -\frac{1}{2}\lambda^T\lambda + \lambda^T\delta_\theta^i\nabla_\theta Q(s^i,a^i;\theta). \tag{17}$$

To simplify notation, we denote $f_i(\theta,\lambda) = -\frac{1}{2}\lambda^T\lambda + \lambda^T\delta_\theta^i\nabla_\theta Q(s^i,a^i;\theta)$. Then, (17) becomes

$$\max_{\lambda}\min_{\theta}\frac{1}{n}\sum_{i=1}^{n} f_i(\theta,\lambda), \tag{18}$$

which is in a finite-sum form. We can show that if (θ^*,λ^*) is a stationary solution of problem (18), then θ^* is a stationary solution of the problem (13). This is because, if (θ^*,λ^*) is a stationary solution of (18), then, $\lambda^* = \frac{1}{n}\sum_i \delta_\theta^i \nabla Q^*(s^i,a^i)$. Further, we have that $\nabla_\theta(\frac{1}{n}\sum_i \delta_\theta^i \nabla Q^*(s^i,a^i))^T\lambda^* = 0$. Combining these two equations we see that θ^* is a stationary solution of problem (13).

Building upon the reduced variance methods for finite-sum problems discussed in Appendix A, we can define here too the intermediate variables $(\tilde{\theta}_i^r,\tilde{\lambda}_i^r) = (\theta^r,\lambda^r)$ if $i = i_r$, and $(\tilde{\theta}_i^r,\tilde{\lambda}_i^r) = (\tilde{\theta}_i^{r-1},\tilde{\lambda}_i^{r-1})$ otherwise. Since we are maximizing over λ and minimizing over θ, the gradient of the objective in (18) becomes

$$G^r = \begin{bmatrix} G_\theta^r \\ -G_\lambda^r \end{bmatrix}, \tag{19}$$

where

$$G_\theta^r = \frac{1}{n}\sum_{i=1}^{n}\nabla_\theta f_i(\tilde{\theta}_i^{r-1},\tilde{\lambda}_i^{r-1}) + \frac{1}{np_{i_r}}[\nabla_\theta f_{i_r}(\theta^r,\lambda^r) - \nabla_\theta f_{i_r}(\tilde{\theta}_{i_r}^{r-1},\tilde{\lambda}_{i_r}^{r-1})] \tag{20}$$

$$G_\lambda^r = \frac{1}{n}\sum_{i=1}^{n}\nabla_\lambda f_i(\tilde{\theta}_i^{r-1},\tilde{\lambda}_i^{r-1}) + \frac{1}{np_{i_r}}[\nabla_\lambda f_{i_r}(\theta^r,\lambda^r) - \nabla_\lambda f_{i_r}(\tilde{\theta}_{i_r}^{r-1},\tilde{\lambda}_{i_r}^{r-1})]. \tag{21}$$

Let us associate a new parameter $\eta_i > 0$ with the i-th data point and select the stepsize to be $\alpha = \frac{1}{\sum_{i=1}^n \eta_i}$, where the selection of the parameter η_i is discussed in the next section. Then, the update of the variables θ and λ takes the form:

$$\theta \leftarrow \theta - \alpha_\theta G_\theta^r, \tag{22a}$$
$$\lambda \leftarrow \lambda + \alpha_\lambda G_\lambda^r, \tag{22b}$$

where α_θ and α_λ are the learning rates for θ and λ, respectively.

The proposed algorithm is summarized in Alg. 1. Specifically, first the empirical data buffer is generated as follows [lines 1-14, Algorithm 1]. At the beginning of each episode, the current MDP state s is set to the initial MDP state s_0 and the current rabin state q is set to the initial rabin state q_0. Then, $m = (s,q)$ represents the current product state in the product MDP [line 2, Algorithm 1]. The current action a is determined by an exploration policy β [lines 4, Algorithm 1]. Then, action a is performed and the next MDP state s' is observed [line 5, Algorithm 1]. The next rabin state q' and the reward r, given the current Rabin state q, are determined by the function `CheckRabin`(s,q,s') in Algorithm 2, which is discussed in detail later in the text [line 6, Algorithm 1]. Then, the next product state $m' = (s',q')$ is constructed and the tuple (m,a,r,m') is stored in the empirical data buffer $\mathcal{E}^n$ [lines 7-8, Algorithm 1]. Then s,q,m are set to s',q',m', respectively [line 9, Algorithm 1]. If m is a terminal state, the current episode is completed [lines 10-12, Algorithm 1] and the next episode starts. Note that we terminate every episode when $q \in \mathcal{G}_i$, since the goal of the robot is to learn a policy so that the states $q \in \mathcal{G}_i$ are visited infinitely often, as this satisfies the accepting condition of the DRA. When all episodes are completed, we finish appending data to the empirical data buffer $\mathcal{E}^n$. The reduced variance optimization steps are summarized in [lines 15-31, Algorithm 1]. In particular, in this part of the algorithm a *a gradient descent* and a *a gradient ascent* step are performed over the Lagrangian function $L(\theta,\lambda)$ with respect to variables θ and λ, respectively. Our algorithm is an Arrow-Hurwicz-Uzawa type algorithm [19].

The function `CheckRabin`(s,q,s') used in [line 6, Algorithm 1] to obtain the next state q' and the reward r given an action a is described in Algorithm 2. This algorithm requires as an input the current MDP state s, the current rabin state q, the next MDP state s' after taking action a, the labeling function L from the MDP, and the DRA reward function $\mathcal{R}_\phi$. First, the next rabin state q' is determined based on the DRA transition rule δ (see Definition 2.1) given the atomic propositions that are true at the next state s' [line 2, Algorithm 2]. If q' is a deadlock state, then this means that the LTL specification is violated. In this case, following the same logic as in [30], we reset the Rabin state to the Rabin state q of the previous step. In [line 10, Algorithm 2], the reward is determined by the reward function $R_\mathcal{R}$ embedded in DRA $\mathcal{R}_\phi$.

4 CONVERGENCE ANALYSIS

In this section we provide the convergence analysis for the proposed non-convex Arrow-Hurwicz-Uzawa type algorithm. To simplify presentation, we focus on the general form of the finite-sum problem given in (36) in Appendix A, where we define $x = (\theta,\lambda)$. We first make the following assumptions.

Assumptions A. We assume that

A1. For all $i = 1,2,\cdots,n$, the functions g_i are gradient Lipschitz continuous, i.e., there exists constant $L_i > 0$ such that

$$\|\nabla g_i(x) - \nabla g_i(y)\| \le L_i\|x-y\| \;\forall\; x,y \in \text{dom}\,(g_i).$$

A2. The function g is lower bounded.

From Assumption A1 we conclude that the function $g = \frac{1}{n}\sum_i g_i$ is also gradient Lipschitz continuous with constant $L > 0$. Assumption A holds when the functions g_i are smooth and dom (g_i) is compact. In our problem, this means that the iterates θ^r and λ^r are always bounded, which is expected when a nonlinear function is properly designed to approximate the value function, [7]. If this assumption is violated, then the variables can grow unbounded which means that the model is overfitting. In this case, finding the optimal θ and λ becomes meaningless.

Problem (36) is a nonconvex optimization problem, therefore we can define the ϵ-stationary solution as the point x^* such that $\mathbb{E}\|\nabla g(x^*)\| \leq \epsilon$. Consequently, we can define the stationarity gap at iteration r as

$$\Psi^r := \mathbb{E}\|\nabla g(x^r)\|^2. \tag{23}$$

Moreover, define a potential function to characterize the behavior of the algorithm as

$$\Lambda^r := \frac{1}{n}\sum_{i=1}^n g_i(x^r) + \sum_{i=1}^n \frac{4}{p_i\eta_i n^2}\left\|\nabla g_i(y_i^{r-1}) - \nabla g_i(x^r)\right\|^2.$$

Also, we define the filtration $\mathcal{F}^r$ as the σ-field generated by $\{i_t\}_{t=1}^{r-1}$. Throughout this section the expectations are taken with respect to i_r conditioning on $\mathcal{F}^r$ unless otherwise stated. In the first lemma we show that under appropriate selection of the stepsize α, the potential function decreases as the algorithm proceeds.

Lemma 4.1. *Suppose Assumptions A holds true, and set*

$$p_i = \frac{\eta_i}{\sum_{i=1}^n \eta_i} \quad \text{and} \quad \alpha \leq \frac{1}{(\sum_{i=1}^n \sqrt{5L_i})^2}. \tag{24}$$

Then, the following holds:

$$\mathbb{E}[\Lambda^r - \Lambda^{r-1}] \leq -\frac{1}{100\alpha}\mathbb{E}\|x^r - x^{r-1}\|^2 - \sum_{i=1}^n \frac{1}{\eta_i n^2}\left\|\nabla g_i(x^{r-1}) - \nabla g_i(y_i^{r-2})\right\|^2. \tag{25}$$

The Proof of Lemma 4.1 is shown in Appendix B. The next theorem presents the main convergence results pertaining to the proposed algorithm.

Theorem 4.2. *Suppose Assumption A holds true and pick p_i and α as in* (24)*. Then, we have*

1. *Every limit point of Algorithm 1 is a stationary solution of problem* (36) *almost surely.*
2. *Algorithm 1 converges with a sublinear rate, i.e.,*

$$\mathbb{E}_u[\Psi^u] \leq \frac{200\mathbb{E}[\Lambda^1 - \Lambda^{T+1}]}{\alpha T},$$

where u is a uniformly random number from $\{1, 2, \cdots, T\}$.

Proof. First we prove the convergence. Using Assumption A2 we can verify that the potential function Λ^r is lower bounded for every $r \geq 1$. Therefore, there exists $\underline{\Lambda}$ such that $\Lambda^r - \underline{\Lambda} \geq 0$. From Lemma 4.1 it is clear that Λ^r is diminishing. Therefore, $\{\Lambda^r - \underline{\Lambda}\}$ is a nonnegative supermartingale. Applying the supermartingale Convergence Theorem [6, Proposition 4.2] we conclude that $\{\Lambda^r\}$ converges almost surely. Further, we have that for all i almost surely

$$\left\|\nabla g_i(x^{r-1}) - \nabla g_i(y_i^{r-2})\right\|^2 \to 0, \ \mathbb{E}\|x^r - x^{r-1}\| \to 0. \tag{26}$$

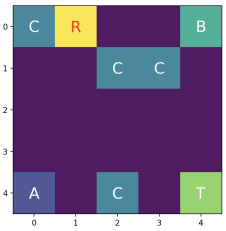

Figure 1: 5 × 5 Simulation Environment. States $(4,0),(0,4),(4,4)$ are labeled with A, B, T, respectively. States $(0,0),(1,2),(1,3),(4,2)$ are labeled with C. R represents the current position of the robot.

Combining this result with the update equation (37) we conclude that $\|\nabla g(x^r)\| \to 0$ almost surely.

Next we prove the second part of the theorem. We bound the stationarity gap defined in (23) as follows:

$$\begin{aligned}\Psi^r &= \mathbb{E}\|\nabla g(x^r)\|^2 \\ &= \mathbb{E}\Big\|\frac{1}{n}\sum_{i=1}^n \nabla g_i(x^r) - G^r + G^r\Big\|^2 \\ &\leq 2\mathbb{E}\|G^r\|^2 + 2\mathbb{E}\Big\|\frac{1}{n}\sum_{i=1}^n \nabla g_i(x^r) - G^r\Big\|^2 \\ &\leq \frac{2}{\alpha^2}\mathbb{E}\|x^{r+1} - x^r\|^2 \\ &\quad + \sum_{i=1}^n \frac{2\alpha}{\eta_i n^2}\left\|\nabla g_i(x^r) - \nabla g_i(y_i^{r-1})\right\|^2, \end{aligned} \tag{27}$$

where in the second inequality we utilize equations (37) and (39). Combining equation (25) and (27) we obtain the following relationship between the stationary gap and the potential function:

$$\Psi^r \leq \frac{200}{\alpha}\mathbb{E}[\Lambda^r - \Lambda^{r+1}]. \tag{28}$$

Taking the sum over $r = 1, \cdots, T$ and dividing both sides by T we have

$$\frac{1}{T}\sum_{r=1}^T \Psi^r \leq \frac{200\mathbb{E}[\Lambda^1 - \Lambda^{T+1}]}{\alpha T}.$$

Since u is a uniform random number in $\{1, 2, \cdots, T\}$ we have

$$\mathbb{E}_u[\Psi^u] \leq \frac{200\mathbb{E}[\Lambda^1 - \Lambda^{T+1}]}{\alpha T} \tag{29}$$

which proves the second part of the theorem. □

5 NUMERICAL EXPERIMENTS

In this section, we illustrate the proposed algorithm on LTL planning problems for a single robot. Algorithm 1 was implemented in Python 2.7, and the neural networks were defined using Tensorflow [1] and trained on the Amazon Web Services server using an Nvidia Tesla K80 graphics card and a quad-core Intel Xeon CPU E5-2686 2.30GHz. In what follows, we examine the performance of Algorithm 1 in a 5×5 and a 10×10 discrete grid world, where each discrete point is associated with a state of a labeled MDP that models the robot; see Figure 1 and Figure 2. The set $\mathcal{AP}$ of atomic

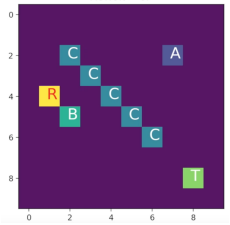

Figure 2: 10×10 Simulation Environment. States $(2,7),(5,2),(8,8)$ are labeled with A, B, T, respectively. States $(2,2),(3,3),(4,4),(5,5),(6,6)$ are labeled with C. R represents the current position of the robot.

propositions is defined as $\mathcal{AP} = \{A, B, C, T, \varnothing\}$, where the atomic propositions A, B, T are observed in the respective regions shown in Figure 1 and Figure 2, while $\varnothing$ denotes that nothing is observed.

In both environments, the robot can take 4 actions: *UP, RIGHT, DOWN, LEFT*. We assume the robot has a noisy controller, which can only execute the desired action with probability of 0.7, and a random action among the other available ones is taken with probability of 0.3. The initial location of the robot is at $(0,1)$ in the 5×5 environment and $(4,1)$ in the 10×10 environment, respectively. In what follows, we consider two case studies. The first case study pertains to the 5×5 environment and an LTL specification corresponding to a DRA with 5 states. We compare our algorithm to a stochastic gradient descent (SGD) method [22] and a Momentum-SGD method [28] that are typically used in deep learning, the model-based algorithm proposed in [30] and tabular Q-Learning approach [34] The second case study pertains to the 10×10 environment and a LTL specification corresponding to a DRA with 16 states. In this case, we implement our algorithm in a online growing-batch [24] fashion to address complexity of the larger state space. Note that in both cases, AMEC-based methods, such as [12], cannot be applied to design policies that maximize the probability of satisfying the considered LTL formulas since AMECs do not exist.

The structure of the neural networks remains the same for both case studies. Specifically, both networks have 2 hidden layers with 400 and 300 nodes, respectively, and a biased layer is associated with each hidden layer. All the weights of the hidden layers are initialized uniformly between 0 and 0.1 and the weights of the output layer are initialized uniformly between 0 and 0.01. All the weights of the bias layers are initialized at 0. The activation functions of the hidden layers are selected to be hyperbolic tangent functions. A batch normalization layer [17] is applied after each one of the two hidden layers to adjust and scale the activation output. The reward function parameters are selected as follows: $r_{\mathcal{G}} = -1$, $r_{\mathcal{B}} = -10$, $r_d = -100$, $r_o = -2$.

5.1 Case Study I

In this case study, we assume that the robot must learn a policy to satisfy the following LTL task:

$$\phi_1 = \Diamond(A \wedge \Diamond T) \wedge \Box\neg C \tag{30}$$

in the 5×5 environment shown in Figure 1. In words, the LTL task ϕ_1 requires the robot to visit A first, then visit T while always avoiding C. The DRA $\mathcal{R}_{\phi_1}$ corresponding to the LTL ϕ_1 has 5 states,

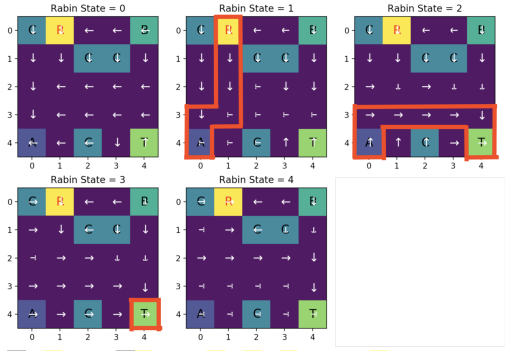

Figure 3: Graphical depiction of the policy designed by Algorithm 1 for the 5×5 environment considered in Case Study I. Each subfigure shows the policy with respect to each rabin state $Q = \{q_0, \ldots, q_4\}$. The arrows show the action, i.e., the direction in which the robot should move, according to the designed policy. The optimal trajectory following the optimal policy is highlighted within red boundaries.

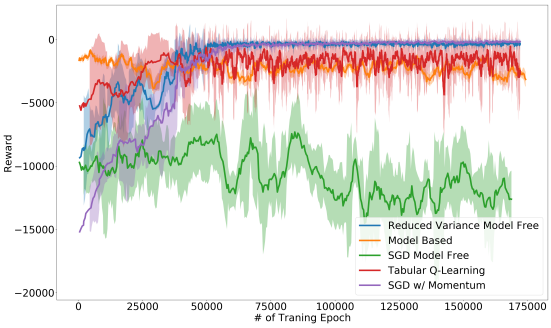

Figure 4: Case Study I: Comparison of the proposed reduced variance model-free learning algorithm, the SGD model-free learning algorithm, the Momentum-SGD model-free learning algorithm, the tabular Q-Learning method and the model-based learning algorithm in [30].

i.e., $Q = \{q_0, q_1, q_2, q_3, q_4\}$ and 33 edges, where the state q_0 is the deadlock state, the state q_1 is the initial state and the state q_4 is the terminal/accepting state that needs to be visited infinitely often. The robot needs to satisfy $A \wedge \neg C$, $T \wedge \neg C$ and $\neg C$, to transition from q_1 to q_2, q_2 to q_3 and q_3 to q_4, respectively.

The policy computed for this environment is shown in Figure 3. Specifically, Figure 3 shows the policy separately for each rabin state q, since the robot is operating in the product MDP state-space that collects states of the form $m = (s, q)$; see Definition 2.9. Observe that the policy learned by the robot in the rabin states q_1, q_2 and q_3 eventually leads to the accepting states with the minimum number of movements due to the design of the reward function (9). Figure 4 compares the proposed reduced variance model-free learning algorithm to the SGD method [22] with TD model-free learning [34], the Momentum-SGD method [28] with TD model-free learning [34], tabular Q-Learning [34] and the model-based learning algorithm

proposed in [30]. It can be seen that the proposed Arrow-Hurwicz-Uzawa type model free learning algorithm converges faster than the other methods and achieves a better approximation of the state-action value functions, as can be seen by the lower variance in the reward.

5.2 Case study II

In this case study, the robot must learn a policy to satisfy the following LTL task:

$$\phi_2 = \Diamond(A \wedge \Diamond(B \wedge \Diamond T)) \wedge \Box\Diamond A \wedge \Box\Diamond B \wedge \Box\neg C. \tag{31}$$

in the 10×10 environment shown in Figure 2. In words, the LTL task ϕ_2 requires the robot to first visit A, then visit B and T, at last travel between A and B infinitely often while always avoiding C. The DRA $\mathcal{R}_{\phi_2}$ corresponding to the LTL ϕ_2 has 16 states, i.e., $Q = \{q_0, q_1, q_2, q_3, \ldots, q_{15}\}$ and 241 edges, where the state q_2 is the deadlock state, the state q_3 is the initial state, the state q_1 is the terminal/accepting state that needs to be visited infinitely often and $\mathcal{G} = \{q_2, q_3, q_6, q_8, q_{10}, q_{11}, q_{12}\}$ is the set of states that need to be visited only finitely often. The robot needs to satisfy $A \wedge \neg C$, $B \wedge \neg C$, $T \wedge \neg C$, $A \wedge \neg C$, $B \wedge \neg C$, $\neg C$, $A \wedge \neg C$, $B \wedge \neg C$, $A \wedge \neg C$ and $\neg C$ to transition from q_3 to q_6, q_6 to q_8 , q_8 to q_{10}, q_{10} to q_{11}, q_{11} to q_{12}, q_{12} to q_{15}, q_{15} to q_{14}, q_{14} to q_{13}, q_{13} to q_1 and q_1 to q_{14}, respectively. The rabin states not mentioned above do not have any physical meaning in this particular simulation environment. An accepting run of the corresponding rabin automaton $\mathcal{R}_{\phi_2}$ is $\mathcal{T}_{\phi_2} = q_3 q_6 q_8 q_{10} q_{11} q_{12} q_{15} q_{14} q_{13} q_1 (q_{14} q_{13} q_1)^{\omega}$.

Note that during the learning process, the agent typically interacts with the environment and new data is collected at each time step. To take full advantage of new data and improve the training efficiency, we implement Algorithm 1 in an online fashion, using the growing batch method described in [24]. The convergence and convergence rate analysis of this online implementation of Algorithm 1 is left for future research. To implement Algorithm 1 in an online way, first we model $\mathcal{E}^n$ as a queue. Specifically, each time new data is collected, we replace the gradient information of the oldest data, $\nabla f_{old}(\theta, \lambda)$, with the new one, $\nabla f_{new}(\theta, \lambda)$, in $\mathcal{E}^n$. Then, we update G_θ and G_λ in (19) as follows

$$G_\theta^r = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta f_i(\tilde{\theta}_i^{r-1}, \tilde{\lambda}_i^{r-1}) + \frac{1}{n}[\nabla_\theta f_{new}(\theta^r, \lambda^r) - \nabla_\theta f_{old}(\tilde{\theta}_{old}^{r-1}, \tilde{\lambda}_{old}^{r-1})] \tag{32}$$

$$G_\lambda^r = \frac{1}{n}\sum_{i=1}^{n} \nabla_\lambda f_i(\tilde{\theta}_i^{r-1}, \tilde{\lambda}_i^{r-1}) + \frac{1}{n}[\nabla_\lambda f_{new}(\theta^r, \lambda^r) - \nabla_\lambda f_{old}(\tilde{\theta}_{old}^{r-1}, \tilde{\lambda}_{old}^{r-1})]. \tag{33}$$

The policy computed for this environment is shown in Figure 5. Observed that according to the optimal learned policy, the robot learns to keep some distance from region C while traveling between the states that need to be visited infinitely. As in Case Study I, we also compare the proposed reduced variance TD model-free learning algorithm, with the SGD method [22] with TD model-free learning [34], the Momentum-SGD method [28] with TD model-free learning [34], the tabular Q-Learning method [34] and the model-based learning algorithm proposed in [30]. The results are

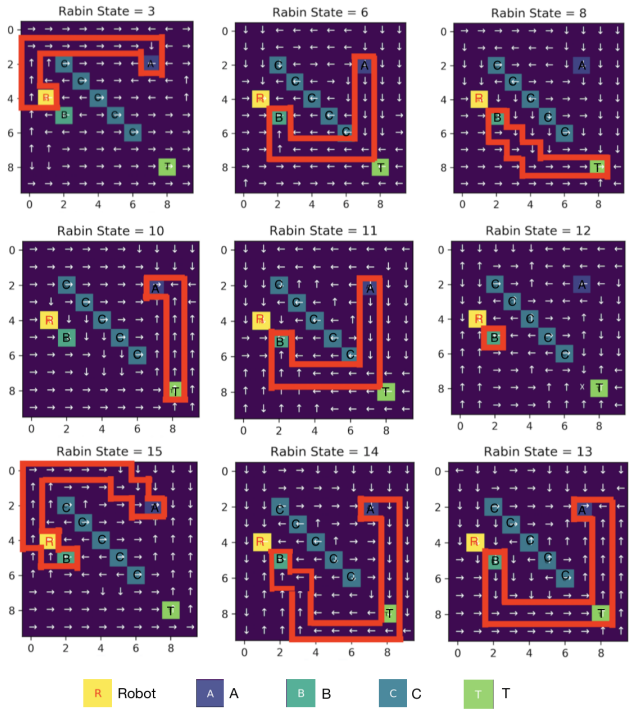

Figure 5: Graphical depiction of the policy designed by Algorithm 1 for the 10×10 environment considered in Case Study II. Each subfigure shows the policy with respect to the rabin states that appear in $\mathcal{T}_{\phi_2}$ except for the accepting state q_1, $Q = \{q_3, q_6, q_8, q_{10}, q_{11}, q_{12}, q_{15}, q_{14}, q_{13}\}$. The arrows show the action, i.e., the direction in which the robot should move, according to the designed policy. The optimal trajectory following the optimal policy is highlighted within red boundaries.

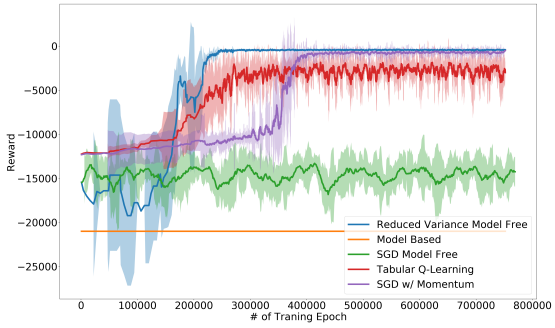

Figure 6: Case Study II: Comparison of the proposed reduced variance model-free learning algorithm, the SGD model-free learning algorithm, the Momentum-SGD model-free learning algorithm, the tabular Q-Learning method and the model-based learning algorithm in [30].

shown in Figure 6. It can be seen that our method converges faster than the other methods and achieves a better approximation of the

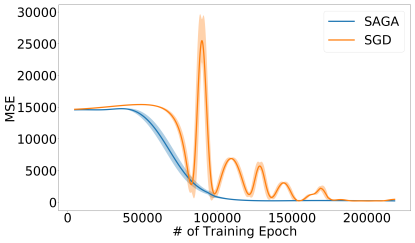

Figure 7: Comparison of the proposed reduced variance algorithm with the SGD algorithm in a policy evaluation setting.

state-action value functions, as can be seen by the lower variance in the reward. A possible reason that the model-based approach does not converge is that the data gathered during the exploration stage is not sufficient to reconstruct the MDP accurately.

5.3 Policy Evaluation

In this section, we compare the error of evaluating the policy generated by Algorithm 1 and SGD for the example considered in Case Study II. The error is defined as the mean square error between the ground truth state values for all product states, $V^{\pi}(s_p)\ \forall s_p \in \mathcal{S}_p$, calculated by dynamic programming [34] and the estimated state values for all states, $\tilde{V}^{\pi}_{Alg_1}(s_p)$ and $\tilde{V}^{\pi}_{sgd}(s_p)\ \forall s_p \in \mathcal{S}_p$, using Algorithm 1 and SGD. The ground truth value $V^{\pi}(s_p)$ is defined as the expectation of the accumulated return over a finite horizon starting from stage 0 and ending at stage $T > 0$ given a state s_p and a policy π, i.e.,

$$V^{\pi}(s_p) = \mathbb{E}[\sum_{i=0}^{T} \gamma^i r_{i+1} | s_p, \pi],\ \forall s_p \in \mathcal{S}_p. \tag{34}$$

and can be calculated using the following dynamic programming update equation [34]

$$V^{\pi}(s_p) \leftarrow \sum_{a \in \mathcal{A}_p} \pi(a|s_p) \sum_{s'_p \in \mathcal{S}_p} \left(R_{\mathcal{P}}(s_p, a, s'_p) + \gamma p(s'_p|s_p, a) V^{\pi}(s'_p) \right), \tag{35}$$

where $p(s'_p|s_p, a) \in \mathcal{P}_p$ are the transition probabilities, and γ is the discounting factor. To calculate $\tilde{V}^{\pi}_{Alg_1}(s_p)$ and $\tilde{V}^{\pi}_{sgd}(s_p)\ \forall s_p \in \mathcal{S}_p$, we sample a fixed number of (s_p, a, r, s'_p) tuples by employing policy π in the environment considered in Case Study II, where $s_p, s'_p \in \mathcal{S}_p,\ a \in \mathcal{A}_p,\ a \sim \pi$ and $r \in R_p$. At last, [Algorithm 1, line 16-31] and the SGD method [22] are applied to calculate $\tilde{V}^{\pi}_{Alg_1}(s_p)$ and $\tilde{V}^{\pi}_{sgd}(s_p)$, $\forall s_p \in \mathcal{S}_p$. The results are shown in Figure 7 and it shows that our algorithm converges faster and the training process is more stable.

6 CONCLUSION

In this paper, we proposed a reduced variance model-free deep reinforcement learning method to synthesize control policies for mobile robots modeled by Markov Decision Process (MDP) with unknown transition probabilities that satisfy Linear Temporal Logic (LTL) specifications. Unlike SGD algorithms that are often used in deep RL, our method can estimate the gradients of an unknown loss function more accurately, improving the stability of the training process. To the best of our knowledge, this is the first model-free reduced variance deep reinforcement learning algorithm that can solve LTL planning problems even if AMECs do not exist. Unlike relevant works, our method does not require learning the transition probabilities in the MDP, constructing a product MDP, or computing Accepting Maximal End Components (AMECs). This significantly reduces the computational cost and also renders our method applicable to planning problems where AMECs do not exist. Simulation studies verified the convergence performance of the proposed algorithm.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. 2008. *Principles of model checking*. MIT press.

[3] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. 2017. *Formal Methods for Discrete-Time Dynamical Systems*. Vol. 89. Springer.

[4] D. Bertsekas. 1999. *Nonlinear programming*. Athena scientific Belmont.

[5] D. Bertsekas. 2000. Incremental Gradient, Subgradient, and Proximal Methods f or Convex Optimization: A Survey. (2000). LIDS Report 2848.

[6] D. P. Bertsekas and J. N. Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

[7] S. Bhatnagar, D. Precup, D. Silver, R. Sutton, H.R. Maei, and C. Szepesvári. 2009. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*. 1204–1212.

[8] R. H. Crites and A. G. Barto. 1995. Improving Elevator Performance Using Reinforcement Learning. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*. 1017–1023.

[9] A. Defazio, F. Bach, and S. Lacoste-Julien. 2014. SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. In *The Proceeding of NIPS*.

[10] Xuchu Ding, Stephen L Smith, Calin Belta, and Daniela Rus. 2014. Optimal control of Markov decision processes with linear temporal logic constraints. *IEEE Trans. Automat. Control* 59, 5 (2014), 1244–1257.

[11] S. Du, J. Chen, L. Li, L. Xiao, and D. Zhou. 2017. Stochastic variance reduction methods for policy evaluation. *arXiv preprint arXiv:1702.07944* (2017).

[12] Jie Fu and Ufuk Topcu. 2014. Probably approximately correct mdp learning and control with temporal logic constraints. *arXiv preprint arXiv:1404.7073* (2014).

[13] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. 2016. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*. 2829–2838.

[14] Meng Guo and Dimos V Dimarogonas. 2015. Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research* 34, 2 (2015), 218–235.

[15] Meng Guo and Michael M Zavlanos. 2018. Probabilistic Motion Planning under Temporal Tasks and Soft Constraints. *IEEE Trans. Automat. Control* PP, 99 (2018), 11. DOI:10.1109/TAC.2018.2799561

[16] D. Hajinezhad, M. Hong, T. Zhao, and Z. Wang. 2016. NESTT: A Nonconvex Primal-Dual Splitting Method for Distributed and Stochastic Optimization. In *Advances in Neural Information Processing Systems 29*. 3215–3223.

[17] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. 448–456.

[18] R. Johnson and T. Zhang. 2013. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. In *the Proceedings of the Neural Information Processing (NIPS)*.

[19] L. Hurwicz K. J. Arrow and H. Uzawa. 1958. *Studies in Linear and Non-linear Programming*. Stanford University Press.

[20] Yiannis Kantaros and Michael M Zavlanos. 2018. Sampling-based optimal control synthesis for multi-robot systems under global temporal tasks. *IEEE Trans. Automat. Control* PP, 99 (7 2018). https://doi.org/10.1109/TAC.2018.2853558

[21] Sertac Karaman and Emilio Frazzoli. 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30, 7 (2011), 846–894.

[22] J. Kiefer and J. Wolfowitz. 1952. Stochastic Estimation of the Maximum of a Regression Function. *Ann. Math. Statist.* 23, 3 (09 1952), 462–466. https://doi.org/10.1214/aoms/1177729392

[23] G. Lan and Y. Zhou. 2017. An optimal randomized incremental gradient method. *Mathematical Programming* (2017).

[24] Sascha Lange, Thomas Gabel, and Martin Riedmiller. 2012. Batch reinforcement learning. In *Reinforcement learning*. Springer, 45–73.

[25] Xiao Li, Yao Ma, and Calin Belta. 2017. A Policy Search Method For Temporal Logic Specified Reinforcement Learning Tasks. *arXiv preprint arXiv:1709.09611* (2017).

[26] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. 2017. Reinforcement learning with temporal logic rewards. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 3834–3839.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[28] Boris Polyak. 1964. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics* 4 (12 1964), 1–17. https://doi.org/10.1016/0041-5553(64)90137-5

[29] S. Reddi, S. Sra, B. Poczos, and A. Smola. 2016. Fast incremental method for nonconvex optimization. *arXiv preprint arXiv:1603.06159* (2016).

[30] Dorsa Sadigh, Eric S Kim, Samuel Coogan, S Shankar Sastry, and Sanjit A Seshia. 2014. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 1091–1096.

[31] M. Schmidt, N. Le Roux, and F. Bach. 2013. Minimizing Finite sums with the stochastic average gradient. (2013). Technical report, INRIA.

[32] S. Shalev-Shwartz and T. Zhang. 2013. Proximal Stochastic Dual Coordinate Ascent Methods for Regularzied Loss Minimization. *Journal of Machine Learning Rsearch* 14 (2013), 567–599.

[33] R. Sutton, H.R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora. 2009. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 993–1000.

[34] Richard S Sutton and Andrew G Barto. 2011. Reinforcement learning: An introduction. (2011).

[35] G. Tesauro. 1995. Temporal Difference Learning and TD-Gammon. *Commun. ACM* 38, 3 (1995), 58–68.

[36] Alphan Ulusoy, Tichakorn Wongpiromsarn, and Calin Belta. 2014. Incremental controller synthesis in probabilistic environments with temporal logic constraints. *The International Journal of Robotics Research* 33, 8 (2014), 1130–1144.

[37] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning.. In *AAAI*, Vol. 16. 2094–2100.

[38] Jing Wang, Xuchu Ding, Morteza Lahijanian, Ioannis Ch Paschalidis, and Calin Belta. 2015. Temporal logic motion control using actor–critic methods. *The International Journal of Robotics Research* 34, 10 (2015), 1329–1344.

[39] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).

[40] A. Zhu and E. Hazan. 2016. Variance Reduction for Faster Non-convex Optimization. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. 699–707.

A FINITE-SUM OPTIMIZATION

Consider the optimization problem:

$$\min_{x\in\mathbb{R}^d} \; g(x) = \frac{1}{n}\sum_{i=1}^{n} g_i(x), \tag{36}$$

where $g_i : \mathbb{R}^d \to \mathbb{R}$ is a smooth possibly nonconvex function. A standard method for solving (36) is the gradient descent algorithm which performs the update $x^{r+1} = x^r - \alpha^r \frac{1}{n}\sum_{i=1}^{n} \nabla g_r(x^r)$, where α^r is the stepsize. This method converges to the stationary solution of the problem (36) when α^r is chosen appropriately [4]. However, when n is large the cost of evaluating the gradient is very expensive. One approach to deal with this issue is using the stochastic gradient descent (SGD) algorithm, in which the full gradient is replaced with its unbiased estimation [5]. In particular, we sample i_r from $\mathcal{E}^n$, see Definition 2.8, and the update rule becomes $x^{r+1} = x^r - \alpha^r \nabla g_{i_r}(x^r)$. This significantly reduces the cost of evaluating the gradient compared to the GD algorithm. However, the variance of the estimation can be very high. To reduce the variance of the estimation, a diminishing step size can be employed, which however makes the algorithm very slow. To decrease the variance and maintain a cheap iteration, reduced variance algorithms have been proposed [9, 18, 31]. The key idea is to store the gradient information as the algorithm proceeds and use this information to reduce the variance of stochastic gradient approximation. In particular, in the SAGA algorithm [9] we sample i_r uniformly randomly from $\mathcal{E}^n$, and define the intermediate variable $y_i^r := x^r$ if $i = i_r$, and $y_i^r := y_i^{r-1}$ otherwise. Then, the algorithm performs the following update:

$$x^{r+1} = x^r - \alpha G^r, \tag{37}$$

where $G^r = \frac{1}{n}\sum_{i=1}^{n} \nabla g_i(y_i^{r-1}) + \nabla g_{i_r}(x^r) - \nabla g_{i_r}(y_{i_r}^{r-1})$, and α is a constant stepsize. It can be proved that G^r is an unbiased estimation of the full gradient. Also, the variance of G^r vanishes when $r \to \infty$ [9, 31]. Notice that in standard form sampling in the SGD and SAGA algorithms is uniform; however, it has been shown that in some cases nonuniform sampling might improve the convergence rate. See for example [16]. In particular, let p_i denote the probability of choosing the ith data. Then, in the nonuniform sampling SAGA (NU-SAGA) the moving direction G^r will be

$$G^r = \frac{1}{n}\sum_{i=1}^{n} \nabla g_i(y_i^{r-1}) + \frac{1}{np_{i_r}}[\nabla g_{i_r}(x^r) - \nabla g_{i_r}(y_{i_r}^{r-1})]. \tag{38}$$

B PROOF OF LEMMA 4.1

Proof. First we bound the difference between the true gradient and its unbiased estimation G^r. We have that

$$\begin{aligned}
&\mathbb{E}\Big\|\frac{1}{n}\sum_{i=1}^{n}\nabla g_i(x^r) - G^r\Big\|^2 \\
=&\mathbb{E}\Big\|\frac{1}{n}\sum_{i=1}^{n}\nabla g_i(x^r) - \frac{1}{n}\nabla g_i(y_i^{r-1}) - \frac{1}{p_{i_r}n}[\nabla g_{i_r}(x^r) + \nabla g_{i_r}(y_{i_r}^{r-1})]\Big\|^2 \\
\le&\sum_{i=1}^{n}\frac{1}{p_i n^2}\Big\|\nabla g_i(x^r) - \nabla g_i(y_i^{r-1})\Big\|^2,
\end{aligned} \tag{39}$$

where in the last inequality we used the fact that $\mathbb{E}[\nabla g_{i_r}(x^r) - \nabla g_{i_r}(y_{i_r}^{r-1})] = \sum_{i=1}^n p_i[\nabla g_i(x^r) - \nabla g_i(y_i^{r-1})]$ as well as the inequality $\mathbb{E}\|x - \mathbb{E}[x]\|^2 \le \mathbb{E}\|x\|^2$, which holds true for any random variable x.

From the definition of the potential function Λ^r we have

$$\begin{aligned}\mathbb{E}[\Lambda^r - \Lambda^{r-1}] &= \mathbb{E}\Big[\frac{1}{n}\sum_{i=1}^n g_i(x^r) - g_i(x^{r-1})\Big] \\ &+ \mathbb{E}\left[\frac{4}{n^2}\sum_{i=1}^n \frac{1}{p_i\eta_i}\left\|\nabla g_i(x^r) - \nabla g_i(y_i^{r-1})\right\|^2\right] \\ &- \mathbb{E}\left[\frac{4}{n^2}\sum_{i=1}^n \frac{1}{p_i\eta_i}\left\|\nabla g_i(x^{r-1}) - \nabla g_i(y_i^{r-2})\right\|^2\right]. \end{aligned} \tag{40}$$

Now we bound equation (40) term by term. For the first term we have

$$\begin{aligned}&\frac{1}{n}\sum_{i=1}^n [g_i(x^r) - g_i(x^{r-1})] \\ &\le \left\langle \frac{1}{n}\sum_{i=1}^n \nabla g_i(x^{r-1}), x^r - x^{r-1}\right\rangle + \frac{\sum_{i=1}^n L_i}{2n}\|x^r - x^{r-1}\|^2 \\ &= \left\langle \frac{1}{n}\sum_{i=1}^n \nabla g_i(x^{r-1}) + \frac{1}{\alpha}(x^r - x^{r-1}), x^r - x^{r-1}\right\rangle \\ &\quad - \left(\frac{1}{\alpha} - \frac{\sum_{i=1}^n L_i}{2n}\right)\|x^r - x^{r-1}\|^2 \\ &= \left\langle \frac{1}{n}\sum_{i=1}^n \nabla g_i(x^{r-1}) - G^{r-1}, x^r - x^{r-1}\right\rangle \\ &\quad - \left(\frac{1}{\alpha} - \frac{\sum_{i=1}^n L_i}{2n}\right)\|x^r - x^{r-1}\|^2 \\ &\le \frac{1}{2\tau}\left\|\frac{1}{n}\sum_{i=1}^n \nabla g_i(x^{r-1}) - G^{r-1}\right\|^2 \\ &\quad - \left(\frac{1}{\alpha} - \frac{\sum_{i=1}^n L_i}{2n} - \frac{\tau}{2}\right)\|x^r - x^{r-1}\|^2, \end{aligned} \tag{41}$$

where in the first inequality we used the Lipschitz continuity of the gradients of g_i's, and in the last inequality we applied the following identity which holds true for any vector a and b and for any constant $\tau > 0$: $\langle a, b\rangle \le \frac{1}{2\tau}\|a\|^2 + \frac{\tau}{2}\|b\|^2$. Selecting $\tau = \frac{1}{2\alpha}$ and taking the expectation on both sides of (41), we have

$$\begin{aligned}&\frac{\alpha}{2}\mathbb{E}\left\|\frac{1}{n}\sum_{i=1}^n \nabla g_i(x^{r-1}) - G^{r-1}\right\|^2 - \left(\frac{3}{4\alpha} - \frac{\sum_{i=1}^n L_i}{2n}\right)\mathbb{E}\|x^r - x^{r-1}\|^2 \\ &\le \sum_{i=1}^n \frac{\alpha}{p_i n^2}\left\|\nabla g_i(x^{r-1}) - \nabla g_i(y_i^{r-2})\right\|^2 - \left(\frac{3}{4\alpha} - \frac{\sum_{i=1}^n L_i}{2n}\right)\mathbb{E}\|x^r - x^{r-1}\|^2. \end{aligned} \tag{43}$$

where in the last inequality we used equation (39). Next we bound the second term (40) as

$$\begin{aligned}&\mathbb{E}\|\nabla g_i(x^r) - \nabla g_i(y_i^{r-1})\|^2 \\ &= \mathbb{E}\|\nabla g_i(x^r) + \nabla g_i(x^{r-1}) - \nabla g_i(x^{r-1}) - \nabla g_i(y_i^{r-1})\|^2 \\ &\le (1+\tau_i)\mathbb{E}\|\nabla g_i(x^r) - \nabla g_i(x^{r-1})\|^2 \\ &+ \left(1 + \frac{1}{\tau_i}\right)\mathbb{E}\|\nabla g_i(y_i^{r-1}) - \nabla g_i(x^{r-1})\|^2 \\ &= (1+\tau_i)\mathbb{E}\|\nabla g_i(x^r) - \nabla g_i(x^{r-1})\|^2 \\ &+ (1-p_i)\left(1 + \frac{1}{\tau_i}\right)\|\nabla g_i(y_i^{r-2}) - \nabla g_i(x^{r-1})\|^2, \end{aligned} \tag{44}$$

where in the first inequality we used the following identity which holds true for any vector a and b: $\|a+b\|^2 \le (1+\tau)\|a\|^2 + (1 + \frac{1}{\tau})\|b\|^2$, for every $\tau > 0$. The last equality is true because for each i there is a positive probability p_i such that the ith data is picked. Therefore, $\nabla g_i(y_i^{r-1}) - \nabla g_i(x^{r-1}) = 0$, otherwise with probability $1 - p_i$ we have $\nabla g_i(y_i^{r-1}) = \nabla g_i(y_i^{r-2})$.

Setting $\tau_i = \frac{2}{p_i}$, overall the second and third term of (40) can be bounded as

$$\begin{aligned}&+ \mathbb{E}\left[\frac{4}{n^2}\sum_{i=1}^n \frac{1}{p_i\eta_i}\left\|\nabla g_i(x^r) - \nabla g_i(y_i^{r-1})\right\|^2\right] \\ &- \mathbb{E}\left[\frac{4}{n^2}\sum_{i=1}^n \frac{1}{p_i\eta_i}\left\|\nabla g_i(x^{r-1}) - \nabla g_i(y_i^{r-2})\right\|^2\right] \\ &\le \sum_{i=1}^n \frac{4L_i^2(2+p_i)}{p_i^2\eta_i n^2}\mathbb{E}\|x^r - x^{r-1}\|^2 \\ &- \frac{2p_i(1-p_i)}{\eta_i n^2}\left\|\nabla g_i(y_i^{r-2}) - \nabla g_i(x^{r-1})\right\|^2. \end{aligned} \tag{45}$$

Combining (43) and (45), and using the relationship of $\frac{1}{\alpha} = \sum_{i=1}^n \eta_i$ we have

$$\begin{aligned}&\mathbb{E}[\Lambda^r - \Lambda^{r-1}] \\ &\le \sum_{i=1}^n \left(\frac{\alpha}{p_i n^2} - \frac{2p_i(1-p_i)}{\eta_i n^2}\right)\left\|\nabla g_i(x^{r-1}) - \nabla g_i(y_i^{r-2})\right\|^2 \\ &+ \sum_{i=1}^n \left(\frac{-3\eta_i}{4} + \frac{L_i}{2n} + \frac{4L_i^2(2+p_i)}{p_i^2\eta_i n^2}\right)\mathbb{E}\|x^r - x^{r-1}\|^2. \end{aligned} \tag{46}$$

Now we look at the constants on the right-hand-side in (46). Setting $p_i = \frac{\eta_i}{\sum_{i=1}^n \eta_i}$, we obtain that

$$\frac{\alpha}{p_i n^2} - \frac{2p_i(1-p_i)}{\eta_i n^2} \le -\frac{1}{\eta_i n^2}.$$

Also, selecting $\eta_i \ge \frac{5L_i}{np_i}$ we get $\frac{-3\eta_i}{4} + \frac{L_i}{2n} + \frac{4L_i^2(2+p_i)}{p_i^2\eta_i n^2} \le \frac{\eta_i}{100}$. Substituting $p_i = \frac{\eta_i}{\sum_{i=1}^n \eta_i}$ in this equation we have $\eta_i \ge \sqrt{L_i \sum_{i=1}^n \eta_i}$. Summing over i and simplifying the results we obtain $\sum_{i=1}^n \eta_i \ge (\sum_{i=1}^n \sqrt{5L_i})^2$, which completes the proof. □