Self-Preserving Genetic Algorithms for Safe Learning in Discrete Action Spaces

Preston K. Robinette
preston.k.robinette@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Nathaniel P. Hamilton
nathaniel.hamilton@parallaxresearch.org
Parallax Advanced Research
Beavercreek, OH, USA

Taylor T. Johnson
taylor.johnson@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

ABSTRACT

Self-Preserving Genetic Algorithms (SPGA) combine the evolutionary strategy of a genetic algorithm with safety assurance methods commonly implemented in safe reinforcement learning (SRL), a branch of reinforcement learning (RL) that accounts for safety in the exploration and decision-making process of the agent. Safe learning approaches are especially important in safety-critical environments, where failure to account for the safety of the controlled system could result in the loss of millions of dollars in hardware or bodily harm to people working nearby, as is true of many cyber-physical systems. While SRL is a viable approach to safe learning, there are many challenges that must be taken into consideration when training agents, such as sample efficiency, stability, and exploration—an issue that is easily addressed by the evolutionary strategy of a genetic algorithm. By combining GAs with the safety mechanisms used with SRL, SPGA offers a safe learning alternative that is able to explore large areas of the solution space, addressing SRL's challenge of exploration. This work implements SPGA with both action masking and run time assurance safety strategies to evolve safe controllers for three types of discrete action space environments applicable to cyber physical systems (control, routing, and operations) and under various safety conditions. Training and testing evaluation metrics are compared with results from SRL trained controllers to validate results. SPGA and SRL controllers are trained across 5 random seeds and evaluated on 500 episodes to calculate average wall time to train, average expected return, and percentage of safe action evaluation metrics. SPGA achieves comparable reward and safety performance results with significantly improved training efficiency (55x faster on average), demonstrating the effectiveness of this safe learning approach.

KEYWORDS

genetic algorithms, safe learning, safe reinforcement learning, run time assurance, action masking

ACM Reference Format:
Preston K. Robinette, Nathaniel P. Hamilton, and Taylor T. Johnson. 2023. Self-Preserving Genetic Algorithms for Safe Learning in Discrete Action Spaces. In *ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2022) (ICCPS '23), May 9–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3576841.3585936

ICCPS '23, May 9–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0036-1/23/05…$15.00
https://doi.org/10.1145/3576841.3585936

1 INTRODUCTION

Reinforcement learning (RL) is a branch of machine learning (ML) based upon the concept of operant conditioning, which uses feedback to reinforce behavior in a system. With recent advances in machine learning, sensing, and algorithm development, RL as a method has been increasingly adopted in various domains, including traffic control [21], robotics [15], chemistry [31], and advertising [12], and it has been used to solve previously intractable problems, such as in high-dimensional state spaces like Go [24] and real-time strategy games like Starcraft [30]. These recent advances have opened the door for the application of RL across various types of cyber physical systems (CPS), including CPS with complex dynamics which are difficult and costly to address by using traditional approaches [18]. Current use cases of RL for CPS range from plant operational control [11], transportation safety [5], electric power systems [7], and communication and networking systems [19]. Despite tremendous advances in RL applied to CPS, the continued adoption of RL trained systems is limited by safety and reliability concerns—concerns that are reminiscent of many black box systems, where a lack of transparency in the decision-making process leaves people wary of the decision made. To ensure the continued widespread adoption of RL systems, safety and reliability must be taken into consideration, especially in regard to safety-critical CPS.

Toward this end, *Safe Reinforcement Learning* (SRL) is a branch of reinforcement learning that takes safety into consideration during the training and/or execution of a policy. These methods include shielding [1], run time assurance [16], and action masking [9, 13]. Although these mechanisms keep the training agent safe, they also magnify issues associated with RL, such as sample efficiency, stability, and most commonly—agent exploration. In regard to exploration, this issue can drastically affect an agent's ability to learn an optimal policy. During training, an agent must decide between exploring new actions and states and exploiting previously learned information. By focusing on exploitation, the learned policy will likely get stuck in local minima. Similarly, if the agent focuses on exploration, positive behaviors are likely not reinforced in the learned policy. Even if the optimal policy is explored, this means nothing if the agent is not able to purposefully replicate this strategy with its policy. This tradeoff between exploration and exploitation is further magnified with the use of safety mechanisms, which obstruct much of the state space and hinder agent exploration [4].

In addition to RL, *Genetic Algorithms* (GAs) are also used to solve complex problems, especially where system dynamics are too complicated or unknown. GAs are a subset of evolutionary algorithms that use nature-inspired operators such as selection, crossover, mutation, and update to evolve iteratively better solutions to a given problem. GAs have been used to learn controllers

for problems ranging from industrial systems [14, 20] to planning [8, 28, 29]. In an interesting contrast to RL, GAs can be likened to an educated guessing strategy, also known as guess-and-check. They are exploratory in nature due to the stochastic nature of the *selection, crossover, and mutation* operators. This allows GAs to quickly and effectively explore the solution space to the problem at hand. As "good" traits are more likely to be passed on in nature, traits that lead to "safe" solutions can be selected for in a population of potential solutions, permeating through generations as they evolve.

Taking inspiration from SRL methods, this work integrates *run time assurance* (RTA) and *action masking* safety methods into the evolutionary process of a genetic algorithm, termed *Self-Perserving Genetic Algorithms* (SPGA). By combining the safety methods of SRL with the exploration benefits of GAs, we believe that SPGA will provide a faster safe-learning alternative to SRL, one that can quickly and safely explore large parts of the solution space. We test and compare SPGA and SRL in discrete action control, routing, and operations problems that can be applied to smart grid (control), smart transportation (routing), and smart factories (operations). To the best of the authors' knowledge, this is the first work to implement both action masking and RTA in the GA evolutionary process.

As such, this work builds upon a standard set of environments from OpenAI Gym [3] and OR-Gym [10] to introduce, develop, and evaluate Self-Preserving Genetic Algorithms in discrete action spaces. The **primary contributions** of this work are:

(1) the introduction and development of the SPGA safe learning framework,
(2) an implementation and evaluation of SPGA in three different discrete action environments,
(3) an implementation and comparison of SRL safety assurance approaches in three discrete action environments, and
(4) an analysis and comparison of the SPGA and SRL safe learning methods across the different environments and methods.

The results of this analysis give insight into the effectiveness of SPGAs as a method of safe learning in discrete action spaces.

2 PRELIMINARIES

In this section, we introduce background information related to reinforcement learning, genetic algorithms, safety, action masking, and run time assurance.

2.1 Learning Methods Overview

The learning methods used in this work are reinforcement learning and genetic algorithms.

2.1.1 ***Reinforcement Learning.*** Reinforcement learning (RL) uses the concept of operant conditioning to train agents to complete designated tasks, or goals [25]. Reinforcements via positive and/or negative rewards are dependent upon an agent's behavior in the environment. These rewards act as feedback to the agent's actions in the environment, and with more experience, the agent learns how to act to maximize the return (i.e. cumulative, discounted reward) it achieves in an episode.

During training, an agent interacts with an environment by taking an action $a_t \in A$ in an observation state $s_t \in S$. The agent receives feedback from the environment in the form of a reward $r = R(s_t, a_t)$ or $r = R(s_t, a_t, s_{t+1})$ and an observation state $s_{t+1} \in S$. The combination of a state, action, reward, and next state transition is known as an experience. Experiences are collected by various strategies and used to intermittently update an agent's policy, π, which determines the best action to take given an observation state, $a_t = \pi(s_t)$. The overall goal of the agent is to learn a policy that maximizes the cumulative, discounted reward gained per episode in the environment [27].

An *episode* consists of a time ordered set of experiences:

$$\{[s_0, a_0, r_0, s_1], [s_1, a_1, r_1, s_2], ..., [s_{n-1}, a_{n-1}, r_{n-1}, s_n]\}$$

where $n \leq T$, the maximum number of timesteps possible per episode. In an episode, the initial observation state s_0 is sampled from the initial conditions of the environment. The system is then actuated by an agent at each timestep with actions a_t until a termination condition is true. A termination condition can either mean that the controller successfully achieved the goal, a negative observation state was detected, or that the total timesteps allotted for the episode were reached.

Deep reinforcement learning (DRL) uses the same concepts and strategies as traditional RL but utilizes deep neural networks (DNN) to approximate function estimators (e.g. transition models, policy, value function, etc.), which are too large to represent with hash maps in complex environments. This work uses a model-free, policy optimization method known as proximal policy optimization (PPO) [23]. PPO uses one neural network to approximate the policy (actor) and a second neural network to approximate the value function (critic). Where the policy network is used to select actions from a given state, the value function is an estimate of the expected return an agent should receive from that state until the end of the episode, or rather how "good" that state is to be in. The actor controls the behavior of the agent, and the critic is used to evaluate that behavior. PPO also uses a clipping parameter to ensure that policy updates are relatively small, increasing the stability of the training process. The general learning framework of PPO consists of **collecting experiences** with the current policy, **policy improvement** based upon those experiences, and **policy evaluation**, which evaluates the actor and critic models as shown in Figure 1.

This work uses the implementation of the PPO algorithm from RLlib [17]. RLlib is an open-source RL library built upon Ray that facilitates and scales parallel computing across available computational resources during training.

2.1.2 ***Safe Reinforcement Learning.*** To address the need for safety in RL, safe reinforcement learning (SRL) incorporates safety into the training and/or execution of the agent. Recent research has addressed SRL by focusing on novel techniques in training [6], such as shielding strategies, action masking, and run time assurance (RTA) modules. This work implements and evaluates action masking and the RTA Simplex Architecture, which are described in Section 2.3.1 and Section 2.3.2.

2.1.3 ***Genetic Algorithms.*** Genetic algorithms (GAs) have also been used to solve RL-type problems [26], and the RL terms described above apply to this learning strategy as well. GAs mimic the evolutionary process of natural selection and are effective in a variety of problems, including scheduling and planning. Whereas RL

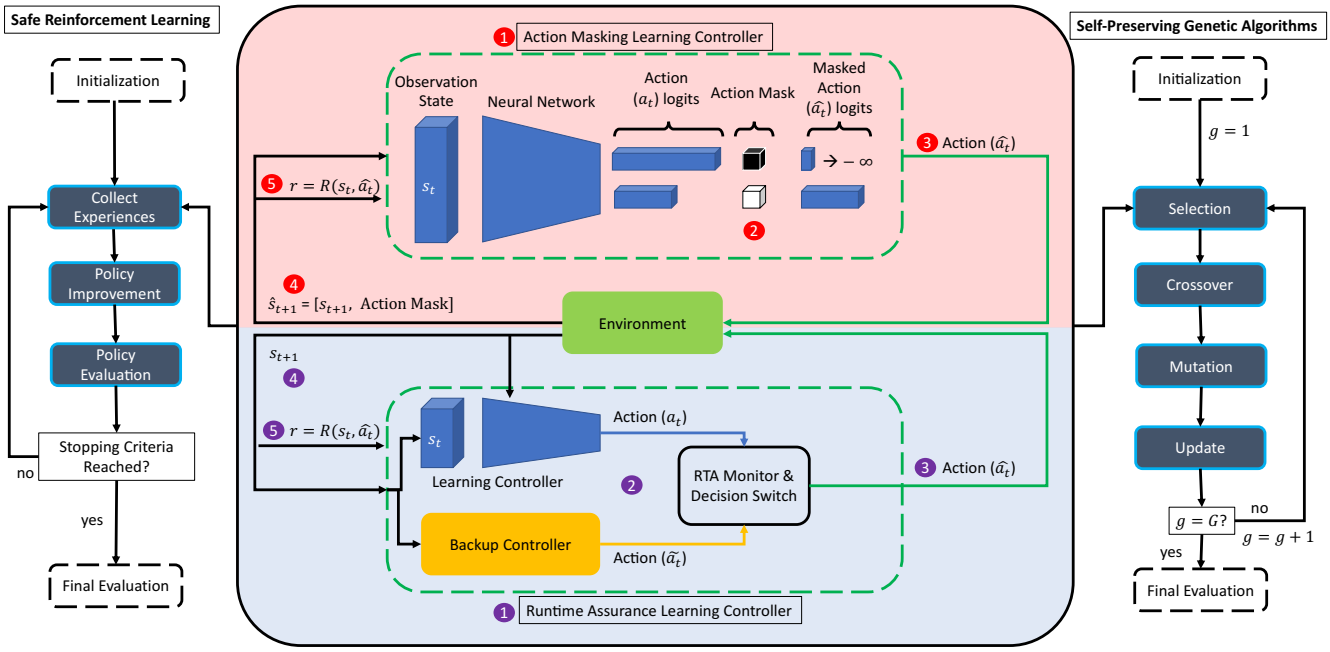


Figure 1: Action masking and RTA controllers in the training framework of Safe Reinforcement Learning and Self-Preserving Genetic Algorithms. There are 4 main parts to this diagram. The red, topmost section of the central block describes an *action masking learning controller* as indicated by the *red 1*. Using the current state of the system s_t as input, action logits are generated from the neural network controller. An action mask is applied to the action logits (*red 2*), where the black cube represents a mask of value 0, and the white cube represents a mask of value 1. From the resulting masked action logits, a safe action is sampled $\hat{a}_t$ (*red 3*) and passed to the environment. In response, the environment returns a next state $\hat{s}_{t+1}$ (*red 4*) and a reward r (*red 5*). The next state contains both the actual observation state and the corresponding action mask. The blue, bottom most section of the central block describes a *RTA learning controller* as indicated by the *purple 1*. Both the learning controller and the backup controller receive the current state s_t as input. The learning controller selects an action a_t, and the backup controller selects an action $\tilde{a}_t$, which are passed to the RTA Monitor & Decision Switch (*purple 2*). Based on the switching mechanism described in Section 2, a safe action, $\hat{a}$, is determined and passed to the environment (*purple 3*). In response, the environment returns a next state s_{t+1} (*purple 4*) and a reward r (*purple 5*). The leftmost side of the diagram shows the general learning strategy of *Safe Reinforcement Learning*, and the rightmost side of the diagram shows the general learning strategy of *Self-Preserving Genetic Algorithms*, both of which use experiences from the learning controllers in the central block.

uses experiences to update function approximator network weights, GAs are better understood as an educated guess-and-check learning strategy.

As shown in Figure 1, the general framework for a GA strategy consists of three main stages: **initialization**, **evolutionary cycle**, and **final evaluation**. Each of these stages is described in more detail below.

(1) **Initialization:** The generation value is set to $g = 1$ to represent the first generation of the evolutionary process, and a population $\mathcal{P}_g$ of n individuals is randomly generated for the first generation. Each individual is a potential solution to the problem at hand. For example, an individual could be an array of 10 floating point numbers such that $x \in [0, 1]$, where x is a single "gene" in the array, as shown below. $\mathcal{P}_1$ would then consist of n of these randomly generated arrays.

$$individual_1 = [.2, .4, .1, .6, .7, .3, .5, .4, .8, .9]$$
$$\vdots$$
$$individual_n = [.7, .2, .2, .3, .9, .5, .1, .6, .7, .3]$$
$$\mathcal{P}_1 = \{individual_{1:n}\}$$

(2) **Evolutionary Cycle:** The evolutionary cycle continues for G generations or until a stopping criterion has been reached.

(a) **Selection:** Each individual (*a particular solution*) of the population is scored according to a user defined fitness function f. For the example above, the fitness function could be $f = sum(individual)$, so the closer the individual is to all 1's, the more "fit" the individual is and the better the score. Based on this fitness score, individuals are selected to "reproduce offspring" for the next generation. While there are many selection strategies, this work

utilizes tournament selection, which randomly pairs individuals from $\mathcal{P}_g$ and the highest scoring individual is chosen as a parent. This selection strategy is used to select $2 \times n$ parents. Individuals can be selected multiple times to be a parent, reflective of nature. After enough parents have been selected to reproduce offspring, the crossover stage begins.

(b) **Crossover:** In crossover, the "genes" of the paired parents combine to create a new potential solution, or child, for the problem at hand. There are many strategies for crossover, including one-point, two-point, and uniform schemes. This work utilizes a two-point crossover strategy demonstrated below by the *TP* operator. Two points are randomly selected from the parents, and the solutions are combined at those two points.

$$parent_1 = [\mathbf{.2, .4, .1, .6, .7, .3, .5, .4, .8, .9}]$$
$$parent_2 = [.7, .2, .2, .3, .9, .5, .1, .6, .7, .3]$$
$$child = TP(parent_1, parent_2)$$
$$child = [\mathbf{.2, .4, .1}, .3, .9, .5, .1, .6, \mathbf{.8, .9}]$$

(c) **Mutation:** After n of children have been produced by crossover, each child solution is then randomly mutated. The rate of mutation m is designated by the user. For instance, a 20% mutation rate will change a gene 20% of the time. An example of the mutation is shown below by the M operator. Mutation helps to prevent the solution space from getting stuck in a local optimum.

$$child = [.2, .4, .1, .3, .9, .5, .1, .6, .8, .9]$$
$$child : M(child) = [.2, \mathbf{.7}, .1, .3, .9, \mathbf{.4}, .1, .6, .8, .9]$$

(d) **Update:** Once each child has been randomly mutated, the generation number is incremented by one ($g = g + 1$), and the current population $\mathcal{P}_g$ is set to the children just produced.

(3) **Final Evaluation:** After the set number of generations G has evolved or the stopping criteria has been reached, the resulting population of solutions is evaluated for multiple trials, and the highest scoring individual is the selected solution for the problem.

In this work, an individual is a neural network controller, which is evaluated on its capability to safely actuate a control system. Please see Section 4 for more details on the SPGA implementation.

2.1.4 ***Self-Preserving Genetic Algorithms.*** The same safety concerns associated with RL agents are important to consider with GA controllers. As such, this work applies the same safety concepts, RTA and action masking, to the GA-derived controllers during the training cycle. By incorporating safety into the fitness function and selection process, the authors believe that safer agents will be more likely to *survive*, leading to the derivation of a "safe" controller. This training paradigm has thus been labeled, *Self-Preserving Genetic Algorithms* (SPGA).

2.2 Safety Definition

In this work, a *safety condition* is a predicate over the state variables (the variables contained in the observation state) of the system, which is a boolean-valued function over those variables. For instance, a safety condition predicate used in this work is: ($x \geq -1.5 \wedge x \leq 1.5$). Here, x is a state variable in the observation state, which the learning controller uses to determine what action a_t to take at that timestep. A controller is considered *safe* if the safety condition holds true for all timesteps of an episode.

From this definition of safety, the percentage of safe actions can also be monitored. The percentage of safe actions, therefore, is the percentage of safe actions from the total actions taken in an episode that evaluate to $True$ from the boolean valued safety condition. The percentage of safe actions is used as an evaluation metric in this work.

2.3 Safety Mechanisms

The safety mechanisms used in this work are action masking and run time assurance.

2.3.1 ***Action Masking.*** Action masking is a safety technique that incorporates known knowledge about an environment into the training process of a controller [9] as shown in the red, topmost region of Figure 1. Based on the user's knowledge of the environment, a mask update function is added to the environment and reflected in the observation state. When a controller receives a state as input, it is also receiving this action mask, as shown by the **red 4** in Figure 1. The actual observation state of the environment is still used as input to the neural network policy of the controller. Once propagated through the network, the mask is applied to the output logits of the policy, as demonstrated by the **red 2** in Figure 1. The mask filters out unsafe actions by adjusting the probability distribution of possible actions. The mask is an array of 0s and/or 1s, where a 1 corresponds to a safe action and a 0 corresponds to an unsafe action. This array is then the input to a logarithmic function, which turns the 1s into 0s and the 0s into $-\infty$. These values are then added to the output logits of the controller, making the selection of unsafe actions 0. After the mask is applied, the resulting masked action logits are sampled from in order to calculate a safe action $\hat{a}_t$ that is passed to the environment, as shown by the **red 3** in Figure 1. Once the safe action has been executed in the environment, the environment responds with an updated observation state, including an updated action mask. The action mask not only prevents dangerous actions, but it also increases the probability of the safe action being taken in the policy when an action mask is no longer in use, as the neural network parameters are updated to reflect this action.

2.3.2 ***Run Time Assurance: Simplex Architecture.*** *Run time assurance* (RTA) methods ensure the safe behavior of safety critical systems. RTA monitors the system state and intervenes at runtime to assure specified constraints on the system are never violated. In this work, we focus on one approach to RTA, the *simplex architecture* [22].

The simplex architecture relies on two controllers, primary and backup, and a decision monitor, as shown by the **purple 2** in Figure 1. At run time, the decision monitor simulates the result of executing the output of the primary controller (in our case, the learning controller). If the resulting next state results in a $True$ boolean valued evaluation of the safety condition, then the primary

control output is passed through to the system ($\hat{a} = a_t$). If the simulated next state is not determined safe, then the backup control output is passed to the system instead ($\hat{a} = \tilde{a}_t$). This allows for the use of a complex controller while still maintaining the guarantees of the safety controller, which is designed to prevent unsafe states from occurring. One possible implementation for a simplex RTA filter is constructed as follows,

Simplex Filter

$$\hat{a} = \begin{cases} a_t & \text{if} \quad C(\phi(s_{t+1})) = 1 \\ \tilde{a}_t & \text{otherwise} \end{cases} \tag{1}$$

Here, $\phi(s_{t+1})$ represents a prediction of the future state, $C(\phi(s_{t+1}))$ is a boolean-valued evaluation of whether the predicted future state of the action a_t generated by the learning controller satisfies the safety condition, and $\tilde{a}_t$ represents the action of the backup controller if this evaluation is $False$.

3 DISCRETE ACTION ENVIRONMENTS AND SAFETY

The discrete action environments used in this work are from three different domains applicable to cyber physical systems (CPS)—control, routing, and operations. A discrete action environment means that the controller has discrete output options that are environment dependent. For instance, a *gridworld* controller can only actuate up, down, left, or right (discrete actions). Each of the discrete action environments use either an OpenAI Gym or an OR-Gym implementation of a classic problem from the problem space. Gym is a collection of environments used for developing and comparing reinforcement learning algorithms, and OR-Gym is a collection of class operations research problems for RL. In each of the environments, the learning agents are trained on varying levels of complexity or safety conditions. The different versions of each environment and the max reward obtainable, or solved score, for each are described in Table 1.

Table 1: Environment Versions

Env. Type	Env. Version	Solved Score
CartPole-v0 (*Control*)	$b = 0.25$	200
	$b = 0.50$	200
	$b = 0.75$	200
	$b = 1.00$	200
	$b = 1.25$	200
	$b = 1.50$	200
FrozenLake-v1 (*Routing*)	8x8 Grid	1.0
	16x16 Grid	1.0
	32x32 Grid	1.0
Knapsack-v0 (*Operations*)	5 Item	36.0
	50 Item	103.0
	100 Item	104.0

3.1 Control Environment

The control problem used in this work is the OpenAI Gym implementation of the classic *CartPole-v0* as described in [2]. The goal of this environment is to keep the pole upright for as long as possible ($T_{max} = 200$) by actuating the system with a leftward force or a rightward force based upon an observation state s_t that acts as an input to the controller. The observation state consists of the cart position x, cart velocity $\dot{x}$, the pole angle θ, and the pole angular velocity ω. The safety conditions associated with this environment are limits placed on the x state variable, referred to as boundaries b, as shown in Table 1. In the boolean-valued safety function discussed in Section 2.2, b corresponds to the predicate value to which x is compared ($x \geq -b \wedge x \leq b$).

3.2 Routing Environment

The routing problem used in this work is derived from *FrozenLake-v1*, where an agent must make it safely across a frozen lake toward a goal location while avoiding holes in the frozen lake by moving left (0), down (1), right (2), or up (3). The goal of this environment is to plan a route in order to safely reach the goal destination while traversing the dangerous terrain. In this environment, we look at three different sizes of grids as shown in Table 1. The larger the grid, the harder it should be for the agent to train as the number of possible routes increases from $8 \times 8 = 64^4 = 16777216$ to $32 \times 32 = 1024^4 = 1.10e^{12}$. The safety conditions associated with this environment are evaluations of the state types, which include *ice*, *hole*, or *goal*. If the state is a *hole*, it is considered unsafe. The boolean-valued safety function is therefore: ($stateType \neq hole$).

3.3 Operations Environment

The operations research environment used in this work is the classic Unbounded Knapsack problem from OR-Gym, which consists of several items with a weight and value. The agent must select from among these items with replacement to maximize the total value of the items selected while not exceeding the max weight allowed by the environment. Similar to the FrozenLake-v1 environment, the Knapsack-v0 environment focuses on three different versions that increase in complexity as shown in Table 1. The safety condition associated with this environment is an evaluation of the max weight possible. If the max weight is exceeded, it is considered unsafe. The boolean-valued safety function is therefore: ($currentWeight \leq maxWeight$).

4 EXPERIMENTAL METHODS

The SRL and SPGA implementations used in this work are discussed in more detail below.

4.1 SPGA Implementation

The genetic algorithm framework used in this work is described below.

(1) **Initialization:** A population of $n = 30$ neural networks are initialized with random weights from a uniform distribution. The number of input nodes and output nodes are environment dependent, where the number of input nodes corresponds to the observation state dimensions and the

number of output nodes corresponds to the action space dimensions. Each neural network has 1 hidden layer with 16 nodes. For example, the neural network dimensions for the CartPole environment are: 4 input, 16 hidden, and 2 output. The generation value is set to $g = 1$, representing the first generation of the evolutionary process.

(2) **Evolutionary Cycle:** The evolutionary cycle ends when $g = G = 500$ or when the average of the fittest individual from the last 30 generations is equal to the solved score of the environment.
   (a) **Selection:** The fitness function f is the cumulative reward, or return, acquired for a single episode. Each of the fitness functions incorporates the safety mechanism used during training (i.e., RTA or AM), where if triggered, it results in a negative reward. Tournament selection is used to select parents for crossover. n pairs of parents are selected for crossover.
   (b) **Crossover:** The pairs of parents are crossed over using the two-point strategy discussed in Section 2.1.3 to produce a child neural network. Here, two random numbers are selected between 0 and 16 (number of nodes in the hidden layer) using a uniform distribution. The values of parent one's weights up until the first crossover point are applied to the child, the second parent's weights are applied until the second crossover point, and the first parent's weights are applied from thereafter. This child is then added to the new generation of individuals. After each pair of parents has been crossed-over, there will be n children in the new generation.
   (c) **Mutation:** The individuals in the new generation are mutated with a mutation rate of $m = 0.2$. This randomly changes a value of the individual's neural network weights using a uniform distribution of generated values.
   (d) **Update:** the current population is set to the new generation, and g is incremented by 1 ($g = g + 1$).

(3) **Final Evaluation:** The resulting population of solutions is evaluated for one more episode. The individual achieving the highest cumulative reward is selected as the learned controller from the evolutionary process.

While this learning framework utilizes hyperparameters such as mutation rate m, generations G, population size n, and one neural network architecture, tuning methods were not utilized because the chosen values performed well across all experiments. These values, however, could be tuned with state-of-the-art tuning methods to improve results in the future.

4.2 PPO Implementation

The PPO implementation used in this work is from RLlib. The hyperparameter and architecture values are kept as the default PPO RLlib values[1] to match the scheme utilized with SPGA.

Training is stopped when the max number of training steps ($N = 500$) is reached or when the average reward of the last 30 training steps is greater than or equal to the solved score of the environment. The solved scores for each environment are listed in Table 1.

[1]https://github.com/ray-project/ray/blob/master/rllib/algorithms/ppo/ppo.py

4.3 Safety Implementation

With both safety mechanisms (i.e., RTA and AM), the agent receives a negative reward if the mechanism is triggered. For instance, if the RTA module described in Section 2.3.2 switches to the backup controller, the agent will receive a negative environment dependent reward. Similarly, if the action mask contains a 0, meaning that the action mask is in use to prevent the agent from making an unsafe action, the agent will also receive a negative environment dependent reward.

The reward functions used in these experiments were not extensively shaped, meaning that they could have been optimized further to increase agent performance. A simple grid search of reward functions was used to select the negative rewards associated with the safety mechanisms in this work. Both SRL and SPGA use the same reward function.

5 EXPERIMENTAL OVERVIEW

In order to evaluate and compare SPGA, the following methods described in Section 4[2] are implemented:

(1) *SPGA-AM*: SPGA with Action Masking
(2) *SPGA-RTA*: SPGA with Run Time Assurance
(3) *SRL-AM*: SRL (PPO) with Action Masking
(4) *SRL-RTA*: SRL (PPO) with Run Time Assurance

Each of these methods are trained in each version of CartPole-v0, FrozenLake-v1, and Knapsack-v0. Each version increases in complexity, providing insight into the robustness of each of the training methods. These experiments were conducted on an 2.3 GHz 8-Core Intel Core i9 processor with 16 GB 2667 MHz DDR4 of memory.

5.1 Training

Each of the implementations (i.e., SRL-AM, SRL-RTA, SPGA-AM, SPGA-RTA) are trained on the same 5 random seeds in each environment and with each safety condition mentioned above. A seed is used to set the randomization of both the environment and distributions of random number generators. The training time across these five seeds is used to obtain an average wall training time metric. The five random seeds generated with *Numpy Random Randint* in this work were 2, 4, 27, 36, and 98.

5.2 Testing

To test the learned controllers, a random seed is then selected from the training seeds to be used across all trained agents for episode rollouts. Using the *Numpy Random Choice* function, seed 4 was selected from the five seeds above as the indicated trained controller to use for every episode. Each of the selected learned controllers were evaluated on 500 episodes. An episode starts from a random initialization of the system, and the controller actuates the system until a termination condition is reached. The average of the cumulative reward across all episodes, or expected return, is then used as an evaluation metric. Additionally, the number of safe actions in each episode is averaged to calculate the percentage of safe actions, where the safety conditions are discussed in Section 3.

[2]All code is available at https://doi.org/10.5281/zenodo.7706773, or https://github.com/pkrobinette/spga

Table 2: Analysis of Self-Preserving Genetic Algorithms and Safe Reinforcement Learning

			SPGA			SRL			
Environment	Safety Method	Variant/Safety Condition	Training Time (s)	Expected Return	(%) Safe Actions	Training Time (s)	Expected Return	(%) Safe Actions	**xSpeedUp**
CartPole-v0	RTA	1.5	11.09 ± 2	200.0 ± 0	100.0 ± 0	178.03 ± 12	198.8 ± 0.8	100.0 ± 0	**16.05**
(Control)		1.25	10.90 ± 1	200.0 ± 0	100.0 ± 0	192.71 ± 10	196.4 ± 1.3	100.0 ± 0	**17.68**
		1.0	10.57 ± 1	199.7 ± 0.2	100.0 ± 0	184.73 ± 4	199.8 ± 0.2	100.0 ± 0	**17.48**
		0.75	15.41 ± 4	166.5 ± 2.0	100.0 ± 0	149.72 ± 7	200.0 ± 0	100.0 ± 0	**9.72**
		0.5	14.71 ± 10	199.9 ± 0.0	100.0 ± 0	122.97 ± 1	199.6 ± 0.5	100.0 ± 0	**8.36**
		0.25	12.84 ± 8.65	158.5 ± 3.2	96.1 ± 0.6	150.36 ± 10	199.9 ± 0.2	100.0 ± 0	**11.71**
	AM	1.5	38.68 ± 9	200.0 ± 0	100.0 ± 0	147.90 ± 19	198.5 ± 1.0	100.0 ± 0	**3.82**
		1.25	56.46 ± 18	200.0 ± 0	100.0 ± 0	418.02 ± 162	199.46 ± 0.3	100.0 ±0	**7.40**
		1.0	58.24 ± 15	200.0 ± 0	100.0 ± 0	246.78 ± 8	200.0 ± 0	100.0 ± 0	**4.24**
		0.75	51.23 ± 14	200.0 ± 0	100.0 ± 0	418.55 ± 159	199.71 ± 0.4	100.0 ± 0	**8.17**
		0.5	167.73 ± 19	200.0 ± 0	100.0 ± 0	159.06 ± 3	196.44 ± 1.4	100.0 ± 0	0.95
		0.25	132.44 ± 68	200.0 ± 0	100.0 ± 0	156.45 ± 3	200.0 ± 0	100.0 ± 0	1.18
FrozenLake-v1	RTA	8x8 Grid	5.49 ± 0.0	1.0 ± 0.0	100.0	18344.54 ± 7396.0	1.0 ± 0.0	100.0	**3341.45**
(Routing)		16x16 Grid	13.63 ± 5.0	0.86 ± 0.1	100.0	18196.31 ± 8220.0	1.0 ± 0.0	100.0	**1335.02**
		32x32 Grid	55.39 ± 27.0	0.54 ± 0.34	100.0	20273.63 ± 10866.0	0.0 ± 0.0	100.0	**366.02**
	AM	8x8 Grid	8.76 ± 0.0	1.0 ± 0.0	100.0	109.17 ± 4.0	1.0 ± 0.0	100.0	**12.46**
		16x16 Grid	12.72 ± 1.0	1.0 ± 0.0	100.0	368.78 ± 160.0	1.0 ± 0.0	100.0	**28.99**
		32x32 Grid	19.99 ± 1.0	1.0 ± 0.0	100.0	5323.37 ± 6448.0	1.0 ± 0.0	100.0	**266.3**
Knapsack-v0	RTA	5 Items	1.05 ± 0.0	36.0 ± 1.0	100.0	557.43 ± 361.0	36.0 ± 5.0	100.0	**530.89**
(Operations)		50 Items	50.95 ± 44.0	95.0 ± 1.0	100.0	15493.49 ± 15085.0	102.0 ± 22.0	100.0	**304.09**
		100 Items	55.33 ± 3	99.0 ± 0.0	100.0	17245.11 ± 19016.0	99.0 ± 13.0	100.0	**177.77**
	AM	5 Items	2.47 ± 0.0	36.0 ± 0.0	100.0	114.53 ± 4.0	34.2 ± 3.0	100.0	**46.37**
		50 Items	21.13 ± 29.0	94.0 ± 0.0	100.0	458.39 ± 163.0	103.0 ± 18.0	100.0	**21.69**
		100 Items	31.98 ± 48.0	104.0 ± 6.0	100.0	2285.08 ± 1706.0	104.0 ± 25.0	100.0	**71.45**

5.3 Evaluation Metrics

From training and testing, we evaluate each safe learning method using three metrics:

(1) **Training Time:** The average wall time across 5 random seeds to complete training.
(2) **Expected Return:** The average episode reward for 500 episode rollouts using a learned policy.
(3) **Safe Actions (%):** The number of safe actions from the total actions taken in an episode that evaluate to *True* from the boolean valued safety condition. This value is averaged across 500 episode rollouts.

Table 3: Geometric Mean xSpeedUp of SPGA vs. SRL

Environment	Safety Method	Mean xSpeedUp
CartPole-v0	RTA	12.95
	AM	3.21
FrozenLake-v1	RTA	1177.54
	AM	45.82
Knapsack-v0	RTA	306.16
	AM	41.58
All Environments		**55.28**

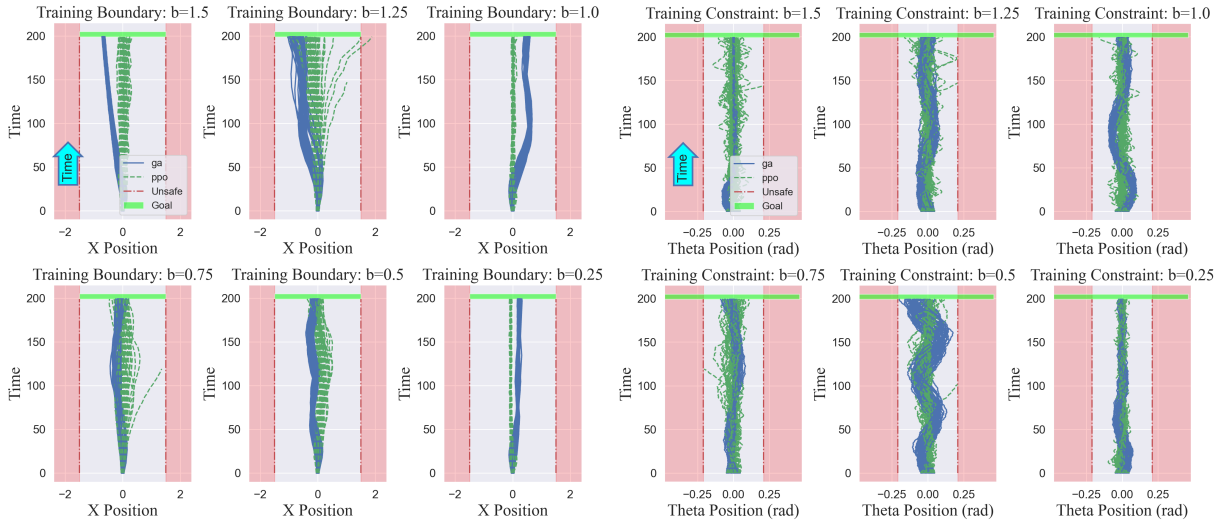
6 RESULTS AND DISCUSSION

Table 2 shows the training and test results of SPGA and SRL in the three discrete action environments using three different metrics described in Section 5.3: *Training Time*, *Expected Return*, and *Safe Actions*. The results of each environment are evaluated further in the following sections.

6.1 Control Results (CartPole-v0)

Figure 2 compares 50 episode trajectories of the SRL-AM trained controller and the SPGA-AM trained controller across different safety conditions. A trajectory, in this case, is the time ordered set of state variables for each timestep in an episode. The red zones in Figure 2a indicate termination conditions of that state variable for an episode, and the green zones highlight the goal condition ($t = T_{max} = 200$). While not trained for, the angle of the state is an environment termination condition, which gives insight into why an episode might not reach the max timesteps during a rollout, shown by a trajectory stopping abruptly. Because the safety conditions are not considered as termination conditions, the trajectory of a controller could reach an "unsafe state" but still continue past this timestep. If a termination condition is reached, however, the episode is terminated, resulting in a lower cumulative reward for that episode.

The plots in Figure 2 highlight the effects of training under different safety conditions b. As the size of the safe region decreases during training, resulting in a smaller safe range, the learned controller actuates closer to the origin. Additionally, almost all the



(a) Position (x) **(b) Angle (θ)**

Figure 2: CartPole-v0 SPGA-Action Mask (SPGA-AM) vs. SRL-Action Mask (SRL-AM) for 50 episode trajectories in different training boundary conditions b. 2a describes the x state variable and 2b describe the θ state variable of the observation state. The red zones indicate the environment termination condition threshold, and the green zone near the top of each plot highlights the goal condition of the controller, which is to successfully actuate the system for 200 timesteps.

episodes that terminate early are a result of the θ termination condition being violated, except in the *Training Boundary: b=1.25* plot of Figure 2a. Here, the trajectory of the SRL-AM agent violates the safety condition. In this case, SPGA-AM is both safer and trains faster. Most of the time, however, SPGA and SRL are the same in terms of safety, but SPGA tends to train faster as shown in Table 2.

6.2 Routing Results (FrozenLake-v1)

Each of the learning methods are able to successfully learn a safe route in each version of the FrozenLake-v1 environment, as shown in Table 2. The routes learned by the SPGA controllers, however, are more efficient, as fewer actuations are needed to reach the goal state. For instance, in the FrozenLake-v1 16x16 Grid shown in Figure 3, SPGA-RTA changes direction 2 times whereas SRL-RTA changes direction 14 times. In addition to having fewer actuations, the SPGA controllers are faster to train, as highlighted in Table 3. SPGA-RTA is 1177.54x faster than SRL-RTA, and SPGA-AM is 45.82 times faster than SRL-AM, demonstrating the benefit of the exploration strategy of a genetic algorithm.

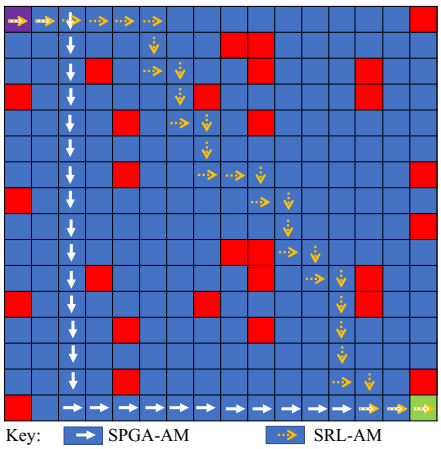
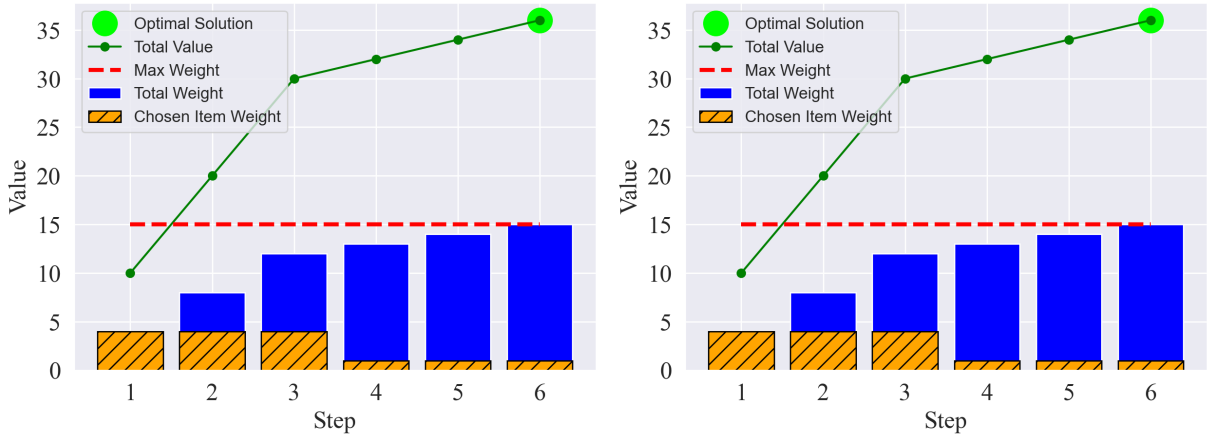


Figure 3: FrozenLake-v1 SPGA-Action Mask (SPGA-AM) vs. SRL-Action Mask (SRL-AM) learned path trajectories for the 16x16 Grid.

6.3 Operations Results (Knapsack-v0)

In the Knapsack-v0 environments, SPGA trained learning methods are faster to train when compared to the SRL trained agents and have equivalent safety evaluations as shown in Table 2. While not as significant an improvement compared to the FrozenLake-v1 environment, SPGA-RTA is 306.16x faster than SRL-RTA, and SPGA-AM is 41.58x faster than SRL-AM. In each version of the knapsack environment, both learning methods key in on one or two items with high value to weight ratio, picking between them in each step as demonstrated in Figure 4. Each safe learning method learns the optimal solution of selected items in each version of Knapsack-v0, except for in the 50 Item version. In the 50 Item version, SPGA-RTA and SRL-RTA are unable to learn the optimal solution during training. This could be due to the random nature of RTA. If the RTA safe controller is triggered, a viable next state is randomly selected, preventing the learning agent from using a strategy to select between the available options. In action masking, though, the agent is still able to apply its knowledge to selecting the next state, resulting in a more self-guided exploration of the state space.



(a) SPGA-RTA **(b) SRL-RTA**

Figure 4: A comparison of SPGA-RTA and SRL-RTA in the 5 Item Knapsack-v0 environment. The orange striped bars represent the weight of the item picked for that step, the blue solid bars represent the cumulative weight of the knapsack, the green solid line shows the cumulative value of the knapsack, the dashed red line shows the max weight allowed in the knapsack, and the lime green circle represents the optimal value of the environment. Both learning strategies end up learning the same policy; SPGA-RTA, however, does so in less time.

6.4 Training Time

The training time of the SPGA learned controllers, SPGA with Action Masking (SPGA-AM) and SPGA with Run Time Assurance (SPGA-RTA), is significantly faster compared to SRL by a geometric mean of 55.28x. SPGA achieves a speedup of up to 3341.45x the average wall training time compared to the SRL methods. With respect to the safety assurance methods, RTA usually trains faster than its action masking counterpart for each experiment in SPGA, even though both safety methods have extra computational steps: specifically, RTA has the probe step, and action masking has calculating the mask. Also, the overall effect of using SPGA for safe learning increases as the complexity of the environment increases, as shown by the correlation between environment difficulty and speedup value. In terms of safety conditions, the smaller the safety condition, the smaller the speedup observed. As the safety condition decreases, the range of available safe states decreases. This could result in more time needed for both learning methods to learn a more precise controller, one that can operate in a smaller range.

6.5 Safety of Controllers

In terms of safety, SPGA and SRL demonstrate similar results, achieving high levels of safety within each of the environments. In reference to the safety methods used in this work, controllers trained with action masking achieve higher percentages of safe actions than those trained with RTA, as shown in Table 2.

6.6 Risks to Validity

While this work provides an extensive evaluation of SPGA, there are design choices that could affect these results. Both SPGA and SRL utilize early stopping criterion in the training process. While they both rely on the last thirty training steps or evolutionary cycles, SRL is using the mean of episode rollouts while SPGA is using the max score of the fitness function of the population. The mean of SRL is necessary because the episode rollouts are stemming from the same controller policy, which can vary in performance as a result of the initial state of the system. Additionally, the max is necessary in SPGA because of the mutation operator, which leads to poor performing individuals as well as high performing individuals for every generation.

This work also uses default hyperparameter values for each learning method. We leave architecture and hyperparameter tuning for future work. Even without hyperparameter tuning, though, SPGA trains quickly.

7 CONCLUSIONS AND FUTURE WORK

This work presents a novel approach to safe learning termed Self-Preserving Genetic Algorithms (SPGA) and demonstrates the effectiveness of this approach in three different discrete action control environments compared to Safe Reinforcement Learning (SRL) methods. SPGA achieves similar performance results to SRL trained controllers in each of the discrete action environments used in this work, while training significantly faster (an average of 55.28x). There are many potential use cases for using SPGA, including routing, operations, and control problems for cyber physical systems. The ability to solve these types of problems will be paramount for the future of smart technologies, such as smart transportation, smart cities, smart buildings, smart grids, and smart factories, which rely on efficient and safe control. A major benefit of using SPGA is with the occurrence of resource constraints (e.g., monetary constraints, time constraints). For instance, if a user is utilizing cloud resources to train a controller for a continuous stirred-tank reactor, the difference between 24 hours of compute time and 0.4 hours

(55.28x faster) of compute can make a drastic difference in computational costs, which magnifies as problems increase in complexity and the required computation time increases.

While this work provides an in-depth analysis of different learning controllers and safety methods, more work is needed to analyze Self-Preserving Genetic Algorithms as a training method for cyber physical systems. Possible areas of future work are listed below.

7.1 Environments

Evaluating SPGA in environments with larger action and state spaces would help to further analyze the robustness of this solution, as well as provide insight into potential use cases.

7.2 SPGA Hyperparameter Optimization

Hyperparameter and architecture tuning methods are used in order to increase overall learning and controller performance. Common optimization algorithms include random search, grid search, and Bayesian optimization methods, which can work in tandem with schedulers like Asynchronous HyperBand, an early-stopping scheduler. Future work could implement hyperparameter and architecture optimization into the SPGA training framework in order to further improve performance.

Overall, this work provides insight into a novel framework for safe learning and lays the groundwork for many interesting areas of future work related to Self-Preserving Genetic Algorithms.

ACKNOWLEDGMENTS

This paper was supported in part by a fellowship award under contract FA9550-21-F-0003 through the National Defense Science and Engineering Graduate (NDSEG) Fellowship Program, sponsored by the Air Force Research Laboratory (AFRL), the Office of Naval Research (ONR), and the Army Research Office (ARO). The material presented in this paper is also based upon work supported by the National Science Foundation (NSF) through grant number 2028001, and the Air Force Office of Scientific Research (AFOSR) under contract numbers FA9550-22-1-0019 and FA9550-23-1-0135. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of DoD or NSF.

REFERENCES

[1] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[2] Andrew G Barto, Richard S Sutton, and Charles W Anderson. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics* 5 (1983), 834–846.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[4] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. 2021. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning* 110, 9 (2021), 2419–2468.

[5] Aidin Ferdowsi, Ursula Challita, Walid Saad, and Narayan B Mandayam. 2018. Robust deep reinforcement learning for security and safety in autonomous vehicle systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 307–312.

[6] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16, 1 (2015), 1437–1480.

[7] Mevludin Glavic, Raphaël Fonteneau, and Damien Ernst. 2017. Reinforcement learning for electric power system decision and control: Past considerations and perspectives. *IFAC-PapersOnLine* 50, 1 (2017), 6918–6927.

[8] Yanrong Hu and Simon X Yang. 2004. A knowledge based genetic algorithm for path planning of a mobile robot. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, Vol. 5. IEEE, 4350–4355.

[9] Shengyi Huang and Santiago Ontañón. 2020. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171* (2020).

[10] Christian D Hubbs, Hector D Perez, Owais Sarwar, Nikolaos V Sahinidis, Ignacio E Grossmann, and John M Wassick. 2020. Or-gym: A reinforcement learning library for operations research problems. *arXiv preprint arXiv:2008.06319* (2020).

[11] Yi Jiang, Jialu Fan, Tianyou Chai, and Frank L Lewis. 2018. Dual-rate operational optimal control for flotation industrial process with unknown operational model. *IEEE Transactions on Industrial Electronics* 66, 6 (2018), 4587–4599.

[12] Junqi Jin, Chengru Song, Han Li, Kun Gai, Jun Wang, and Weinan Zhang. 2018. Real-time bidding with multi-agent reinforcement learning in display advertising. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2193–2201.

[13] Anssi Kanervisto, Christian Scheller, and Ville Hautamäki. 2020. Action space shaping in deep reinforcement learning. In *2020 IEEE Conference on Games (CoG)*. IEEE, 479–486.

[14] Charles Karr and L Michael Freeman. 1998. *Industrial applications of genetic algorithms*. Vol. 5. CRC press.

[15] Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32, 11 (2013), 1238–1274.

[16] Christopher Lazarus, James G Lopez, and Mykel J Kochenderfer. 2020. Runtime safety assurance using reinforcement learning. In *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE, 1–9.

[17] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*. PMLR, 3053–3062.

[18] Xing Liu, Hansong Xu, Weixian Liao, and Wei Yu. 2019. Reinforcement learning for cyber-physical systems. In *2019 IEEE International Conference on Industrial Internet (ICII)*. IEEE, 318–327.

[19] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. 2019. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys & Tutorials* 21, 4 (2019), 3133–3174.

[20] Kim F Man, Kit Sang Tang, and Sam Kwong. 2012. *Genetic algorithms for control and signal processing*. Springer Science & Business Media.

[21] Faizan Rasheed, Kok-Lim Alvin Yau, Rafidah Md Noor, Celimuge Wu, and Yeh-Ching Low. 2020. Deep Reinforcement Learning for Traffic Signal Control: A Review. *IEEE Access* (2020).

[22] Jose G Rivera, Alejandro A Danylyszyn, Charles B Weinstock, Lui R Sha, and Michael J Gagliardi. 1996. *An Architectural Description of the Simplex Architecture*. Technical Report. Carnegie-Mellon University.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[24] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[25] Burrhus F Skinner. 1963. Operant behavior. *American psychologist* 18, 8 (1963), 503.

[26] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567* (2017).

[27] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[28] Lianfang Tian and Curtis Collins. 2004. An effective robot trajectory planning method using a genetic algorithm. *Mechatronics* 14, 5 (2004), 455–470.

[29] Jianping Tu and Simon X Yang. 2003. Genetic algorithm based path planning for a mobile robot. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, Vol. 1. IEEE, 1221–1226.

[30] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.

[31] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N Zare, and Patrick Riley. 2019. Optimization of molecules via deep reinforcement learning. *Scientific reports* 9, 1 (2019), 1–10.