

A Two-Phase Safe Reinforcement Learning Framework for Finding the Safe Policy Space

A. J. Westley^a and Gavin Rens^b

*Computer Science Division, Stellenbosch University, Stellenbosch, South Africa
alexanderjwestley@gmail.com, gavinrens@sun.ac.za*

Keywords: Safe Reinforcement Learning, Constrained Markov Decision Processes, Safe Policy Space, Violation Measure.

Abstract: As reinforcement learning (RL) expands into safety-critical domains, ensuring agent adherence to safety constraints becomes crucial. This paper introduces a two-phase approach to safe RL, Violation-Guided Identification of Safety (ViGIS), which first identifies a safe policy space and then performs standard RL within this space. We present two variants: ViGIS-P, which precalculates the safe policy space given a known transition function, and ViGIS-L, which learns the safe policy space through exploration. We evaluate ViGIS in three environments: a multi-constraint taxi world, a deterministic bank robber game, and a continuous cart-pole problem. Results show that both variants significantly reduce constraint violations compared to standard and β -pessimistic Q-learning, sometimes at the cost of achieving a lower average reward. ViGIS-L consistently outperforms ViGIS-P in the taxi world, especially as constraints increase. In the bank robber environment, both achieve perfect safety. A Deep Q-Network (DQN) implementation of ViGIS-L in the cart-pole domain reduces violations compared to a standard DQN. This research contributes to safe RL by providing a flexible framework for incorporating safety constraints into the RL process. The two-phase approach allows for clear separation between safety consideration and task optimization, potentially easing application in various safety-critical domains.

1 INTRODUCTION


Reinforcement learning (RL) is a subfield of machine learning wherein an autonomous agent explores a given environment with the goal of learning how to complete a task. RL has seen a large increase in popularity over the last decade due to its use in the fields of robotics as well as video game and board game AI (Silver et al., 2016). RL has a key advantage: programmers are not required to specify how the agent completes its task (Kaelbling et al., 1996). Instead, they need only design a system that rewards the agent for good actions and punishes it for bad actions; a punishment can be seen as a negative reward.


With RL being applied to an ever-increasing array of problems, safety has become a growing concern. Regular RL models seek only to maximise their rewards, and thus provide no guarantees that they will take safety into account when learning or performing tasks (Srinivasan et al., 2020). This is often not an issue, but in environments where safety is paramount the agent should also endeavour to stick within the

given safety constraints. For example, if you are building a self-driving car, then crashing even once could result in severe damage to the car. Under standard RL regimes, the only way for the car to know how not to crash would be to receive negative rewards for crashing, which requires the car to crash. On the other hand, if the car seeks only to take the safest action, then the safest action would likely be to remain still and never leave its parking space. In this case, the car will never complete its task. This underscores the need for effective safe RL techniques.

Safe RL is a subtopic of RL wherein algorithms endeavour to keep a balance between safety and performance across a wide array of problem spaces (García and Fernández, 2015). Designing algorithms that can strike this balance is a difficult task, and the necessary approach is often dependent on the problem at hand. To approach these problems various techniques have emerged, including constraint optimisation, model-based RL, providing predefined safety rules, or modifying the reward by heavily penalising unsafe actions.

This research proposes a two-phase RL approach designed to ensure safety during both the learning

^a  <https://orcid.org/0009-0009-5497-1375>

^b  <https://orcid.org/0000-0003-2950-9962>

and implementation phases. The algorithm, called Violation-Guided Identification of Safety (ViGIS), first learns a safe policy space, then performs standard RL within the predetermined safe policy space. The process of learning the safe policy space is based on Constrained Markov Decision Processes (Altman, 1999), using a violation measure (Nana, 2023) to quantify each policy’s safety. There are two main novelties to ViGIS: it is agnostic of the underlying RL algorithm used during phase-two, and violation tolerances to be specified for each constraint, thus forming a preferential order between the constraints. From these constraints, the agent can precompute or learn the safe policy space in silico, providing a safety bootstrap for the goal-learning process. With this in mind, we address the following questions: How can we analytically determine the safe policy space when the transition function and constraints are given? How does analytically determining the safe policy space perform compared to learning it empirically? How does learning only within the safe policy space affect an agent’s performance?

The rest of this paper is structured as follows: Section 2 sets out the necessary theoretical background for the paper by explaining Markov Decision Processes and Constrained Markov Decision Processes, then introducing the notion of policies and safe policy spaces. Section 3 provides an overview of recent safe RL approaches, with a particular focus on a Master’s thesis, which serves as inspiration to the research presented in this paper (Nana, 2023). In Section 4, the two-phase algorithm is introduced, its underlying mathematics is explained, and a pseudocode is provided. Section 5 describes the various testing environments along with their respective safety constraints and success criteria, after which it evaluates the algorithm in these testing environments. The two-phase agent is compared to a regular RL agent to assess its safety, as well as its overall performance and efficiency. To end off, Section 6 summarises the work done in this paper and provides some closing remarks on potential extensions of the work.

2 BACKGROUND

2.1 Markov Decision Processes

A Markov Decision Process (MDP) is a framework for modelling fully observable, stochastic, sequential decision problems. In reinforcement learning, MDPs are used to model interactions between an agent and its environment. MDPs have Markovian state transitions, meaning the next state depends only

on the current state (and action). Constraints can be added to a standard MDP, making it a constrained MDP (CMDP). Throughout this paper, unless explicitly stated otherwise, MDP refers to an unconstrained MDP.

2.1.1 Unconstrained MDPs

An MDP can be defined by the tuple $\langle \mathbf{S}, \mathbf{A}, \mathbf{T}, \mathbf{R} \rangle$ (Russell and Norvig, 2010):

- \mathbf{S} is the set of all states which exist within the MDP – the state space.
- \mathbf{A} is the set of all actions which can be performed in each state – the action space.
- $\mathbf{T} : \mathbf{S} \times \mathbf{A} \rightarrow \text{Distr}(\mathbf{S})$ is the transition function which gives the probabilities of transitioning from some state s to another state s' when an action a is performed. For a particular s, s' and a , this function is defined as:

$$P_a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a) \quad (2.1)$$

- $\mathbf{R} : \mathbf{S} \times \mathbf{A} \rightarrow \mathbb{R}$ is the reward function that calculates the value of each state-action pair. This expresses the instantaneous benefit of performing action a in state s . The reward gained at a particular time t is notated as R_t .

The goal of solving an MDP is to find a sequence of actions which maximise the total reward gained according to \mathbf{R} .

2.1.2 Constrained MDPs

In environments where some states are undesirable, we can extend the MDP framework by adding constraints, leading to a CMDP. In this case, the tuple becomes $\langle \mathbf{S}, \mathbf{A}, \mathbf{T}, \mathbf{R}, \mathbf{C} \rangle$ (Altman, 1999). All elements of the tuple remain the same, but for the addition of \mathbf{C} , which is expressed as

$$\{c_i : \mathbf{S} \times \mathbf{A} \rightarrow \mathbb{R} | i = 1, \dots, k\} \quad (2.2)$$

where k is the number of constraints in the system (Ge et al., 2019). \mathbf{C} specifies which states in the state space are not safe. It can reduce the state and action spaces that may be explored in the MDP or it can alter the reward function directly. With \mathbf{C} added to the MDP framework the objective is no longer to only maximise the reward gained according to \mathbf{R} , but to also abide by the constraints specified by \mathbf{C} . These constraints can be set for various reasons, but in this study they will mostly be safety constraints. Fig. 1 shows how CMDPs fit into the context of RL.

2.2 Policy and Safe Policy Space

The goal of any RL algorithm is for the agent to find the best actions in order to reach a given objective.

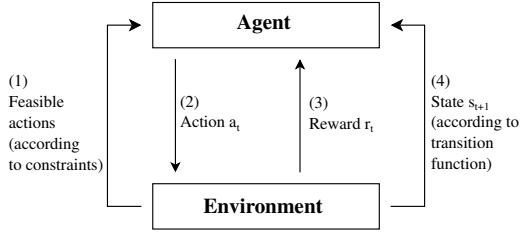


Figure 1: An abstract representation of a CMDP through the lens of reinforcement learning.

These actions are usually expressed in the form of a policy. A policy is a set of state-action pairs which indicates the best action to perform from any given state.

The set of all possible policies for a particular problem is called the policy space of the problem. The policy space grows exponentially in size with the action and state spaces, and thus can become enormous for discrete problems, and even infinite for continuous problems (Mutti et al., 2022). All policies, even those which do not complete the goal are included in the policy space. In safe RL, we are not interested in policies which would violate the safety constraints, so we can rather look at the subset of policies which adhere to these constraints. This policy subspace is called the *safe policy space*.

3 RELATED WORKS

3.1 Safe Reinforcement Learning

Safe RL has existed as a topic for over a decade and thus there has been a significant amount of research done in the field (García and Fernández, 2012; Gaskett, 2003; Driessens and Džeroski, 2004; Geramifard et al., 2011). In this part, we summarise and discuss different safe RL approaches with reference to a taxonomy proposed by García and Fernández (2015). According to this taxonomy, there are two overarching categories of Safe RL techniques, which entail modifying either the *optimization criterion* or the *exploration process*. We end off the section by discussing where ViGIS fits into the field of safe RL.

3.1.1 Optimization Criterion

One over-arching approach to safe RL is to directly modify the optimization criterion. The three most common modifications are: *worst case*, *risk-sensitive*, and *constrained* criterion.

A *worst case* criterion is one which aims to find a policy which maximizes the expected reward in the

worst-case scenario, when there is uncertainty in the environment. Gaskett (2003) addressed the stochastic environment problem by proposing a modified Q-Learning algorithm, called β -pessimistic Q-Learning. β -pessimistic Q-Learning aims to strike a balance between maximising the reward of the worst case and the best case. It does this using the following update equation:

$$Q_{\beta}(s, a) \leftarrow (1 - \alpha)Q_{\beta}(s, a) + \alpha \left[R_{t+1} + \gamma \left((1 - \beta_p) \max_{a' \in \mathbf{A}} Q(s', a') + \beta_p \min_{a' \in \mathbf{A}} Q(s', a') \right) \right], \quad (3.1)$$

where $\beta_p \in [0, 1]$ is the hyperparameter which tunes the agent’s consideration of risk. Larger β_p values make the agent more averse to risk.

Risk-sensitive criteria contain a hyperparameter which tunes the agent’s consideration of risk. This hyperparameter can be tuned by the programmer to specify a subjective balance between risk and reward. Inside the criterion, the hyperparameter is usually incorporated into an exponential function, or a weighted sum of risk and reward.

A *constrained* criterion is similar to a typical constrained optimization problem, as the goal is to find the optimal solution which satisfies a given set of constraints. As per the name, these criteria usually take the form of a CMDP.

3.1.2 Exploration Process

The other approach is to modify the exploration process directly. This is usually done by providing the agent with external knowledge in the form of: *initial knowledge*, or *teacher advice*.

Providing the agent with *initial knowledge* gives it a leg-up at the beginning of the learning process. The agent can use this knowledge to avoid some undesirable parts of the state space completely, which reduces the size of the state space which needs to be explored.

An agent can also be provided with *teacher advice* during the learning process. This approach requires the presence of some form of teacher: human or otherwise. The teacher can then provide the agent with advice whenever the agent asks or the teacher deems it necessary. An advantage of this approach is that it allows the teacher to manually prevent the agent from making choices which could have dangerous consequences.

3.1.3 Model-Based Look-Ahead Approaches

In addition to the approaches discussed above, there have been some promising model-based algorithms to

preempt any constraint violations. These approaches are the most similar to ViGIS, but ViGIS differs from these because it functions as a framework wherein other RL methods can be used, and it finds the safe policy space of a problem domain, rather than employing a direct look-ahead strategy. One example that has shown promise is the notion of probabilistic shields (Yang et al., 2023). This method involves logically modeling the environment and its constraints, then using this model to determine the respective probabilities of violating these constraints. An alternative to shielding is to use model-based rollouts to predict future violations. This approach, called Safe Model-Based Policy Optimization (SMBPO), was presented by (Thomas et al., 2024). SMBPO uses Gaussian dynamics models during rollouts to assess the safety of each trajectory, then heavily penalises trajectories that are deemed unsafe. In addition to the reasons already discussed, SMBPO also differs from ViGIS in that ViGIS does not make use of any rollout scheme. One last notable approach is Safe Upper Confidence Bound Value Iteration (SUCBVI), proposed by (Xiong et al., 2023). SUCBVI is an extension of UCBVI to incorporate step-wise constraints. SUCBVI is similar to ViGIS in its use of step-wise constraint avoidance as opposed to long-term violation minimisation. It also bears similarity to ViGIS due to it operating as a framework, rather than a stand-alone algorithm.

3.2 Violation Measure

ViGIS makes use of the violation measure that was formulated by Nana (2023). This violation measure is centred around the idea that, in many problems, it may not be possible to know with certainty whether some state-action pairs will lead to a safety violation. To deal with these problems it makes sense to model risk as the probability that performing action a in state s will lead to a dangerous situation. In the violation measure formulation, a tolerance is specified for each constraint of how high this probability can be, which allows the weight of each constraint to be carefully tuned. The violation measure function is expressed as

$$V_C(s, a) = \frac{1}{|C|} \sum_{(c_i, p_i) \in C} \max\{0, \hat{P}(c_i|s, a) - p_i\} \quad (3.2)$$

where V_C refers to the violation measure as per constraint set C , $|C|$ is the cardinality of that set, $\hat{P}(c_i|s, a)$ is the probability that constraint c_i will be violated by performing action a in state s , and p_i is the tolerance threshold for constraint c_i . For example, if the taxi-routing system might be more willing to route the taxi through a storm than an area with a high crime rate,

in which case the constraint set would be:

$$C = \{(c_{\text{crime}}, 0.1), (c_{\text{storm}}, 0.5)\}. \quad (3.3)$$

Since V_C is essentially the average of a set of probabilities, it will always produce a real number between zero and one. The violation measure, itself, is a constrained criterion that expresses the average violation probability across all states that can be reached from the current state. Nana applies the violation function in two different ways: inside the reward function to modify the reward function, and outside the reward function as a risk-sensitive criterion. The risk sensitive criterion leads to the following expression for the optimal policy:

$$\pi^*(s) = \operatorname{argmax}_{a \in A} [Q(s, a) - \beta U(s, a)] \quad (3.4)$$

where β is the tuning hyperparameter and $U(s, a)$ is the U-function, which is formulated similarly to the Q-function. The only difference with the U-function is that instead of maximising the reward function, it aims to minimize the violation measure according to the following update equation.

$$U(s, a) \leftarrow (1 - \alpha)U(s, a) + \alpha(V_C(s, a) + \gamma \cdot \min_{a' \in A} U(s', a')). \quad (3.5)$$

4 THE TWO-PHASE ALGORITHM

In Section 3.2, the violation measure is used to modify the optimization criterion, but this is not the only way it can be used. Here we propose a two-phase algorithm, ViGIS, that uses the violation measure to modify the exploration process by providing initial knowledge. ViGIS works as follows:

1. The violation function is used to find the problem's safe policy space.
2. RL is performed within the safe policy space.

We also describe some variants of ViGIS and end off the section by summarising the base algorithm and these variants.

4.1 Setup

Before we derive the algorithm, it is important to lay out any necessary assumptions, and prior knowledge. ViGIS assumes the existence of a *violation delta function* which is expressed as follows:

$$\delta(c, s) = \begin{cases} 1, & s \text{ violates constraint } c \\ 0, & \text{otherwise} \end{cases} \quad (4.1)$$

This violation delta function expresses whether or not a state s violates a particular constraint c .

ViGIS also assumes that the following components of a CMDP are present and known: the state space \mathbf{S} , the action space \mathbf{A} , a reward function \mathbf{R} , and a set of constraints \mathbf{C} . There are two different approaches to this algorithm, one of which assumes that the transition function $\mathbf{T} = P(s'|s, a)$ is known. If \mathbf{T} is unknown in this case, it can be learnt using RL techniques or estimated by Monte Carlo simulation.

4.2 Phase One: Finding the Safe Policy Space

To find the safe policy space, we will explore two different approaches: we can precalculate the safety of every state-action pair in the policy space, or we can learn it using RL itself.

4.2.1 ViGIS-P: Precalculating the Safe Policy Space

ViGIS-P is the base form of the ViGIS algorithm, wherein the agent precalculates the violation measure for every possible state-action pair. This algorithm defines the safe policy space in terms of the violation function discussed in Section 3.2. If Π is the policy space of a particular problem, then the safe policy space is the set of all policies, $\pi \in \Pi$, in which all state-action pairs have a violation measure that falls below some predefined violation threshold β . This threshold functions as a hyperparameter that can be used to tune the agent's overall aversion to risk. Formally, the safe policy space can be defined as:

$$\Pi_{\text{safe}} = \{\pi \in \Pi | s \in \mathbf{S}, a \in \mathbf{A}, (s, a) \in \pi, V_C(s, a) \leq \beta\} \quad (4.2)$$

where Π_{safe} is the safe policy space, (s, a) is a state-action pair in policy π , and $\beta \in [0, 1]$ is the violation threshold. This is not the full picture, however, as V_C must still be calculated.

There are three main parameters in the violation function¹: \mathbf{C} , p_i , and $\hat{P}(c_i|s, a)$. Since \mathbf{C} and p_i are both assumed to be known, the only unknown is $\hat{P}(c_i|s, a)$. Let us then redefine $\hat{P}(c_i|s, a)$ in terms of what is already known.

Theorem 1. *Let \mathbf{S} be the state space of an MDP, $\delta(c, s) = P(c|s)$ be a violation delta function, and $P(s'|s, a)$ be the transition function, then the probability of violating constraint c by performing action a in state s , $\hat{P}(c|s, a)$, can be expressed as*

$$\hat{P}(c|s, a) = \sum_{s' \in \mathbf{S}} P(s'|s, a) \delta(c, s'). \quad (4.3)$$

¹ see Equation 3.2

Proof.

$$\begin{aligned} \hat{P}(c|s, a) &= \sum_{s' \in \mathbf{S}} P(c, s'|s, a) \\ &= \sum_{s' \in \mathbf{S}} P(c|s', s, a) P(s'|s, a) \\ &\quad c \text{ is independent of } s \text{ and } a \text{ given } s'. \\ \therefore \hat{P}(c|s, a) &= \sum_{s' \in \mathbf{S}} P(c|s') P(s'|s, a) \\ &= \sum_{s' \in \mathbf{S}} P(s'|s, a) \delta(c, s') \end{aligned}$$

□

Algorithm 1: Determine whether or not the state-action pair (s, a) is safe.

```

Function IS_SAFE( $s, a, \mathbf{C}, \beta$ ):
     $\mathbf{c} \leftarrow$  set of constraints in  $\mathbf{C}$ 
     $\mathbf{p} \leftarrow$  set of tolerance thresholds in  $\mathbf{C}$ 
     $V_C \leftarrow 0$ 
    for  $i = 1$  to  $|\mathbf{C}|$  do
         $\hat{P} \leftarrow 0$ 
        forall  $s'$  in  $\mathbf{S}$  do
             $\hat{P} \leftarrow \hat{P} + P(s'|s, a) \cdot \delta(\mathbf{c}[i], s')$ 
        end
         $V_C \leftarrow V_C + \max(0, \hat{P} - \mathbf{p}[i])$ 
    end
     $V_C \leftarrow \frac{1}{|\mathbf{C}|} \cdot V_C$ 
    if  $V_C \leq \beta$  then
        return true
    else
        return false
    end
end

```

Algorithm 2: Determine the safe policy space.

```

Function POLICY_SPACE( $\mathbf{S}, \mathbf{A}, \mathbf{C}, \beta$ ):
     $\Pi_{\text{safe}} \leftarrow |\mathbf{S}| \times |\mathbf{A}|$  boolean matrix
    forall  $s$  in  $\mathbf{S}$  do
        forall  $a$  in  $\mathbf{A}$  do
             $\Pi_{\text{safe}}[s][a] \leftarrow \text{IS\_SAFE}(s, a, \mathbf{C}, \beta)$ 
        end
    end
    return  $\Pi_{\text{safe}}$ 
end

```

Algorithm 1 shows how a single state-action pair can be evaluated as either safe or unsafe, using Theorem 1. Using this, Algorithm 2 determines the safe policy space by evaluating every possible state-action pair. As shown in Algorithm 2, a Boolean matrix is used to store whether a state-action pair is safe or not, but it may also be useful to store the violation measures themselves instead. If we store a V_C table, its values can still be compared to our β parameter on

the fly to determine a state’s safety, while also allowing the potential for comparing or updating these values during the agent’s learning phase. This will be explored further in Section 4.3.

4.2.2 ViGIS-L: Learning the Safe Policy Space

ViGIS-L is an alternative approach to finding the safe policy space, where the safe policy space is found using RL, rather than a full precalculation. This would be akin to training an agent in a safety simulation before transferring it to a physical robot to learn its task. In this regime, we learn the safe policy space using a recursive policy space update function which resembles that of Q-Learning:

$$\hat{V}_C(s, a) \leftarrow (1 - \alpha)\hat{V}_C(s, a) + \alpha(V_C(s, a) + \eta \cdot \min_{a' \in \mathbf{A}} \hat{V}_C(s', a')), \quad (4.4)$$

where η is a discount factor. Instead of maximising rewards like typical Q-Learning, this method learns the $\hat{V}(s, a)$ values by minimising violations. Since Q-Learning is model-free, this method does not require a transition function to operate. This is advantageous, because in most stochastic environments the transition function is not known beforehand. To exclude the transition function from $V_C(s, a)$, we replace it with the following:

$$W(s') = \frac{1}{|\mathbf{C}|} \sum_{(c_i, p_i) \in \mathbf{C}} \max\{0, \delta(c, s') - p_i\}. \quad (4.5)$$

This formulation expresses the violation measure of a single given state s' . The resulting update equation, using $W(s')$, is

$$\hat{V}_C(s, a) \leftarrow (1 - \alpha)\hat{V}_C(s, a) + \alpha(W(s') + \eta \cdot \min_{a' \in \mathbf{A}} \hat{V}_C(s', a')). \quad (4.6)$$

In this approach, an agent in state s takes an action a and arrives at subsequent state s' , then updates $\hat{V}_C(s, a)$ based on the perceived level of violation in state s' . This formulation is valid because, in the limit of infinite exploration, the agent will make all state transitions with their relative frequencies. This approach still requires violations on the part of the agent. This being said, this phase can be performed *in silico* before the agent is transferred to the target environment where learning the task is likely to be more accurate. Just like with reward-based RL, this approach should not only learn if a state-action pair will cause an immediate safety violation, but also if it may cause a violation further into the future. This will result in the RL approach learning different violation values to the precalculation method. More specifically, the \hat{V}_C values could exceed one. In this

case, \hat{V}_C can no longer be treated like probabilities. This means that either *beta* may need to be tuned to a value greater than one, or \hat{V}_C must be normalised. In this paper, we normalise \hat{V}_C empirically for all tabular Q-Learning agents, and leave it unnormalised for the Deep Q-Network agent. There is likely an analytical approach to normalising \hat{V}_C , but we leave this for future work.

There is one last aspect of this approach which is important to discuss. Since this phase of the algorithm is only interested in learning the environment’s safe policy space, we do not take rewards into account yet. This introduces a practical issue: if the agent is not learning how to complete the goal, allowing it to explore until it reaches an end state is not very effective. In this investigation, the issue is addressed by ending each trial after the agent has performed a set number of actions. To ensure that the agent explores the state space relatively evenly, it also begins each trial in a random state. There are likely alternative solutions to this problem which can be explored in future work.

4.3 Phase Two: Maximising Reward

Once the safe policy space has been computed or learnt, the agent enters the learning phase. During the learning phase, the agent explores its environment as per regular RL methods except it remains within the safe policy space as much as possible. This change only affects how the agent selects its next action, and thus can be implemented for a variety of standard RL algorithms. Algorithm 3 shows an example of how this change can be implemented for ϵ -greedy Q-Learning.

Algorithm 3: Q-Learning within the safe policy space.

```

Function ACT( $s, \epsilon, Q, \mathbf{A}$ ):
   $r \leftarrow$  random number  $\in [0, 1]$ 
   $\mathbf{A}_{\text{safe}} \leftarrow \{a \in \mathbf{A} \mid (s, a) \in \Pi_{\text{safe}}\}$ 
  if  $r < \epsilon$  then
    | return random  $a \in \mathbf{A}_{\text{safe}}$ 
  else
    | return  $\operatorname{argmax}_{a \in \mathbf{A}_{\text{safe}}} Q_s(a)$ 
  end
end

```

The flexibility of this learning phase also allows for the use of safe RL algorithms as an additional layer of safety. If a table of violation measures is maintained instead of a Boolean table, as discussed earlier, then those violation measures can be used to form a risk sensitive criterion (García and Fernández, 2015). Using a weighted sum approach, the criterion would take the same form as Equation 3.4.

4.4 Time Complexity

Now that the basic structure of ViGIS has been laid out, we can discuss its time complexity. ViGIS-P, as is, must precompute the safety of every single state-action pair. By inspecting Algorithm 1, one can see that it has a time complexity of $O(|S| \cdot |C|)$. Since Algorithm 2 uses Algorithm 1 internally, it has a time complexity of $O(|S|^2 \cdot |A| \cdot |C|)$. This means that for large state and action spaces, the algorithm can become computationally expensive. Even a 20-by-20 maze with two constraints and four actions (up, down, left, right), will require roughly 1280000 calculations to compute the safe policy space. An empirical analysis of the runtime is shown in Figure 2, which corresponds to the analytical analysis. The time complexity of ViGIS-L is not directly dependent on the state-space, action-space or constraints, but rather by the number of trials and number of steps per trial. These two parameters are highly environment dependent, which makes a formal time complexity analysis quite difficult.

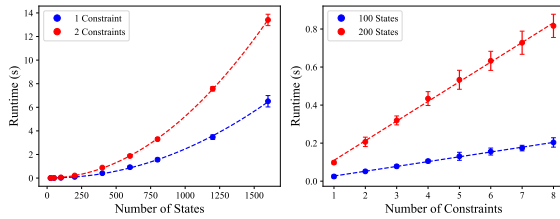


Figure 2: The empirical runtime for phase one of the ViGIS-P algorithm for different numbers of states (left) and constraints (right). Both plots show the best fit polynomials, which are quadratic and linear respectively.

4.5 Safe Policy Space Dead Ends

A problem that could rear its head in some environments is: What does the agent do when it has no safe actions to choose from? If safety were of no concern, then the agent would just haphazardly choose the action of highest reward. An example situation where this could arise might be if a physical robot only has enough battery power to perform one action, but no recharge stations are nearby. If a depleted battery is deemed as a safety constraint, then none of the available actions would be within the safe policy space. Since we restrict the agent’s action choices to those that fall within the safe policy space, there would then be no actions to choose from. One solution to this issue would be for the agent to take no action. In this case, there would need to be an external means of assistance. If the agent is digital, for example, then the agent can restart or begin the next trial, but if the agent is physical then it might need to be manually

re-positioned. Alternatively, the agent could attempt to make the best choice with the options it has. In this case, the means of choosing the best unsafe action could be explicitly specified, or the agent could resume its regular RL process while ignoring the relative danger of each action. Which of these approaches is best would depend on the environment at hand. For example, if safety is a higher priority than completing the task, then it might be more beneficial for the agent to take no action.

5 EVALUATION

We applied ViGIS to three environments: (1) a taxi grid world with up to four constraints, (2) a bank robber game, (3) a cart balancing a pole. To evaluate the performance of each method, each experiment was conducted 30 times and the average performance over all these runs is reported. ViGIS-P and ViGIS-L are compared to a regular Q-Learning agent and a β -pessimistic Q-Learning agent, except in the cart pole environment where ViGIS-L is compared to a Deep Q-Network agent. The evaluation metrics are the total reward gained and number of violations each episode.

5.1 Taxi World

We performed four experiments, all of which take place in a 10x10 grid with up to four constraints. The experiments resemble an autonomous taxi choosing a route to its destination. The goal of the agent is to find its way to the exit, while remaining within the safe regions of road. Once the agent reaches the destination it receives a large positive reward, and whenever a safety constraint is violated the agent receives a penalty (negative reward). The agent receives a default reward of -1 for all other actions, to encourage the agent to learn the quickest route. Whenever the agent chooses a direction to move, it has a 0.8 chance of actually moving in that direction, and a 0.1 chance of moving 90° clockwise or anticlockwise from the desired direction, respectively. In the first environment there is only one constraint, and each subsequent environment includes one more constraint than the last. In order, the constraints are cliffs, crime, potholes, and storms, all with tolerance thresholds of 0. Each episode ends when the agent either falls off a cliff or reaches the goal destination. All agents in this environment make use of a tabular ϵ -greedy Q-Learning algorithm. The ViGIS-L agent also uses this algorithm to learn the safe policy space. Both the ViGIS-P and ViGIS-L methods were tested for $\beta \in \{0, 0.2, 0.5, 0.8\}$. The β -pessimistic agent has its

β_p value set to 0.5. A diagram of the environment with all constraints is shown in Fig. 3.

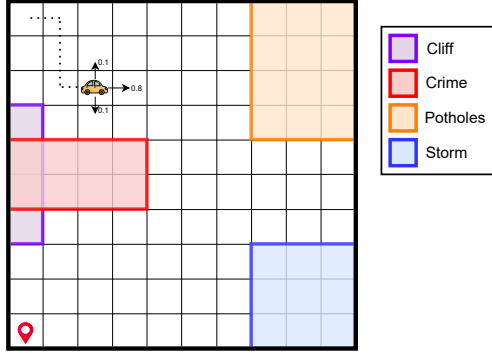


Figure 3: The taxi world environment with all constraints.

Figures 4 and 5 show the performance of the ViGIS-P and ViGIS-L agents, respectively, in the 4-constraint taxi world environment. As would be expected, the agents with higher β values generally commit more violations. Interestingly, the $\beta = 0.5$ agents commit more violations early-on than the $\beta = 0.8$ agents. It is not clear what exactly causes this behaviour, but it suggests that the choice of β should be carefully tuned. The $\beta = 0$ agents are able to make relatively few violations compared to the other agents, but still does violate the safety constraints in spite of $\beta = 0$ indicating zero tolerance of danger. Upon closer investigation, it was found that all of these violations occur at positions (1,3) and (1,6) on the grid: the corner where the pit and crime constraints overlap. In this case the agent's safest actions are to move upwards or right, both of which incur a 10% probability of violating a safety constraint. The agent is then caught in a safe policy dead end, as described earlier. The agents in Figures 4 and 5 are programmed to take the action which corresponds to the highest reward when in one of these dead ends, but other methods are assessed in Figure 8. Comparing the violation and reward rates, it is evident that the agents which commit fewer violations also acquire fewer rewards. This can be expected, because an agent would likely need to take a sub-optimal path in order to avoid the constraints.

Figure 6 shows the performance of the ViGIS-P and ViGIS-L agents compared to a regular RL agent. Both ViGIS agents have their β values set to zero, since this is the β value where both agents commit the fewest violations. The ViGIS-P agent commits an average of 0.71 violations per trial, whereas the ViGIS-L agent commits 0.20, which is $\approx 71\%$ fewer violations. If we compare these agents to the β -pessimistic Q-Learning agent, which makes ≈ 1.51 violations per trial, the ViGIS algorithm makes $\approx 53\%$ fewer viola-

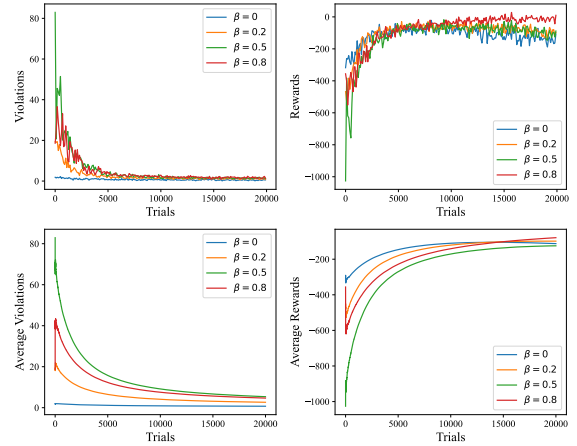


Figure 4: The number of violations (left) and rewards (right) per trial of ViGIS-P agent in the taxi world with all 4 constraints, and their respective running averages underneath.

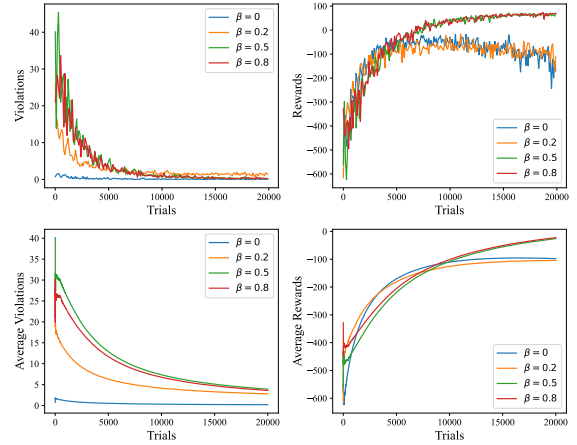


Figure 5: The number of violations (left) and rewards (right) per trial of ViGIS-L agent in the taxi world with all 4 constraints, and their respective running averages underneath.

tions, albeit at the cost of having a much less stable reward curve. It is also worth noting that the regular Q-Learning agent vastly outperforms all three safe algorithms in terms of rewards.

Figure 7 illustrates how both ViGIS algorithms perform in the taxi world environment for all numbers of constraints. As would be expected, both agents tend to violate the constraints more frequently as more constraints are added. The number of violations committed by ViGIS-P increases significantly when the number of constraints is increased from 1 to 2, but further increases only increase the number of violations slightly. As was discussed earlier, all violations made by the ViGIS-P agent are in those two safe policy dead end locations. Adding more constraints to the taxi world does not add any more dead ends, but the state space that the agent explores be-

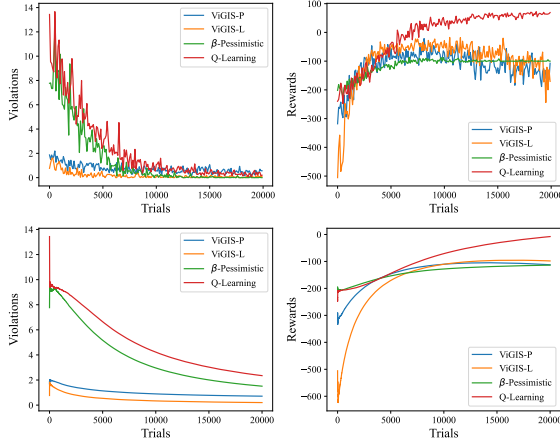


Figure 6: The number of violations (left) and rewards (right) per trial of ViGIS-P and ViGIS-L compared to β -pessimistic Q-Learning and regular Q-Learning. Results are from the taxi world with all 4 constraints, and their respective running averages are beneath.

comes effectively smaller. This causes the agent to be slightly more likely to commit enter these two dead ends, which causes the slight further increase in violations when there are 3 or 4 constraints. On the other hand, ViGIS-L shows an increase in violations that is directly proportional to the number of constraints. The ViGIS-L agent commits fewer violations than the ViGIS-P agent for all constraint configurations.

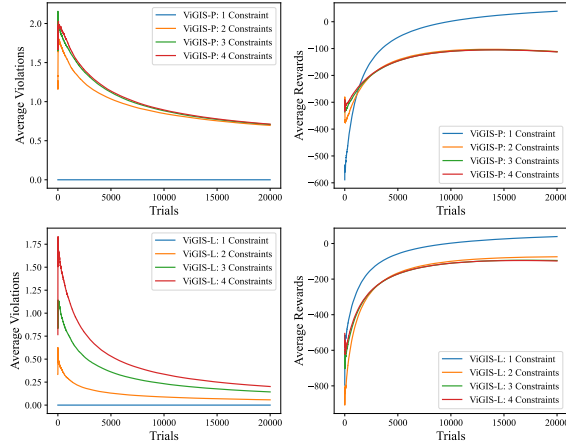


Figure 7: A performance comparison between ViGIS-P (top) and ViGIS-L (bottom) for different numbers of constraints. The average number of violations (left) and rewards (right) per trial in the taxi world are shown.

Figure 8 compares three different methods of handling the safe policy dead end issue: choosing the action with the highest expected reward, choosing the action with the lowest expected violation, and choosing no action. For both ViGIS agents, the lowest violation method performs causes fewer violations than the highest reward method. The no action method is

able to avoid all violations, which shows promise for this algorithm in safety-critical environments.

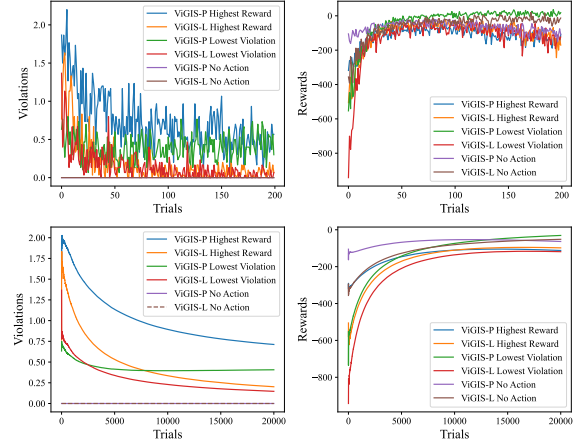


Figure 8: A comparison between different methods of handling safe policy dead ends. The number of violations (left) and rewards (right) per trial in the taxi world with all 4 constraints are shown, with their respective running averages underneath.

5.2 Bank Robber

In this domain, the agent's goal is to find the bank vault, take the money, then return to the entrance - avoiding the security guard as it does so. The agent's actions are completely deterministic but the security guard patrols stochastically. After the agent chooses a direction to move in, the security guard chooses a random adjacent open square, with uniform probability, and moves to it. When the agent reaches the exit with the money it receives a large reward, when it takes the money it receives a small reward, and when it is caught it receives a large penalty. Like in the taxi world, the agent receives a default reward of -1 for all other actions. Figure 9 shows an illustration of the environment. The agents applied to this environment are the same as for the taxi world environment. The tolerance threshold for being caught and β are both set to zero. The β -pessimistic agent has its β_p value set to 0.5.

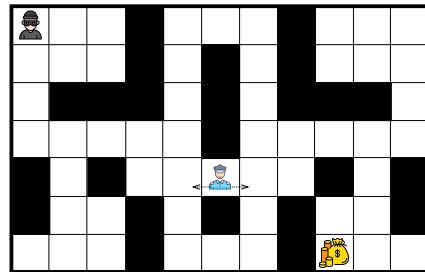


Figure 9: The Bank Robber environment.

The performance of ViGIS in the bank robber environment compared to Q-Learning and β -pessimistic Q-Learning is shown in Figure 10. Evidently, both ViGIS-P and ViGIS-L make no violations at all, and achieve a very high average reward by the end of training. This shows the potential of ViGIS for environments where the agent has complete deterministic control over its own actions. ViGIS-P and ViGIS-L produce almost identical reward curves, which shows that both algorithms perform equally as well when they find the same safe policy space.

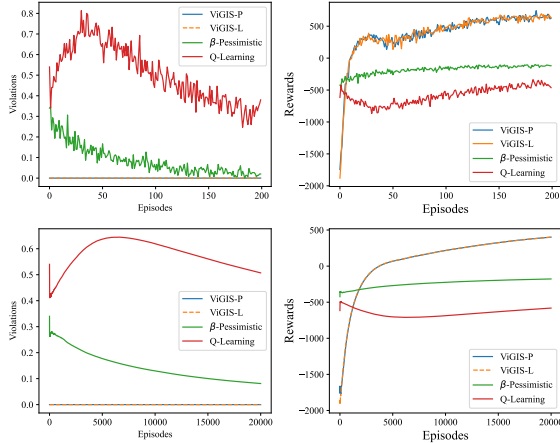


Figure 10: The number of violations (left) and rewards (right) per trial of ViGIS-P and ViGIS-L compared to β -pessimistic Q-Learning and regular Q-Learning. Results are from the bank robber environment with, and their respective running averages are beneath.

5.3 Cart Pole

For the final problem domain, we make use of the OpenAI Gym’s Cart Pole environment (Brockman et al., 2016) with an added constraint: the agent’s position must remain between -0.5 and 0.5 . In this environment, the agent operates a cart with an attached pole and must keep the pole upright by only moving the cart left or right. The agent receives a reward at the end of each trial proportional to how long the pole remained upright. At each step, the agent receives a reward of -1 if it is outside the bounds of -0.5 and 0.5 , otherwise it receives a reward of zero. Figure 11 shows an illustration of the cart pole environment. In this environment, we only test ViGIS-L against a regular agent. Each agent uses a Double Deep Q-Network (DDQN); ViGIS-L also uses a DDQN to learn the safe policy space. These DDQNs are comprised of 4 input neurons, 3 hidden layers with 128 neurons each, and 2 output neurons. The ViGIS-L agent’s β value is set to zero: minimal tolerance for violations.

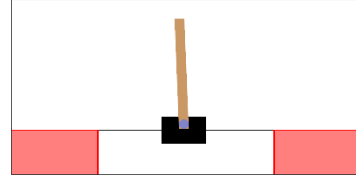


Figure 11: The cart pole environment with the added constraint indicated in red.

Figure 12 shows the performance of a DDQN-based ViGIS-L agent in the cart pole environment compared to a regular DDQN agent. The violation plot shows that after about 10 trials, both agents begin to learn how to keep the pole up for long enough to leave the boundaries specified by the constraints. ViGIS-L commits significantly fewer violations than the regular agent during this phase, which is more clear from the running average. By the end of training, the regular agent has committed 7.92 violations per trial on average, whereas the ViGIS-L agent has committed 5.65. This shows a violation reduction of more than 28%. Both agents achieve similar performance reward-wise, however the regular agent does outperform the ViGIS-L agent on average after about 350 trials. These results suggest that the ViGIS-L algorithm was able to better avoid constraint violations, albeit at a slight cost to performance. These results show promise for ViGIS-L in deep RL.

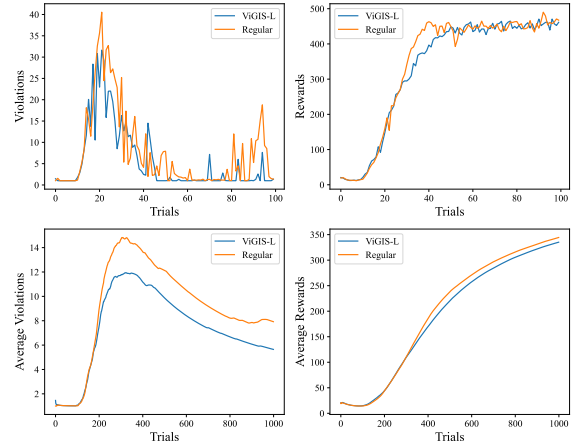


Figure 12: The violations (left) and rewards (right) per trial in the cart pole environment, as well as their respective running averages beneath.

6 CONCLUSION

This report presented a two-phase approach to safe RL, ViGIS, which focuses on first finding the safe policy space, then performing RL within the safe policy space. We described two implementations of ViGIS,

named ViGIS-P and ViGIS-L. ViGIS-P requires prior knowledge of the transition function but does not need to simulate constraint violations, whereas ViGIS-L needs violations to be performed in silico, but does not require prior knowledge of the transition function. The fact that violations should be simulated for ViGIS-L brings into question why we would not perform both phases in simulation. This is of course possible, but by separating the safety learning from reward learning the agent is able to learn the reward on-site, where the rewards would be more accurate. Both ViGIS approaches were tested in two discrete environments and ViGIS-L was also tested in a continuous environment.

Both ViGIS-P and ViGIS-L showed fewer constraint violations than the regular and β -pessimistic Q-Learning agents across all environments. Often, the better safety adherence lead the ViGIS agents to achieve a lower average reward. ViGIS does encounter the problem of safe policy dead ends, but various methods of dealing with these dead ends were assessed and it was found that choosing the safest action significantly reduces the number of committed violations. It is also possible, in some environments and configurations, for ViGIS to avoid violations, which was shown for the bank robber environment and the taxi world environment for the *No Action* ViGIS agents. ViGIS-L was able to make 28% fewer constraint violations than the regular DDQN agent in the cart pole environment, which shows its potential in the realm of deep RL.

There are multiple avenues where the research in this paper can be extended in future work:

- *Scalability*: investigate methods to optimize ViGIS-P for larger state spaces, potentially through approximation or parallelism.
- *Deep RL*: assess the performance of ViGIS with a broader range of deep RL architectures.
- *Normalising \hat{V}_C* : develop analytical methods for normalizing ViGIS-L's violation measure.
- *Hybrid Approaches*: explore combinations of ViGIS with other safe RL techniques to further enhance safety and performance.
- *Environments*: Assess the performance of ViGIS in more environments.
- *Benchmarking*: Compare ViGIS to additional state-of-the-art safe RL algorithms.

REFERENCES

- Altman, E. (1999). *Constrained Markov Decision Processes*. Chapman and Hall.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI gym. *arXiv preprint arXiv:1606.01540*.
- Driessens, K. and Džeroski, S. (2004). Integrating guidance into relational reinforcement learning. *Machine Learning*, 57:271–304.
- García, J. and Fernández, F. (2012). Safe exploration of state and action spaces in reinforcement learning. *J. Artif. Int. Res.*, 45(1):515–564.
- García, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42):1437–1480.
- Gaskett, C. (2003). Reinforcement learning under circumstances beyond its control.
- Ge, Y., Zhu, F., Ling, X., and Liu, Q. (2019). Safe Q-Learning method based on constrained Markov decision processes. *IEEE Access*, 7:165007–165017.
- Geramifard, A., Redding, J., Roy, N., and How, J. (2011). UAV cooperative control with stochastic risk models.
- Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.
- Mutti, M., Del Col, S., and Restelli, M. (2022). Reward-free policy space compression for reinforcement learning. In Camps-Valls, G., Ruiz, F. J. R., and Valera, I., editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 3187–3203. PMLR.
- Nana, H. P. Y. (2023). Reinforcement learning by minimizing constraint violation.
- Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.
- Srinivasan, K., Eysenbach, B., Ha, S., Tan, J., and Finn, C. (2020). Learning to be safe: Deep RL with a safety critic.
- Thomas, G., Luo, Y., and Ma, T. (2024). Safe reinforcement learning by imagining the near future. In *Proceedings of the 35th International Conference on Neural Information Processing Systems, NIPS '21*, Red Hook, NY, USA. Curran Associates Inc.
- Xiong, N., Du, Y., and Huang, L. (2023). Provably safe reinforcement learning with step-wise violation constraints. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S., editors, *Advances in Neural Information Processing Systems*, volume 36, pages 54341–54353. Curran Associates, Inc.
- Yang, W.-C., Marra, G., Rens, G., and De Raedt, L. (2023). Safe reinforcement learning via probabilistic logic shields. In Elkind, E., editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 5739–5749. International Joint Conferences on Artificial Intelligence Organization. Main Track.