

## Todo list

Review: Added some sentences on results. . . . .	2
Phil: reword this based on figure . . . . .	3
Review: Happy with $G_t$ ? . . . .	4
Review: Notation clearer? . . . . .	4
Overview of IC3, other invariant finding techniques for interlockings . . . . .	4
Review: Some introductory babble . . . . .	5
Explain why the LTS in James et al. [17] uses four elements, but we ignore ... . . . .	5
Phil: Does the transition description make sense?	5
Performance plots for PPO on pelican_19 . . . .	7

# Towards Reinforcement Learning of Invariants for Model Checking of Interlockings

Ben Lloyd-Roberts, Phillip James, Michael Edwards

*Department of Computer Science, Swansea University UK*

*Email: {ben.lloyd-roberts, p.d.james, michael.edwards}@swansea.ac.uk*

**Abstract**—The application of formal methods to verify that railway signalling systems operate correctly is well established within academia and is beginning to see real applications in industry. However, the uptake is yet to make a substantial impact due to current approaches often producing so-called false positives that cause the need for lengthy manual analysis during verification. It is accepted that so-called invariants, properties which hold for all scenarios in which a signalling system operates, can help reduce the number of such false positives. These invariants are often bespoke, manually designed properties created by experienced engineers and automating their generation remain a challenge. In this work we present first steps towards using reinforcement learning to explore state spaces of signalling systems and to generate a dataset of state sequences from which we envisage invariants could be mined. Our results suggest the viability of reinforcement learning algorithms in both maximising state space coverage and the longest loop free path for state spaces up to a theoretical size of  $2^{50}$ . Proximal Policy Optimisation (PPO) demonstrates the most stable learning, particularly in large environments where the optimal behaviour function is most complex. Distributed, multi-agent algorithms, such as Asynchronous Advantage Actor-Critic (A3C) result in the greatest state coverage.

**Review: Added some sentences on results.**

## 1. Introduction

Model checking is a formal verification technique stemming from the need to systematically check whether certain properties hold for different configurations (states) of a given system. Given a transition system  $T$  and a formula (or property)  $F$ , model checking attempts to verify through refutation that  $s \vdash F$  for every system state  $s \in T$ , such that  $T \vdash F$ .

The application of model checking in order to verify railway interlockings is well established within academia and is beginning to see real applications in industry. As early as 1995, Groote et al. [1] applied formal methods to verify an interlocking for controlling the Hoorn-Kersenboogher railway station. Newer approaches to interlocking verification have also been proposed in recent years [2], [3], [4]. This includes work by Linh et al. [5] which explores the verification of interlockings written in a similar language to Ladder Logic using SAT-based model checking. In spite of

this, such approaches still lack widespread use within the Rail industry.

In particular, one of the limitations of such model checking solutions is that verification can fail due to over approximation, typically when using techniques such as inductive verification [6]. Inductive verification fails to consider whether system states which violate a given property are indeed reachable by the system from a defined initial configuration. These false positive often require manual inspection. One solution is to introduce so-called invariants to suppress false positives [5]. Invariants are properties that hold for sub-regions of the state space. The aim is to introduce invariants that help bound the region of reachable states when model checking. However generating sufficiently strong invariants automatically is complex [7].

In this work we take first steps towards using machine learning to generate invariants by providing a first formal mapping of interlocking based state spaces to a reinforcement learning (RL) environment. We then explore how such state spaces can be generated in a controlled manner to test the scalability of our approach on environments where the number of reachable states is known. Finally we provide an analysis of how various reinforcement learning algorithms can be used to effectively explore state spaces in terms of their coverage. We see this as a first step towards mining invariants from such state spaces as this approach would indeed require reasonable coverage. Finally we reflect upon future works in directing our approach to improve exploration and learn invariants from a dataset of state sequences generated by our RL agents.

## 2. Preliminaries

We now briefly discuss model checking of railway interlockings and reinforcement learning. For further details we refer the reader to [6], [7] and [8] respectively.

### 2.1. Ladder Logic & Interlockings

Interlockings serve as a filter or ‘safety layer’ between inputs from railway operators, such as route setting requests, ensuring proposed changes to the current railway state avoid safety conflicts. As a vital part of any railway signalling system, interlockings are critical systems regarded with the highest safety integrity level (SIL4) according to the CENELEC 50128 standard.

Ladder logic is a graphical language widely used to program Programmable Logic Controllers [?] and in particular the Siemens interlocking systems we consider in this work. From an abstract perspective, ladder logic diagrams can be represented as propositional formulae. Here we follow the definition of James et al [?]. A ladder logic rung consists of the following entities. *Coils* represent boolean values that are stored for later use as output variables from the program. A coil is always the right most entity of the rung and its value is computed by executing the rung from left to right. *Contacts* are the boolean inputs of a rung, with *open* and *closed* contacts representing the values of un-negated and negated variables respectively. The value of a coil is calculated when a rung fires, making use of the current set of inputs – input variables, previous output variables, and output variables already computed for this cycle – following the given connections. A horizontal connection between contacts represents logical conjunction and a vertical connection represents logical disjunction.

A interlocking executes such a program from top-to-bottom over and over, indefinitely.

More formally, following [?] a ladder logic program is constructed in terms of disjoint finite sets  $I$  and  $C$  of input and output/state variables. We define  $C' = \{c' \mid c \in C\}$  to be a set of new variables in order to denote the output variables computed by the interlocking in the current cycle.

**Defn 1. Ladder Logic Formulae:** A ladder logic formula  $\psi$  is a propositional formula of the form

$$\psi \equiv ((c'_1 \leftrightarrow \psi_1) \wedge (c'_2 \leftrightarrow \psi_2) \wedge \dots \wedge (c'_n \leftrightarrow \psi_n))$$

Where each conjunct represents a rung of the ladder, such that the following holds for all  $i, j \in \{1, \dots, n\}$ :

- $c'_i \in C'$  (i.e.  $c'$  is a coil)
- $i \neq j \rightarrow c'_i \neq c'_j$  (i.e. coils are unique)
- $vars(\psi_i) \subseteq I \cup \{c'_1, \dots, c'_{i-1}\} \cup \{c_i, \dots, c_n\}$  (i.e. the output variable  $c'_i$  of each rung  $\psi_i$ , may depend on  $\{c_i, \dots, c_n\}$  from the previous cycle, but not on  $c_j$  with  $j < i$ , due to the nature of the ladder logic implementation, those values are overridden.)

## 2.2. Transition Systems and Model Checking for Ladder Logic

For this work, we have concentrated on trying to produce invariants for the approaches taken by Kanso et al. [6] and James et al. [7]. Here we include their model of ladder logic based railway interlocking programs as we use this as a basis for defining a learning environment.

Building upon the propositional representation of a ladder logic program given in Section ??, we can define, following [?], the semantics of a ladder logic program in terms of labelled transition systems.

Let  $\{0, 1\}$  represent the set of boolean values and let

$$Val_I = \{\mu_I \mid \mu_I : I \rightarrow \{0, 1\}\} = \{0, 1\}^I$$

$$Val_C = \{\mu_C \mid \mu_C : C \rightarrow \{0, 1\}\} = \{0, 1\}^C$$

be the sets of valuations for input and output variables.

The semantics of a ladder logic formula  $\psi$  is a function that takes the two current valuations and returns a new valuation for output variables.

$$[\psi] : Val_I \times Val_C \rightarrow Val_C$$

$$[\psi](\mu_I, \mu_C) = \mu'_C$$

where  $\mu'_C$  is computed as follows: the value of each variable  $c_i$  is computed using the  $i$ th rung of the ladder,  $\psi_i$ , using the valuations  $\mu_C$  and  $\mu_I$  from the last cycle and the current valuations restricted to those evaluated before the current variable. We refer the reader to [?] for full details.

**Defn 2. Ladder Logic Labelled Transition System:** We define the labelled transition system  $LTS(\psi)$  for a ladder logic formula  $\psi$  as the tuple  $(Val_C, Val_I, \rightarrow, Val_0)$  where

- $Val_C = \{\mu_C \mid \mu_C : C \rightarrow \{0, 1\}\}$  is a set of states.
- $Val_I = \{\mu_I \mid \mu_I : I \rightarrow \{0, 1\}\}$  is a set of transition labels.
- $\rightarrow \subseteq Val_C \times Val_I \times Val_C$  is a labelled transition relation, where  $\mu_C \xrightarrow{\mu_I} \mu'_C$  iff  $[\psi](\mu_I, \mu_C) = \mu'_C$ .
- $Val_0 \subseteq Val_C$  is the set of initial states.

We write  $s \xrightarrow{t} s'$  for  $(s, t, s') \in R$ . A state  $s$  is called *reachable* if  $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \dots \xrightarrow{t_{n-1}} s_n$ , for some states  $s_0, \dots, s_n \in Val_C$ , and labels  $t_0, \dots, t_{n-1} \in Val_I$  such that  $s_0 \in Val_0$ .

Phil: reword this based on figure

The shaded region of Figure ?? gives an example of a simple ladder Logic LTS for controlling a pelican crossing as considered by James et al [?]. Here we can see states contain Boolean valuations for the ladder logic variables and that transitions are labelled with inputs and their Boolean valuations. We can also one initial state (the state with dotted edges), where in this case all variables are set to false.

## 2.3. Reinforcement Learning and MDPs

Reinforcement Learning (RL) is a machine learning paradigm with demonstrably impressive capacity for modelling sequential decision making problems as the optimal control of some incompletely-known Markov Decision Process (MDP) [9].

**Defn 3 Markov Decision Process** A finite discounted Markov Decision Process  $M$  is a five tuple  $\langle S, \mathcal{A}, P_a(s, s'), R_a(s, s'), \gamma \rangle$ , where

- $S$ , is a finite set of states, known as the observation space or state space, representing the environment at discrete time steps.
- $\mathcal{A}$ , describes the action space; a set of actions performable at discrete time steps, used to compute new states from the observation space.
- $P_a(s, s') = Pr(s_{t+1} = s' \mid s_t, a_t)$ , describe state transition probabilities; the likelihood of observing state  $s_{t+1}$  given action  $a_t$  is taken from state  $s_t$ .
- $R_a(s, s')$  is a reward function feeding a scalar signal back to the agent at each time step  $t$ .

- $\gamma \in [0, 1]$  is a discount scalar successively applied at each time step.

In an RL setting we refer to the MDP as our environment,  $\mathcal{E}$  where through simulation, software agents sample actions  $a \in \mathcal{A}$ , observe changes in states,  $s \in \mathcal{S}$  and learn to optimise some objective based on rewards,  $r$  issued over discrete time steps  $t$ . Simulation can be continuous, where agents indefinitely interact with the environment until some termination criterion is met, such as reaching some reward threshold. Alternatively tasks may be episodic, where training is conducted over a sequence of episodes, defined by a finite number of time steps. An agent's trajectory,  $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_h, a_h, r_h)$ , summarises experience accumulated over a single episode or continuous training run. Here,  $h$  refers to the horizon; a time step beyond which rewards are no longer considered. Learning may then conclude having achieved adequate performance or resume from some initial state following an environment reset. Rewards observed from time step  $t$  up to some terminal time step  $T$ , are denoted

**Review: Happy with  $G_t$ ?**

$G_t = \sum_{i=t}^T \gamma^{i-t} r_i$ , and referred to as the return. Discount factor  $\gamma \in [0, 1]$  downscales rewards over return time steps while future rewards are reduced as the discount exponent  $i - t$  increases. This helps prioritise immediate rewards over distant ones and enumerate returns over a potentially infinite horizon.

Two principle challenges of RL are those of prediction and control. The first refers to approximating the value function used to estimate state values  $v(s)$  or state-action values  $q(s, a)$ , describing the benefit of being in that state. Values are differentiable and updated based on an empirical average, known as the expectation, taken over observed returns,  $\mathbb{E}[G_t | S_t = s, A_t = a]$ .

**Review: Notation clearer?**

In other words the expectation over return  $G_t$  depends being in state  $S_t = s$ , having taken action  $A_t = a$  during the present time step. Observed returns depend on the action(s) sampled at discrete time steps. The second challenge of control concerns optimising this selection process via the policy  $\pi(a|s)$ ; a probability distribution mapping states to actions most likely to maximise the reward objective. Ultimately an optimal value function will converge to an optimal policy []. In practice to scale with complex environments these functions are parametric and approximated with gradient-based methods, such as stochastic gradient ascent.

Determining the reachable state space for LLPs is often intractable, making the complete MDP model unknown to us without computing all transitions. Consequently we trial several gradient-based learning methods. Policy gradient methods [] utilise parameterised policies which update with respect to their parameter vector and objective function []. Actor-critic methods also approximate a parametric value function, separating the learning of action selection from value estimates. In this work we explore the efficacy of basic Deep Q-Networks [11], Proximal Policy Optimisation

(PPO) [12], Advantage Actor-Critic (A2C) and its asynchronous counterpart (A3C) [8] in navigating our set of generated LLPs. We discuss the merits and drawbacks of each approach for our setting in Section ??.

### 3. Related Work

Here we briefly highlight key contributions within related literature, addressing the invariant finding problem for interlocking programs and contemporary RL strategies for environment exploration.

#### 3.1. Invariant Finding

Overview of IC3, other invariant finding techniques for interlockings

From software engineering techniques [24], [25] to hybrid methods incorporating machine learning [26], researchers have proposed various approaches to invariant finding with varying degrees of success.

#### 3.2. Exploration in Reinforcement Learning

Game testing, applied algorithms to 3D environment, position and player movement used as observations with aim of state coverage for bug finding. FPS paper [27] applies vanilla PPO with no reset logic to identify game exploits. HMI paper [28] apply curiosity function based on an empirical count of state-action pairs to maximise traversal of state transitions over a deterministic finite state automaton navigation model. Uses vanilla Q-learning with e-greedy exploration. Similar use in web testing [29] simulate user actions to navigate web application, tracking 'uncertain' state transitions for backtracking. Also use vanilla Q-learning and count-based reward scaling during for Q-function updates. [30] decompose exploration tasks over large, adaptive and partially observable environments into two sub-problems - adaptive exploration policy for region selection and separate policy for exploitation of an area of interest. Others [31] incorporate recurrent networks in policy design, using temporality to recall the performance of past actions and their subsequent consequences according to the reward function. ACER algorithm used to make the A3C algorithm off-policy and constrain updates with TRPO like parameter updates. [32] apply a range of state-of-the-art learning algorithms to help robot navigation in procedurally generated environments. PPO found to have near human level performance, recalling which states are 'expensive' to reach, minimising the frequency of rediscovery. Works such as [33] have illustrated the efficacy of RL methods in learning heuristics over large graph structures. Others have employed distributing learning [34] for accelerated performance and prioritising novelty when exploration unfamiliar environments [23], [35], [36].

## 4. Mapping Formal Methods to Reinforcement Learning

In this section we introduce a mapping of modelling ladder logic programs for formal verification to a model where reinforcement learning can be applied, such that properties learned in the RL setting translate to model checking problems.

Review: Some introductory babble

Explain why the LTS in James et al. [17] uses four elements, but we ignore ...

### 4.1. Ladder Logic Markov Decision Process

We now define the finite Markov Decision Process (MDP), or environment  $\mathcal{E}$  used to represent the LLP. A Ladder Logic MDP  $M(\psi)$  is a five tuple, where our observation space is the union of program inputs and ladder variables,  $\mathcal{S} = V_I \cup V_C$ . Any unique valuation under the dynamics of a LLP constitutes a distinct state. Our action space,  $\mathcal{A} = V_I$ , describes the set of ladder logic inputs used to compute new valuations after program execution.  $P_a(s, s') = Pr(s_{t+1} = s' | s_t, a_t)$ , describes the state transition function in terms of probabilities of observing  $s_{t+1}$  given action  $a_t$  is taken from state  $s_t$ .  $R_a(s, s')$  is our reward function responsible for issuing positive feedback back to the agent at each time step  $\gamma$  is a discount scalar applied to the reward estimates for future time steps.

Subsequently the environment unfolds as a set of reachable states for the respective LLP as agents build a behaviour policy  $\pi(s|a, \theta)$  according to the reward function and state transitions observed under  $P_a(s, s')$ . To motivate continued exploration before model performance can converge, a balance is maintained between stochastic action sampling and selecting predicted actions output by the policy network.

Phil: Does the transition description make sense?

Aiming to maximise state space coverage we implement a reward scheme which positively rewards novel observations and dissuade loops with negative reward and episode termination. Training episodes spawn agents at some initial state, either with all variables set to 0 or from some randomly sampled state from a set of previous observations. Workers are initialised with separate environment instances to accumulate experience independently. A global set of observations is asynchronously updated by workers periodically to compile shared experiences. Early stopping terminates training if a set number of consecutive model updates do not improve in terms of coverage or reward. Second, training ends in our artificially generated programs if all reachable states have been observed at least once. To aid exploration, on episode resets we randomly initialise workers in some previously visited state, as demonstrated in [23].

TABLE 1. A3C COVERAGE METRICS

Environment			Agent	
States (Theoretical)	States (Reachable)	Actions	K-bound	Coverage
$2^{14}$	15	2	14	100.0
$2^{16}$	31	4	28	100.0
$2^{18}$	63	6	48	100.0
$2^{20}$	127	8	33	100.0
$2^{22}$	255	10	76	100.0
$2^{24}$	511	12	49	100.0
$2^{26}$	1023	14	306	100.0
$2^{28}$	2047	16	538	100.0
$2^{30}$	4095	18	1418	99.731
$2^{32}$	8191	20	1712	96.532
$2^{34}$	16383	22	1498	95.550
$2^{36}$	32767	24	2879	84.694
$2^{38}$	65535	26	1969	89.071
$2^{40}$	131071	28	2692	82.884
$2^{42}$	262143	30	1406	76.033
$2^{44}$	524287	32	1782	62.137
$2^{46}$	1048575	34	1593	64.053
$2^{48}$	2097151	36	1598	57.547
$2^{50}$	4194303	38	2566	41.483

### 4.2. Environment Generation

Given exhaustive search of large state spaces is often computationally intractable, we have generated a set of LLPs where the number of reachable states and recurrence reachability diameter are known. This has enabled us to analyse the performance of our approach against well understood state spaces. Using existing models of ladder logic structures as a base template [7], we derive progressively larger programs by sequentially introducing additional rungs. This way a constrained yet predictable pattern of growth is devised. If  $|S(p_i)|$  represents the number of reachable states for a program  $p_i$ , a subsequently generated program  $p_{i+1}$  with one additional rung, has  $2|S(p_i)| + 1$  reachable states. Through a series of training runs on each environment we record the number of states observed by workers to gauge the overall state space coverage.

## 5. Results

We now present a set of results from applying our approach to a series of generated LLPs, modelled as learning environments.

Performance plots show PPO and A2C algorithms to compare single agent advantage actor-critic against the trust region method.

### 5.1. Algorithmic Trends

Tried DQN on smaller environments to test the viability of RL in our problem formulation and it solved them (with respect to our reward objective) within an acceptable time frame. DQN doesn't scale well and is too sensitive to hyperparameter tuning. buffer size environment dependent, training delay and gradient step size contingent on complexity of learnable function, which we'd expect to increase with



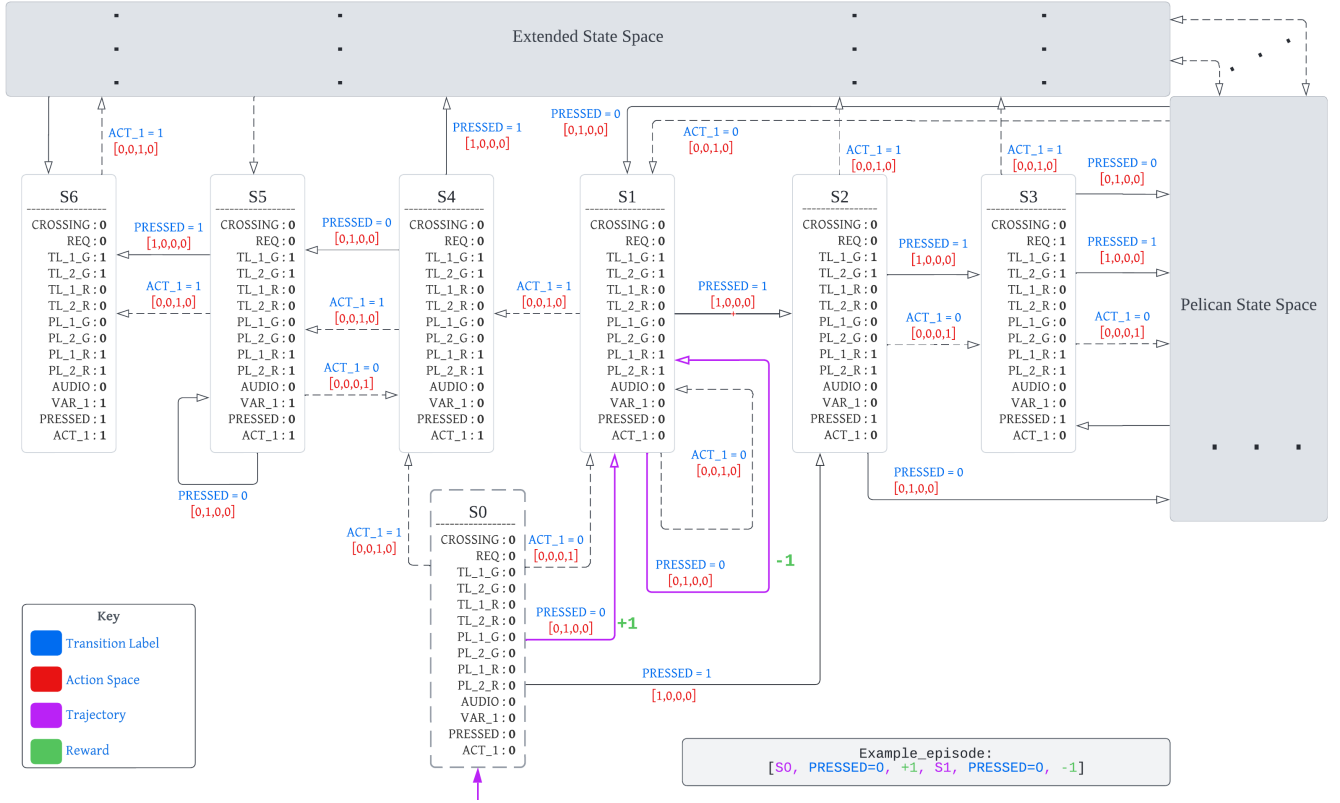


Figure 1. Simulation results for the network.

environment size. Algorithm is also off-policy - trains from experiences generated by old (presumably less optimal) policies. Exploration is governed explicitly by the epsilon-greedy strategy and guaranteed to anneal toward a minimum threshold over a significant number of timesteps - this could happen with large regions of an environment unobserved. Methods exist to improve the sample efficiency and learning stability, such as Prioritised experience replay or duelling DQN.

We also try two Actor-critic methods, Advantage actor-critic (A2C) and its asynchronous counterpart (A3C) to improve sample efficiency in distributing environment interaction among several workers. Having several workers and an on-policy learning algorithm removes the need of replay memory and training delay to accumulate sufficient experience. Actor and critic networks, approximating the behaviour policy and value function, provide better convergence guarantees at the cost of some additional complexity. Paired with randomised reset logic, both algorithms accumulate experience faster than Vanilla DQN, achieving good coverage on a range of medium to large state spaces. They are susceptible to parameter updates pushing the model into an unfamiliar region of policy space from which subsequent updates are unable to recover, potentially triggering model collapse.

We then move to trust region methods which constrain the magnitude of gradient updates within some clip range or

according to KL-divergence between the current and most recent policy parameters, making the model more resilient to collapse.

## 5.2. Asynchronous Exploration

Preliminary results applying our approach to a number of generated programs are outlined in Table 1. ‘Actions’ referenced in the third column refer to the number of possible assignments over input variables in each LLP, from every state. ‘K’, refers to the greatest number of steps taken before repeating observations in the environment, across all workers.

Coverage metrics are expectedly maximised for environments with a small number of reachable states with acceptable levels of coverage for programs with a theoretical state space up to  $2^{40}$ . Interestingly, we observed longer training durations occasionally increased coverage beyond a certain threshold. It is possible workers learn an optimal search strategy within a subregion of the state space. Additionally, performance in terms of max  $k$  and states reached increased by approx. 5% when decreasing the total number of episodes from  $3 \times 10^5$  to  $1.5 \times 10^5$  episodes. This may be a product of random episode initialisation spawning workers in more desirable states where stochastic action sampling happened to lead to unfamiliar subregions of the environment.

Performance plots illustrating the cumulative reward which failed to maximise coverage often increased linearly before collapsing to some suboptimal reward. This may be due to tendencies for large network updates to shift the network gradients into a bad local minima, from which performance does not recover within the allotted training duration. The on-policy nature of actor critic means trajectories generated via an old policy are no longer sampled during minibatch updates for the current policy, thus biasing behaviour to the most recent model updates and introducing sample inefficiency. Adding experience replay [37] may help avoid this in future applications

Given the A3C algorithm requires workers to asynchronously update their shared network every  $T_{\max}$  steps or on episode termination, larger values for  $T_{\max}$  consolidate more information regarding worker trajectories before applying gradient updates to their local network. We found the most significant improvements to performance in terms of coverage metrics and increasing the  $k$  bound when introducing workers to larger environments, was lower update frequencies and random start state initialisation. Prior to these adjustments workers, irrespective of their number, seldom covered 80% of most smaller environments. Similarly, for the largest environment with  $2^{50}$  states, coverage improved from 3.2% to 41.48%

### 5.3. Trust Regions and Reward Shaping

#### Performance plots for PPO on pelican\_19

- Advantage actor critic methods seem to perform better in terms of coverage. Distributed nature of algorithm and no update clipping may allow for greater exploration.
- PPO optimises better to increasing the k-bound for the explored subregion while slowly increasing state coverage.
- A2C increases state coverage significantly faster than PPO but struggles to learn optimal loop free paths within its discovered subregion.
- Value function loss is more stable for PPO compared to A2C. This measures the TD error between the current value function and actual observed returns.
- A2C entropy loss converges toward 0 sooner than PPO, a possible indication the policy learned under A2C experiences less uncertainty when making predictions. Could mean PPO policy maintains it knows very little about the environment but still maximises rewards better than its A2C counterpart.

## 6. Conclusion & Future Work

In this paper we have applied a basic asynchronous and trust region deep reinforcement learning methods to maximise program state coverage, motivated by a reward function of state novelty.

In light of our findings, we aim to improve several aspects of our approach, predominantly concerning learning

stability, sample efficiency and training speed. Experience replay for distributed learning may improve on-policy bias and sample efficiency.

The low dimensionality of our state spaces representation may allow us to introduce count-based exploration models to dampen the reward issued for states repeatedly observed [35]. Intrinsic motivation has also illustrated successes in environment exploration [38].

Applying IMPALA [39] to improve both sample efficiency over A3C and robustness to network architectures and hyperparameter adjustments. The adoption of a Long Short-Term Memory model (LSTM) also improves performance given GPU acceleration is maximised on larger batch updates.

For invariant finding this work has indicated the viability of reinforcement learning algorithms in increasing state space coverage. In our subsequent works we envisage the current learning framework to serve as a means of dataset generation, from which patterns or sequences of can be mined from agent trajectories.

## Acknowledgments

We thank Tom Werner and Andrew Lawrence at Siemens Mobility UK & EPSRC for their support in these works.

## References

- [1] J. F. Groote, S. F. van Vlijmen, and J. W. Koorn, "The safety guaranteeing system at station hoorn-kersenboogerd," in *COMPASS'95 Proceedings of the Tenth Annual Conference on Computer Assurance Systems Integrity, Software Safety and Process Security*. IEEE, 1995, pp. 57–68.
- [2] A. Fantechi, W. Fokkink, and A. Morzenti, "Some trends in formal methods applications to railway signaling," *Formal methods for industrial critical systems: A survey of applications*, pp. 61–84, 2012.
- [3] A. Ferrari, G. Magnani, D. Grasso, and A. Fantechi, "Model checking interlocking control tables," in *FORMS/FORMAT 2010*. Springer, 2011, pp. 107–115.
- [4] A. E. Haxthausen, M. L. Bliguet, and A. A. Kjær, "Modelling and verification of relay interlocking systems," in *Monterey Workshop*. Springer, 2008, pp. 141–153.
- [5] M. Awedh and F. Somenzi, "Automatic invariant strengthening to prove properties in bounded model checking," in *2006 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 1073–1076.
- [6] K. Kanso, F. Moller, and A. Setzer, "Automated verification of signalling principles in railway interlocking systems," *Electronic Notes in Theoretical Computer Science*, vol. 250, no. 2, pp. 19–31, 2009.
- [7] P. James, A. Lawrence, F. Moller, M. Roggenbach, M. Seisenberger, A. Setzer, K. Kanso, and S. Chadwick, "Verification of solid state interlocking programs," in *International Conference on Software Engineering and Formal Methods*. Springer, 2013, pp. 253–268.
- [8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.
- [9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [10] M. Geist and O. Pietquin, "Algorithmic survey of parametric value function approximation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 6, pp. 845–867, 2013.

- [11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
- [12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.
- [13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.
- [14] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [15] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.
- [16] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.
- [17] M. Bloesch, J. Humplik, V. Patraucean, R. Hafner, T. Haarnoja, A. Byravan, N. Y. Siegel, S. Tunyasuvunakool, F. Casarini, N. Batchelor *et al.*, "Towards real robot learning in the wild: A case study in bipedal locomotion," in *Conference on Robot Learning*. PMLR, 2022, pp. 1502–1511.
- [18] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, p. 105400, 2021.
- [19] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.
- [20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [21] S. M. Kakade, "A natural policy gradient," *Advances in neural information processing systems*, vol. 14, 2001.
- [22] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2017.
- [23] C. Girdillo, J. Bergdahl, K. Tollmar, and L. Gisslén, "Improving playtesting coverage via curiosity driven reinforcement learning agents," *arXiv preprint arXiv:2103.13798*, 2021.
- [24] M. L. Case, A. Mishchenko, and R. K. Brayton, "Automated extraction of inductive invariants to aid model checking," in *Formal Methods in Computer Aided Design (FMCAD'07)*. IEEE, 2007, pp. 165–172.
- [25] S. Bensalem, Y. Lakhnech, and H. Saidi, "Powerful techniques for the automatic generation of invariants," in *International Conference on Computer Aided Verification*. Springer, 1996, pp. 323–335.
- [26] P. Garg, D. Neider, P. Madhusudan, and D. Roth, "Learning invariants using decision trees and implication counterexamples," *ACM Sigplan Notices*, vol. 51, no. 1, pp. 499–512, 2016.
- [27] J. Bergdahl, C. Girdillo, K. Tollmar, and L. Gisslén, "Augmenting automated game testing with deep reinforcement learning," in *2020 IEEE Conference on Games (CoG)*, 2020, pp. 600–603.
- [28] Y. Cao, Y. Zheng, S.-W. Lin, Y. Liu, Y. S. Teo, Y. Toh, and V. V. Adiga, "Automatic hmi structure exploration via curiosity-based reinforcement learning," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 1151–1155.
- [29] Y. Zheng, Y. Liu, X. Xie, Y. Liu, L. Ma, J. Hao, and Y. Liu, "Automatic web testing using curiosity-driven reinforcement learning," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 423–435.
- [30] A. Peake, J. McCalmon, Y. Zhang, D. Myers, S. Alqahtani, and P. Pauca, "Deep reinforcement learning for adaptive exploration of unknown environments," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2021, pp. 265–274.
- [31] K. G. S. Apuroop, A. V. Le, M. R. Elara, and B. J. Sheu, "Reinforcement learning-based complete area coverage path planning for a modified htrihex robot," *Sensors*, vol. 21, no. 4, 2021. [Online]. Available: <https://www.mdpi.com/1424-8220/21/4/1067>
- [32] D. I. Koutras, A. C. Kapoutsis, A. A. Amanatiadis, and E. B. Kosmatopoulos, "Marsexplorer: Exploration of unknown terrains via deep reinforcement learning and procedurally generated environments," *Electronics*, vol. 10, no. 22, 2021. [Online]. Available: <https://www.mdpi.com/2079-9292/10/22/2751>
- [33] S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. Singh, "Learning heuristics over large graphs via deep reinforcement learning," *arXiv preprint arXiv:1903.03332*, 2019.
- [34] M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli *et al.*, "Acme: A research framework for distributed reinforcement learning," *arXiv preprint arXiv:2006.00979*, 2020.
- [35] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," 2017.
- [36] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.
- [37] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," 2017.
- [38] R. Houthoofd, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," 2017.
- [39] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," 2018.