Policy-Specific Abstraction Predicate Selection in Neural Policy Safety Verification

Marcel Vinzent¹, Min Wu², Haoze Wu², Jörg Hoffmann¹,³

¹ Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
² Department of Computer Science, Stanford University, Stanford, United States
³ German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany

{vinzent, hoffmann}@cs.uni-saarland.de, {minwu, haozewu}@cs.stanford.edu

Abstract

Neural networks (NN) are an increasingly important representation of action policies π. Recent work has introduced policy predicate abstraction (PPA) to prove safety of such π, through over-approximating the state space subgraph induced by π. Counterexample-guided abstraction refinement (CEGAR) drives the verification process, iteratively building the PPA for the current set of predicates, finding an abstract unsafe path σπ, and selecting new predicates based on sources of spuriousness in σπ. Compared to standard predicate abstraction, this involves a new source of spuriousness, where the necessary concrete transition exists but the neural network π selects a different action. Prior work tackles this source of spuriousness naïvely, based on the concrete states involved. Here we introduce a method that instead selects new abstraction predicates specific to policy behavior, trying to characterize π’s decision boundary. Inspired by ideas from formal explainable AI (XAI) in computer vision, we use SMT reasoning to identify a state variable set Vrel relevant to π, in the sense that π’s decision is invariant against perturbation of the irrelevant variables V \ Vrel. We then add new predicates based on v ∈ Vrel. We introduce and empirically evaluate a number of variants of this approach. The results show that the approach can sometimes improve CEGAR performance, though there is no clear winning configuration.

1 Introduction

Neural networks (NN) are an increasingly important representation of action policies in many contexts, including AI planning (Issakkimuthu, Fern, and Tadepalli 2018; Groshev et al. 2018; Garg, Buijpai, and Mausam 2019; Toyer et al. 2020; Stahlberg, Bonet, and Geffner 2022). But how to verify that such a policy is safe? Given a policy π, a start condition φ₀, and an unsafety condition φₙ, how to verify whether a state sₙ₊₁ |= φₙ is reachable from a state s₀ |= φ₀ under π? Such verification is potentially very hard as it compounds the state space explosion with the difficulty of analyzing even single NN decision episodes.

Research has recently begun to address this question. A prominent line of works addresses neural controllers of dynamical systems, where the NN outputs a vector u of reals forming input to a continuous state-evolution function f (Sun, Khedr, and Shoukry 2019; Tran et al. 2019; Huang et al. 2019; Dutta, Chen, and Sankaranarayanan 2019). Recent work extends this thread to hybrid systems, addressing smooth (tanh/sigmoid) activation functions by compilation into such systems (Ivanov et al. 2021). Another thread explores bounded-length verification of neural controllers (Akintunde et al. 2018, 2019; Amir, Schapira, and Katz 2021).

Here we follow up on the work by Vinzent et al. (2022, 2023) (henceforth: Vea), which tackles NN policies π with ReLU activation functions taking discrete action choices in non-deterministic state spaces over bounded-integer state variables V. The approach extends predicate abstraction (PA) (Graf and Sádi 1997; Ball et al. 2001; Henzinger et al. 2004) to what Vea baptizes policy predicate abstraction (PPA). Like PA, PPA builds an over-approximating abstraction defined through a set \( P \) of predicates, where each \( p \in P \) is a linear constraint over the state variables (e.g. \( x = 7 \) or \( x \leq y \)). Unlike PA however, PPA abstracts not the full state space Θ, but the policy-restricted state space Θπ, i.e., the state-space subgraph containing only the transitions taken by π. Vea’s method builds the fragment of Θπ - the predicate abstraction of Θπ reachable from φ₀, and checks whether φₙ is reached. If this is not the case, then π is safe.

To find a suitable predicate set \( P \) automatically, Vea employ counterexample-guided abstraction refinement (CEGAR) (Clarke et al. 2003), iteratively building the PPA for the current set of predicates, finding an abstract unsafe path σπ, and selecting new predicates based on sources of spuriousness in σπ (if no σπ exists then π is proved safe, if σπ is not spurious then π is proved unsafe). Vea show, on a collection of benchmarks adapted from AI planning, that this approach tends to outperform encodings into the state-of-the-art verification tool NuXmv (Cavada et al. 2014), in particular being the only approach that succeeds in proving policies safe.

In the present paper, we hone in on the sources of spuriousness in the abstract unsafe path σπ, and the adequate selection of new predicates to address these sources. This is an important and subtle problem because, whereas in standard PA spurious behavior is always due to the over-approximation in abstract state transitions, in PPA the reason may instead be a decision of the neural network π. To see this, consider a concrete state \( s \) in an abstract state \( s_π \), and say that the abstract path next proceeds to \( s_π' \) via label \( l \). In PA, this is possible iff there is a concrete transition \((s, l, s')\) s.t. \( s' \in s_π' \). In PPA, we additionally need that \( π(s) = l \).
If this is not the case, then the needed transition behavior is there, but the needed NN decision is not.

This source of spuriousness calls for new predicates specific to policy behavior: distinguishing states within $s_P$ where $\pi$ selects vs. does not select $l$, and therewith characterizing the relevant decision boundary of the neural network $\pi$. Vea did not tackle this challenge. Instead, they opted for a naïve method, based on a witness state $s_w \in s_P$ (found by PPA) for the abstract transition $(s_P, l, s_P)$. They introduce new predicates allowing to distinguish $s$ from $s_w$. This disregards the question which aspects of $s_w$ are actually relevant to $\pi$’s decision.

Here we tackle this question in detail. Inspired by ideas from formal explainable AI (XAI) in computer vision (Wu, Wu, and Barrett 2022), we say that a set of state variables $V_{irr} \subseteq V$ is $\epsilon$-irrelevant if the decision of $\pi$ in $s_w$ is invariant against perturbation of $V_{irr}$ by at most $\epsilon$ while keeping all other variables fixed. If $V_{irr}$ is maximal, we say $V_{rel} = V \setminus V_{irr}$ is $\epsilon$-relevant. We use SMT reasoning specialized to NN (Katz et al. 2017, 2019) to compute an $\epsilon$-relevant set $V_{rel}$ in a loop over $\epsilon \in V$. We get rid of the algorithm parameter $\epsilon$ through an outer loop using binary search to identify a minimal $\epsilon$ such that an $\epsilon$-relevant $V_{rel}$ exists. The incurred SMT overhead is negligible compared to the SMT effort required in PPA anyhow, so that this methodology is feasible.

We also experiment with the form of new predicates introduced, in particular relying on $\epsilon$ to identify a value between $s_w$ and $s$ – intuitively, estimating the location of $\pi$’s decision boundary – to be used in the new abstraction predicate.

We evaluate our techniques on Vea’s benchmarks, comparing a broad range of algorithm configurations. The results show that the approach can sometimes improve CEGAR performance, though there is no clear winning configuration.

2 Preliminaries

We consider the policy verification setting as introduced by Vea. A state space is a tuple $(V, L, O)$ of state variables $V$, action labels $L$, and operators $O$. For each variable $v \in V$ the domain $D_v \neq \emptyset$ is a bounded integer interval. We denote by $\text{Exp}$ the set of linear integer expressions over $V$, of the form $d_1 \cdot v_1 + \cdots + d_r \cdot v_r + c$ with $d_1, \ldots, d_r, c \in \mathbb{Z}$. $C$ denotes the set of linear integer constraints over $V$, i.e., constraints of the form $e_1 \prec e_2$ with $\prec \in \{\leq, =, \geq\}$ and $e_1, e_2 \in \text{Exp}$, and Boolean combinations thereof. An operator $o \in O$ is a tuple $(g, l, u)$ with label $l \in L$, guard $g \in C$, and (partial) update $u : V \rightarrow \text{Exp}$.

A (partial) variable assignment $\sigma$ over $V$ is a function with domain $\text{dom}(\sigma) \subseteq V$ and $\sigma(v) \in D_v$ for $v \in \text{dom}(\sigma)$. Given $s_1, s_2$, we denote by $s_1 [s_2]$ the update of $s_1$ by $s_2$, i.e., $\text{dom}(s_1 [s_2]) = \text{dom}(s_1) \cup \text{dom}(s_2)$ with $s_1 [s_2](v) = s_2(v)$ if $v \in \text{dom}(s_2)$, else $s_1 [s_2](v) = s_1(v)$. By $e(\sigma)$ we denote the evaluation of $e \in \text{Exp}$ over $\sigma$, and by $\phi(\sigma)$ the evaluation of $\phi \in C$. If $\phi(\sigma)$ evaluates to true, we write $s \models \phi$.

The state space of $(V, L, O)$ is a labeled transition system (LTS) $\Theta = (S, L, T)$. The set of states $S$ is the (finite) set of all complete variable assignments over $V$. The set of transitions $T \subseteq S \times L \times S$ contains $(s, l, s')$ iff there exists an operator $o = (g, l, u)$ such that $s \models g$ and $s' = s[u(s)]$, where $u(s) = \{ v \mapsto u(v)(s) \mid v \in \text{dom}(u) \}$. We also write $s \models o$ for $s \models g$, and abbreviate $s[o] = s[u(s)]$.

Observe that the separation between action labels and operators allows both, state-dependent effects (different operators with the same label $l$ applicable in different states) and action outcome non-determinism (different operators with the same label $l$ applicable in the same state).

A policy $\pi$ is a function $S \rightarrow L$. We consider $\pi$ represented by a neural network (NN). While Vea’s approach applies, in principle, to arbitrary NN, its current implementation is restricted to feed-forward NN with rectified linear unit (ReLU) activation functions $\text{ReLU}(x) = \max(x, 0)$. These NN consist of an input layer, arbitrarily many hidden layers, and an output layer with one neuron per label $l$; $\pi$ is obtained by applying argmax to the output layer.

We next introduce basic safety notations pertaining to $\Theta$ as a whole; we will later modify these to consider policy safety instead. A safety property is a pair $\rho = (\phi_0, \phi_u)$ where $\phi_0, \phi_u \in C$. Here, $\phi_u$ identifies the set of unsafe states that should be unreachable from the set of possible start states represented by $\phi_0$. $\Theta$ is unsafe with respect to $\rho$ iff there exists a path $(s_0^0, l_0^0, \ldots, s_n^{n-1}, l^{n-1}, s_n^n)$ in $\Theta$ such that $s_0^0 \models \phi_0$ and $s_n^n \models \phi_u$. Otherwise $\Theta$ is safe.

Predicate abstraction (Graf and Saidi 1997) is a well-established technique to prove safety. Assume a set of predicates $P \subseteq C$. An abstract state $s_P$ is a (complete) truth value assignment over $P$. The abstraction of a (concrete) state $s \in S$ is the abstract state $s_P$ with $s_P(p) = p(s)$ for each $p \in P$. Conversely, $[s_P] = \{ s' \in S \mid s_P | s' = s_P \}$ denotes the set of all concrete states represented by $s_P$. The predicate abstraction of $\Theta$ over $P$ is then the LTS $\Theta_P = (S_P, L, T_P)$, where $S_P$ is the set of all abstract states over $P$, and $T_P = \{ (s_P, l, s'_P) \mid (s, l, s') \in T \}$. To compute $\Theta_P$, one must solve what we call the abstract transition problem for every possible abstract transition: $(s_P, l, s'_P) \in T_P$ iff there exists an $l$-labeled operator $o \in O$ and a concrete state $s_w \in [s_P]$, a transition witness, such that $s_w \models o$ and $s_w[o] \in [s'_P]$. Such abstract transition problems are routinely encoded into satisfiability modulo theories (SMT) (Barrett and Tinelli 2018), using off-the-shelf solvers like Z3 (de Moura and Bjørner 2008).

Analogously to safety in $\Theta$, the abstract state space $\Theta_P$ is said to be unsafe with respect to a safety property $\rho = (\phi_0, \phi_u)$ iff there exists a path $\sigma_P = (s_0^0, l_0^0, \ldots, l^{n-1}, s_n^n)$ in $\Theta_P$ such that $s_0^0 \models \phi_0$, $s_n^n \models \phi_u$, and $s_i \models \phi$ iff there exists $s \in [s_P]$ such that $s \models \phi$. Otherwise $\Theta_P$ is safe, in which case $\Theta$ is safe as well.

A concrete path $(s_i^0, l_i^0, \ldots, l_i^{n-1}, s_i^n)$ for $i \in \{0, \ldots, n\}$ and $s_0^0 \models \phi_0$, $s_n^n \models \phi_u$ is a concretization of $\sigma_P$. If there does not exist a concretization for $\sigma_P$, then $\sigma_P$ is spurious. Counterexample-guided abstraction refinement (CEGAR) (e.g. (Clarke et al. 2003; Henzinger et al. 2004; Gupta and Strichman 2005; Podelski and Rybalchenko 2007)) iteratively removes such spurious abstract paths by refining $P$, until either the abstraction is proven safe or a non-spurious abstract path is found.
3 PPA & CEGAR (Vea)

We next briefly revisit Vea as the direct background for our work.

Policy Predicate Abstraction

Safety of a given policy \( \pi \) is proved via safety of the policy-restricted state space \( \Theta^\pi \), the subgraph \( \langle S, \mathcal{L}, T^\pi \rangle \) of \( \Theta \) with \( T^\pi = \{(s, l, s') \in T \mid \pi(s) = l\} \). In turn, safety of \( \Theta^\pi \) is proved via safety of the policy predicate abstraction of \( \Theta^\pi \), \( \Theta_P^\pi = \langle S_P, \mathcal{L}, T_P^\pi \rangle \) where \( T_P^\pi = \{(s|p, l, s'|p) \mid (s, l, s') \in T, \pi(s) = l\} \).

Vea (2022) introduce an algorithm to compute the fragment of \( \Theta_P^\pi \) reachable from the start condition \( \phi_0 \). The abstract transition problems in this computation involve checking whether \( \pi \) selects the correct label: \( (s|p, l, s'|p) \in T_P^\pi \) iff there exists an \( l \)-labeled operator \( o \in \mathcal{O} \) and transition witness \( s_w \in [s|p] \), such that \( s_w \models o \) and \( \pi(s_w) = l \). This is a key new source of complexity as the SMT sub-formula representing the neural network \( \pi \) contains one disjunction for every neuron. Vea show how this can be dealt with through approximate SMT checks. In particular, this pertains to continuous relaxations of the integer state variables, solved with Marabou (Katz et al. 2019), an SMT solver tailored to NN analysis.

CEGAR

To find a suitable predicate set \( \mathcal{P} \) automatically, Vea (2023) employ counterexample-guided abstraction refinement (CEGAR) (Clarke et al. 2003). Starting from an initially coarse predicate set \( \mathcal{P} = \emptyset \), they iteratively search for an abstract unsafe path \( \sigma_P \) in \( \Theta_P^\pi \). If \( \sigma_P \) is spurious, i.e., without concretization in \( \Theta^\pi \), they refine \( \mathcal{P} \) by adding predicates based on the source of spuriousness in \( \sigma_P \), and iterate. If \( \sigma_P \) is not spurious then \( \pi \) is proved safe. Conversely, if no \( \sigma_P \) exists then \( \pi \) is proved safe.

For refinement, Vea distinguish two sources of spuriousness, over-approximation of the transition behavior and spuriousness induced by the policy. Figure 1 shows an illustration. Consider a non-spurious prefix \( \sigma_P = (s_p^0, l^0, \ldots, l^{i-1}, s_p^i) \) of \( \sigma_P \), i.e., there exists a prefix concretization \( (s^0, l^0, \ldots, s^i(l^{i-1}, s^w_i)) \) in \( \Theta^\pi \). Moreover, say \( \sigma_P \) next proceeds to \( s_{p}^{i+1} \) via label \( l^i \). This is possible due to some transition witness \( s_w^i \in [s^i|p] \) with outgoing transition \( (s_w^i, l^i, s_{p}^{i+1}) \) such that \( s_{w}^{i} \in [s_{p}^{i+1}] \). Spuriousness due to over-approximation of the transition behavior occurs, if no such witness is reachable under concretizations of \( s_{p}^{i} \), i.e., there does not exist a concrete path \( (s^0, l^0, \ldots, l^{i-1}, s^i) \) of spuriousness also occurs in standard PA. Here, Vea refine based on weakest precondition computation over \( (s^0, l^0, \ldots, l^{i-1}, s^i) \) (cf. (Podelski and Rybalchenko 2007)).

In PPA, we additionally need that \( \pi(s_w^i) = l^i \), i.e., \( \pi \) selects the respective action label in the witness state. If there exists a state \( s^i \in [s^i|p] \) with \( (s^i, l^i, s_{p}^{i+1}) \) (i.e., a witness in standard PA) reachable via concretizations, but \( \pi(s^i) \neq l^i \) for any such \( s^i \), then the spuriousness is induced by the policy. For refinement of policy-induced spuriousness, Vea apply a technique they baptize witness splitting. They consider an individual concretization \( (s^0, l^0, \ldots, l^{i-1}, s^i) \) of \( \sigma_P \) for which action label selection deviates, \( \pi(s^i) \neq l^i \), and introduce predicates that enable to distinguish \( s^i \) and \( s_{w}^{i} \) (the witness state found by PPA). Intuitively, these predicates approximate the decision boundary of \( \pi \) within \( s_{w}^{i} \) by the difference between \( s_{w}^{i} \) and \( s^i \), splitting \( s^i \) in the refined abstraction. Vea show that, while a spurious counterexample may persist after a single refinement step based on this approximation, it is guaranteed to be removed within finitely many refinement steps, and hence the approach is complete.

Open Questions

Vea demonstrate, on a collection of benchmarks adapted from AI planning, that this approach tends to outperform encodings into the state-of-the-art non-NN-tailored verification tool NuXmv (Cavada et al. 2014), in particular being the only approach that succeeds in proving policies safe. These results are encouraging. However, in their policy refinement, they disregard the question of which aspects (i.e., state variables) are actually relevant to \( \pi \)’s decision in \( s_{w}^{i} \). Instead, they introduce one box constraint for each variable \( v \in V \); \( v \leq s_{w}^{i}(v) \) if \( s_{w}^{i}(v) < s^i(v) \) and \( v \geq s_{w}^{i}(v) \) if \( s_{w}^{i}(v) > s^i(v) \).

This leaves us with the question whether performance can be improved via a more targeted selection of (i) the variables and (ii) the split predicate \( v \leq c \). For (i), Vea’s approach is complete as long as at least one variable is selected. For (ii), they split right at the witness \( s_{w}^{i}(v) \), but, in principle, any split point \( c \) between \( s_{w}^{i}(v) \) and \( s^i(v) \) can be chosen. In our work, we investigate these questions. Specifically, we (i) provide a variable selection technique inspired by ideas from formal XAI (Sec. 4) and (ii) experiment with different heuristics for selecting the split point (Sec. 5).

For the remainder of this paper, we consider an abstract path prefix \( (s_p^0, l^0, \ldots, l^{i-1}, s_p^i) \) with policy-induced spuriousness in \( s_p^i \), prefix concretization \( (s^0, l^0, \ldots, l^{i-1}, s^i) \), and witness \( s_w^i \in [s^i|p] \). For readability, we drop the index \( i \) and abbreviate \( \pi(s_w^i) \) by \( l_w^i \).
4 Variable Selection for Policy Refinement

In this section, we provide a variable selection technique, that partitions the state variables \(\mathcal{V}\) into a set of relevant variables \(\mathcal{V}_{\text{rel}}\) and irrelevant variables \(\mathcal{V}_{\text{irr}}\) for \(\pi\) in \(s_w\). The motivation is to perform witness splitting using the relevant subset \(\mathcal{V}_{\text{rel}}\) only.

The technique is inspired by ideas from formal explainable AI, specifically VERIX (VERified exPlanability) (Wu, Wu, and Barrett 2022). VERIX computes a minimal subset of input features that explains the behavior of a neural network for a given input, such as an image in computer vision or a text in natural language processing. In essence, VERIX finds explanations by identifying sets of irrelevant features. Given a perturbation bound \(\epsilon\), a set of features is irrelevant, if the neural model behavior is invariant against arbitrary \(\epsilon\)-bounded perturbations of all irrelevant features while fixing the other features, i.e., the explanation. If the set of irrelevant features is maximal, i.e., no feature can be added, the explanation is optimal. In other words, the features in the explanation are relevant.

In the following, we first spell out an adaption of VERIX to our witness splitting setting. Subsequently, we contribute a binary search framework \(\text{BS}\), to get rid of the \(\epsilon\) parameter. Their combination constitutes our variable selection technique.

\(\epsilon\)-Relevant Variables

The notion of relevant and irrelevant features translates to state variables in an intuitive manner. In the witness splitting context, we are interested in perturbations “from” the witness \(s_w\) “towards” the concretization state \(s\) only. We maintain a relative perturbation bound \(\epsilon \in [0, 1]\) and normalize the absolute perturbation bound for each variable to the interval between \(s_w\) and \(s\). That is, the value for variable \(v\) in the perturbed state \(\hat{s}\) is bounded by \(\vert \hat{s}(v) - s_w(v) \vert \leq \epsilon \cdot \vert s(v) - s_w(v) \vert \) and \(s_w(v) \leq \hat{s}(v) \leq s(v)\) if \(s_w(v) \leq s(v)\) respectively \(s_w(v) \geq \hat{s}(v) \geq s(v)\), abbreviated \(\hat{s}(v) \in D^\epsilon_v(s_w, s)\).

**Definition 1 (\(\epsilon\)-Relevant Variables).** Given states \(s_w, s \in \mathcal{S}\) with \(\pi(s_w) = l_w \neq \pi(s)\) and a perturbation bound \(\epsilon \in [0, 1]\), \(\mathcal{V}_{\text{irr}} \subseteq \mathcal{V}\) is \(\epsilon\)-relevant for \(\pi\) in \(s_w\) if for any state \(\hat{s} \in \mathcal{S}\) such that \(\hat{s}(v) \in D^\epsilon_v(s_w, s)\) for each \(v \in \mathcal{V}_{\text{irr}}\) and \(\hat{s}(v) = s_w(v)\) for each \(v \in \mathcal{V} \setminus \mathcal{V}_{\text{irr}}\), it holds \(\pi(\hat{s}) = l_w\).

\(\mathcal{V}_{\text{irr}}\) is maximal if no superset of \(\mathcal{V}_{\text{irr}}\) is irrelevant. If \(\mathcal{V}_{\text{irr}}\) is maximal, its complement \(\mathcal{V}_{\text{rel}} = \mathcal{V} \setminus \mathcal{V}_{\text{irr}}\) is \(\epsilon\)-relevant.

We provide a graphical illustration in Figure 2. Intuitively, a set of state variables \(\mathcal{V}_{\text{irr}}\) is \(\epsilon\)-irrelevant if we can perturb \(\mathcal{V}_{\text{irr}}\) in \(s_w\) up to bound \(\epsilon\) (while fixing \(\mathcal{V} \setminus \mathcal{V}_{\text{irr}}\)) with the policy still selecting \(l_w\) in the perturbed state \(\hat{s}\). If \(\mathcal{V}_{\text{irr}}\) is maximal, i.e., no variable can be added, then its complement \(\mathcal{V}_{\text{rel}} = \mathcal{V} \setminus \mathcal{V}_{\text{irr}}\) is \(\epsilon\)-relevant. We remark that Definition 1 is local in that for a given bound \(\epsilon\) there may exist more than one \(\epsilon\)-relevant set \(\mathcal{V}_{\text{rel}}\). An interesting problem would be to find \(\mathcal{V}_{\text{rel}}\) with minimum size, which is, however, computationally extremely expensive. Following Wu et al. (2022), we instead settle for the computationally cheaper option of computing \(\mathcal{V}_{\text{rel}}\) greedily given a fixed traversal order of the variables.

![Figure 2: Graphical illustration of VERIX deployed in policy predicate abstraction, adapted from (Wu, Wu, and Barrett 2022). The big blue “+” denotes the witness state \(s_w\), and the small “+”s are its variants with \(\epsilon\)-perturbations on state variables. Perturbing the irrelevant variables (blue “+”) will definitely not change the action label, whereas perturbing both (orange “+”) may lead to a label change. Particularly, when the irrelevant variable set \(\mathcal{V}_{\text{irr}}\) is maximal (i.e., its complement \(\mathcal{V}_{\text{rel}}\) is \(\epsilon\)-relevant) perturbing \(\mathcal{V}_{\text{irr}}\) plus any single variable in \(\mathcal{V}_{\text{rel}}\) can breach the decision boundary of the policy \(\pi\), as shown by the variants connected by dotted lines.](image)

**Algorithm 1: VERIX** (Wu, Wu, and Barrett 2022) (adapted to WS setting).

**Input:** States \(s_w, s \in \mathcal{S}\) such that \(\pi(s_w) = l_w \neq \pi(s)\).

**Parameter:** perturbation-bound \(\epsilon \in [0, 1]\).

**Output:** \(\epsilon\)-relevant \(\mathcal{V}_{\text{rel}} \subseteq \mathcal{V}\).

1. \((\mathcal{V}_{\text{rel}}, \mathcal{V}_{\text{irr}}) \leftarrow (\emptyset, \emptyset)\)
2. **for** \(v' \in \text{TRAVERSALORDER}(s_w)\) **do**
   
   3. \(\mathcal{V}_{\text{irr}} \leftarrow \mathcal{V}_{\text{irr}} \cup \{v'\}\)
   
   4. \(\phi \leftarrow \bigwedge_{v \in \mathcal{V}_{\text{irr}}} \hat{s}(v) \in D^\epsilon_v(s_w, s)\)
   
   5. \(\phi \leftarrow \phi \bigwedge_{v \in \mathcal{V} \setminus \mathcal{V}_{\text{irr}}} \hat{s}(v) = s_w(v)\)

6. **if** \(\text{HOLD} \leftarrow \text{CHECK}(\phi \Rightarrow \pi(\hat{s}) = l_w)\)**
   
   7. **if** \(\text{HOLD} \text{ then**}
   
   8. \[\mathcal{V}_{\text{irr}} \leftarrow \mathcal{V}_{\text{irr}} \cup \{v'\}\]
   
   9. **else**
   
   10. \[\mathcal{V}_{\text{rel}} \leftarrow \mathcal{V}_{\text{rel}} \cup \{v'\}\]

**VERIX Algorithm.** Algorithm 1 shows the adaption of the VERIX algorithm to our setting. We are given a witness \(s_w\) and a concretization state \(s\) with \(\pi(s_w) = l_w \neq \pi(s)\) as well as a perturbation bound \(\epsilon\). VERIX maintains two sets \(\mathcal{V}_{\text{irr}}\) and \(\mathcal{V}_{\text{rel}}\), initialized to \(\emptyset\) (line 1). It then iterates each state variable \(v'\) according to a traversal order (see below) fixed for \(s_w\) (line 2), and checks whether \(v'\) can be added to \(\mathcal{V}_{\text{irr}}\). For this purpose, it constructs a formula \(\phi\) over the perturbation state \(\hat{s}\) where each \(v \in \mathcal{V}_{\text{irr}}\) as well as \(v'\) is \(\epsilon\)-bounded (line 4) and all remaining variables are fixed (line 5). It then checks via a call to a sub-procedure \(\text{CHECK}\) whether \(\phi\) implies \(\pi(\hat{s}) = l_w\) (line 6), i.e., whether \(\pi\) selecting \(l_w\) is invariant to perturbation to \(\mathcal{V}_{\text{irr}} \cup \{v'\}\). In practice, \(\text{CHECK}\) can be instantiated with an off-the-shelf
SMT solver (e.g., Marabou (Katz et al. 2019)). If CHECK holds, \( \mathcal{V}_{irr} \cup \{ v' \} \) is indeed irrelevant and \( v' \) can be added to \( \mathcal{V}_{irr} \) (line 8). If not, \( \mathcal{V}_{rel} \) is added to \( \mathcal{V}_{rel} \) (line 10). After all variables are traversed, each \( v \in \mathcal{V} \) is placed in either of the two sets \( \mathcal{V}_{irr} \) or \( \mathcal{V}_{rel} \); \( \mathcal{V}_{irr} \) is a maximal \( \epsilon \)-irrelevant set and \( \mathcal{V}_{rel} \) is \( \epsilon \)-relevant.

**Theorem 2 (Correctness)**. Algorithm 1 computes a \( \epsilon \)-relevant variable set \( \mathcal{V}_{rel} \subseteq \mathcal{V} \).

Theorem 2 follows analogously to explanation optimality for original VERIX (Wu, Wu, and Barrett 2022). We provide a formal proof in an online technical report (TR).

**Traversal Order**. The traversal order can significantly affect the size and shape of \( \mathcal{V}_{irr} \) respectively \( \mathcal{V}_{rel} \). Following Wu et al. (2022), we use a \( s_w \)-dependent heuristic that aims to produce small \( \mathcal{V}_{rel} \) based on variable sensitivity.

**Definition 3 (Variable Sensitivity)**. The variable sensitivity for \( v \in \mathcal{V} \) in \( s_w \in \mathcal{S} \) is

\[
sens(v) = \sum_{l \in L} \left| \pi^l(s_w[v \mapsto up_v]) - \pi^l(s_w[v \mapsto lo_v]) \right|
\]

where \( lo_v \) and \( up_v \) are the lower and upper bound of \( v \), \( s_w[v \mapsto c] \) is the update of \( s_w \) by \( v \mapsto c \), and \( \pi^l(s) \) is the value of the output layer neuron \( l \) in the neural network \( \pi \) given input \( s \).

Variable sensitivity is inspired by feature-level sensitivity (Wu, Wu, and Barrett 2022). Intuitively, we measure how sensitive the policy \( \pi \) is to perturbations of \( v \) in \( s_w \), comparing the output layer difference for maximal and minimal perturbation (\( up_v \) and \( lo_v \)).

Given \( sens, \text{TRAVERSALORDER}(s_w) \) iterates each state variable \( v \) from least to most sensitive. This is expected to lead to \( \mathcal{V}_{irr} \) and smaller \( \mathcal{V}_{rel} \).

We remark that \( sens \) in isolation can already be used to select a subset of heuristically-relevant variables for witness splitting (e.g., the most sensitive variable only). We investigate this option as a baseline in our experiments.

**\( \epsilon \)-Minimal Relevant Variable Set**

VERIX leaves the perturbation bound \( \epsilon \) as a user-parameter to customize. In our case, however, we do not consider a specific variable selection problem \((s_w, s)\) in isolation. Instead, we want to repeatedly select a subset of relevant variables for changing pairs of \( s_w \) and \( s \) each time we refine policy-induced spuriousness throughout an invocation of CEGAR. Ultimately, we want \( \epsilon \) to automatically adapt to the variable selection problem at hand.

We achieve this via a binary search around VERIX to find a minimal perturbation bound \( \epsilon \) such that a non-empty \( \epsilon \)-relevant \( \mathcal{V}_{rel} \) exists. Intuitively, this enables to select the subset of variables that is relevant to \( \pi \)'s decision near \( s_w \).

---

1It is also possible to first run an adversarial attack algorithm to try to quickly falsify \( \phi \Rightarrow \pi(\delta) = l_w \) before deploying the SMT solver, but on the benchmarks that we study those SMT calls do not incur a large overhead.


3We remark that this optimization criteria is not equivalent to finding \( \epsilon \) with non-empty \( \mathcal{V}_{rel} \) that is minimal in size.

<table>
<thead>
<tr>
<th>Algorithm 2: BSEARCH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: States ( s_w, s \in \mathcal{S} ) such that ( \pi(s_w) = l_w \neq \pi(s) ).</td>
</tr>
<tr>
<td><strong>Parameter</strong>: ( \epsilon )-precision ( \tau \in (0, 1) ).</td>
</tr>
<tr>
<td><strong>Output</strong>: ( \epsilon )-minimal relevant ( \mathcal{V}_{rel} \subseteq \mathcal{V} ).</td>
</tr>
<tr>
<td>1. ( (\mathcal{V}<em>{rel}, \epsilon</em>{lo}, \epsilon, \epsilon_{up}) \leftarrow (\emptyset, 0, 0.5, 1) )</td>
</tr>
<tr>
<td>2. while ( \epsilon_{up} - \epsilon_{lo} &gt; \tau ) do</td>
</tr>
<tr>
<td>3. ( \mathcal{V}_{rel} \leftarrow \text{VERIX}(s_w, s, \epsilon) )</td>
</tr>
<tr>
<td>4. if ( \mathcal{V}_{rel} = \emptyset ) then</td>
</tr>
<tr>
<td>5. ( (\epsilon_{lo}, \epsilon, \epsilon_{up}) \leftarrow (\epsilon, \epsilon + (\epsilon_{up} - \epsilon) / 2, \epsilon_{up}) )</td>
</tr>
<tr>
<td>6. else</td>
</tr>
<tr>
<td>7. ( (\mathcal{V}<em>{rel}, \epsilon</em>{lo}, \epsilon, \epsilon_{up}) \leftarrow (\mathcal{V}<em>{rel}, \epsilon</em>{lo}, \epsilon_{lo} + (\epsilon - \epsilon_{lo}) / 2, \epsilon) )</td>
</tr>
<tr>
<td>8. if ( \mathcal{V}<em>{rel} = \emptyset ) then ( \mathcal{V}</em>{rel} \leftarrow \text{VERIX}(s_w, s, 1) )</td>
</tr>
</tbody>
</table>
be non-empty. We remark that in our empirical study, this corner case never occurred.

**Theorem 4** (Correctness). Algorithm 2 computes a non-empty $\epsilon$-relevant variable set $V_{rel} \subseteq V$ such that $\epsilon$ is minimal up to $\tau$.

Due to space constraints, we move the formal proof to the online TR.

**Proof sketch.** Let $V_{rel}$ be the variable set computed by Algorithm 2. Case distinction.

- $V_{rel}$ is set (non-empty) in the loop (line 7): Whenever $V_{rel}$ is updated, $\epsilon_{up}$ is set to current $\epsilon$. Hence, $V_{rel}$ is $\epsilon_{up}$-relevant at loop termination. $\epsilon_{lo}$ is $0$ ($0$ is $0$-relevant) or updated for $V_{rel} = \emptyset$ (line 5). Hence, there does not exist $\epsilon' \leq \epsilon_{lo}$ with a non-empty $\epsilon'$-relevant variable set. Since additionally $\epsilon_{up} - \epsilon_{lo} < \tau$, $\epsilon_{up}$ is minimal up to $\tau$.

- $V_{rel}$ is set after loop termination (line 8): Since $V_{rel} = \emptyset$ at loop termination, we have $\epsilon_{up} = 1$. Analogously, to the first case, we have that there does not exist $\epsilon' \leq \epsilon_{lo}$ with a non-empty $\epsilon'$-relevant variable set. Since also $\epsilon_{up} - \epsilon_{lo} < \tau$, $1$ is minimal up to $\tau$.

\[ \square \]

# 5 Split Predicate Selection

In the previous section, we have provided a technique to select a subset of relevant state variables $V_{rel}$ for witness splitting. In this section, we propose different heuristics for selecting the split predicate $v \geq c$ for each $v \in V_{rel}$. In principle, witness splitting can be applied using any split predicate $v \leq c$ with $s_w(v) \leq c < s(v)$ if $s_w(v) < s(v)$, and $v \geq c$ with $s_w(v) \geq c > s(v)$ if $s_w(v) > s(v)$, respectively.

- **(AT-WITNESS)** In their original approach, Veas split right at the witness $s_w(v)$ for each state variable $v$. That is, split predicate $v \leq s_w(v)$ if $s_w(v) < s(v)$ and $v \geq s_w(v)$ if $s_w(v) > s(v)$. Intuitively, this choice is motivated in that $s_w(v)$ is known to “lie” within the decision boundary of $\pi$ selecting the desired action label, namely $\pi(s_w)$. In contrast, due to the complex structure of NN, no ad-hoc predication on $\pi$’s behavior for split points $c$ beyond $s_w(v)$ can be made.

- **(MID-SPLIT)** A disadvantage of AT-WITNESS is that it may heavily underestimate the distance to the decision boundary from $s_w$ towards $s$. This may result in unnecessarily many iterations of $\pi$-induced refinement; stepwise approximating the decision boundary. To mitigate such behavior, we experiment with a heuristic that splits in the middle between $s_w$ and $s$. That is, $v \leq \frac{s_w(v) + s(v)}{2} : (s(v) - s_w(v))$ for $s_w(v) < s(v)$ (and analogously for $s_w(v) > s(v)$). While there is no guarantee in terms of $c$’s relation to $\pi$’s decision boundary, middle-point based approximations can be observed to perform well in various contexts.

- **($\epsilon$-SPLIT)** Our variable selection technique computes, in addition to the relevant variable subset $V_{rel}$, also a corresponding perturbation bound $\epsilon$. Broadly speaking, $\epsilon$ can be considered as an estimate of the decision boundary of $\pi$: $\epsilon$-bounded perturbation to any $v \in V_{rel}$ (plus $V_{irr}$) allows for perturbed states $s$ such that $\pi(s) \neq \pi(s_w)$; crossing the decision boundary. We, hence, experiment with a heuristic that uses a split point relative to $\epsilon$. That is, $v \leq s_w(v) + \epsilon : (s(v) - s_w(v))$ for $s_w(v) < s(v)$ (and analogously for $s_w(v) > s(v)$).

# 6 Experiments

We implemented our approach on top of Veas’s C++ code base, which uses Z3 (de Moura and Bjørner 2008) and Marabou (Katz et al. 2019) to solve the various SMT queries during PPA computation and refinement. For Check (Algorithm 1, line 6) we query Marabou. All experiments were run on machines with Intel Xenon E5-2650 processors at 2.2 GHz, with time and memory limits of 12 h and 4 GB. We next describe the experiments setup, then summarize our results.

**Experimental Setup**

**Configurations.** We compare a broad range of algorithmic configurations for refinement of policy-induced spuriousness during CEGAR combining different variable selection techniques and split point heuristics.

- **Vea** is the default configuration used by Vea, i.e., selecting all state variables and using AT-WITNESS for splitting.

- **Vea** still selects all variables but uses MID-SPLIT for splitting.

- **Vs** applies the variable selection technique introduced in Section 4, i.e., BSEARCH around VERIX. For splitting it uses Vs’s default AT-WITNESS. For Vs, all following configurations using BSEARCH, we set $\epsilon$-precision $\tau = 0.05$.

- **Vs** also applies variable selection, though, using MID-SPLIT for splitting.

- **Vs** applies variable selection and uses $\epsilon$-SPLIT for splitting.

- **Vs** uses $\epsilon$-SPLIT to select the most-sensitive variable only, splitting based on MID-SPLIT. It serves as a simple baseline to the more advanced variable selection based on VERIX.

- As the ultimate baseline, Rand selects a random number of variables and a random split point between $s_w(v)$ and $s(v)$ for each selected $v$.

**Randomization.** The underlying CEGAR refinement approach involves two sources of non-determinism.

- Each refinement step is dependent on the detected spurious counterexample. Depending on the order in which the abstract state space is explored, different counterexamples, i.e., abstract unsafe paths, may be found. Veas provide a simple hamming distance heuristic, greedily

---

Our tool (and all experiments) are available at request.

State variables $v \in V$ such that $s_w(v) = s(v)$ are always ignored, since no split point is possible.
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>NN Safe</th>
<th>Vea^w</th>
<th>Vea^mid</th>
<th>BsVerix^w</th>
<th>BsVerix^mid</th>
<th>BsVerix^c</th>
<th>sens^mid</th>
<th>Rand</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Blocks</td>
<td>✓</td>
<td>5.1 ± 0.1</td>
<td>4.1 ± 0.1</td>
<td>6.6 ± 0.1</td>
<td>8.3 ± 0.0</td>
<td>6.8 ± 0.0</td>
<td>8.6 ± 0.0</td>
<td>6.0 ± 0.0</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>✓</td>
<td>1.9 ± 0.1</td>
<td>2.5 ± 0.1</td>
<td>3.0 ± 1.1</td>
<td>3.5 ± 1.1</td>
<td>3.4 ± 1.1</td>
<td>3.6 ± 1.1</td>
</tr>
<tr>
<td>(cost-ign)</td>
<td>64</td>
<td>✓</td>
<td>1.6 ± 2.1</td>
<td>1.6 ± 2.1</td>
<td>2.8 ± 2.1</td>
<td>2.6 ± 2.1</td>
<td>2.8 ± 2.1</td>
<td>2.6 ± 2.1</td>
</tr>
<tr>
<td>6 Blocks</td>
<td>✓</td>
<td>9.2 ± 1.1</td>
<td>2.8 ± 3.3</td>
<td>7.5 ± 1.1</td>
<td>9.8 ± 1.1</td>
<td>9.2 ± 1.1</td>
<td>2.4 ± 2.1</td>
<td>1.8 ± 2.1</td>
</tr>
<tr>
<td>(cost-ign)</td>
<td>32</td>
<td>✓</td>
<td>1.5 ± 2.1</td>
<td>1.5 ± 2.1</td>
<td>1.5 ± 2.1</td>
<td>1.5 ± 2.1</td>
<td>1.5 ± 2.1</td>
<td>1.4 ± 2.1</td>
</tr>
<tr>
<td>8 Blocks</td>
<td>✓</td>
<td>1.3 ± 4.3</td>
<td>1.7 ± 4.3</td>
<td>1.8 ± 4.3</td>
<td>1.9 ± 4.3</td>
<td>2.1 ± 4.3</td>
<td>3.2 ± 4.3</td>
<td>1.7 ± 4.3</td>
</tr>
<tr>
<td>(cost-ign)</td>
<td>64</td>
<td>✓</td>
<td>1.3 ± 4.4</td>
<td>1.4 ± 4.4</td>
<td>1.4 ± 4.4</td>
<td>1.3 ± 4.4</td>
<td>1.3 ± 4.4</td>
<td>1.4 ± 4.4</td>
</tr>
<tr>
<td>8-puzzle</td>
<td>✓</td>
<td>4.2 ± 1.1</td>
<td>3.9 ± 1.1</td>
<td>4.2 ± 1.1</td>
<td>3.8 ± 1.1</td>
<td>3.9 ± 1.1</td>
<td>3.5 ± 1.1</td>
<td>4.4 ± 1.1</td>
</tr>
<tr>
<td>(cost-ign)</td>
<td>32</td>
<td>✓</td>
<td>3.2 ± 4.4</td>
<td>3.3 ± 4.4</td>
<td>3.3 ± 4.4</td>
<td>3.3 ± 4.4</td>
<td>4.3 ± 4.4</td>
<td>4.3 ± 4.4</td>
</tr>
<tr>
<td>4 Blocks</td>
<td>✓</td>
<td>1.5 ± 1.1</td>
<td>2.0 ± 1.0</td>
<td>1.7 ± 1.0</td>
<td>2.2 ± 1.1</td>
<td>1.5 ± 1.0</td>
<td>1.2 ± 1.0</td>
<td>3.8 ± 1.1</td>
</tr>
<tr>
<td>(cost-awa)</td>
<td>32</td>
<td>✓</td>
<td>1.4 ± 3.2</td>
<td>2.0 ± 3.3</td>
<td>1.4 ± 4.4</td>
<td>6.3 ± 2.2</td>
<td>8.9 ± 2.2</td>
<td>8.1 ± 2.3</td>
</tr>
<tr>
<td>6 Blocks</td>
<td>✓</td>
<td>3.2 ± 4.4</td>
<td>2.3 ± 4.4</td>
<td>3.2 ± 4.4</td>
<td>8.2 ± 4.4</td>
<td>8.2 ± 4.4</td>
<td>8.2 ± 4.4</td>
<td>8.2 ± 4.4</td>
</tr>
<tr>
<td>(cost-awa)</td>
<td>64</td>
<td>✓</td>
<td>9.5 ± 2.3</td>
<td>1.7 ± 3.3</td>
<td>1.7 ± 3.3</td>
<td>2.3 ± 3.3</td>
<td>1.9 ± 3.3</td>
<td>7.1 ± 2.3</td>
</tr>
</tbody>
</table>

Table 1: Runtime results in seconds for the evaluated CEGAR configurations (Vea^w, Vea^mid, BsVerix^w, BsVerix^mid, BsVerix^c, sens^mid, Rand) for different benchmarks and NN policies averaged over 10 randomized invocations, where each result t ± 10^-j specifies the mean t · 10^-j and the magnitude of the observed standard deviation 10^j. Timeouts (exceeding the time limit of 12 h) are weighted as 24h. - indicates timeouts for all invocations. For Vea^w and BsVerix^w we additionally compare the number of CEGAR iterations (#It), the number of policy-induced refinement steps (π-It), the size of the final predicate set (|P|), the number of predicates added during policy-induced refinement (π-|P|) – averaged over 10 randomized invocations.

exploring abstract states for which the distance to an unsafe state is estimated to be small, to speed up the detection of unsafe paths. However, distinct abstract states may obtain the same heuristic estimate. In this case, Vea break ties non-deterministically. To account for this source of non-determinism and to investigate the impact of different spurious counterexamples, we invoke each of the configurations above with multiple seeds.

For policy-induced spuriousness in particular, each refinement step is also influenced by the deployed witness s_w – an abstract transition may have multiple witnesses – as well as its counterpart s in the deployed prefix concretization – there may be multiple prefix concretizations. Both of these states are extracted from the underlying SMT solvers (Marabou and Z3) as solutions found for the problem at hand (PPA transition problem and prefix concretization). To account for this additional source of non-determinism – the SMT solver is essentially a black-box – we also invoke each configuration with different seeds for Z3 and Marabou.

In total, we invoke each configuration with 10 different seeds.

**Benchmarks.** Due to the large configuration space (variable & split point selection compounded with randomization), we focus on a subset of Vea’s benchmarks. To give a brief overview, the benchmarks are variants of the planning domains Blocksworld and SlidingTiles. Actions moving a block/tile x may probabilistically fail, in which case the cost of moving x (represented by a state variable) is incremented. These probabilistic transitions (over which the policy is learned) are abstracted into non-deterministic ones for the purpose of verification, amounting to a worst-case analysis. In both domains, the start conditions impose a partial order on the block/tile positions. In Blocksworld, a state is unsafe if the number of blocks on the table exceeds a fixed limit. In SlidingTiles, unsafe states are specified in terms of a set of unsafe tile positions.

**Experimental Results**

Table 1 shows our results. On 16 out of the 18 problem instances at least one configuration wins. The conclusion are:

**Variable Selection.** Policy-specific variable selection (BsVerix, sens^mid) can improve performance over the default applied by Vea. However, there is no clear winning configuration. The BsVerix-based configurations (BsVerix^w, BsVerix^mid, BsVerix^c) have at least one best performing configuration (smallest mean) on 7 problem instances, while the configurations selecting all variables (Vea^w, Vea^mid) win on 5 instances. On 3 instances, the baseline based on variable sensitivity (sens^mid) performs best. The baseline based on random selection (Rand) still achieves best performance on one instance. On 9 instances, all configurations – including Rand – achieve performance in the same magnitude as the best performing configuration. Moreover, for each configuration we observe a standard deviation in the same order of magnitude.
magnitude as the average runtime on at least 8 out of the 16 problem instances.

In summary, these results are highly inconclusive. While the high standard deviation hints at a certain instability of the approach – presumably due to the policy-induced refinement using witness states extracted from the underlying SMT solvers – the performance of Rand may, at least to some degree, indicate that the current benchmark basis does not address the problem at sufficient scale.

Split Predicate Selection. Focusing on split predicate selection, the picture becomes less opaque. AT-WITNESS is the overall better performing heuristic (compared to MID-SPLIT and $\epsilon$-SPLIT). Vea$^w$ outperforms Vea$^{mid}$ on 11 problem instances – covering one instance on which Vea$^{mid}$ does not terminate. BsVerix$^w$ outperforms BsVerix$^{mid}$ and BsVerix$^c$ on 8 instances – BsVerix$^{mid}$ and BsVerix$^c$ only dominating on 4 respectively 3 instances. These results indicate that overall the more conservative split point estimate of AT-WITNESS turns out to be more stable. That said, there still is no clear winner. Vea configurations and BsVerix configuration both perform in the same order of magnitude on more than 10 instances.

Vea vs. BsVerix While there is no clear picture runtime-wise, the impact of policy-specific variable selection can be clearly measured in terms of policy-induced refinement steps and the number of predicates added during these steps. Table 1 illustrates this by comparing Vea$^w$ and BsVerix$^c$. As intended, for BsVerix$^c$ the predicate selection becomes more cautious – fewer predicates $\pi \cdot [P]$ are added during policy-induced refinement steps while the number of refinement steps $\pi \cdot \#It$ grows. The latter also results in an increase of the total number of CEGAR iterations ($\#It$). Interestingly, the total number of predicates $|P|$ is usually similar in size. That is, the decrease in the number of predicates added during $\pi$-refinement $\pi \cdot |P|$ results in an increase of the number of predicates added for refinement of standard spuriousness. In accordance with this intuition, the increase in $\#It$ tends to be larger than the increase in $\pi \cdot \#It$. For other Vea-BsVerix-pairs similar observations can be made.

7 Conclusion Neural policy verification via policy predicate abstraction involves intricate problems in the choice of abstraction predicates, which naturally, for spuriousness caused by policy decisions, should target the neural decision boundary. We have introduced and evaluated a range of methods operationalizing this intuition. For the time being, while improvements over the previous simpler predicate selection method can be obtained, the results are largely inconclusive in that there is no winning configuration. To a degree at least, this is probably due to the current benchmark basis, which does not seem to exhibit the problem addressed at sufficient scale – a strong indication being that, in more than half of the benchmark instances, a random strategy achieves average performance in the same order of magnitude as the respective winning configuration. Future work will have to expand the benchmark basis, and reveal whether there exist more consistently well-performing configurations (potentially also based on different ideas, like minimum-distance states on the other side of the decision boundary as illustrated in Figure 2 (Croce and Hein 2020; Madry et al. 2018; Szegedy et al. 2014)). Portfolio methods are of interest too, as they might manage to reap the benefits of different strategies across benchmarks.

Acknowledgments
This work was funded by DFG Grant 389792660 as part of TIRR 248 – CPEC (https://perspicuous-computing.science). This work has received funding from the European Union’s Horizon Europe Research and Innovation program under the grant agreement TUPLES No 101070149.

References


