Debugging a Policy: Automatic Action-Policy Testing in AI Planning

Marcel Steinmetz,[1] Daniel Fiser,[1] Hasan Ferit Eniser,[2] Patrick Ferber,[1,4] Timo P. Gros,[1] Philippe Heim,[1] Daniel Höller,[1] Xandra Schuler,[1] Valentin Wüstholz,[3] Maria Christakis,[2] Jörg Hoffmann[1]

[1] Saarland University, [2] MPI-SWS, [3] ConsenSys, [4] University of Basel

{steinmetz,fiser,timopgros,hoeller,hoffmann}@cs.uni-saarland.de, {s8phheim,s8xaschu}@stud.uni-saarland.de, patrick.ferber@unibas.ch, {hfeniser, maria}@mpi-sws.org, wuestholz@gmail.com

DNNs Are Taking Over Control!

How to gain trust in the policy's learned decisions? Here: Testing!

The Assumptions

- **Action policy** π maps *states* $s \in \mathcal{S}$ to *actions* $\pi(s) \in \mathcal{A}$.
- **Testing objective** V measurable in terms of a **value function** $V^\pi \colon \mathcal{S} \to \mathbb{R} \cup \{\infty\}$.
- **Optimal value function** for testing objective V: $V^*(s) = \min_\pi V^\pi(s)$.
- Example: classical planning, where π induces path $\sigma^\pi(s) = \langle s, \pi(s), s_1, \pi(s_1), \ldots, s_n \rangle$
  - Quantitative goal reachability (plan cost)
$$V^\pi(s) := \begin{cases} c(\sigma^\pi(s)) & \text{if } s_n \models \mathcal{G} \\ \infty & \text{otherwise} \end{cases}$$
  - Qualitative goal reachability
$$V^\pi(s) := \begin{cases} 1 & \text{if } s_n \not\models \mathcal{G} \\ 0 & \text{otherwise} \end{cases}$$
  - Temporal failure property ϕ:
$$V^\pi(s) := \begin{cases} 1 & \text{if } \sigma^\pi(s) \models \phi \\ 0 & \text{otherwise} \end{cases}$$

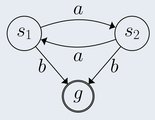
Policy Bugs: Definition

Definition (Bug)

A state s is a **bug in** π if $|V^\pi(s) - V^*(s)| > 0$

$\Rightarrow \pi$ is bug-free iff π is optimal

Note: π can have bugs despite each $\pi(s)$ being optimal individually. Consider $\pi(s_1) = \pi(s_2) = a$, assuming a has 0 cost, and the objective is to reach g:



But, how can we actually

→ **confirm** whether a given state s is a bug?

→ **generate** suitable candidates for testing?

Policy Bugs: Examples

- Quantitative goal reachability: s is a bug if path $\sigma^\pi(s)$ is not an optimal plan.
- Qualitative goal reachability: s is a bug if $\sigma^\pi(s)$ does not reach the goal, $V^\pi(s) = 1$, but s is solvable, $V^*(s) = 0$.
- Failure formula ϕ: $\sigma^\pi(s)$ makes true the formula, $V^\pi(s) = 1$, but there exists another policy π' such that $V^{\pi'}(s) = 0$; the failure is avoidable.

Bug Confirmation

Definition (Bug Confirmation)

Bug confirmation is the problem of deciding whether, given a state s and policy π, s is a bug in π.

- Solving this problem exactly ⇔ solving $V^*(s)$
- **Test oracles**: sufficient criteria confirming some – but not all – bugs
- Exploit extensive work on planning heuristics approximating V^*!

  → Admissible (lower) bound h: $h(s) \leq V^*(s)$, for all s

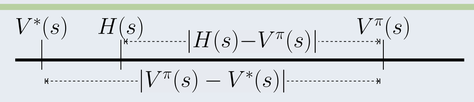
  → Upper bound H: $V^*(s) \leq H(s)$, for all s

Bug Confirmation via V^* Bounds

Idea: Derive lower bound $L \leq |V^\pi(s) - V^*(s)|$

Proposition (Bug Confirmation)

If $H(s) \leq V^\pi(s)$, then $L_H := |V^\pi(s) - H(s)|$ satisfies $L_H \leq |V^\pi(s) - V^*(s)|$.



Fuzzing Bugs

Idea: inspired by *fuzzing* in program testing

- Given some input state s_0.
- Mutate this state in order to obtain s, likely to be a bug.

Definition (Fuzzing Bug):

A state s is a **fuzzing bug** relative to some state s_0 if
$$|V^\pi(s) - V^*(s)| > |V^\pi(s_0) - V^*(s_0)|$$

Observe:

1. If s is a fuzzing bug relative to some s_0, s is a bug.
2. Every bug s with non-minimal testing-objective gap is a fuzzing bug relative to some s_0.

Conclusion & Outlook

- Decision-making via NNs becomes increasingly popular.

⇒ Raises the need to gain trust in such decisions.

- First step in the exploration of *testing* in a planning context.
- Analysis of use of planning heuristics for bug confirmation.
- Experiments in classical planning; showcasing that even with the present tools, we are already able to find bugs effectively.

What now?

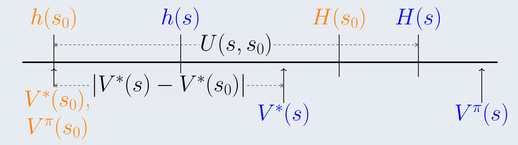
- Bug confirmation methods.
- Fuzzing approaches.
- Fault localization.
- What to do with the found bugs? (Retraining)

Fuzzing-Bug Confirmation via V^* Bounds

Observe: s is a fuzzing-bug relative to s_0 iff **(a)** $|V^\pi(s) - V^\pi(s_0)| > |V^*(s) - V^*(s_0)|$ and $V^\pi(s) \geq V^\pi(s_0)$, or **(b)** $|V^\pi(s) - V^\pi(s_0)| < |V^*(s) - V^*(s_0)|$ and $V^*(s) \leq V^*(s_0)$, or **(c)** $|V^\pi(s) - V^\pi(s_0)| = |V^*(s) - V^*(s_0)| \neq 0$ and $V^\pi(s) \geq V^\pi(s_0)$ and $V^*(s) \leq V^*(s_0)$.

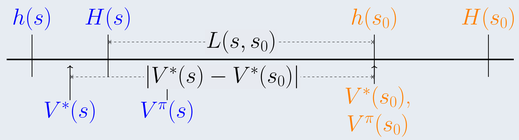
Proposition (Fuzzing-Bug Confirm. (a))

Let $\delta_1 := |h(s) - H(s_0)|$, $\delta_2 := |H(s) - h(s_0)|$, and $U(s, s_0) := \max(\delta_1, \delta_2)$. Then s is a fuzzing-bug relative to s_0 if $|V^\pi(s) - V^\pi(s_0)| > U(s, s_0)$ and $V^\pi(s) \geq V^\pi(s_0)$.



Proposition (Fuzzing-Bug Confirm. (b))

Say that $H(s) < h(s_0)$, and let $L(s, s_0) := |H(s) - h(s_0)|$. Then s is a fuzzing bug relative to s_0 if $|V^\pi(s) - V^\pi(s_0)| < L(s, s_0)$.



Testing Classical Planning ASNet Policies

(A) Benchmarks			(B) Pool P		(C) Oracles Comparison recall (%)						(D) PolQualBias recall (%)				(E) NoveltyFilter recall (%)				(E) NoveltyFilter bugs w/ unique fact pair (%)				(E) NoveltyFilter #bug regions			
					qual $P \setminus P_\pi$			quant P_π			qual P		quant P_π		qual P		quant P_π		qual		quant		qual		quant	
Domain	Σ	Σ_π	$\lvert P\rvert$	$\lvert P_\pi\rvert$	undo	look	perf	inv	look	aras	off	on	off	on	off	on	off	on	off	on	off	on	off	on	off	on
Toyer et al. (2020) ASNet Policies																										
Blocksw	30	24	144.6	28.3	100	65.3	100	5.0	49.0	20.1	8.5	64.4	33.8	56.5	64.3	64.4	61.6	56.5	64.2	70.9	96.2	96.4	4.4	7.6	3.4	4.2
MatchBl	51	6	139.3	8.0	–	0.5	1.1	–	8.3	60.0	5.9	1.1	53.5	60.0	4.8	1.1	50.0	60.0	89.4	100	100	100	0.7	0.5	0.5	1.0
Own ASNet Policies on IPC Benchmarks																										
Floortil	20	14	186.2	1.5	–	0.7	4.6	–	33.7	49.0	4.1	4.6	37.9	49.0	6.1	4.6	50.0	49.0	90.6	91.9	97.6	97.6	0.4	0.4	0.6	0.6
Gripper	35	35	58.3	58.3				0.5	15.2	90.1	0.0	0.0	87.1	90.2	0.0	0.0	87.8	90.2			66.9	78.3	0.0	0.0	18.4	29.2
Satellite	20	16	114.0	9.3	100	88.2	100	1.0	70.5	37.4	16.9	85.7	38.3	74.9	88.7	85.7	70.0	74.9	65.9	72.9	85.3	97.7	4.4	5.2	0.9	2.4
Scanaly	50	50	63.9	19.2	100	40.7	100	0.1	12.3	7.9	4.0	28.5	13.8	16.3	29.0	28.5	15.3	16.3	64.9	73.0	99.0	100	3.3	5.8	1.9	2.0
Spanner	40	40	142.8	2.9	–	0.0	0.0	–	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0					0.0	0.0	0.0	0.0
Storage	30	7	145.3	58.4	100	88.5	100	0.4	13.9	23.4	21.0	44.5	17.1	23.4	51.3	44.5	24.7	23.4	38.8	58.3	98.8	98.3	1.1	7.1	4.9	5.6
Transp	60	24	151.2	51.4	100	97.8	100	1.5	37.1	48.6	17.7	50.9	46.7	48.8	56.2	50.9	46.4	48.8	37.0	58.4	78.7	92.8	1.1	5.7	8.7	14.8
VisitAll	79	21	78.7	3.2	100	19.8	100	2.9	12.2	15.4	79.9	86.9	28.8	23.1	87.1	86.9	26.2	23.1	71.2	81.9	100	100	6.3	9.5	0.5	0.7

- Test pool generation via random walks
- **Policy quality bias**: bias walks to states with larger $V^\pi(s)$ values.
- **Novelty filter**: only add a random-walk end-state into the pool if that contributes some new fact pair.