Debugging a Policy: A Framework for Automatic Action Policy Testing

Marcel Steinmetz, Timo P. Gros, Philippe Heim, Daniel Höller, Jörg Hoffmann

July 5, 2021
What is a “Bug”? But what about trust in a learned neural action policy?
what is a “bug”? 

outlook references 

e.g. PRL, e.g. [Toyer et al. (2018); Issakkimuthu et al. (2018); Groshev et al. (2018); Garg et al. (2019); Rivlin et al. (2020); Toyer et al. (2020)] 

but what about trust in a learned neural action policy?
What is a “Bug”?

Bug Confirmation

Outlook

References

- PRL, e.g. Toyer et al. (2018); Issakkimuthu et al. (2018); Groshev et al. (2018); Garg et al. (2019); Rivlin et al. (2020); Toyer et al. (2020)

But what about trust in a learned neural action policy?
What is a "Bug"?

But what about trust in a learned neural action policy?
e.g. PRL, e.g. [Toyer et al. (2018); Issakkimuthu et al. (2018); Groshev et al. (2018); Garg et al. (2019); Rivlin et al. (2020); Toyer et al. (2020)]
e.g. PRL, e.g. [Toyer et al. (2018); Issakkimuthu et al. (2018); Groshev et al. (2018); Garg et al. (2019); Rivlin et al. (2020); Toyer et al. (2020)]

But what about trust in a learned neural action policy?
- Explanation, e.g. [Chakraborti et al. (2019); Agogino et al. (2019)]
- Visualization, e.g. [Gros et al. (2020)]
- Shielding, e.g. [Könighofer et al. (2017); Alshiekh et al. (2017); Fulton and Platzer (2018)]
- Verification, e.g. [Katz et al. (2017); Gehr et al. (2018); Akintunde et al. (2018); Vinzent and Hoffmann (2021)]
• Explanation, e.g. [Chakraborti et al. (2019); Agogino et al. (2019)]
• Visualization, e.g. [Gros et al. (2020)]
• Shielding, e.g. [Könighofer et al. (2017); Alshiekh et al. (2017); Fulton and Platzer (2018)]
• Verification, e.g. [Katz et al. (2017); Gehr et al. (2018); Akintunde et al. (2018); Vinzent and Hoffmann (2021)]
• Testing
Explanation, e.g. [Chakraborti et al. (2019); Agogino et al. (2019)]

Visualization, e.g. [Gros et al. (2020)]

Shielding, e.g. [Könighofer et al. (2017); Alshiekh et al. (2017); Fulton and Platzer (2018)]

Verification, e.g. [Katz et al. (2017); Gehr et al. (2018); Akintunde et al. (2018); Vinzent and Hoffmann (2021)]

Testing

e.g. [Julian et al. (2020)] in model-free (blackbox environment) setting for image-based NN controllers.

But in ICAPS community? None that I know of.
Explanation, e.g. [Chakraborti et al. (2019); Agogino et al. (2019)]

Visualization, e.g. [Gros et al. (2020)]

Shielding, e.g. [Könighofer et al. (2017); Alshiekh et al. (2017); Fulton and Platzer (2018)]

Verification, e.g. [Katz et al. (2017); Gehr et al. (2018); Akintunde et al. (2018); Vinzent and Hoffmann (2021)]

Testing

e.g. [Julian et al. (2020)] in model-free (blackbox environment) setting for image-based NN controllers.

But in ICAPS community? None that I know of.

→ “Is this planning?”
Explanation, e.g. [Chakraborti et al. (2019); Agogino et al. (2019)]

Visualization, e.g. [Gros et al. (2020)]

Shielding, e.g. [Könighofer et al. (2017); Alshiekh et al. (2017); Fulton and Platzer (2018)]

Verification, e.g. [Katz et al. (2017); Gehr et al. (2018); Akintunde et al. (2018); Vinzent and Hoffmann (2021)]

Testing
e.g. [Julian et al. (2020)] in model-free (blackbox environment) setting for image-based NN controllers.

But in ICAPS community? None that I know of.

→ “Is this planning?” Some of it surely is (you’ll see).
Explanation, e.g. [Chakraborti et al. (2019); Agogino et al. (2019)]

Visualization, e.g. [Gros et al. (2020)]

Shielding, e.g. [Könighofer et al. (2017); Alshiekh et al. (2017); Fulton and Platzer (2018)]

Verification, e.g. [Katz et al. (2017); Gehr et al. (2018); Akintunde et al. (2018); Vinzent and Hoffmann (2021)]

Testing

e.g. [Julian et al. (2020)] in model-free (blackbox environment) setting for image-based NN controllers.

But in ICAPS community? None that I know of.

→ “Is this planning?” Some of it surely is (you’ll see).
→ “Is this PRL?”
Explanation, e.g. [Chakraborti et al. (2019); Agogino et al. (2019)]

Visualization, e.g. [Gros et al. (2020)]

Shielding, e.g. [Könighofer et al. (2017); Alshiekh et al. (2017); Fulton and Platzer (2018)]

Verification, e.g. [Katz et al. (2017); Gehr et al. (2018); Akintunde et al. (2018); Vinzent and Hoffmann (2021)]

Testing

  e.g. [Julian et al. (2020)] in model-free (blackbox environment) setting for image-based NN controllers.

  But in ICAPS community? None that I know of.

→ “Is this planning?” Some of it surely is (you’ll see).

→ “Is this PRL?” You tell me :-)
Explanation, e.g. [Chakraborti et al. (2019); Agogino et al. (2019)]

Visualization, e.g. [Gros et al. (2020)]

Shielding, e.g. [Könighofer et al. (2017); Alshiekh et al. (2017); Fulton and Platzer (2018)]

Verification, e.g. [Katz et al. (2017); Gehr et al. (2018); Akintunde et al. (2018); Vinzent and Hoffmann (2021)]

Testing
e.g. [Julian et al. (2020)] in model-free (blackbox environment) setting for image-based NN controllers.

But in ICAPS community? None that I know of.

→ “Is this planning?” Some of it surely is (you’ll see).
→ “Is this PRL?” You tell me :-) New workshop Trusted AIP?
1. Context & Notation
2. What is a “Bug”?  
3. Bug Confirmation  
4. Outlook
Agenda

1  Context & Notation

2  What is a “Bug”? 

3  Bug Confirmation

4  Outlook
Planning Models Addressed

Everything.
Planning Models Addressed

Everything.

- Classical planning
- Contingent planning
- Oversubscription planning
- Discounted-reward/MaxProb MDPs
- ⟨InsertYourFavoriteModelHere⟩
Planning Models Addressed

Everything.

- Classical planning
- Contingent planning
- Oversubscription planning
- Discounted-reward/MaxProb MDPs
- $\langle$InsertYourFavoriteModelHere$\rangle$

$\rightarrow$ All we assume is that learning a policy $\pi : states \mapsto actions$ makes sense, and that a value function $V^\pi : states \mapsto \mathbb{R}$ can be defined which captures the quality of $\pi$ run on $s$. 
Generic (Cross-Planning-Model) Notation

Qualitative value function:

\[ V^\pi(s) := \begin{cases} 
0 & \text{no run of } \pi \text{ on } s \text{ reaches the goal} \\
0.5 & \text{some runs of } \pi \text{ on } s \text{ reach the goal} \\
1 & \text{all runs of } \pi \text{ on } s \text{ reach the goal} 
\end{cases} \]
Generic (Cross-Planning-Model) Notation

Qualitative value function:

\[ V^\pi(s) := \begin{cases} 
0 & \text{no run of } \pi \text{ on } s \text{ reaches the goal} \\
0.5 & \text{some runs of } \pi \text{ on } s \text{ reach the goal} \\
1 & \text{all runs of } \pi \text{ on } s \text{ reach the goal} 
\end{cases} \]

Optimal value function:

\[ V^*(s) := \begin{cases} 
\min_\pi V^\pi(s) & \text{objective is minimization} \\
\max_\pi V^\pi(s) & \text{objective is maximization} 
\end{cases} \]
Generic (Cross-Planning-Model) Notation

**Qualitative value function:**

\[
V^\pi(s) := \begin{cases} 
0 & \text{no run of } \pi \text{ on } s \text{ reaches the goal} \\
0.5 & \text{some runs of } \pi \text{ on } s \text{ reach the goal} \\
1 & \text{all runs of } \pi \text{ on } s \text{ reach the goal}
\end{cases}
\]

**Optimal value function:**

\[
V^*(s) := \begin{cases} 
\min_\pi V^\pi(s) & \text{objective is minimization} \\
\max_\pi V^\pi(s) & \text{objective is maximization}
\end{cases}
\]

**Generic “is better than” notation:** (for the record)

\[
V(s') \prec V(s) : \text{iff } \begin{cases} 
V(s') < V(s) & \text{objective is minimization} \\
V(s') > V(s) & \text{objective is maximization}
\end{cases}
\]
Agenda

1. Context & Notation
2. What is a “Bug”?
3. Bug Confirmation
4. Outlook
Definition: Bug

Definition (Bug)

A state $s$ is a **bug** in policy $\pi$ if $\Delta := |V^\pi(s) - V^*(s)| > 0$. 
Definition: Bug

A state $s$ is a **bug** in policy $\pi$ if $\Delta := |V^\pi(s) - V^*(s)| > 0$.

- Classical planning, qualitative: $\Delta = 1 \equiv \pi$ does not reach the goal on solvable state.
- Contingent planning, qualitative: $\Delta = 0.5 \equiv \pi$ does not reach the goal on some solvable states.
- Oversubscription planning/rewards: $\Delta$ rewards less than possible.
- MaxProb MDPs: reach goal with $\Delta$ less probability than possible.
Definition: Bug

A state $s$ is a **bug** in policy $\pi$ if $\Delta := |V^\pi(s) - V^*(s)| > 0$.

- Classical planning, qualitative: $\Delta = 1 \equiv \pi$ does not reach the goal on solvable state.
- Contingent planning, qualitative: $\Delta = 0.5 \equiv \pi$ does not reach the goal on some solvable states.
- Oversubscription planning/rewards: $\Delta$ rewards less than possible.
- MaxProb MDPs: reach goal with $\Delta$ less probability than possible.

**Notes:**

- Bug-free $\Rightarrow$ optimal.
Definition: Bug

Definition (Bug)

A state $s$ is a **bug** in policy $\pi$ if $\Delta := |V^\pi(s) - V^*(s)| > 0$.

- Classical planning, qualitative: $\Delta = 1 \equiv \pi$ does not reach the goal on solvable state.
- Contingent planning, qualitative: $\Delta = 0.5 \equiv \pi$ does not reach the goal on some solvable states.
- Oversubscription planning/rewards: $\Delta$ rewards less than possible.
- MaxProb MDPs: reach goal with $\Delta$ less probability than possible.

**Notes:**

- Bug-free $\Rightarrow$ optimal.
- This would *not* be the case for bug $:= \text{action starting optimal policy}$.
Definition: Fuzzing Bug

Definition (Fuzzing Bug)

A state $s'$ is a **fuzzing-bug** relative to $s$ if

$$\Delta := |V^\pi(s') - V^*(s')| - |V^\pi(s) - V^*(s)| > 0.$$
Definition: Fuzzing Bug

Definition (Fuzzing Bug)

A state \( s' \) is a **fuzzing-bug** relative to \( s \) if

\[
\Delta := |V^\pi(s') - V^*(s')| - |V^\pi(s) - V^*(s)| > 0.
\]

**Observe:** (trivial)

1. If \( s' \) is a fuzzing-bug relative to some \( s \), then \( s' \) is a bug.
Definition: Fuzzing Bug

Definition (Fuzzing Bug)

A state $s'$ is a fuzzing-bug relative to $s$ if

$$\Delta := |V^\pi(s') - V^*(s')| - |V^\pi(s) - V^*(s)| > 0.$$ 

Observe: (trivial)

1. If $s'$ is a fuzzing-bug relative to some $s$, then $s'$ is a bug.
2. Every bug $s'$ with non-minimal optimality gap $|V^\pi(s) - V^*(s)|$ is a fuzzing-bug relative to some $s$. 
Definition: Fuzzing Bug

Definition (Fuzzing Bug)

A state $s'$ is a **fuzzing-bug** relative to $s$ if

$$\Delta := |V^\pi(s') - V^*(s')| - |V^\pi(s) - V^*(s)| > 0.$$ 

Observe: (trivial)

1. If $s'$ is a fuzzing-bug relative to some $s$, then $s'$ is a bug.
2. Every bug $s'$ with non-minimal optimality gap $|V^\pi(s) - V^*(s)|$ is a fuzzing-bug relative to some $s$.

Why?

- Natural situation in fuzzing algorithms.
- 2. does not hold under restrictions on reachability of $s'$ from $s$ by such algorithms.
Definition: Fuzzing Bug

A state $s'$ is a **fuzzing-bug** relative to $s$ if
$$
\Delta := |V^\pi(s') - V^*(s')| - |V^\pi(s) - V^*(s)| > 0.
$$

**Observe:** (trivial)

1. If $s'$ is a fuzzing-bug relative to some $s$, then $s'$ is a bug.
2. Every bug $s'$ with non-minimal optimality gap $|V^\pi(s) - V^*(s)|$ is a fuzzing-bug relative to some $s$.

**Why?**

- Natural situation in **fuzzing** algorithms.
- 2. does not hold under restrictions on reachability of $s'$ from $s$ by such algorithms.
- Can this definition help in **bug confirmation**?
Agenda

1. Context & Notation
2. What is a “Bug”? 
3. Bug Confirmation 
4. Outlook

Jörg Hoffmann

Debugging a Policy
Definition (Bug Confirmation)

**Bug confirmation** is the problem of deciding, given a state $s$, whether or not $s$ is a bug.

$\rightarrow$ Obviously, solving this problem exactly involves solving $s$ optimally. (I told you some of it is planning, didn’t I?)
**Definition (Bug Confirmation)**

*Bug confirmation* is the problem of deciding, given a state $s$, whether or not $s$ is a bug.

→ Obviously, solving this problem exactly involves solving $s$ optimally. (I told you some of it is planning, didn’t I?)

**So we approximate . . .** [Patrik Haslum, AIPS’00]
Definition (Bug Confirmation)

**Bug confirmation** is the problem of deciding, given a state $s$, whether or not $s$ is a bug.

→ Obviously, solving this problem exactly involves solving $s$ optimally. (I told you some of it is planning, didn’t I?)

So we approximate . . . [Patrik Haslum, AIPS’00]

With $H_* \succeq V^*(s)$ and $h_\pi(s) \preceq V^\pi(s)$ pessimistic approximation of $V^*$ and optimistic approximation of $V^\pi$ respectively:

Proposition (Bug Confirmation)

Say that $V^*(s) \preceq H_*(s)$ and $h_\pi(s) \preceq V^\pi(s)$. Say that $h_\pi(s) \succeq V^*(s)$ and $H_*(s) \preceq V^\pi(s)$. Then $|h_\pi(s) - H_*(s)| \leq |V^\pi(s) - V^*(s)|$. 
**Bug Confirmation**

**Definition (Bug Confirmation)**

**Bug confirmation** is the problem of deciding, given a state $s$, whether or not $s$ is a bug.

→ Obviously, solving this problem exactly involves solving $s$ optimally. (I told you some of it is planning, didn’t I?)

**So we approximate . . .** [Patrik Haslum, AIPS’00]

With $H_* \geq V^*(s)$ and $h_\pi(s) \leq V^\pi(s)$ pessimistic approximation of $V^*$ and optimistic approximation of $V^\pi$ respectively:

**Proposition (Bug Confirmation)**

*Say that $V^*(s) \leq H_*(s)$ and $h_\pi(s) \leq V^\pi(s)$. Say that $h_\pi(s) \geq V^*(s)$ and $H_*(s) \leq V^\pi(s)$. Then $|h_\pi(s) - H_*(s)| \leq |V^\pi(s) - V^*(s)|$.*

→ Boils down to: “evaluate $V^\pi(s)$, and try to find a better policy for $s$”.
Proposition (Fuzzing Bug Confirmation)

(a) \( I_*(s) \cap I_*(s') = \emptyset \), \( s' \) is a fuzzing-bug relative to \( s \) if \( H_*(s') < h_*(s) \) and either \( V^\pi(s') \succeq V^\pi(s) \) or \( |V^\pi(s') - V^\pi(s)| < |H_*(s') - h_*(s)| \).

(b) \( s' \) is a fuzzing-bug relative to \( s \) if \( V^\pi(s') \succeq V^\pi(s) \) and \( |V^\pi(s') - V^\pi(s)| > U_*(s,s') \).
**Proposition (Fuzzing Bug Confirmation)**

(a) If $I_*(s) \cap I_*(s') = \emptyset$, $s'$ is a fuzzing-bug relative to $s$ if $H_*(s') < h_*(s)$ and either $V^\pi(s') \succeq V^\pi(s)$ or $|V^\pi(s') - V^\pi(s)| < |H_*(s') - h_*(s)|$.

(b) $s'$ is a fuzzing-bug relative to $s$ if $V^\pi(s') \succeq V^\pi(s)$ and $|V^\pi(s') - V^\pi(s)| > U_*(s,s')$.

**Theorem (It’s All in Vain)**

Boils down to “evaluate $V^\pi(s)$, and try to find a better policy for $s$”.

---

Jörg Hoffmann  
PRL'21  
Debugging a Policy  
13/21
Bug Confirmation, ctd.

Proposition (Fuzzing Bug Confirmation)

(a) If \( I_*(s) \cap I_*(s') = \emptyset \), \( s' \) is a fuzzing-bug relative to \( s \) if \( H_*(s') < h_*(s) \) and either \( V^\pi(s') \succeq V^\pi(s) \) or \( |V^\pi(s') - V^\pi(s)| < |H_*(s') - h_*(s)| \).

(b) \( s' \) is a fuzzing-bug relative to \( s \) if \( V^\pi(s') \succeq V^\pi(s) \) and \( |V^\pi(s') - V^\pi(s)| > U_*(s, s') \).

Theorem (It’s All in Vain)

Boils down to “evaluate \( V^\pi(s) \), and try to find a better policy for \( s \)”.

So what?

- Many special cases with “\( V^* \) oracle” (e.g. all states known to be solvable; enough time during at testing to run symbolic planner).
- In general case, plug in plan-quality improvement algorithms [Bäckström (1998); Do and Kambhampati (2003); Nakhost and Müller (2010); Siddiqui and Haslum (2015)].
Agenda

1. Context & Notation
2. What is a “Bug”?  
3. Bug Confirmation  
4. Outlook
Ok, so now let’s actually do this!
Ok, so now let’s actually do this!

- Develop fuzzing methods!
- Develop bug confirmation paradigms (metamorphosic testing etc)!
- See what all this does in all your favorite planning and learning scenarios!
Thanks for listening.

Questions?


References II


