Program Synthesis from Rewards using Probe and FrAngel

Impact of Exploration-Exploitation Configurations on Probe and FrAngel in Minecraft

1. Introduction

Program synthesis - Field where a program is searched that satisfies a user specification. The specification can be input/output examples, incomplete programs, or natural language.
Novel user specification - **learning from rewards**

Probe - a bottom-up synthesizer that uses partial solutions to update a probabilistic grammar that guides the search.
FrAngel - a random-search iterator that *"mines"* fragments from partial solutions to bias the search towards promising programs.

Exploitation vs Exploration - Excessive exploration implies too many programs, while excessive exploitation results in too few programs.
MineRL - a python library for reinforcement learning in Minecraft

2. Research questions

"How do different exploration-exploitation configurations affect the performance of FrAngel and Probe in MineRL?"

"How do we use existing synthesizers to learn from rewards?

3. Methodology

Probe and FrAngel were **generalized** to allow reward-based specifications and different search iterators.
The Probe algorithm was adapted to use rewards instead of examples, while FrAngel discretized the reward space into individual test cases.

Integration of the algorithms in the Minerl environment.
No image processing was done -> Learn ***only*** from rewards -> Play Minecraft **blindfolded**

4. Experiment setup

A **navigation** task from MineRL was used to evaluate different configurations of exploitation-exploration tradeoffs.
The player has to move 64 blocks to reach a diamond.
Rewards indicate how much closer the player is to the target after each step.

5.1 Results Probe

An **alternating iterator** was implemented that, with probability *p*, generates random programs and otherwise uses the bottom-up iterator.
The configuration with p = 0.3 performs better than others.

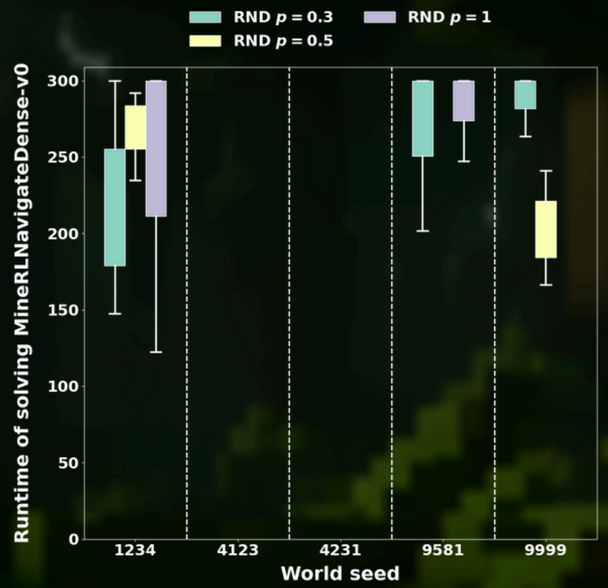

Figure 1. Box plot of the runtime of solving the navigation task for different probabilities and world seeds. The timeout is 300 seconds.

Figure 1 suggests that using random search for ***30%*** of the time provides a good balance between exploration and exploitation.
The downside of adding random exploration is that the standard deviation increases.

5.2 Results FrAngel

At each step, FrAngel chooses to use existing **fragments** or generate random programs.
If fragments are chosen, they can remain intact or be randomly modified to allow for more exploration.

5.2 Continuation

This experiment uses different **mutation probabilities** for fragments.
The configuration with a probability of 0 achieves the most consistent performance across different world seeds.

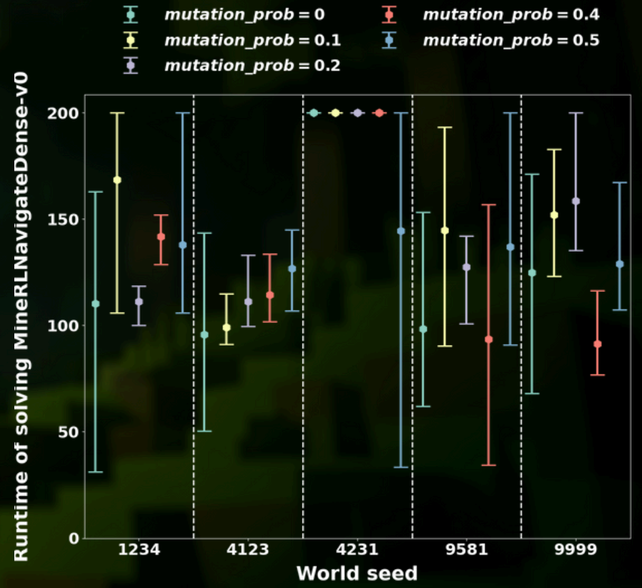

Figure 2. Plot with the maximum, minimum, and mean runtime of FrAngel when using different mutation probabilities. The timeout is 200 seconds.

Figure 2 shows that random mutation can help, as the run with a probability of 0.5 is the only one able to solve seed 4231.
However, less mutation performs better on seeds 1234 and 4123.

6. Conclusion

Both algorithms proved effective when applied to Minecraft navigation tasks.
For Probe, exploring random programs 30% of the time proves to be a good option.

For FrAngel, increasing fragment exploration only helps in some cases as the algorithm already uses random search.

Author
Nicolae Filat
n.filat@student.tudelft.nl

Responsible Professor
Sebastijan Dumančić
s.dumancic@tudelft.nl

Supervisor
Tilman Hinnerichs
t.r.hinnerichs@tudelft.nl