Evolving a Search Procedure for Program Synthesis

1 Background

Program Synthesis

A task of constructing programs meeting formal specifications [1].

- Three different domains with hundreds of tasks are considered: robot path-planning, string transformations and ASCII art.
- For each task, a set of examples consisting of **input and output** is provided. The aim of the program synthesizer is to construct programs that, given the input, return the desired output.
- Each of the domains is assigned a separate set of tokens comprising the Domain-Specific Language (**DSL**) from which the programs are constructed.
- To search the program space, a range of **different search procedures** can be used or even combined.

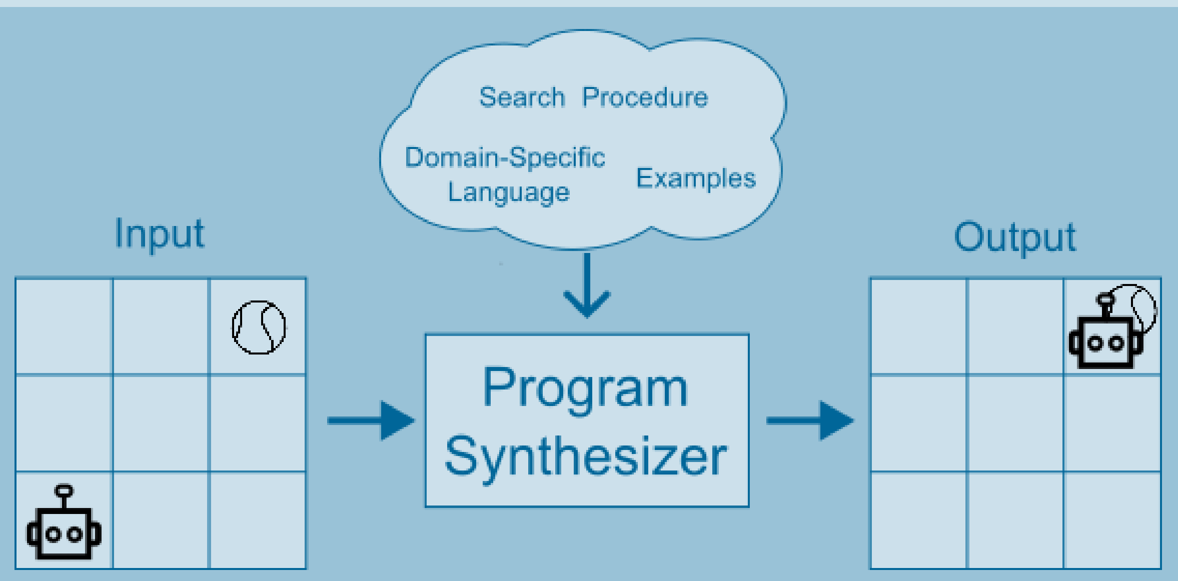


Fig. 1: Visualization of program synthesis in the robot path-planning domain.

Genetic Algorithms

Starting from a set of the randomly generated **individuals** (the first generation), a genetic algorithm performs the **selection** of the **fittest** specimen and propagates their **genes** further using **mutation** and **crossover** operators [2]. The process continues until a desired individual is found.

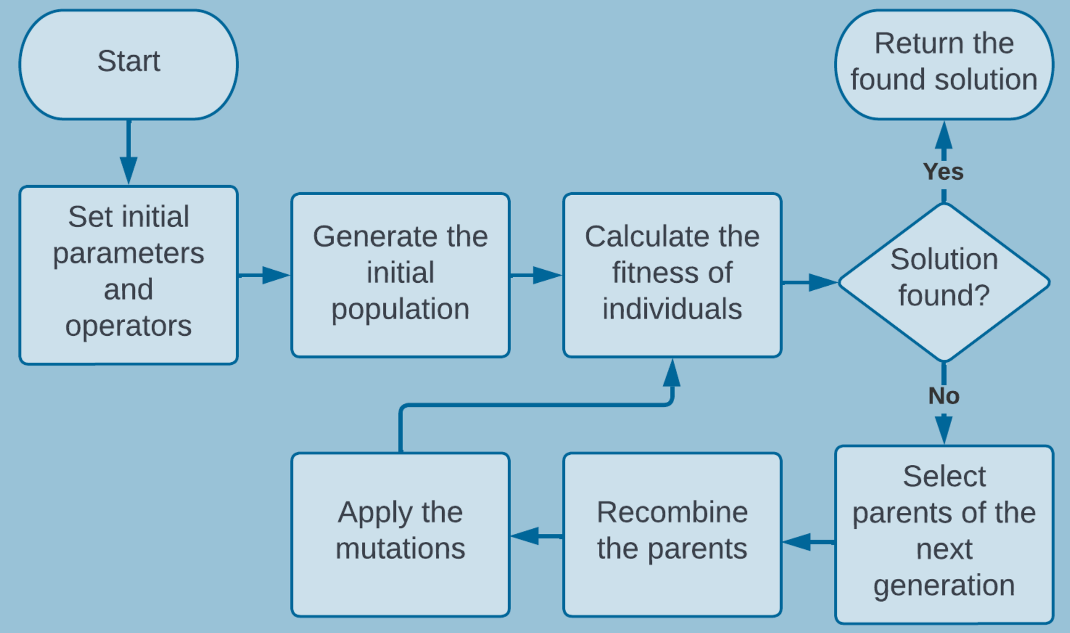

Start
Set initial parameters and operators
Generate the initial population
Calculate the fitness of individuals
Solution found?
Yes
Return the found solution
No
Select parents of the next generation
Recombine the parents
Apply the mutations

Fig. 2: Graph depciting steps of a genetic algorithm.

2 Research Question

Is it possible to develop an efficient search procedure as a combination of simpler search procedures by means of genetic algorithms?

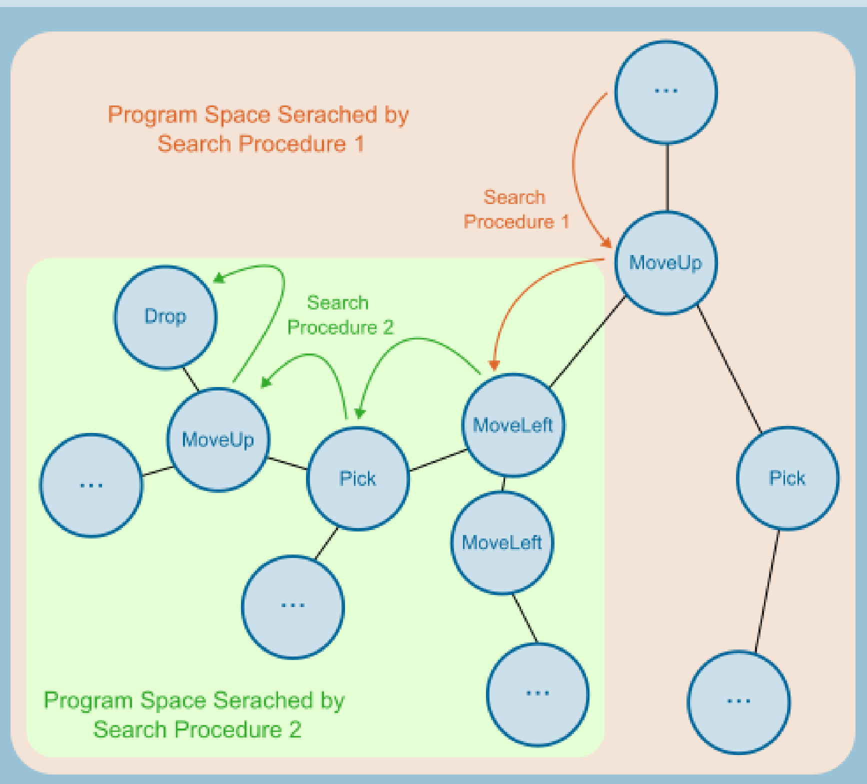

Program Space Serached by Search Procedure 1
Search Procedure 1
Search Procedure 2
Program Space Serached by Search Procedure 2
...
MoveUp
Drop
MoveUp
Pick
MoveLeft
MoveLeft
Pick
...

Fig. 3 Combining multiple search procedures to search the program space consisting of tokens (blue nodes).

3 Methodology

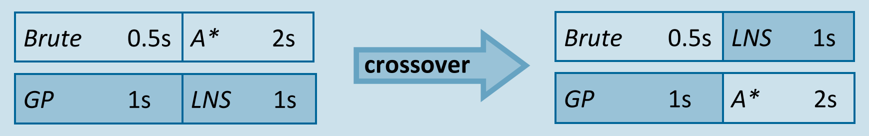
- **Literature study** to discover optimal operators and parameters
- **Designing the** genetic algorithm
- **Performing experiments** to establish optimal value of mutation probability and compare fitness functions
- **Evolving** the most optimal search procedures on three different domains and comparing results

4 Evolution Process

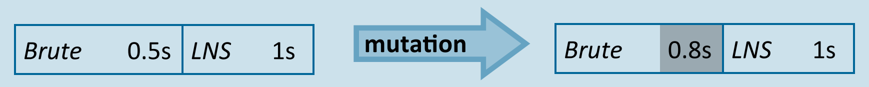
1. Chromosome are defined as a chain of ordered search procedures and timeouts. Initial population of fixed size is chosen randomly.

LNS	1s	*MH*	0.5s	*A**	0.2s

2. Each chromosome is assigned fitness depending on how fast and how many tasks the search procedure solves. Two different fitness functions are considered.
3. Parents of the next generation are selected using tournament selection.
4. Two-point crossover is used to combine the parents with a probability of 0.8.



5. A range of mutation operators is used to mutate the resulting individuals. Multiple mutation probabilities are tested.

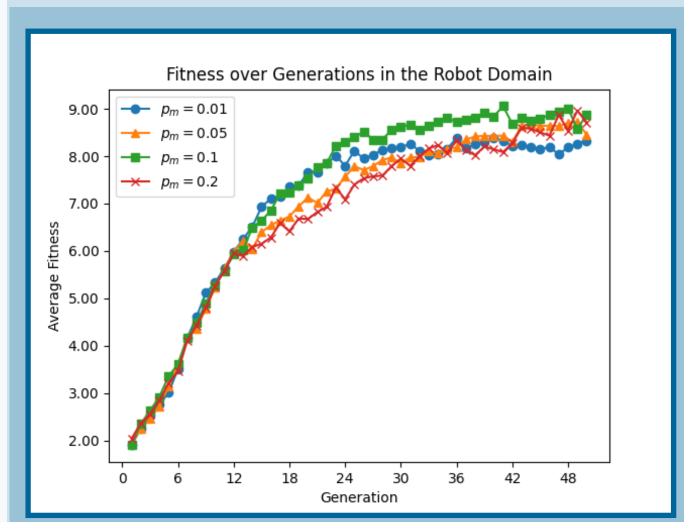


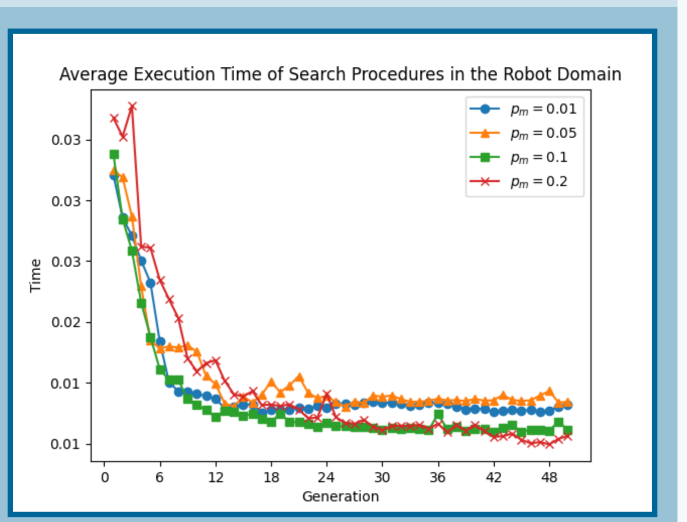
6. The process is repeated for around 30 to 50 generations, depending on the domain.

5 Results

Different Mutation Probabilities

Several values of a mutation proability have been tested in the robot domain. This parameter represents how likely each chromosome is to mutate before being placed in the next generation.





Figs. 4/5: Average fitness and execution times over generations with different mutation probabilities.

The **mutation probability of 0.1** manages to find a perfect degree of randomness without descending into a chaotic search. This claim has been confirmed by experiments performed in two other domains.

Two fitness functions

$$fit_{weak} = \begin{cases} \frac{succ}{time} & \text{if succ} \geq 0.5 \\ 0 & \text{if succ} < 0.5 \end{cases} \qquad fit_{strong} = \begin{cases} 0.5 * succ & \text{if succ} < 1 \\ 0.5 * succ + \frac{0.5}{time} & \text{if succ} = 1 \end{cases}$$

Where *succ* is the success rate, *time* is the summed timeout.

Weak fitness function manages to find a balance between the number of solved tasks and the synthesis time. One of the chains synthesised by this function is highlighted below.

Strong fitness function aims for programs that solve all the tasks. No combined search procedures have been found to be more efficient using this method.

Evolved Combination of Search Methods

Combination of Brute and A* algorithms result manages to solve more tasks than their singular counterparts in the string domain. Brute is a greedy search algorithm. First, the search space is trimmed by Brute to then focus on a smaller part of it with a more exploratory A*.

Seq (ms)	Avg Time (ms)	Acc (%)
AS 38.96 → AS 56.74	14.92	52.48
Brute 38.96 → Brute 56.74	30.08	51.38
Brute 38.96 → AS 56.74	22.25	65.97

Table 1: Comparison between a combined search procedure of Brute and A* and two procedures consisting of their singular equivalents.

Fitness	Search	Timeout (s)	Avg_t (s)	Acc (%)
Strong	AS	2.267	0.721	99.1
Weak	AS	0.501	0.366	47.8

Table 2: Comparison between searches evolved using two different fitness functions. Acc stands for accuracy, Avg_ t is the averate execution time per task.

Both fitness methods can be considered valid and their suitability relies on the preference of the user, who needs to answer the question of whether sacrificing half of the tasks is worth the difference in the execution time.

6 Conclusion

The performed experiments have proven that **it indeed is possible to develop an efficient, combined search procedure with the use of genetic algorithms.** However, the results are strongly dependent on the domain, considered search procedures, and the definition an 'efficient search procedure'.

Michał Teodor Okoń

Responsible professor:
Sebastijan Dumančić

[1] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. In Foundations and Trends in Programming Languages, volume 4, chapter Introduction. 2017. doi: 10.1561/2500000010.Michael Negnevitsky.
[2] Evolutionary computation. in Artificial Intelligence A Guide to Intelligent Systems Artificial Second Edition, pages 219–257. 2005