Neuro-Guided Genetic Programming: Prioritizing Evolutionary Search with Neural Networks

Paweł Liskowski, Iwo Błądek, Krzysztof Krawiec

Institute of Computing Science, Poznań University of Technology

9 October 2018
Outline

1 Introduction

2 Search problem

3 Training neural network

4 Neuro-Guided Genetic Programming

5 Experiments
In this presentation, we will describe work originally presented at GECCO 2018 conference in Kyoto, 15-19.07.2018.

This work is also described in the peer-reviewed publication:

Automatic program synthesis: a general class of problems where the goal is to find a program (executable procedure) that satisfies a given specification.

**Specification**

2, [1 5 3] → 4  
1, [1 8 3 5] → 1  
1, [1 8 3 5 7] → 1  
2, [1 8 3 5 7] → 4  
3, [1 8 3 5 7] → 9  
2, [1 5 3 0 8] → 1  
...  

**Target program**

a ← int  
b ← [int]  
c ← Sort b  
d ← Take a c  
e ← Sum d

**Aspects:** type of specification, programming language.
Genetic programming

- Population of candidate programs.
- In each generation programs are being selected based on their *fitness* and *search operators* modify those programs, which then constitute a new population.

**Example of search operators:**

**Mutation:**

\[
\text{(ite (>= x y) 2 (+ x y))}
\]

\[
\downarrow
\]

\[
\text{(ite (< y 0) 2 (+ x y))}
\]

**Crossover:**

\[
\text{(mod x 2)}
\]

\[
\downarrow
\]

\[
\text{(ite (>= x y) 2 (+ x y))}
\]

\[
\downarrow
\]

\[
\text{(mod (+ x y) 2)}
\]

\[
\text{(ite (>= x y) 2 x)}
\]
Motivation

Problem we wanted to solve:

- Search operators work under assumption that every instruction has the same chance to lead to a correct candidate program (uniform distribution of instructions given the problem instance).
- In practice, this in vast majority of cases does not hold.

Our contribution:

- Search operators (mutation, population initialization) taking into account the conditional probability of instructions given input-output examples from the specification.
- Conditional probability of instructions is obtained by training a neural network on input-output examples.
Outline of our approach

“All” problem instances

1. Train artificial neural network (NN) to estimate conditional probability of program instructions given the I/O examples.
Outline of our approach

“All” problem instances

1. Train artificial neural network (NN) to estimate conditional probability of program instructions given the I/O examples.

Particular problem instance

1. Query the neural network on the I/O examples to obtain probability estimates.

2. Parametrize search operators (mutation, population initialization) of GP with the obtained estimates.

3. Run GP.
Artificial neural network is used, but should the whole proposed solution be treated as a classical machine learning scenario?

Tentative answer: No. Machine learning subcomponent is used to guide search, but in the end this is a search problem.
Outline

1. Introduction
2. Search problem
3. Training neural network
4. Neuro-Guided Genetic Programming
5. Experiments
Search problem

- **Goal**: find such a program in the programming language (*DeepCoder DSL*) that the specification will be met.

- **Specification**: a list of input-output examples.
DeepCoder DSL

Types:
- Int
- List[Int]

Functions:
- (10) operations on lists: Head, Last, Take, Drop, Access, Minimum, Maximum, Reverse, Sort, Sum
- (5) higher-order functions: Map, Filter, Count, ZipWith, Scanl1

Other elements of the language:
- (10) lambdas for Map (add1, sub1, multMinus1, mult2, mult3, mult4, div2, div3, div4, square).
- (4) predicates for Filter and Count (>0, <0, isOdd, isEven).
- (5) lambdas for ZipWith and Scanl (+, −, *, min, max).
DeepCoder DSL

We use the same DSL as was used in the DeepCoder paper [1].

**Program representation:**
- A variant of linear GP.
- A fixed-length sequence of instructions, each of which issues a function call, and stores it’s result in a fresh variable.

**Example program:**

P0: Compute the sum of a smallest numbers from the list $b$.

```plaintext
a ← int
b ← [int]
c ← Sort b
d ← Take a c
e ← Sum d
```

**Input:**
2, [1 8 3 5 7]

**Output:**
4
a ← int
b ← [int]

Declaring program’s input. Variable $a$ will be an arbitrary $Int$ provided by the user, and $b$ will be an arbitrary $List[Int]$. 
Program representation – functions

Every line of the program consists of a single application of a function to the previously defined variables.

For example:

```plaintext
a ← int
b ← [int]
c ← Function {a, b}^+
d ← Function {a, b, c}^+
e ← ...
```

```plaintext
a ← int
b ← [int]
c ← Sort b
d ← Take a c
e ← Sum d
```
Program representation – predicates

Some functions accept certain predicates or lambdas, which are predefined and treated as constant elements of the language.

For example (lambdas in red):

\[
\begin{align*}
a &\leftarrow \text{int} \\ b &\leftarrow [\text{int}] \\ c &\leftarrow \text{Function predicate } \{a, b\}^+ \\ d &\leftarrow \text{Function lambda } \{a, b, c\}^+ \\ e &\leftarrow \ldots
\end{align*}
\]

\[
\begin{align*}
a &\leftarrow [\text{int}] \\ b &\leftarrow [\text{int}] \\ c &\leftarrow \text{Map } (*3) \ a \\ d &\leftarrow \text{ZipWith } (+) \ c \ b \\ e &\leftarrow \text{Maximum } d
\end{align*}
\]
Example problems

P0: Compute the sum of a smallest numbers from the list \( b \).

\[
\begin{align*}
a &\leftarrow \text{int} \\
b &\leftarrow \text{[int]} \\
c &\leftarrow \text{Sort } b \\
d &\leftarrow \text{Take a c} \\
e &\leftarrow \text{Sum } d
\end{align*}
\]

Input:
2, [1 8 3 5 7]
...
Output:
4

P4: Given lists \( a \) and \( b \), compute the minimal area of rectangles of dimensions given in the input lists.

\[
\begin{align*}
x &\leftarrow \text{[int]} \\
y &\leftarrow \text{[int]} \\
c &\leftarrow \text{Sort } x \\
d &\leftarrow \text{Sort } y \\
e &\leftarrow \text{Reverse } d \\
f &\leftarrow \text{ZipWith } (\ast) \ d \ e \\
g &\leftarrow \text{Sum } f
\end{align*}
\]

Input:
[1 2 3], [4 5 6]
...
Output:
28
Outline

1. Introduction
2. Search problem
3. Training neural network
4. Neuro-Guided Genetic Programming
5. Experiments
Network architecture

Attribute Predictions

Final Activations

Pooled

Hiddens 3

Hiddens 2

Hiddens 1

State Embeddings

Program State

Inputs 1

Outputs 1

Inputs 5

Outputs 5

Network training

- Training algorithm: Adam.
- Training lasts up to 100 epochs (full passes over the training set).
- Early stopping condition: validation loss ceased to improve.
Generation of the training set

- All programs up to a certain number of instructions while removing most semantic duplicates.
- Each training case is a tuple (I/O-examples, instructions vector).
- I/O-examples are generated randomly.

**Small training set** – programs up to length 3 with most of the semantic duplicates removed. Total count: 822,582 programs.

**Large training set** – programs up to length 4 with most of the semantic duplicates removed. Total count: 5,004,532 programs.
Test sets

- 10,000 programs not present in the training set.
- Several neural architectures and learning algorithms were tested and we selected the one with the highest accuracy on the test set.

<table>
<thead>
<tr>
<th>training set</th>
<th>total programs</th>
<th>accuracy on test set (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>822,582</td>
<td>92.48</td>
</tr>
<tr>
<td>large</td>
<td>5,004,532</td>
<td>90.85</td>
</tr>
</tbody>
</table>
Heatmaps

Small training set:

Large training set:
**P0:** Compute the sum of a smallest numbers from the list \( b \).

**Specification:**
2, \([1\ 5\ 3]\) \(\rightarrow\) 4  
1, \([1\ 8\ 3\ 5]\) \(\rightarrow\) 1  
3, \([1\ 8\ 3\ 5\ 7]\) \(\rightarrow\) 9  
\ldots

**Target program:**

\[ \begin{align*} 
  a &\leftarrow \text{int} \\
  b &\leftarrow \text{[int]} \\
  c &\leftarrow \text{Sort } b \\
  d &\leftarrow \text{Take } a \ c \\
  e &\leftarrow \text{Sum } d 
\end{align*} \]
**P1:** Count the number of points of the winner. \( a \) is a list of wins (3 points), and \( b \) is a list of draws (1 point).

**Specification:**

\[
[1 \ 2], \ [1 \ 2] \rightarrow 8 \\
[1 \ 0 \ 0], \ [1 \ 1 \ 2] \rightarrow 4 \\
[2 \ 2 \ 1 \ 0], \ [1 \ 1 \ 0 \ 0] \rightarrow 7 \\
\ldots
\]

**Target program:**

\[
a \leftarrow [\text{int}] \\
b \leftarrow [\text{int}] \\
c \leftarrow \text{Map} (*3) \ a \\
d \leftarrow \text{ZipWith} (+) \ c \ b \\
e \leftarrow \text{Maximum} \ d
\]
**P4:** Compute the minimal total area of rectangles which are constructed by pairing dimensions given in lists \(a\) and \(b\).

**Specification:**

\[
[1 \ 2 \ 3], \ [1 \ 2 \ 3] \rightarrow 10 \\
[1 \ 2 \ 2], \ [1 \ 1 \ 2] \rightarrow 6 \\
\ldots
\]

**Target program:**

\[
\begin{align*}
a & \leftarrow \text{[int]} \\
b & \leftarrow \text{[int]} \\
c & \leftarrow \text{Sort } a \\
d & \leftarrow \text{Sort } b \\
e & \leftarrow \text{Reverse } d \\
f & \leftarrow \text{ZipWith } (\ast) \ d \ e \\
g & \leftarrow \text{Sum } f
\end{align*}
\]
Outline

1. Introduction
2. Search problem
3. Training neural network
4. Neuro-Guided Genetic Programming
5. Experiments
Neuro-Guided Genetic Programming

- Fixed-length, linear program representation.
- At the beginning, mutation in GP is parametrized with the result returned by network for the input-output examples in the specification.
- Apart from that, GP proceeds normally.
- All programs in a GP run have the same nominal length, computed as: length of the target program + 1.
- *Nop* operation is included, to allow for effectively shorter programs.
Search operators

**Mutation:**
- An instruction is randomly selected in the program.
- The function call is analyzed, and constructed is a set of functions with the matching signature.
- A function to insert and its arguments are selected randomly with the probabilities provided by the network (after normalization).

**Crossover:**
- Exchanging up to \( l_c = 2 \) consecutive instructions between parents.
- Signatures of the instructions must match.
- If there are no such consecutive instructions, then \( l_c \) is decreased.
- If \( l_c = 0 \), then parent programs are returned unchanged.
Outline

1. Introduction
2. Search problem
3. Training neural network
4. Neuro-Guided Genetic Programming
5. Experiments
**Evolution parameters**

- **Preliminary parameter tuning:** the probabilities of mutation and crossover $p_m, p_c \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, population size $\in \{100, 500, 1000\}$; each configuration was ran 30 times.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>1000</td>
</tr>
<tr>
<td>Max generations</td>
<td>200</td>
</tr>
<tr>
<td>Number of runs</td>
<td>50</td>
</tr>
<tr>
<td>Probability of mutation $p_m$</td>
<td>0.8</td>
</tr>
<tr>
<td>Probability of crossover $p_c$</td>
<td>0.0 or 0.5</td>
</tr>
<tr>
<td>Selection method</td>
<td>Tournament (T) or Lexicase (L)</td>
</tr>
<tr>
<td>Tournament size</td>
<td>7</td>
</tr>
<tr>
<td>Max program length</td>
<td>length of target program + 1</td>
</tr>
<tr>
<td>Number of fitness cases</td>
<td>128</td>
</tr>
</tbody>
</table>
### Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Length</strong></td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td><strong>Small training set</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Large training set</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Benchmarks the same as in the DeepCoder paper [1].
Experiment dimensions

- **Small training set** – 822,582 programs up to length 3.
- **Large training set** – 5,004,532 programs up to length 4.

Total tested configurations $= 2 \cdot \ldots$
Experiment dimensions

- **T** – Tournament selection (size 7)
- **L** – Lexicase selection

Total tested configurations $= 2 \cdot 2 \cdot \ldots$
Experiment dimensions

- C – Crossover used ($p_c = 0.5$)
- N – Crossover *not* used ($p_c = 0.0$)

Total tested configurations = $2 \cdot 2 \cdot 2 \cdot \ldots$
Experiment dimensions

- **U** – Search operators biased with a *uniform distribution*

- **P** – Search operators biased with *prior probabilities* reflecting the frequency of instructions in the training set

- **S** – Search operators biased using *NN; only mutation*

- **IS** – Search operators biased using *NN; both mutation and population initialization*

Total tested configurations $= 2 \cdot 2 \cdot 2 \cdot 4 = 32$
Observation 1

- **IS** is much better than **S**.
- Because of that, in the further analysis we present results only for the **IS** variant.

<table>
<thead>
<tr>
<th>configuration</th>
<th>avg success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (mut)</td>
<td>0.574</td>
</tr>
<tr>
<td>IS (mut, pop_init)</td>
<td>0.735</td>
</tr>
</tbody>
</table>
Observation 2

- Crossover does not make much difference for the effectiveness of search.
- Because of that, in the further analysis we focus on the N (no crossover) variant.

<table>
<thead>
<tr>
<th>configuration</th>
<th>avg success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (crossover)</td>
<td>0.573</td>
</tr>
<tr>
<td>N (no crossover)</td>
<td>0.580</td>
</tr>
</tbody>
</table>
Observation 3

- Configurations parametrized with probability estimates were better than baselines.

**Success rates for the small training set.** Legend: T (tournament), L (lexicase), U (unbiased), P (priors baseline), S (search), IS (initialization and search).

<table>
<thead>
<tr>
<th>method cx</th>
<th>$T_U$</th>
<th>$T_P$</th>
<th>$T_{IS}$</th>
<th>$L_U$</th>
<th>$L_P$</th>
<th>$L_{IS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2 (2)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>P5 (2)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.98</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>P0 (3)</td>
<td>0.70</td>
<td>0.34</td>
<td>1.00</td>
<td>0.58</td>
<td>0.40</td>
<td>1.00</td>
</tr>
<tr>
<td>P1 (3)</td>
<td>0.18</td>
<td>0.26</td>
<td>0.54</td>
<td>0.16</td>
<td>0.20</td>
<td>0.96</td>
</tr>
<tr>
<td>P7 (3)</td>
<td>0.16</td>
<td>0.34</td>
<td>0.56</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>P3 (4)</td>
<td>0.14</td>
<td>0.12</td>
<td>1.00</td>
<td>0.52</td>
<td>0.28</td>
<td>1.00</td>
</tr>
<tr>
<td>P6 (4)</td>
<td>0.08</td>
<td>0.06</td>
<td>0.04</td>
<td>0.40</td>
<td>0.82</td>
<td>0.78</td>
</tr>
<tr>
<td>P8 (4)</td>
<td>0.18</td>
<td>0.10</td>
<td>0.28</td>
<td>0.36</td>
<td>0.26</td>
<td>0.82</td>
</tr>
<tr>
<td>P4 (5)</td>
<td>0.14</td>
<td>0.02</td>
<td>0.00</td>
<td>0.52</td>
<td>0.38</td>
<td>0.14</td>
</tr>
<tr>
<td>mean</td>
<td>0.40</td>
<td>0.36</td>
<td>0.60</td>
<td>0.61</td>
<td>0.59</td>
<td>0.86</td>
</tr>
<tr>
<td>rank</td>
<td>10.72</td>
<td>12.00</td>
<td>8.28</td>
<td>8.50</td>
<td>8.17</td>
<td>4.17</td>
</tr>
</tbody>
</table>
Observation 3

- Configurations parametrized with probability estimates were better than baselines.

**Success rates for the large training set.** Legend: T (tournament), L (lexicase), U (unbiased), P (priors baseline), S (search), IS (initialization and search).

<table>
<thead>
<tr>
<th>method cx</th>
<th>T_U</th>
<th>T_P</th>
<th>T_IS</th>
<th>L_U</th>
<th>L_P</th>
<th>L_IS</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2 (2)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>P5 (2)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.98</td>
<td>0.98</td>
<td>1.00</td>
</tr>
<tr>
<td>P0 (3)</td>
<td>0.70</td>
<td>0.34</td>
<td>1.00</td>
<td>0.58</td>
<td>0.54</td>
<td>1.00</td>
</tr>
<tr>
<td>P1 (3)</td>
<td>0.18</td>
<td>0.20</td>
<td>0.58</td>
<td>0.16</td>
<td>0.16</td>
<td>0.98</td>
</tr>
<tr>
<td>P7 (3)</td>
<td>0.16</td>
<td>0.28</td>
<td>0.78</td>
<td>1.00</td>
<td>0.98</td>
<td>1.00</td>
</tr>
<tr>
<td>P3 (4)</td>
<td>0.14</td>
<td>0.10</td>
<td>0.68</td>
<td>0.52</td>
<td>0.46</td>
<td>0.98</td>
</tr>
<tr>
<td>P6 (4)</td>
<td>0.08</td>
<td>0.00</td>
<td>0.12</td>
<td>0.40</td>
<td>0.64</td>
<td>0.72</td>
</tr>
<tr>
<td>P8 (4)</td>
<td>0.18</td>
<td>0.16</td>
<td>0.42</td>
<td>0.36</td>
<td>0.32</td>
<td>0.84</td>
</tr>
<tr>
<td>P4 (5)</td>
<td>0.14</td>
<td>0.02</td>
<td>0.00</td>
<td>0.52</td>
<td>0.52</td>
<td>0.32</td>
</tr>
</tbody>
</table>

| mean      | 0.40| 0.34| 0.62 | 0.61| 0.62| 0.87 |
| rank      | 10.56| 12.33| 7.28 | 8.50| 9.39| 3.56 |
Observation 4

- Average success rate on the selected benchmarks was slightly higher for the small training set.

<table>
<thead>
<tr>
<th>training set</th>
<th>avg success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>0.581</td>
</tr>
<tr>
<td>large</td>
<td>0.573</td>
</tr>
</tbody>
</table>
### Statistical analysis

#### Ranks for the tested configurations (Friedman’s test):

<table>
<thead>
<tr>
<th>Group</th>
<th>Method</th>
<th>L\textsubscript{IS}</th>
<th>L\textsubscript{S}</th>
<th>T\textsubscript{IS}</th>
<th>L\textsubscript{P}</th>
<th>L\textsubscript{U}</th>
<th>T\textsubscript{S}</th>
<th>T\textsubscript{U}</th>
<th>T\textsubscript{P}</th>
</tr>
</thead>
<tbody>
<tr>
<td>small\textsubscript{N} (p = 0.00877)</td>
<td>Rank</td>
<td>2.50</td>
<td>3.06</td>
<td>4.28</td>
<td>4.28</td>
<td>4.56</td>
<td>5.50</td>
<td>5.67</td>
<td>6.17</td>
</tr>
<tr>
<td>small\textsubscript{C} (p = 0.01058)</td>
<td>Rank</td>
<td>2.17</td>
<td>3.61</td>
<td>4.33</td>
<td>4.33</td>
<td>4.72</td>
<td>5.22</td>
<td>5.72</td>
<td>5.89</td>
</tr>
<tr>
<td>large\textsubscript{N} (p = 0.00093)</td>
<td>Rank</td>
<td>2.06</td>
<td>3.61</td>
<td>3.72</td>
<td>4.44</td>
<td>4.83</td>
<td>5.44</td>
<td>5.50</td>
<td>6.39</td>
</tr>
<tr>
<td>large\textsubscript{C} (p = 0.00075)</td>
<td>Rank</td>
<td>2.22</td>
<td>3.50</td>
<td>3.83</td>
<td>4.33</td>
<td>4.56</td>
<td>5.11</td>
<td>5.83</td>
<td>6.61</td>
</tr>
</tbody>
</table>

**Legend:** small/large (training set used), N (no crossover), C (crossover), T (tournament), L (lexicase), U (unbiased), P (priors baseline), S (search), IS (initialization and search).
Neuro-Guided GP first trains the neural network on the family of search problem instances of interest, and then uses this network to guide search.

Neural network is able to generalize beyond the program size it was trained on.

Neuro-Guided GP fared better than standard GP and baselines on a small set of problems.
Thank you for your attention!