Program Boosting: Program Synthesis via Crowd-Sourcing

Robert A. Cochran, Loris D. Antoni, Benjamin Livshits, David Molnar, and Margus Veanes
How can we use crowd-sourcing to boost program accuracy where the program’s initial specification may be open to interpretation?
Key Insight

Regular Expressions

➔ Lack an easy-to-formalize specification
➔ Different regexes cover different cases
➔ Surprisingly difficult to implement addressing all the tricky corner cases
➔ Plenty of room for ambiguity

CrowdBoost

➔ Pose a tricky programming task as a crowdsourcing challenge
➔ Describe the task in question in a very loose form of specification
➔ Provide positive and negative examples ("the golden set"), giving a partial specification
➔ Blend imperfect solutions together to yield a solution of higher quality using a genetic programming approach
➔ Refine the result using a two-crowd approach
Approach

- Task
- Skilled Crowd (developers)
- Candidate Programs
- Blend Programs
- More Accurate Program
- Training Set
- Untrained Crowd (regular computer users)
Example

→ Post a programming task to Bountify: “Generate a regular expression to validate phone numbers”

→ Take the first 3 submissions
A regular expression for a phone number

\^[2-9] \d{2}-\d{3}-\d{4}$

- Character class indicates our first character must be any digit from 2 to 9
- Then there should be two more digits that can be any value from 0 - 9
- Followed by 3 digits
- Lastly, dash and 4 digits

This indicates, string can't contain other characters that don't belong to phone number.
Example

Genetic Programming Algorithm
Example

➔ Represent each regular expression as a Symbolic Finite Automaton

➔ Manipulate each SFA within a genetic programming algorithm
Example

→ Perform crossover operations on the candidates
Example

Perform mutation operations on the candidates

before mutation: A B C D E F G H
after mutation: A D E C B F G H
Example

➔ Filter the new examples by human evaluation via Mechanical Turk, and update our candidates.

➔ Run fitness tests on the candidates and select the best.

➔ Submit our resulting population back to the head of the main algorithm.

➔ Repeat the process until we find a perfect solution, or we run out of money.
Example

Task: Phone numbers

Fitness: 0.897959183673469
Evaluation: Experimental Analysis

**Specification:** Phone, Email, Date, or URL

**Golden Set Initial Examples**

94 + & 111 -

Candidate regexes obtained from developers from Bountify (14), Regexlib (4), and other sources (15)

**Candidate regexes obtained from developers from Bountify (14), Regexlib (4), and other sources (15)**

**465 Experimental Pairs**

Select Successful Candidate Programs

Measure fitness using golden and refined example sets

Output Program with best fitness

Classify new examples using Mechanical Turk as Valid or Invalid and add to evolving training set

Blend Candidate Programs: Crossover & Mutation

Updated set of considered programs and examples
Evaluation: Initial Data

Specifications provided to Bountify developers. The last two columns capture the number of positive and negative examples (a subset of the golden set) given to workers in the task specifications.

<table>
<thead>
<tr>
<th>Task</th>
<th>Specification</th>
<th>Examples +</th>
<th>Examples -</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone numbers</td>
<td><a href="https://bountify.co/5b">https://bountify.co/5b</a></td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Dates</td>
<td><a href="https://bountify.co/5v">https://bountify.co/5v</a></td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Emails</td>
<td><a href="https://bountify.co/6c">https://bountify.co/6c</a></td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>URLs</td>
<td><a href="https://bountify.co/5f">https://bountify.co/5f</a></td>
<td>14</td>
<td>9</td>
</tr>
</tbody>
</table>

The number of examples in the golden set and the number of candidate regexes in each case study.

<table>
<thead>
<tr>
<th></th>
<th>Golden set +</th>
<th>Golden set -</th>
<th>Candidate regexes</th>
<th>Candidate regex source:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone numbers</td>
<td>20</td>
<td>29</td>
<td>8</td>
<td>Bountify: 3</td>
</tr>
<tr>
<td>Dates</td>
<td>31</td>
<td>36</td>
<td>6</td>
<td>Bountify: 3</td>
</tr>
<tr>
<td>Emails</td>
<td>7</td>
<td>7</td>
<td>10</td>
<td>Bountify: 4</td>
</tr>
<tr>
<td>URLs</td>
<td>36</td>
<td>39</td>
<td>9</td>
<td>Bountify: 4</td>
</tr>
</tbody>
</table>

Distribution of initial accuracy (fitness) of candidate regular expressions by source. Overall, initial fitness values hover between .5 and .75, with none of the regexes being either “too good” or “too bad”.

Summarized size and complexity of the candidate regexes by length and by number of states in each resulting SFA.

<table>
<thead>
<tr>
<th>Task</th>
<th>Regex character length 25%</th>
<th>Regex character length 50%</th>
<th>Regex character length 75%</th>
<th>Regex character length Max</th>
<th>SFA state count 25%</th>
<th>SFA state count 50%</th>
<th>SFA state count 75%</th>
<th>SFA state count Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone numbers</td>
<td>44.75</td>
<td>54</td>
<td>67.75</td>
<td>96</td>
<td>14.75</td>
<td>27</td>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>Dates</td>
<td>154</td>
<td>288</td>
<td>352.25</td>
<td>434</td>
<td>19</td>
<td>39.5</td>
<td>72</td>
<td>78</td>
</tr>
<tr>
<td>Emails</td>
<td>33.5</td>
<td>68.5</td>
<td>86.75</td>
<td>357</td>
<td>7.25</td>
<td>8.5</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>URLs</td>
<td>70</td>
<td>115</td>
<td>240</td>
<td>973</td>
<td>12</td>
<td>25</td>
<td>30</td>
<td>80</td>
</tr>
</tbody>
</table>
Evaluation: Findings

The regular expressions for each of the tasks were tested for accuracy on positive and negative examples in two sets, the golden set and the evolved set.

Golden set can be manipulated by adding and removing examples to influence accuracy measurements.

Evolved set is more representative since it evolves naturally through refinement and crowd consensus.

High-level results obtained from the boosting process are consistent across all tasks showing an average boost of 16.25%.
Evaluation: Findings

In each task category, boosting results (mean) are shown via fitness values measured on either the golden set or the evolved set for three separate regexes: initial, “no crowd” and “crowd”.

Characterizing the boosting process in three dimensions: the number of generations, the number of generated strings, and the measured consensus for classification tasks.

Statistics for the crossover and mutation process across the tasks. The number of crossovers produced during boosting is in ten of thousands, but only a small percentage of them survive to the next generation. The number of mutations is smaller (single thousands), and their survival rate is somewhat higher. This can be explained by the fact that mutations are relatively local transformations and are not nearly as drastic as crossovers.
Evaluation: Findings

Running times:
Pair-wise boosting for each task averaged from about 4 minutes and 37 minutes per pair.

Overall cost:
Performing program boosting across all four tasks ranged between 41 cents and 3 dollars per pair.

Left: Running times for each task
Right: Costs for Mechanical Turk
In Summary

➔ A semi-automatic program synthesis technique using a set of initial crowd-sourced programs that finds the best result by crowd-sourcing a training set for a measure of fitness

➔ An implementation for program boosting algorithm involving a genetic programming technique with crossover and mutation algorithms

➔ CrowdBoost represents regular expressions using Symbolic Finite Automata (SFAs). This is most likely the first work to use genetic programming on automata over a complex alphabet, UTF-16 in this case

➔ An evaluation of this program boosting technique over four case studies, which yielded an average program boost of 16.25% over 465 pairs of regular expressions. The results also showed consistency across the tasks and sources of regular expressions, giving support to the generality of their approach
How can crowd-sourcing programs and examples go wrong and affect program boosting?
Discussion

How will this technique scale on pieces of code?
Discussion

How do you know when to stop crowd-sourcing?
Discussion

Is this approach worth the amount of time it takes to get the results?
Do we know that the final program is the most fit?
References

Program Boosting Powerpoint Presentation → google.com
Symbolic Automata → cs.wisc.edu
Regular Expressions → code.tutsplus.com
Genetic Programming → geneticprogramming.com, wikipedia.org
Pairwise Testing → tutorialspoint.com
Genetic Algorithms vs Genetic Programming → stackoverflow.com