General algorithmic problem solving competitions with crowd-sourced solution validation

A blockchain-based platform design

Efstathios Aliprantis
Athens University of Economics and Business

GEC 2021, Greece
Algorithm Competitions

I have a problem, I need an algorithm

I can create algorithms

A Traditional Approach

We judge submitted algorithms

Experts Committee

Proposed Blockchain-based Platform

I fully moderate the competition & determine the winner

Blockchain-based Platform
Example: Factorization

Verifier(instance, solution?, fail_reason?):

product = 1
for c in solution?:
    if not c is not prime:
        return False
    product = product * c

return product == instance

~ Made by Petitioner

Solver1(instance):

solution = empty_array()

for i in 2..instance:
    while instance mod i == 0:
        instance = instance div i
        solution.append(i)

return solution

~ Made by Creator1

Solver2(instance):

if instance mod 2 == 0:
    return [2, instance div 2]

return [instance]

~ Made by Creator2

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solver1</th>
<th>Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>[2, 17]</td>
<td>True</td>
</tr>
<tr>
<td>25</td>
<td>[5, 5]</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solver2</th>
<th>Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>[2, 17]</td>
<td>True</td>
</tr>
<tr>
<td>25</td>
<td>[25]</td>
<td>False</td>
</tr>
</tbody>
</table>

*Parameter fail_reason is used whenever necessary to speed up computations on Verifier*
Main Observation

\[ \text{Solver is not correct} \iff \exists (\text{instance, fail_reason}) \text{ s.t. Verifier}(\text{instance}, \text{Solver}(\text{instance}), \text{fail_reason}) = \text{False} \]

Basic Flow of a Competition

1) The **Petitioner** submits the **Verifier**

2) A **Creator** submits a **Solver** and **pays a deposit**

3) a. A Tester (human being) finds a “counter-instance”
   \[ \Rightarrow \text{Solver rejected, Tester receives the deposit} \]

   b. No “counter-instance” found **within a predefined time period**
   \[ \Rightarrow \text{Solver accepted, deposit refunded to Creator} \]
Submitting a Verifier

Step 1) Deploy on Ethereum

Source Code → Compile → EVM Bytecode → Deploy → Ethereum

Verifier Smart Contact

0xAddress

Step 2) Submit on the Platform

Source Code → Parse → Parsed Source Code → Submit →

Our Platform’s Smart Contact

0xAddress

EVM Bytecode

Parsed Source Code

Compile & Compare with @0xAddress

Restoring

Source Code

Similar process when submitting a Solver
Enforcing Solver Complexity Constraints
• The compiler outputs instructions that measure steps & space used by Solvers
• The Petitioner submits one more function to limit the allowed steps/space of Solvers: ComplexityConstraints: instance → (max_steps, max_space)

Limitations
• Verifiers & Solvers must be able to be implemented (no pure theoretic constructs)
• Verifiers & Solvers must run in (gas) attainable time/space
  ➢ Hint: Verifiers can utilize the fail_reason parameter to reduce complexity
• Solvers can only be judged per instance, not on average

What’s Next: Proofs of correctness
The Platform will be able to, also, validate proofs of correctness
• Definitely correct accepted algorithms
• Not relying on Testers
• No need to wait for a predefined time period

Computer-based proof validation will be automated thus error-free
Acknowledgments

I'd like to thank my advisor prof. Eugénie Foustoucos for her constant guidance

Some Related Work


• For a list of platforms for “Algorithm Programming Competitions” see https://cs.au.dk/~gerth/code (NOTE: Our Platform differs from listed ones as it aims to solve problems of interest for Petitioners while the listed ones pose problems already solved as a mean to challenge the contestants)

References (Ethereum)
