Solving Symbolic Regression Problems with Formal Constraints

Iwo Błądek  Krzysztof Krawiec

Institute of Computing Science
Poznań University of Technology, Poland

GECCO, Prague, July 15, 2019
1 Introduction

2 CDGP

3 CDGP for regression

4 Experiments
Motivation

Jane the Physicist wants to devise a formula for the force of gravity.

- Jane has a set of observations \((m_1, m_2, r) \mapsto F\).
- Other laws of physics tell her that:
  1. Swapping the masses does not change the force
  2. A force cannot be negative
  3. Increasing one of the masses increases the force
Example problems

Benchmark: gravity

\[ f(m_1, m_2, r) = 6.67408 \cdot 10^{-11} \cdot \frac{m_1 m_2}{r^2} \]

Constraints:

- **s:** \( f(m_1, m_2, r) = f(m_2, m_1, r) \)
- **b:** \( f(m_1, m_2, r) \geq 0 \)
- **m:** strict monotonicity w.r.t. both \( m_1 \) and \( m_2 \).

Test cases:

<table>
<thead>
<tr>
<th>m1</th>
<th>m2</th>
<th>r</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.88386</td>
<td>11.36099</td>
<td>15.74871</td>
<td>0.0000000000033273260096895905</td>
</tr>
<tr>
<td>11.1782</td>
<td>3.21214</td>
<td>7.67932</td>
<td>0.0000000000040635631498539907</td>
</tr>
<tr>
<td>7.22322</td>
<td>0.10104</td>
<td>11.48446</td>
<td>0.00000000000369308431952941</td>
</tr>
</tbody>
</table>
Example problems

Benchmark: \textit{resistance2}

\[
f(r_1, r_2) = \frac{r_1 r_2}{r_1 + r_2}
\]

Constraints:

\begin{itemize}
\item \textbf{s:} \quad f(r_1, r_2) = f(r_2, r_1)
\item \textbf{c}_1: \quad r_1 = r_2 \implies f(r_1, r_2) = \frac{r_1}{2}
\item \textbf{c}_2: \quad f(r_1, r_2) \leq r_1 \land f(r_1, r_2) \leq r_2
\end{itemize}

Test cases:

\begin{tabular}{ccc}
\textbf{r1} & \textbf{r2} & \textbf{out} \\
10.09611 & 17.39521 & 6.388341979690316 \\
0.68719 & 4.75438 & 0.600408042568597 \\
1.42871 & 17.19419 & 1.319102352206155
\end{tabular}
Example problems

Benchmark: **resistance3**

\[ f(r_1, r_2, r_3) = \frac{r_1 r_2 r_3}{r_1 r_2 + r_1 r_3 + r_2 r_3} \]

Constraints:

- **s:** \( f(r_1, r_2, r_3) = \ldots = f(r_3, r_2, r_1) \)
- **c_1:** \( r_1 = r_2 = r_3 \implies f(r_1, r_2, r_3) = \frac{r_1}{3} \)
- **c_2:** \( f(r_1, r_2, r_3) \leq r_1 \land f(r_1, r_2, r_3) \leq r_2 \land f(r_1, r_2, r_3) \leq r_3 \)

Test cases:

<table>
<thead>
<tr>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.31772</td>
<td>8.88437</td>
<td>2.90062</td>
<td>1.771060362583031</td>
</tr>
<tr>
<td>19.09801</td>
<td>17.08167</td>
<td>3.09636</td>
<td>2.3048717244360835</td>
</tr>
<tr>
<td>17.71372</td>
<td>11.53495</td>
<td>6.26835</td>
<td>3.3038401592739173</td>
</tr>
</tbody>
</table>
Problem

Symbolic Regression with Formal Constraints (SRFC)

Given a set of test cases (examples) $T$, a set of formal constraints $C$, and a (possibly infinite) set of functions $\mathcal{F}$, find a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $f \in \mathcal{F}$, that minimizes the approximation error on $T$, while satisfying all constraints in $C$.

Examples of formal constraints:

- Symmetry with respect to arguments: $f(x, y) = f(y, x)$
- Symmetry with respect to domain: $f(x) = f(-x)$
- Range: $f(x, y) \in [0, 1]$
- Monotonicity: $\forall x, y \ x > y \implies f(x) > f(y)$
- Convexity/concavity
- Value of derivative in a given point: $f'(3.7) > 2.5$
Advantages and challenges

Advantages:

- Resulting model is guaranteed to have the requested properties.
- Arbitrary constraints can be used.
- Models can be induced from fewer examples.

Challenges:

- **Feedback bottleneck**
  A constraint is either satisfied or not; not much information for guiding search.

- **Necessity of logical proof**
  A constraint may be specified over an infinite number of points.
Outline

1. Introduction
2. CDGP
3. CDGP for regression
4. Experiments
Counterexample-Driven Genetic Programming (CDGP) \(^1 \ 2 \ 3\)

Module: **Testing**

- Fitness is computed based on the set of test cases $T_c$.
- $T_c$ may initially be empty or seeded with test cases provided by the user.
- During verification phase, a new test may be added to $T_c$. 

![Diagram of Counterexample-Driven Genetic Programming (CDGP)]
Module: **Verification**

- Checks if a candidate solution satisfies the formal constraints.
- Involves a *Satisfiability Modulo Theories (SMT) solver*.

### Formal verification

Proving for a candidate solution $p$ that:

\[
\forall_{in} Pre(in) \implies Post(in, p(in)),
\]

where $p(in)$ is the output returned by $p$ for input $in$, $Pre(in)$ is a precondition, and $Post(in)$ is a postcondition. In practice, often the negated form is disproved:

\[
\exists_{in} Pre(in) \nRightarrow Post(in, p(in)).
\]
An example of SMT verification

Incorrect program (MAX):

```python
if x < y:
    res = x
else:
    res = y
```

Formal specification:

\[
\begin{align*}
max(x, y) & \geq x \land \\
max(x, y) & \geq y \land \\
(max(x, y) = x & \lor max(x, y) = y)
\end{align*}
\]

Solver result:

SAT

\[\begin{align*}
x &= -1 \\
y &= 0
\end{align*}\]

**SAT** means that the program is incorrect and solver provides us a counterexample (-1, 0).
An example of SMT verification

Correct program (MAX):

```python
if x > y:
    res = x
else:
    res = y
```

Formal specification:

\[
\begin{align*}
\text{max}(x, y) & \geq x \ \land \\
\text{max}(x, y) & \geq y \ \land \\
(\text{max}(x, y) = x \ \lor \ \text{max}(x, y) = y)
\end{align*}
\]

Solver result:

UNSAT

**UNSAT** means that the program is **correct** with respect to the specification. No counterexample was found.
Two types of tests may be created from counterexamples:

- **Complete tests**
  Conventional tests of the form \((input, desired \ output)\).

- **Incomplete tests**
  Tests without a specified desired output.
  Evaluated using the SMT solver.

Example 1: \(f(x) = \sqrt{x}\)

<table>
<thead>
<tr>
<th>(x)</th>
<th>(f(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2 or (-2)</td>
</tr>
<tr>
<td>9</td>
<td>3 or (-3)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\(\leftarrow\) There is no single desired output for easy testing
Two types of tests may be created from counterexamples:

- **Complete tests**
  Conventional tests of the form \((\text{input}, \text{desired output})\).

- **Incomplete tests**
  Tests without a specified desired output.
  Evaluated using the SMT solver.

Example 2: \(f(x) > 2x\)

<table>
<thead>
<tr>
<th>(x)</th>
<th>(f(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>9, 10, ...</td>
</tr>
<tr>
<td>9</td>
<td>19, 20, ...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

← There is no single desired output for easy testing
Counterexample-Driven Symbolic Regression (CDSR)

Changes introduced in CDGP to handle SR problems:

1. **Definition of "All passed?"**
   A program is sent to verification if it passes $\alpha$ percent of incomplete tests.

   **Why:** complete tests are ignored in order to avoid arbitrary thresholds on the difference between function’s output and the expected output of the test.
Counterexample-Driven Symbolic Regression (CDSR)

Changes introduced in CDGP to handle SR problems:

2 Definition of termination condition
Search process terminates when both (i) its aggregated error (MSE) on complete tests is below the automatically computed error threshold; (ii) it meets the formal constraints.

Why: solution needs to both have low error on tests and satisfy all the constraints. Absolute threshold leads to problems.

The error threshold for MSE:

\[ \epsilon = (t \cdot \sigma_Y)^2 \]

where \( t \) is called tolerance, and \( \sigma_Y \) is standard deviation of the output variable in the data.
Counterexample-Driven Symbolic Regression (CDSR)

CDSR with tests for properties ($CDSR_{props}$)

- Extends the initial set of tests $T_c$ with incomplete tests, one for each constraint.
- These additional tests are verified by SMT solver, and they are always evaluated either to 0 or 1.
- Uses $\epsilon$-Lexicase\(^4\) selection on both complete and incomplete tests.

<table>
<thead>
<tr>
<th>#</th>
<th>r1</th>
<th>r2</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.09611</td>
<td>17.39521</td>
<td>6.388341979690316</td>
</tr>
<tr>
<td>2</td>
<td>0.68719</td>
<td>4.75438</td>
<td>0.600408042568597</td>
</tr>
<tr>
<td>3</td>
<td>1.42871</td>
<td>17.19419</td>
<td>1.319102352206155</td>
</tr>
<tr>
<td>4</td>
<td>$f(r_1, r_2) = f(r_2, r_1)$</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$r_1 = r_2 \implies f(r_1, r_2) = \frac{r_1}{2}$</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$f(r_1, r_2) \leq r_1 \land f(r_1, r_2) \leq r_2$</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

There are only a handful of approaches which allow GP to synthesize provably-correct programs.

**Approaches to provably-correct synthesis in GP**

- Fitness based on satisfying subconstraints
- Generating counterexamples

(2007) C.G. Johnson, *Genetic Programming with Fitness Based on Model Checking*


(2011) P. He, L. Kang, C.G. Johnson, S. Ying, *Hoare logic-based genetic programming*


1 Introduction

2 CDGP

3 CDGP for regression

4 Experiments
Experiment dimensions

Dimension 1: **synthesis method**
- GP – baseline
- CDSR – our approach
- \( \text{CDSR}_{\text{props}} \) – our approach with added tests for individual constraints

Dimension 2: **number of test cases**
- 3 tests
- 5 tests
- 10 tests

Dimension 3: **tolerance**
- 0.01
- 0.1

**Total # tested configurations:** \( 3 \cdot 3 \cdot 2 = 18 \)
Benchmarks

- Gravity
- Resistance 2
- Resistance 3

With variants:
- b: bound
- m: monotonicity
- s: symmetry
- c1: custom constraint – equal arguments
- c2: custom constraint – output always smaller than any of the arguments

Total # benchmark variants: 8 + 4 + 4 = 16
To make benchmarks more realistic, noise was introduced to both the inputs and the output of the function.

Noise is calculated according to the formula:

\[ \tilde{X} = X + \mathcal{N}(0, \beta \cdot \sigma_X) \]

and in our experiments we assumed \( \beta = 0.1 \), and \( \sigma_X \) is a standard deviation of the variable \( X \) in the data.
## Evolution parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of runs</td>
<td>25</td>
</tr>
<tr>
<td>Population size</td>
<td>500</td>
</tr>
<tr>
<td>Maximum number of generations</td>
<td>∞</td>
</tr>
<tr>
<td>Maximum runtime in seconds</td>
<td>1800</td>
</tr>
<tr>
<td>Probability of mutation</td>
<td>0.5</td>
</tr>
<tr>
<td>Probability of crossover</td>
<td>0.5</td>
</tr>
<tr>
<td>Maximum height of initial programs</td>
<td>4</td>
</tr>
<tr>
<td>Maximum height of trees inserted by mutation</td>
<td>4</td>
</tr>
<tr>
<td>Maximum height of programs in population</td>
<td>12</td>
</tr>
<tr>
<td>Selection method</td>
<td>$\epsilon$-Lexicase$^5$</td>
</tr>
</tbody>
</table>

---


[https://github.com/kkrawiec/CDGP](https://github.com/kkrawiec/CDGP)
Results – success rates

### Properties and MSE

<table>
<thead>
<tr>
<th></th>
<th>GP</th>
<th>CDSR</th>
<th>$CDSR_{props}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr_b</td>
<td>0.07</td>
<td>0.03</td>
<td>0.04</td>
</tr>
<tr>
<td>gr_m</td>
<td>0.00</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>gr_s</td>
<td>0.10</td>
<td>0.13</td>
<td>0.15</td>
</tr>
<tr>
<td>gr_bm</td>
<td>0.00</td>
<td>0.01</td>
<td>0.04</td>
</tr>
<tr>
<td>gr Bs</td>
<td>0.11</td>
<td>0.05</td>
<td>0.11</td>
</tr>
<tr>
<td>gr_ms</td>
<td>0.00</td>
<td>0.04</td>
<td>0.07</td>
</tr>
<tr>
<td>gr_bms</td>
<td>0.00</td>
<td>0.01</td>
<td>0.15</td>
</tr>
<tr>
<td>res2_c1</td>
<td>0.01</td>
<td>0.43</td>
<td>0.45</td>
</tr>
<tr>
<td>res2_c2</td>
<td>0.04</td>
<td>0.25</td>
<td>0.36</td>
</tr>
<tr>
<td>res2_s</td>
<td>0.00</td>
<td>0.22</td>
<td>0.29</td>
</tr>
<tr>
<td>res2_sc</td>
<td>0.03</td>
<td>0.17</td>
<td>0.35</td>
</tr>
<tr>
<td>res3_c1</td>
<td>0.00</td>
<td>0.07</td>
<td>0.21</td>
</tr>
<tr>
<td>res3_c2</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>res3_s</td>
<td>0.00</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>res3_sc</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>mean</td>
<td>0.02</td>
<td>0.10</td>
<td>0.15</td>
</tr>
</tbody>
</table>

### Properties

<table>
<thead>
<tr>
<th></th>
<th>GP</th>
<th>CDSR</th>
<th>$CDSR_{props}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr_b</td>
<td>0.14</td>
<td>0.79</td>
<td>1.00</td>
</tr>
<tr>
<td>gr_m</td>
<td>0.00</td>
<td>0.02</td>
<td>0.91</td>
</tr>
<tr>
<td>gr_s</td>
<td>0.13</td>
<td>0.86</td>
<td>0.99</td>
</tr>
<tr>
<td>gr_bm</td>
<td>0.00</td>
<td>0.01</td>
<td>0.83</td>
</tr>
<tr>
<td>gr Bs</td>
<td>0.19</td>
<td>0.91</td>
<td>0.98</td>
</tr>
<tr>
<td>gr_ms</td>
<td>0.00</td>
<td>0.05</td>
<td>0.91</td>
</tr>
<tr>
<td>gr_bms</td>
<td>0.00</td>
<td>0.02</td>
<td>0.97</td>
</tr>
<tr>
<td>res2_c1</td>
<td>0.01</td>
<td>0.55</td>
<td>0.75</td>
</tr>
<tr>
<td>res2_c2</td>
<td>0.04</td>
<td>0.33</td>
<td>0.83</td>
</tr>
<tr>
<td>res2_s</td>
<td>0.00</td>
<td>0.60</td>
<td>0.99</td>
</tr>
<tr>
<td>res2_sc</td>
<td>0.03</td>
<td>0.22</td>
<td>0.83</td>
</tr>
<tr>
<td>res3_c1</td>
<td>0.00</td>
<td>0.07</td>
<td>0.26</td>
</tr>
<tr>
<td>res3_c2</td>
<td>0.01</td>
<td>0.02</td>
<td>0.11</td>
</tr>
<tr>
<td>res3_s</td>
<td>0.00</td>
<td>0.31</td>
<td>0.50</td>
</tr>
<tr>
<td>res3_sc</td>
<td>0.00</td>
<td>0.01</td>
<td>0.13</td>
</tr>
<tr>
<td>mean</td>
<td>0.04</td>
<td>0.32</td>
<td>0.73</td>
</tr>
</tbody>
</table>

- **b**: bound
- **m**: monotonicity
- **s**: symmetry
- **c1**: custom constraint – equal input arguments
- **c2**: custom constraint – output smaller than any of the inputs
- **c**: conjunction of c1 and c2
Statistical analysis

Friedman statistical test was used to analyze the findings.

MSE below threshold and properties met (Friedman’s test \( p = 7.1 \cdot 10^{-25} \)).

<table>
<thead>
<tr>
<th>method</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDSR(_{props}) _0.1</td>
<td>1.6</td>
</tr>
<tr>
<td>CDSR _0.1</td>
<td>2.5</td>
</tr>
<tr>
<td>GP _0.1</td>
<td>4.0</td>
</tr>
<tr>
<td>CDSR(_{props}) _0.01</td>
<td>4.2</td>
</tr>
<tr>
<td>CDSR _0.01</td>
<td>4.3</td>
</tr>
<tr>
<td>GP _0.01</td>
<td>4.4</td>
</tr>
</tbody>
</table>

Fraction of models that meet the formal properties (\( p = 1.6 \cdot 10^{-40} \)).

<table>
<thead>
<tr>
<th>method</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDSR(_{props}) _0.1</td>
<td>1.5</td>
</tr>
<tr>
<td>CDSR(_{props}) _0.01</td>
<td>1.6</td>
</tr>
<tr>
<td>CDSR _0.1</td>
<td>3.3</td>
</tr>
<tr>
<td>CDSR _0.01</td>
<td>4.1</td>
</tr>
<tr>
<td>GP _0.1</td>
<td>5.1</td>
</tr>
<tr>
<td>GP _0.01</td>
<td>5.4</td>
</tr>
</tbody>
</table>
Observations

Pros:
- CDSR systematically outperforms GP.
- Much lower risk of overfitting.
- Valid models induced from as few as three examples.

Cons:
- Significant computational overhead.
- Limitations of the theorem prover – SMT solvers do not handle well transcendental functions like \( \cos \), \( \log \), etc.
Main points:

- CDSR, a method for solving the task of symbolic regression with formal constraints by finding counterexamples.

- Solutions produced by CDSR generalize well in terms of expected properties.

- $CDSR_{props}$ was best at that, thanks to treating individual constraints as incomplete tests.

Thank you for your attention!

Questions.