Solving Symbolic Regression Problems with Formal Constraints

Iwo Błądek   Krzysztof Krawiec

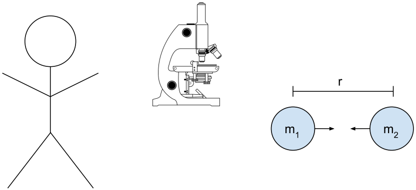
Institute of Computing Science
Poznań University of Technology, Poland

GECCO, Prague, July 15, 2019

Outline

Motivation

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

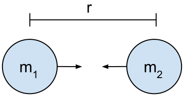


- Jane has a set of observations $(m_1, m_2, r) \mapsto F$.
- Other laws of physics tell her that:
  1. Swapping the masses does not change the force
  2. A force cannot be negative
  3. Increasing one of the masses increases the force

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.

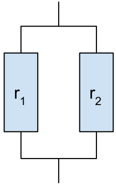
Test cases:

m1	m2	r	out
10.88386	11.36099	15.74871	0.000000000033273260096895905
11.1782	3.21214	7.67932	0.000000000040635631498539907
7.22322	0.10104	11.48446	0.000000000000369308431952941

Example problems

Benchmark: resistance2

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



Constraints:

- s**:** $f(r_1, r_2) = f(r_2, r_1)$
- c_1**:** $r_1 = r_2 \implies f(r_1, r_2) = \frac{r_1}{2}$
- c_2**:** $f(r_1, r_2) \leq r_1 \quad \wedge \quad f(r_1, r_2) \leq r_2$

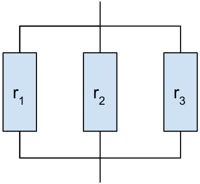
Test cases:

r1	r2	out
10.09611	17.39521	6.388341979690316
0.68719	4.75438	0.600408042568597
1.42871	17.19419	1.319102352206155

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: $f(r_1, r_2, r_3) \leq r_1 \quad \wedge \quad f(r_1, r_2, r_3) \leq r_2 \quad \wedge \quad f(r_1, r_2, r_3) \leq r_3$

Test cases:

r1	r2	r3	out
9.31772	8.88437	2.90062	1.77106036258303l
19.09801	17.08167	3.09636	2.3048717244360835
17.71372	11.53495	6.26835	3.3038401592739173

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 $f : \mathbb{R}^n \to \mathbb{R}$, $f \in \mathcal{F}$, 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:

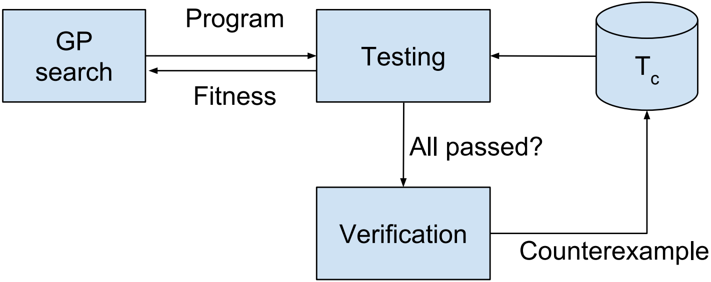
- Resulting model is guaranteed to have the requested properties.
- Arbitrary constraints can be used.
- Models can be induced from fewer examples.

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

Counterexample-Driven Genetic Programming (CDGP)[1] [2] [3]



[1] I.Błądek, K.Krawiec, J.Swan, J.Drake, "Counterexample-Driven Genetic Programming: Stochastic Synthesis of Provably Correct Programs", IJCAI, 2018.

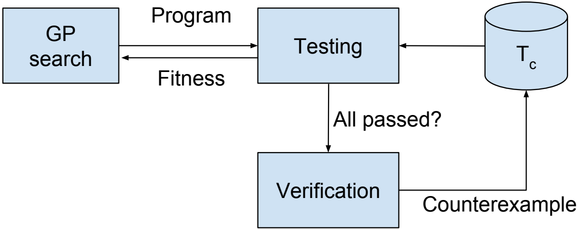
[2] I.Błądek, K.Krawiec, J.Swan, "Counterexample-Driven Genetic Programming: Heuristic Program Synthesis from Formal Specifications", ECJ, 2017.

[3] K.Krawiec, I.Błądek, J.Swan, "Counterexample-Driven Genetic Programming", GECCO, 2017.

Counterexample-Driven Genetic Programming (CDGP)

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.



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:

$$\forall_{in} \ Pre(in) \implies Post(in, p(in)), \tag{1}$$

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:

$$\exists_{in} \ Pre(in) \not\Longrightarrow Post(in, p(in)). \tag{2}$$

An example of SMT verification

Incorrect program (MAX):

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

Formal specification:

$$\begin{gathered} max(x,y) \geq x \quad \wedge \\ max(x,y) \geq y \quad \wedge \\ (max(x,y) = x \ \vee \ max(x,y) = y) \end{gathered}$$

Solver result:

```
SAT
x = -1
y = 0
```

SAT means that the program is incorrect and solver provides us a counterexample (-1, 0).

An example of SMT verification

Correct program (MAX):

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

Formal specification:

$$\begin{gathered} max(x,y) \geq x \quad \wedge \\ max(x,y) \geq y \quad \wedge \\ (max(x,y) = x \;\vee\; max(x,y) = y) \end{gathered}$$

Solver result:

```
UNSAT
```

UNSAT means that the program is correct with respect to the specification. No counterexample was found.

Complete and incomplete tests

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: $f(x) = \sqrt{x}$

x	$f(x)$
4	2 or -2
9	3 or -3
...	...

$\leftarrow$ There is no single desired output for easy testing

Complete and incomplete tests

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 2: $f(x) > 2x$

x	$f(x)$
4	9, 10, ...
9	19, 20, ...
...	...

$\leftarrow$ There is no single desired output for easy testing

Outline

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 α 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 σ_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 ϵ-Lexicase[4] selection on both complete and incomplete tests.

#	r1	r2	out
1	10.09611	17.39521	6.388341979690316
2	0.68719	4.75438	0.600408042568597
3	1.42871	17.19419	1.319102352206155
4	$f(r_1, r_2) = f(r_2, r_1)$		-
5	$r_1 = r_2 \implies f(r_1, r_2) = \frac{r_1}{2}$		-
6	$f(r_1, r_2) \leq r_1 \quad \wedge \quad f(r_1, r_2) \leq r_2$		-

[4] W.L. Cava, L. Spector, K. Danai, "Epsilon-lexicase Selection for Regression", GECCO, 2016.

State of the art

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

- (2007) C.G. Johnson, *Genetic Programming with Fitness Based on Model Checking*
- (2008) G. Katz, D. Peled, *Genetic Programming and Model Checking: Synthesizing New Mutual Exclusion Algorithms*
- (2011) P. He, L. Kang, C.G. Johnson, S. Ying, *Hoare logic-based genetic programming*
- (2016) G. Katz, D. Peled, *Synthesizing, correcting and improving code, using model checking-based genetic programming*

Generating counterexamples

- (2017) I.Błądek, K.Krawiec, J.Swan, *Counterexample-Driven Genetic Programming: Heuristic Program Synthesis from Formal Specifications*

Outline

1. Introduction

2. CDGP

3. CDGP for regression

4. Experiments

Experiment dimensions

Dimension 1: **synthesis method**

- **GP** – baseline
- **CDSR** – our approach
- **CDSR**$_{props}$ – 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: $8 + 4 + 4 = 16$

Noise

- 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:

$$\widetilde{X} = X + \mathcal{N}(0, \beta \cdot \sigma_X)$$

and in our experiments we assumed $\beta = 0.1$, and σ_X is a standard deviation of the variable X in the data.

Evolution parameters

Parameter	Value
Number of runs	25
Population size	500
Maximum number of generations	∞
Maximum runtime in seconds	1800
Probability of mutation	0.5
Probability of crossover	0.5
Maximum height of initial programs	4
Maximum height of trees inserted by mutation	4
Maximum height of programs in population	12
Selection method	ϵ-Lexicase[5]

https://github.com/kkrawiec/CDGP

[5] W.L. Cava, L. Spector, K. Danai, "Epsilon-lexicase Selection for Regression", GECCO, 2016.

Results – success rates

Properties and MSE

	GP	CDSR	$CDSR_{props}$
gr_b	0.07	0.03	0.04
gr_m	0.00	0.01	0.01
gr_s	0.10	0.13	0.15
gr_bm	0.00	0.01	0.04
gr_bs	0.11	0.05	0.11
gr_ms	0.00	0.04	0.07
gr_bms	0.00	0.01	0.15
res2_c1	0.01	0.43	0.45
res2_c2	0.04	0.25	0.36
res2_s	0.00	0.22	0.29
res2_sc	0.03	0.17	0.35
res3_c1	0.00	0.07	0.21
res3_c2	0.01	0.02	0.01
res3_s	0.00	0.03	0.02
res3_sc	0.00	0.00	0.01
mean	0.02	0.10	0.15

Properties

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

- 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 $p = 7.1 \cdot 10^{-25}$).

method	rank
CDSR_{props} _0.1	1.6
CDSR _0.1	2.5
GP_0.1	4.0
CDSR_{props} _0.01	4.2
CDSR _0.01	4.3
GP_0.01	4.4

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

method	rank
CDSR_{props} _0.1	1.5
CDSR_{props} _0.01	1.6
CDSR _0.1	3.3
CDSR _0.01	4.1
GP_0.1	5.1
GP_0.01	5.4

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.

Summary

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.