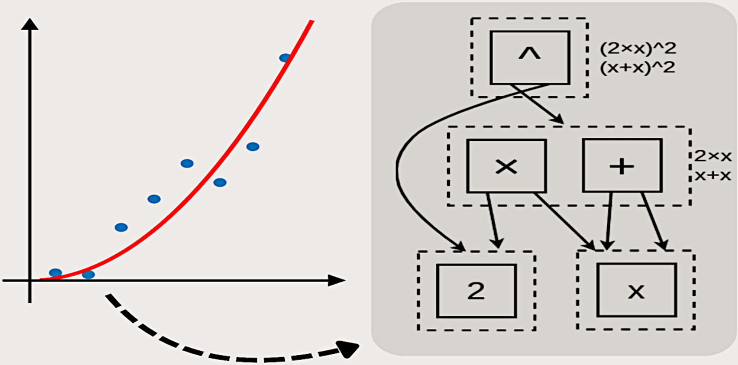
rEGGression - an Interactive and Agnostic Tool for the Exploration of Symbolic Regression Models



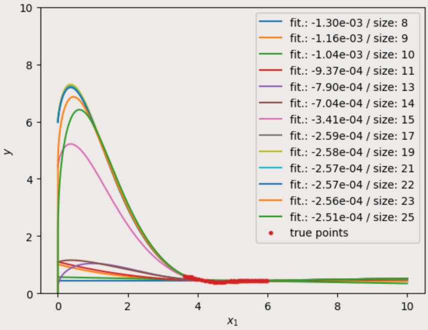
Fabrício Olivetti de França, Gabriel Kronberger

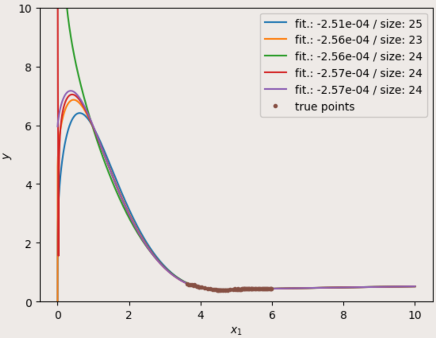
Federal University of ABC
University of Applied Sciences Upper Austria

Symbolic Regression: Going Beyond the Pareto Front

What do we really want?

- Accuracy-size tradeoff: simplest model with a good accuracy.
- The limiting behavior of the function is also important.
- Sometimes the Pareto front hides good alternatives.



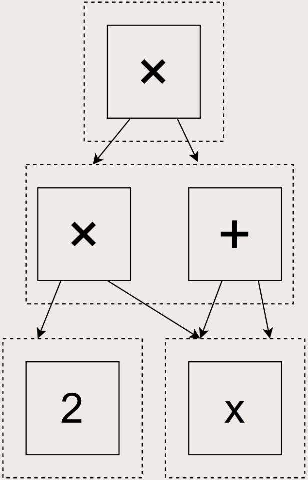


rEGGression: a database system for SR models

Features

- Database storing multiple models using **e-graphs**.
- Pattern matching ability.
- Building blocks explorations.
- Modularity detection.
- Import expressions from:
  - Operon
  - PySR
  - Bingo
  - GOMEA
  - etc.
- Pandas DataFrame.

e-graph



Full-view

	Id	Expression	Numpy	Latex	Fitness	Parameters	Size	DL
0	26974	(Abs((x0 / x1)) ^ (t0 * (Abs(x1) ^ t1)))	np.abs((x[:, 0] / x[:, 1])) ** (t[0] * np.abs(x[:, 1]) ** t[1])	$\left\lvert\frac{r_k}{log_{Re}}\right\rvert^{\left(\theta_0 \cdot \lvert log_{Re}\rvert^{\theta_1}\right)}$	-1.12e-03	[-0.11, 0.5]	9	1.61e+01
1	173811	(Abs((x0 / x1)) ^ (t0 * (Abs(x1) ^ t1)))	np.abs((x[:, 0] / x[:, 1])) ** (t[0] * np.abs(x[:, 1]) ** t[1])	$\left\lvert\frac{r_k}{log_{Re}}\right\rvert^{\left(\theta_0 \cdot \lvert log_{Re}\rvert^{\theta_1}\right)}$	-1.12e-03	[-0.12, 0.47]	9	1.61e+01
2	156288	(Abs((x0 / x1)) ^ (t0 + (t1 * x1)))	np.abs((x[:, 0] / x[:, 1])) ** (t[0] + (t[1] * x[:, 1]))	$\left\lvert\frac{r_k}{log_{Re}}\right\rvert^{(\theta_0 + (\theta_1 \cdot log_{Re}))}$	-1.14e-03	[-0.12, -0.03]	9	1.75e+01
3	69238	(Abs((x0 / x1)) ^ (t0 * (Abs(t1) ^ x1)))	np.abs((x[:, 0] / x[:, 1])) ** (t[0] * np.abs(t[1]) ** x[:, 1])	$\left\lvert\frac{r_k}{log_{Re}}\right\rvert^{\left(\theta_0 \cdot \lvert\theta_1\rvert^{log_{Re}}\right)}$	-1.15e-03	[-0.15, 1.1]	9	1.61e+01

Compact-view

	Id	Latex	Fitness
0	181293	$\frac{1}{\left\|\left\|\frac{\theta_0}{log_{Re}}\right\|^{\theta_1}\right\|^{\left\|\frac{1}{r_k}\right\|^{\left\|\frac{\|log_{Re}\|^{\theta_2}}{\frac{\frac{1}{r_k}}{log_{Re}}}\right\|^{\frac{\theta_3}{(\theta_4+log_{Re})}}}}}$	-5.06e-04
1	191394	$\frac{1}{\left\|\left\|\frac{\theta_0}{log_{Re}}\right\|^{\theta_1}\right\|^{\left\|\frac{1}{r_k}\right\|^{\left\|\frac{\|log_{Re}\|^{\theta_2}}{((r_k \cdot r_k)-(\theta_3+log_{Re}))}\right\|^{\frac{\theta_4}{log_{Re}}}}}}$	-5.32e-04

Pattern Matching

Top Expressions

Give me the very best!

- The *top* command returns the best N expressions filtering by size, and complexity.
- Ability to pattern match:
  - $x_0^{v_0} + \theta_0 v_0 + v_1$
  - x to the power of something plus θ times this **same** something plus **something else**.

Pattern Matching

```
(egraph.top(3,
    filters=["size <= 7"],
    pattern="v0 + x0")
 .style.format(fmt))
```

	Id	Latex	Fitness
0	7202	$\left(\theta_0 \cdot \lvert(x_0 + x_0)\rvert^{\theta_1}\right)$	-0.001309
1	12425	$\lvert(\theta_0 \cdot (x_0 + x_0))\rvert^{\theta_1}$	-0.001309
2	198550	$\left\lvert\frac{x_1}{(x_0 + x_0)}\right\rvert^{\theta_0}$	-0.003112

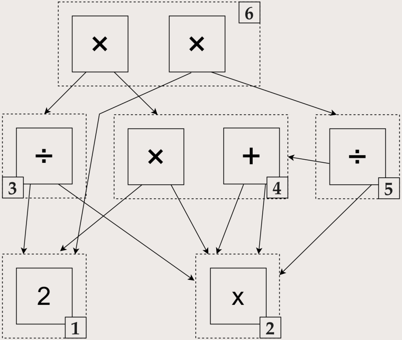
- Expressions **not** matching a certain pattern.
- Or that matches a pattern at the root.

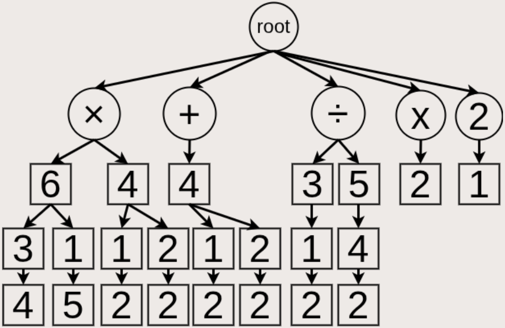
Let's Match[1] [2]

Modern libraries for equality saturation have an e-matching algorithm to pattern match **building blocks**.
Quickly find all the matches of a certain pattern inside an e-graph.

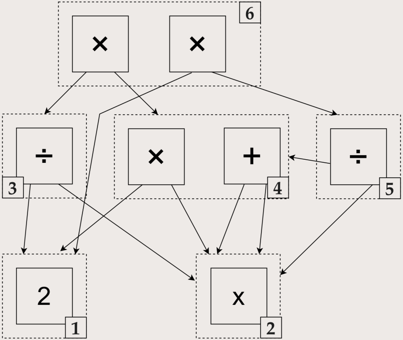
[1] Willsey, Max, et al. "Egg: Fast and extensible equality saturation." Proceedings of the ACM on Programming Languages 5.POPL (2021): 1-29.
[2] Zhang, Yihong, et al. "Relational E-matching." arXiv preprint arXiv:2108.02290 (2021).

Let's Match





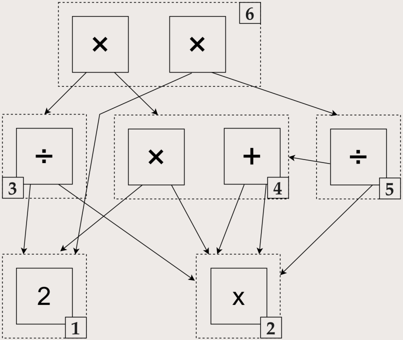
Matching Expressions



$\alpha \times (\beta + \gamma)$

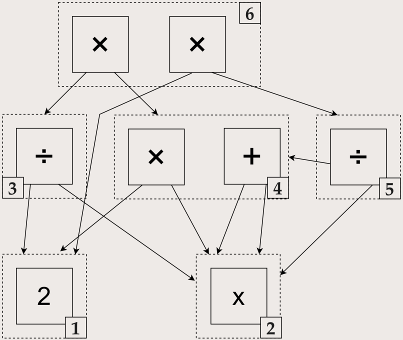
α, β, γ are *match all* variables.

Matching Expressions



$$\{\times \to ?, \alpha \to ?, + \to ?, \beta \to ?, \gamma \to ?\}$$

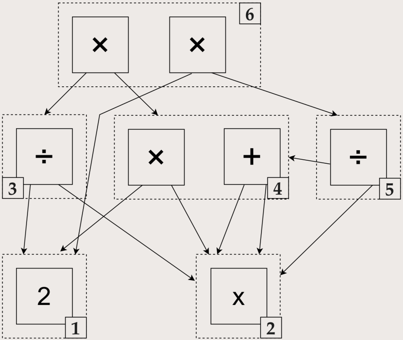
Matching Expressions



$$\{\times \to 4, \alpha \to ?, + \to ?, \beta \to ?, \gamma \to ?\}$$

$$\{\times \to 6, \alpha \to ?, + \to ?, \beta \to ?, \gamma \to ?\}$$

Matching Expressions

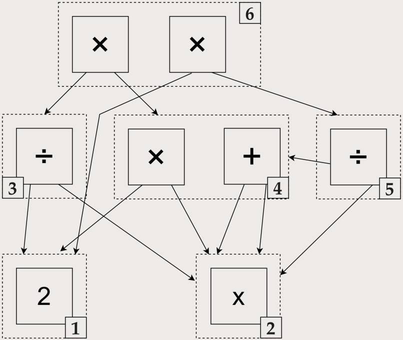


$$\{\times \to 4, \alpha \to 1, + \to ?, \beta \to ?, \gamma \to ?\}$$

$$\{\times \to 6, \alpha \to 3, + \to ?, \beta \to ?, \gamma \to ?\}$$

$$\{\times \to 6, \alpha \to 1, + \to ?, \beta \to ?, \gamma \to ?\}$$

Matching Expressions

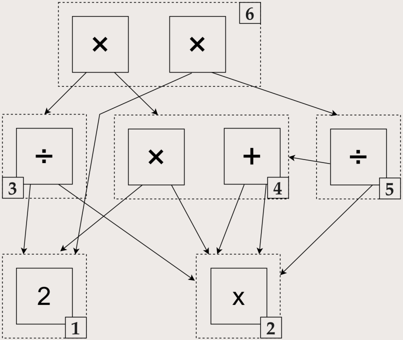


$$\{\times \to 4, \alpha \to 1, + \to 2, \beta \to ?, \gamma \to ?\}$$

$$\{\times \to 6, \alpha \to 3, + \to 4, \beta \to ?, \gamma \to ?\}$$

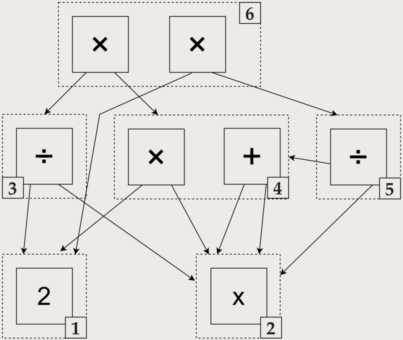
$$\{\times \to 6, \alpha \to 1, + \to 5, \beta \to ?, \gamma \to ?\}$$

Matching Expressions



$$\{\times \to 6, \alpha \to 3, + \to 4, \beta \to ?, \gamma \to ?\}$$

Matching Expressions



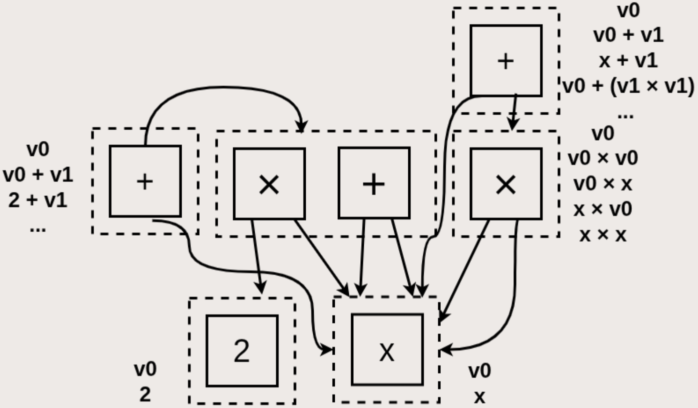
$$\{\times \to 6, \alpha \to 3, + \to 4, \beta \to 2, \gamma \to 2\}$$

Building Blocks

Exploring the Building Blocks

- Generate building blocks by traversing the e-graph and counting the frequency.
- We only need to traverse each node once and count all shared expressions.
- The expression $x + x$ will count x once.

Exploring the Building Blocks



Exploring the Building Blocks

```
egraph.distribution(filters=["size < 6"],
                    limitedAt=100, dsc=True,
                    byFitness=True, atLeast=100,
                    fromTop=1000)
```

Pattern	Count	Avg. Fit.
$\theta_0(v_0 + (\frac{v_1}{v_2})^{\theta_3})$	138	$-4.45 \cdot 10^{-4}$
$(\theta_0(\frac{v_0}{v_1} + v_2))^{v_3}$	610	$-5.17 \cdot 10^{-4}$
$(v_0(v_1 + v_2 + v_3))^{v_4}$	278	$-5.83 \cdot 10^{-4}$

More Building Blocks

We can also extract all building blocks of a single expression:

```
x_0 ^ (t_0 + x_1)
egraph.extractPattern(e_class_id)
```

Pattern
$x_0^{t_0+x_1}$
$v_0^{t_0+x_1}$
$x_0^{v_0}$
$v_0^{t_0+v_1}$
...

More Building Blocks

And this also allows us to count the occurrences of each token:

```
egraph.distributionOfTokens(500)
```

Token	Count	Avg. Fit.
x_0	1273	$-1.36 \cdot 10^{-4}$
$+$	1929	$-9 \cdot 10^{-5}$
$\times$	867	$-1.99 \cdot 10^{-4}$
$\frac{1}{x}$	205	$-3.2 \cdot 10^{-4}$
...		

In-depth Examination

Subtrees, optimize the unevaluated and insert new.

```
egraph.subtrees(100)
```

Expression	Fitness
x_0	-20.23
θ_0	-14.12
$\theta_0 x_0$	-6.53
$x_0^{\theta_0 x_0}$	NaN
$x_0^{\theta_0 x_0} + \theta_1$	NaN
$\theta_1 x_1$	NaN
$x_0^{\theta_0 x_0} + \theta_1 x_1$	$-1.32 \cdot 10^{-3}$

```
egraph.optimize(93)
```

Expression	Fitness
$\theta_1 x_1$	-5.43

```
egraph.insert("x0 ^
    (t0 + x1)")
```

Expression	Fitness
$x_0^{\theta_0 + x_1}$	-0.43

Combining Results from Different Algorithms

Union of All Greatest SR Algorithms

We can run different SR algorithms and import their results into this e-graph:

```
egraph.importFromCSV("equations.operon")
egraph.importFromCSV("equations.pysr")
egraph.importFromCSV("equations.bingo")
```

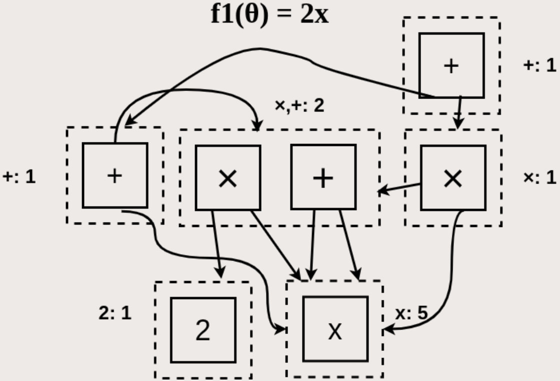
And then we can resume the search, starting with this set using ‘eggp‘ (check our other talk):

```
reg = EGGP(maxSize=25, gen=200, nPop=300,
          loadFrom="seed.egraph", dumpTo="new.egraph")
```

Modularity

Modularity

We can also detect *modular* expressions by traversing the e-graph and counting how many times each e-class id appears in one expression.



Modularity

```
egfinal.modularity(2, filters=[”> 3”])
```

$$\left(\left(\left|z_0\right|^{\theta_0} + \left|z_0\right|^{\theta_1}\right) \cdot \theta_2\right)$$

$$z_0 = (log_{Re} - r_k)$$

$$((f_0(\theta_{0...2}) \cdot (f_0(\theta_{3...5}) \cdot r_k)) + \theta_6)$$

$$|z_0|^{\left(\theta_0 \cdot \left|\frac{1}{z_0}\right|^{\theta_1}\right)}$$

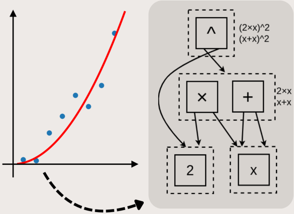
$$f_0(\theta) = \left(\left(\frac{((\theta_0 + r_k) + r_k)}{r_k} \cdot \theta_1\right) + \theta_2\right)$$

$$z_0 = \frac{log_{Re}}{r_k}$$

And more…

- Load and refit all expressions
- Refit everything with another loss function
  - MSE
  - Gaussian
  - Poisson
  - Bernoulli
  - ROXY
- Validation error.
- Simplify using equality saturation.

Final Remarks



symbolic regression + e-graphs = ❤️

🥚

- e-graph brings the essence of relational databases into symbolic regression.
- rEGGression can help us navigate the set of visited expressions during a search .
- many new features on the way.

Questions

Python library and CLI

- pip install eggp
- pip install reggression
- pip install symregg

Open-source:

- **https://github.com/folivetti/eggp**
- **https://github.com/folivetti/reggression**
- **https://github.com/folivetti/symregg**