Genetic and Evolutionary Computation Conference
(GECCO '24)
July 14--18, 2024, Melbourne, Australia

Inexact Simplification of Symbolic Regression Expressions with Locality-sensitive Hashing

Guilherme Seidyo Imai Aldeia
Federal University of ABC
Santo Andre, São Paulo, Brazil
guilherme.aldeia@ufabc.edu.br

Fabrício Olivetti de França
Federal University of ABC
Santo Andre, São Paulo, Brazil
folivetti@ufabc.edu.br

William G. La Cava
Boston Children's Hospital
Harvard Medical School
Boston, Massachusetts, USA
william.lacava@childrens.harvard.edu

UFABC

Boston Children's Hospital
Until every child is well™

AVA Lab

VE RI TAS

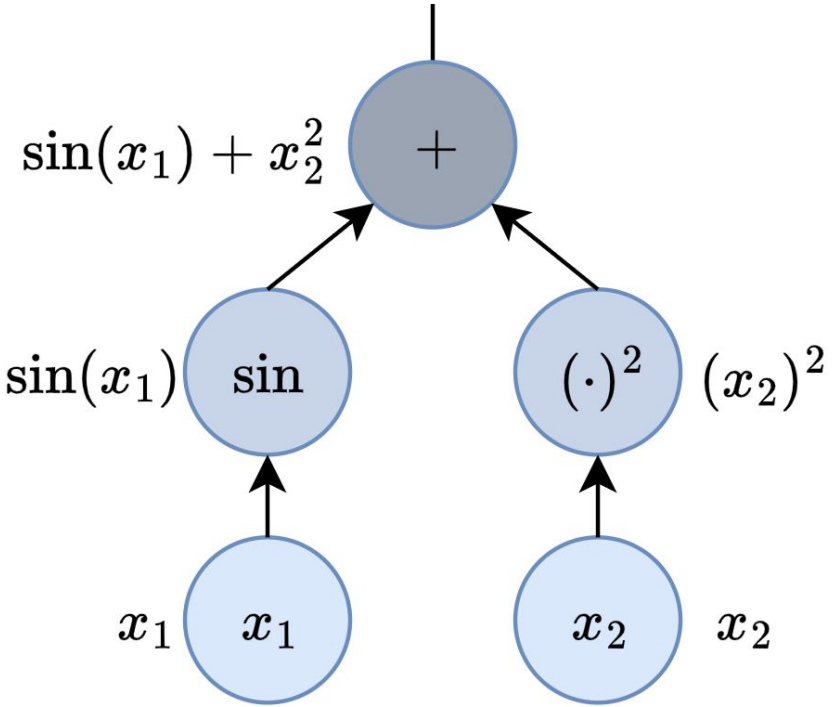
chip Computational Health Informatics Program

CMCC
Centro de Matemática, Computação e Cognição

Symbolic Regression expressions

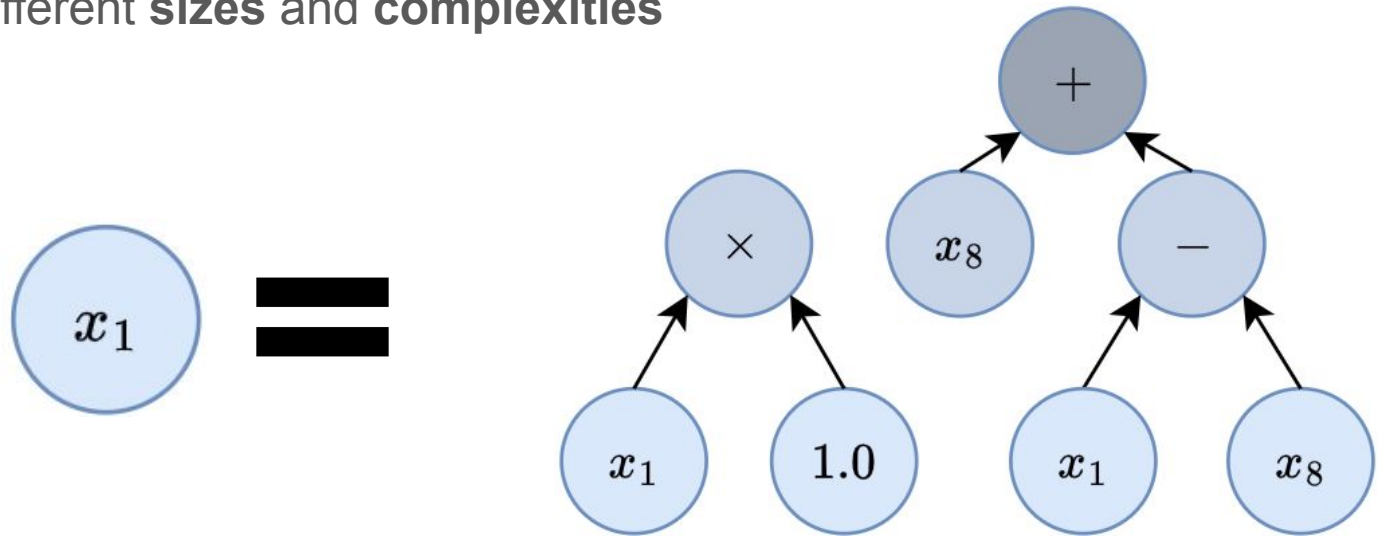
Symbolic regression (SR) is the task of finding a mathematical expression that best fits a given dataset.

We use it when we want interpretable results.



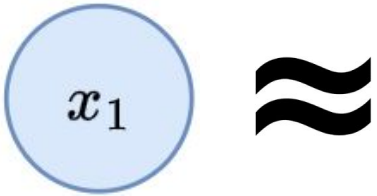
Symbolic Regression expressions

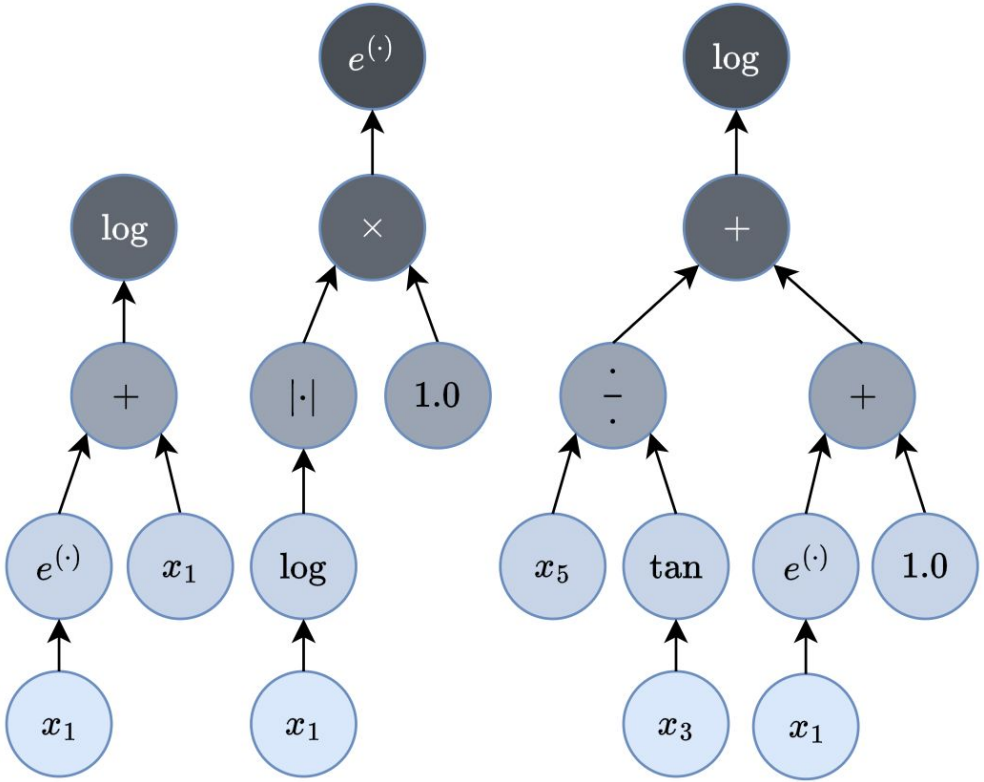
The search space is huge and can contain several equivalent expressions with different **sizes** and **complexities**



Symbolic Regression expressions

There is also expressions with very similar (but not equal) predictions.





Why is it important

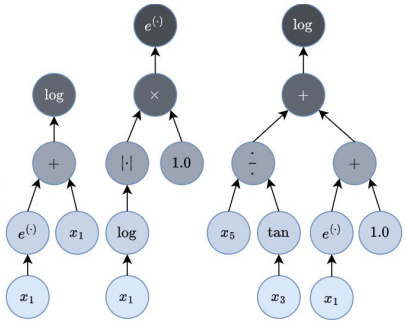
- Removal of **Introns** (parts of the model that does not influence prediction);
- Avoid **bloat** (growth in size with unjustified improvement on error);
- Avoid **overparameterization** (too many parameters to tune).

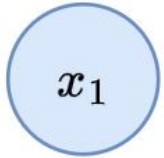
Simpler representation for expressions, improving **interpretability**.

Why is it important

- Removal of **Introns** (parts of the model that does not influence prediction);
- Avoid **bloat** (growth in size with unjustified improvement on error);
- Avoid **overparameterization** (too many parameters to tune).

Simpler representation for expressions, improving **interpretability**.





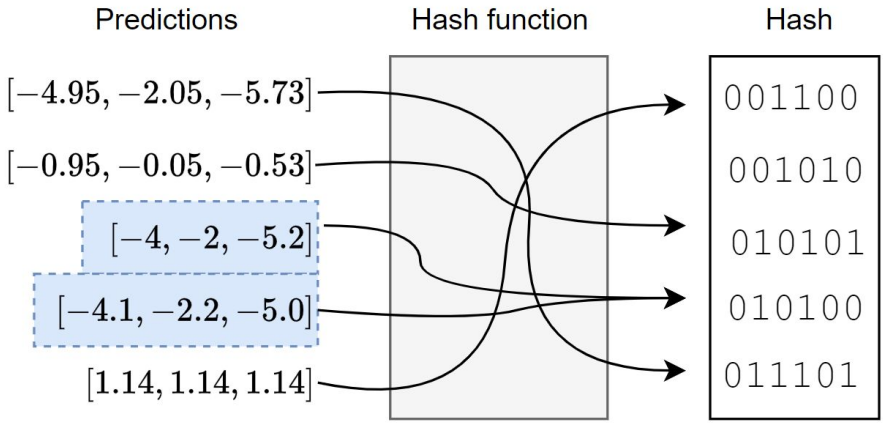
The problem of expression simplification

- It is hard to define a set of algebraic rules for simplification;
- Some simplifications are based on data;
- Depending on the strategy, it is implementation-dependent.

I am lazy and a big fan of ideas that are simple to implement. If only there was some **efficient way** of finding **similarity** between pairs of structures…!

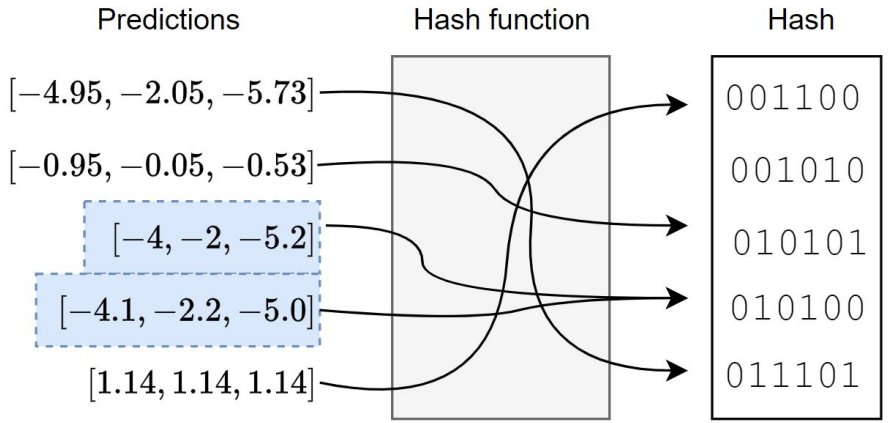
Inexact simplification

Locality-Sensitive Hashing (LSH): find **approximate** neighbors in a high-dimensional space. The hash allows fast store and recovery of similar structures.



Inexact simplification

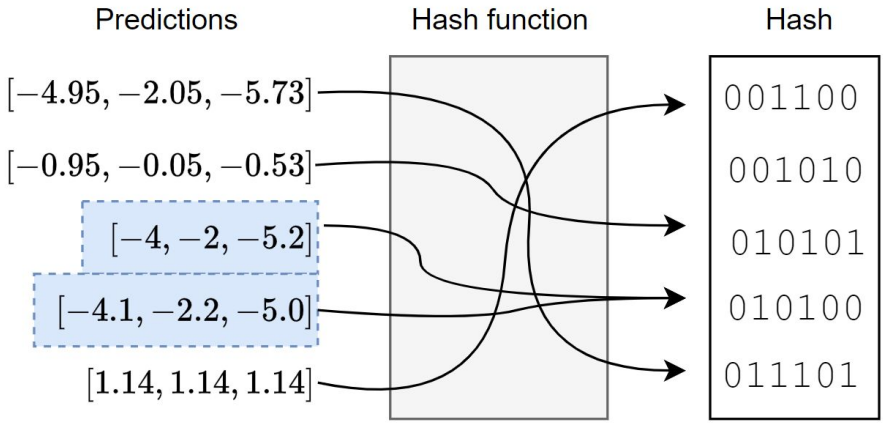
Locality-Sensitive Hashing (LSH): find **approximate** neighbors in a high-dimensional space. The hash allows fast store and recovery of similar structures.



This naturally allow us to use a relaxed definition of equivalent expressions.

Inexact simplification

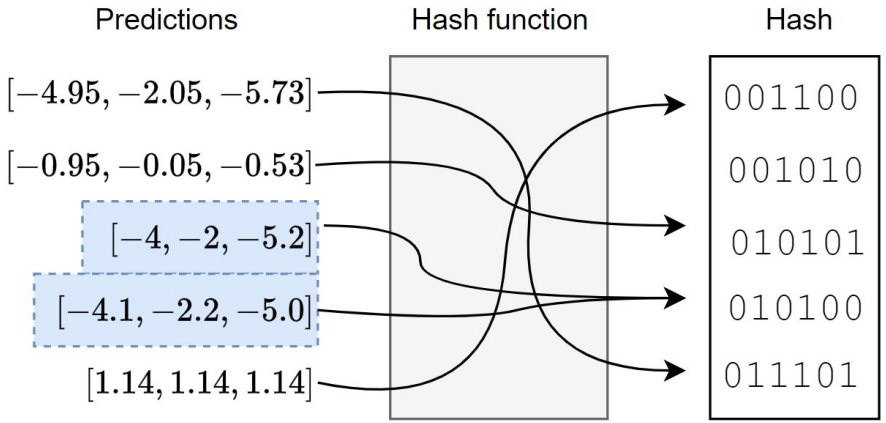
Locality-Sensitive Hashing (LSH): find **approximate** neighbors in a high-dimensional space. The hash allows fast store and recovery of similar structures.



This naturally allow us to use a relaxed definition of equivalent expressions.

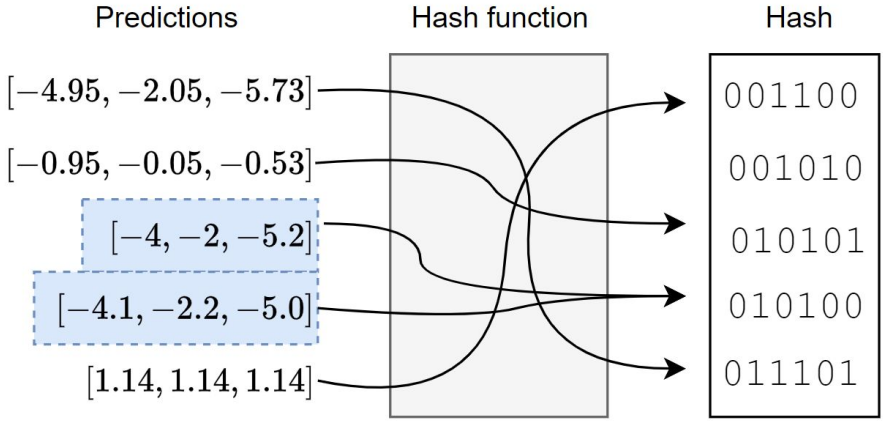
Inexact simplification

SR is usually performed with populational heuristics.



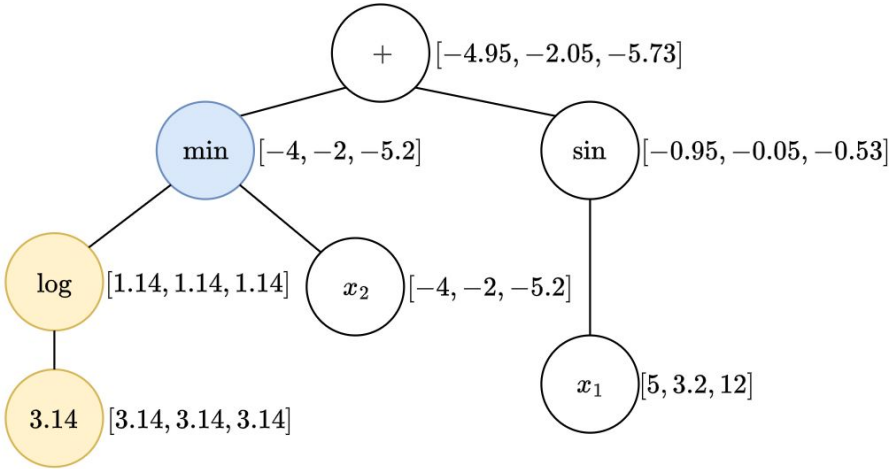
Inexact simplification

SR is usually performed with populational heuristics.



Our **hypothesis**: we can take advantage of the populational aspect of SR to build a collection of models, and we can map them to keys that are fast to calculate.

Our proposal: inexact simplification with LSH

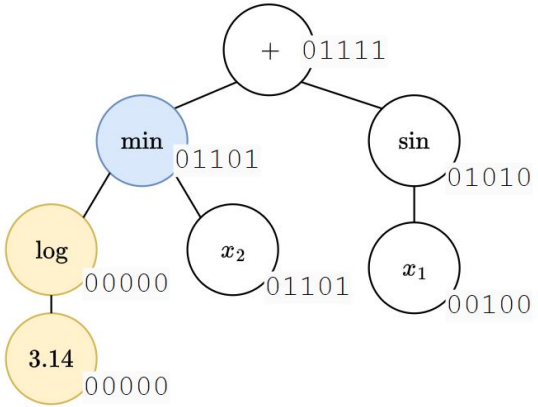


Simplification table	
00000	[cte]
00100	$[x_1]$
01101	$[x_2]$

For each node, we can have a prediction vector based on its sub-tree.

The initial simplification table is filled with constants and original features

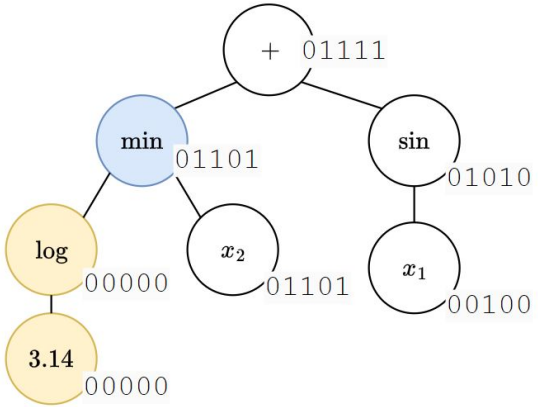
Our proposal: inexact simplification with LSH



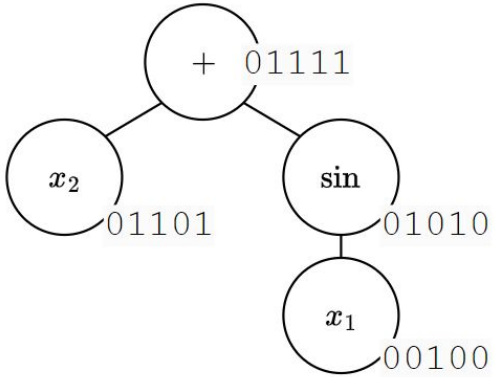
Simplification table	
00000	$[\texttt{cte}, 3.14, \log(3.14)]$
00100	$[x_1]$
01101	$[x_2, \min(\log(3.14), x_2)]$
01010	$[\sin x_1]$
01111	$[x_2 + \sin x_1, \min(\log(3.14), x_2) + \sin x_1]$

We can calculate the hash of each node based on its prediction vector, and store the sub-tree into the table.

Our proposal: inexact simplification with LSH



Simplification table	
00000	$[\texttt{cte}, 3.14, \log(3.14)]$
00100	$[x_1]$
01101	$[x_2, \min(\log(3.14), x_2)]$
01010	$[\sin x_1]$
01111	$[x_2 + \sin x_1, \min(\log(3.14), x_2) + \sin x_1]$



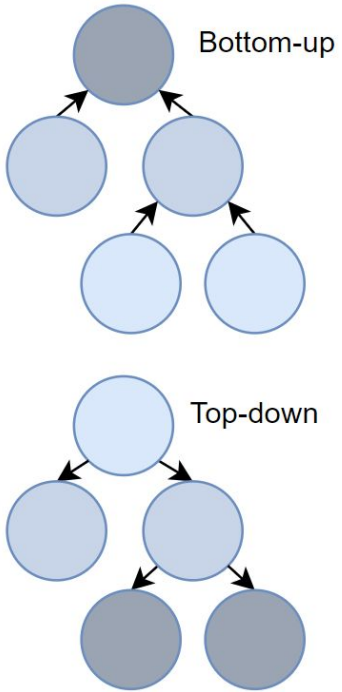
Then, we iterate through the tree replacing each sub-tree by the smallest one with same hash.

Experimental setup

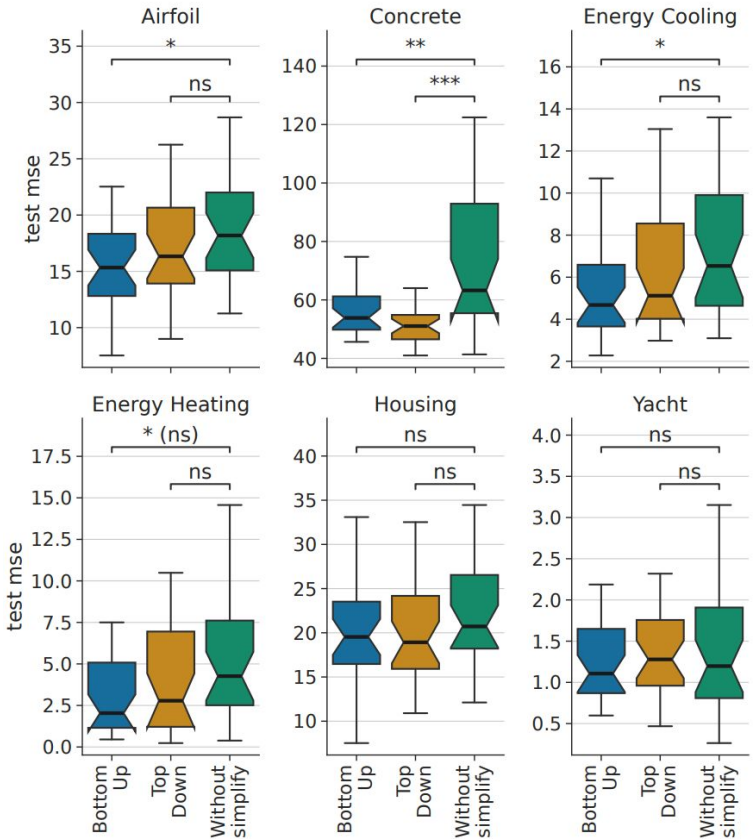
6 datasets, hyperparameters fixed, changing only the simplification method, iterating bottom-up or top-down

Dataset	# samples	# features
Airfoil	1503	5
Concrete	1030	8
Energy Cooling	768	8
Energy Heating	768	8
Housing	506	13
Yacht	308	6

Parameter	Value
pop size (S)	80
max gen (G)	200
max depth ($\max_d$)	7
max size ($\max_s$)	128 (2^7)
tolerance (τ)	$1e-2$
hash_len	256 bits
probabilities	1/5 for each variation operator
objectives	[error (MSE), size (# nodes))]
Function set	$[+, -, *, \div, \lvert\cdot\rvert, \cos^{-1}, \sin^{-1}, \tan^{-1},$ $\cos, \sin, \tan, e^{(\cdot)}$, min, max, $\log, \log(1+\cdot), \exp(1+\cdot), \sqrt{\lvert\cdot\rvert}, (\cdot)^2]$



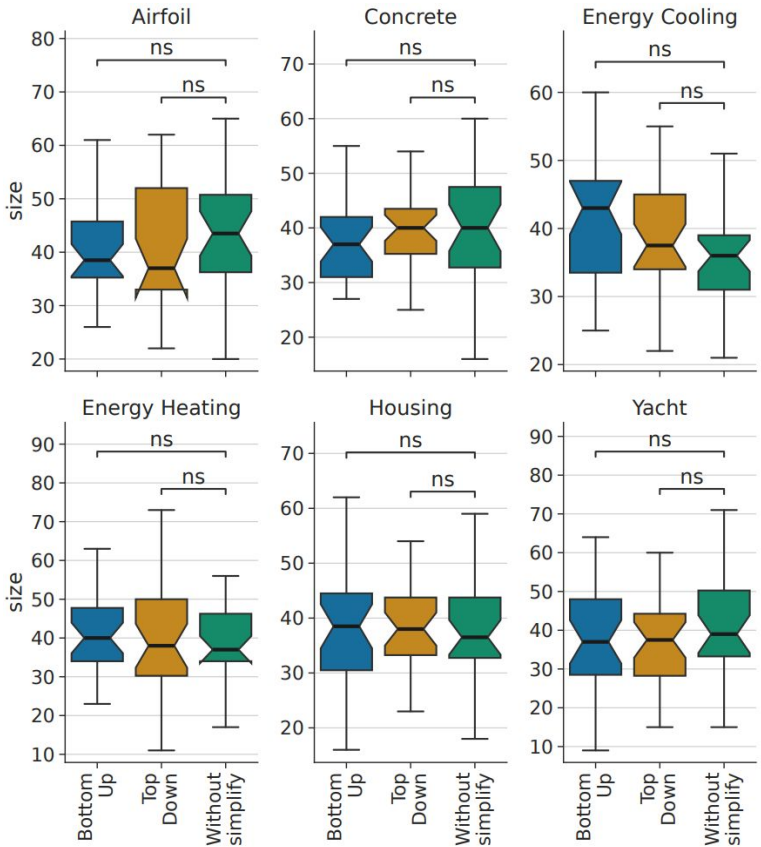
Test MSE



Both simplification strategies can improve MSE on test partition for some problems when compared to without using any simplification.

We observe a median of 20% reduction on MSE.

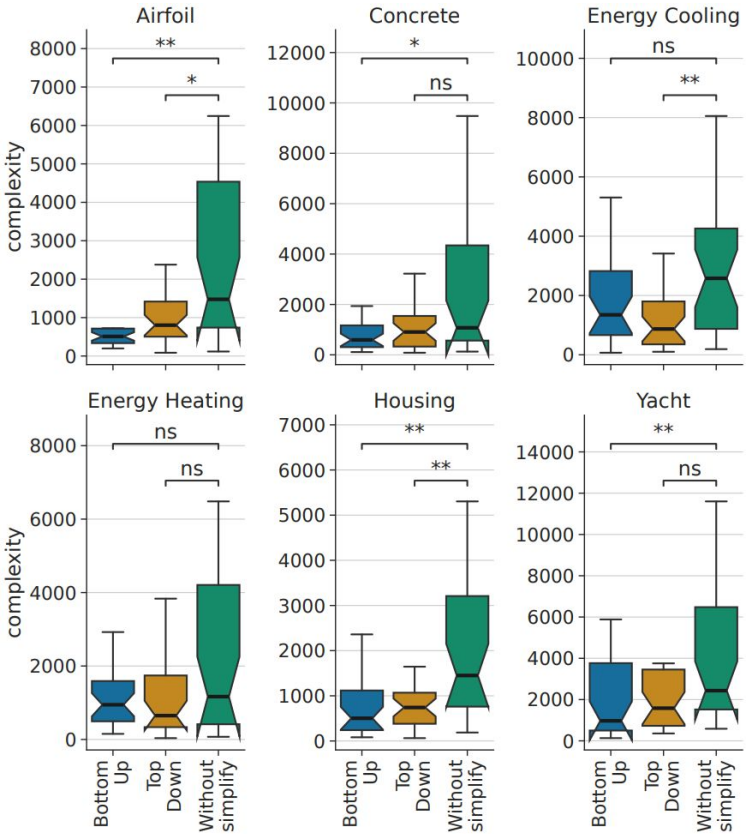
Size



No significant difference on final size.

we believe that simplification removes unnecessary terms of the expressions, and this freeing up space allows the search to find better structures to fill the space.

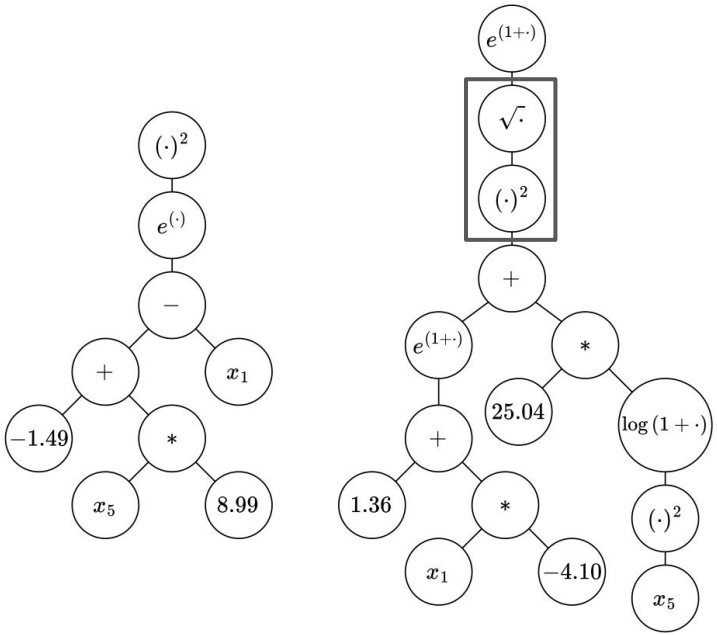
Complexity



Complexity is small when we simplify.

This means that the models are easier to humans to comprehend.

Results w/ and w/o simplification



Smallest expression found by bottom-up (left) and without simplification (right) for the Yacht dataset. The MSE for each expression was 1.98 using the bottom-up simplification and 2.08 without simplification.

Results w/ and w/o simplification

```
1010101111001001...
 - x_2
 - absolute(x_2)
 - minimum(x_2, x_2)
 - minimum(x_2, x_6)
 - minimum(x_2, x_5)
 - add(0.0, x_2)
 - log(exp(x_2))
 - sqrtabs(square(x_2))
 - maximum(-523.249, x_2)
```

```
1010110111001101...
 - square(x_5)
 - multiply(x_5, x_5)
 - absolute(square(x_5))
 - maximum(square(x_5), x_0)
 - maximum(add(-15.455, x_1), square(x_5))
```

```
1110001011011111...
 - multiply(x_1, x_7, x_0, x_4)
 - multiply(x_0, x_4, x_1, x_7)
 - multiply(x_1, x_0, multiply(x_4, x_7))
```

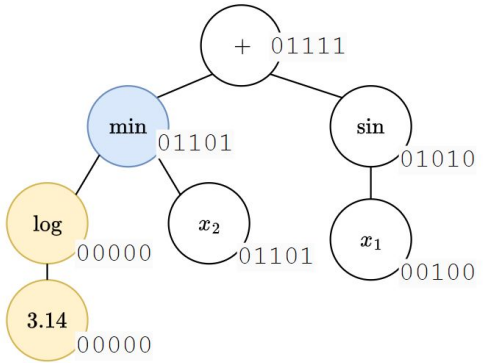
The simplification found 5144 different hashes, with a total of 7324 expressions.

Here we can see some equivalent expressions that occurrences would be replaced by the smallest among them.

Conclusions

Our simplification returned more accurate and less complex expressions.

The drawback is that simplification is inexact, so it can perform substitutions that slightly changes the behavior of the model. The method relies on hash collisions, to be studied in future work.



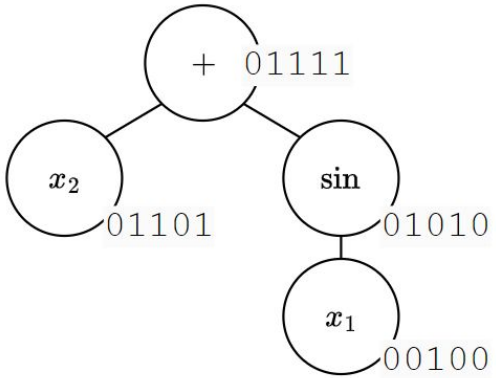
Simplification table	
00000	$[\texttt{cte}, 3.14, \log(3.14)]$
00100	$[x_1]$
01101	$[x_2, \min(\log(3.14), x_2)]$
01010	$[\sin x_1]$
01111	$[x_2 + \sin x_1, \min(\log(3.14), x_2) + \sin x_1]$

Takeaways

- The populational aspect of SR helps filling the simplification table;
- We can have simplification without having to define algebraic rules;
- Final models are simpler and more accurate than when not using any simplification.

Our method can be implemented to SR as long as you can:

1. Iterate through the expression, and
2. Replace parts of it.



Inexact simplification of symbolic expressions

Thank you!

Guilherme Aldeia

guilherme.aldeia@ufabc.edu.br

UFABC

Boston Children's Hospital
Until every child is well™

AVA Lab

VE RI TAS

chip Computational Health Informatics Program

CMCC
Centro de Matemática, Computação e Cognição