

***“ Why Simulation-Based Approachs
with Combined Fitnesses are a Good
Approach for Mining Spaces of Turing
Computable Functions »***

or

***« Why GP is nice for desperate cases
at the limit of computability »***

Olivier Teytaud (TAO, Inria, Lri, UMR Cnrs, Université Paris-Sud)



Slide 1

Why I love GP

The idea of this work

- Genetic Programming is boring :
 - Sometimes it does not find anything
 - Often it is slow

is it possible to put GP in the thrash and to use something else ?



The idea of this work

“ Is it possible to put GP in the thrash and to use something else ?”

We formalize this question, and we essentially give a negative answer.



Organization of this talk

- The framework of Turing-computable functions for symbolic regression
- The computability of optimization in such spaces
- Iterative algorithms
- Complexity



Turing-computable functions

- Usually GP deals with much more restricted spaces of functions (DAGs)
- Here we allow :
 - Loops
 - Tests
 - Subroutines
 - C++, Pascal, Matlab, ...



Symbolic regression

- n pairs « input x output » are provided
- GP looks for a function that maps inputs to outputs
- Troubles :
 - It works for n inputs, but does it work for other possible outputs ?
 - This point has already been studied (e.g. S. Droste, GP98, Teytaud et al, Gecco05) : short programs generalize well under the “i.i.d” framework

Temporary conclusion



We have to find a function

- that works on all examples ;
- that are as small as possible ...

and this program will generalize well.



Organization of this talk

- The framework of Turing-computable functions for symbolic regression
- **The computability of optimization in such spaces**
- Iterative algorithms
- Complexity



Symbolic regression in Turing-computable functions

- What is already known ?
 - Using the smallest program that fits the data generalizes well (Occam's Razor) and lead to Universal Consistency (UC) : as the number of examples runs to infinity, the average error rate in generalization goes to 0
 - Finding the smallest program that fits the data is not possible (Kolmogorov's complexity)
- What else can we show ?
 - Also, the fastest (when it exists) can not be computed
 - Also, the less space-consuming can not be computed
 - Many other criterions lead to the same uncomputability (*degree = 0'*)

Temporary conclusion

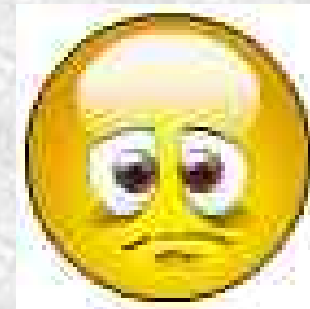


We have to find a function

- that works on all examples ;
- that is the smallest ... ;

and this program will generalize well.

But this is not computable !



What is possible then ?

Using something else than « finite-time algorithms » : *iterative algorithms*

We do not consider

program = Algorithm(inputs, outputs)

but

(program1, program2, ...) = Algorithm(input1, output1, input2, output2, ...)



Organization of this talk

- The framework of Turing-computable functions for symbolic regression
- The computability of optimization in such spaces
- **Iterative algorithms**
- Complexity

Which iterative algorithms ?

Ok, let's try :

(program1, program2, ...) = Algorithm(inputs, outputs, ...)

How to define program(i) ?

program(i) = optimum for some compromise between

(1) average computation time on i examples

(2) length

among functions that fit the inputs/outputs



Is this computable ?

program(i) = optimum for some compromise between

(1) average computation time on i examples

(2) length

among functions that fit the inputs/outputs relevance to inputs

is this computable ?

Yes : Kolmogorov's complexity with bounded resources.



Does this generalize ?

program(i) = optimum for some compromise between

(1) average computation time

(2) length

among functions that fit the inputs/outputs relevance to inputs

does this generalize ?

Yes : VC-dimension increases slowly enough (Sieve methods)



So we are happy ?

program(i) = optimum for some compromise between

(1) average computation time

(2) length

among functions that fit the inputs/outputs relevance to inputs

- program(i) is computable
- When i runs to infinity, a program that generalizes is found

But how long is the computation time ?

GP uses (expensive) simulations : can we get rid of the cost of simulations ?





Temporary conclusion




We have to find a function

- that works on all examples ;
- that is not too slow / large / ... ;

and this program will generalize well.

And this can be computed by iterative algorithms performing simulations !

Organization of this talk

- The framework of Turing-computable functions for symbolic regression
- The computability of optimization in such spaces
- Iterative algorithms
- **Complexity**
 - Can we do much faster than simulations ?
 - Essentially, no. 

Complexity of the optimization ?

We study the computation time required for finding the shortest program among not too slow programs.

With loss of generality (sorry) we study the computation time required for deciding if n can be generated in time T by a machine of size S .



Complexity : formalization

We study the computation time required for deciding if n is (T,S) -complex.

Def: n is (T,S) -complex if there is no Turing-machine of size S that generates x in time S .

Def: consider A an algorithm deciding if x is (T,S) -complex. Then $C(A,t,s) = \sup_x T(A(x,t,s))$

\implies we show lower bounds on $C(A,t,s)$

Complexity : the result

Consider $T(n)$ any more-than-exponential sequence,
and $S(n)$ a given logarithmic sequence.

Then, $C(A, T(n), S(n))$ is at least $T(n)/\text{polynomial}(n)$

Proof: analogous to Berry's paradox.



Berry's paradox

(



Berry's paradox (informal)

Consider y the smallest number that can not be described within 2 lines with this police in this slide.

We have described y in two lines !



Berry's paradox (formal)

Formal version : consider the following program:

P looks at its length D ,

P then tests all programs shorter than D lines,
considers all their outputs y_1, \dots, y_N .

P outputs the first number that is not in y_1, \dots, y_N and stops.

\implies Paradox: P outputs a number that can not be generated in D lines whereas it is made of D lines.

Berry's paradox : conclusions

==> Paradox: P outputs a number that can not be generated in D lines whereas it is made of D lines.

==> we have cheated : it is not possible to know if a given Turing-machine halts.

==> the same reasoning shows that the smallest number that can not be described in D lines is not computable

Berry's paradox

)

--> back to GP



Analogous of Berry's paradox for complexity

Consider the smallest integer that can not be computed within resources (T, S) .

If (T, S) -complexity can be computed with low resources, then this integer can be computed with not so large resources also !

==> paradox



$C(A, T(n), S(n))$ is at least $T(n)/\text{polynomial}(n)$

Proof:

- Consider $y(n)$ the smallest $T(n), S(n)$ -complex integer
- $y(n)$ can be computed in time roughly $y(n) \times C(T(n), S(n))$ by a program of size roughly $S(n)$ (using algorithm A on $1, 2, 3, \dots, y(n)$)
- $y(n) = O(2^{S(n)})$ there are at most $2^{\{S(n)\}}$ integers computable in size $S(n)$
- Roughly $O(2^{S(n)}) C(T(n), S(n)) > T(n)$
- $\implies C(T(n), S(n)) > T(n)/\text{poly}(n)$ □



Conclusion



Iterative algorithms can find solutions in cases in which finite-time algorithms can not, e.g. finding short and fast programs that realize a given task.

The iterative algorithms used in our proofs are simulation-based.

We are not sure that any solution is simulation-based, but we know that a wide family of solutions implies a computational time close to simulation.



Conclusion



Philosophically: the approach with simulations/selections like in GP has drawbacks :

- we perform iterations without knowing when to stop ;
- we use expensive simulations.



Conclusion



But, both conditions are necessary in the sense that :

- the solution must be iterative without stopping criterion (unless you get an oracle for the halting problem !) ;
- the computation time is at least roughly the simulation time, unless there exists an iterative solution that does not solve (T,S) -complexity.