Improvements in Program Synthesis for Integer Sequences

Thibault Gauthier, Miroslav Olšák, Josef Urban

September 4, 2023

This work was partially supported by the CTU Global Postdoc funding scheme (TG), Czech Science Foundation project 20-06390Y (TG), ERC-CZ project POSTMAN no. LL1902 (TG), Amazon Research Awards (TG, JU), EU ICT-48 2020 project TAILOR no. 952215 (JU), and the European Regional Development Fund under the Czech project AI&Reasoning with identifier CZ.02.1.01/0.0/0.0/15_003/0000466 (MO, JU).

Selection of 123 Solved Sequences

https://github.com/Anon52MI4/oeis-alien

Table: Samples of the solved sequences.

https://oeis.org/A317485	Number of Hamiltonian paths in the n-Bruhat graph.
https://oeis.org/A349073	a(n) = U(2*n, n), where U(n, x) is the Chebyshev polynomial of the second kind.
https://oeis.org/A293339	Greatest integer k such that $k/2^n < 1/e$.
https://oeis.org/A1848	Crystal ball sequence for 6-dimensional cubic lattice.
https://oeis.org/A8628	Molien series for A_5.
https://oeis.org/A259445	Multiplicative with $a(n) = n$ if n is odd and $a(2^s) = 2$.
https://oeis.org/A314106	Coordination sequence Gal.6.199.4 where G.u.t.v denotes the coordination sequence for a vertex of type v in tiling number t in the Galebach list of u-uniform tilings
https://oeis.org/A311889	Coordination sequence Gal.6.129.2 where G.u.t.v denotes the coordination sequence for a vertex of type v in tiling number t in the Galebach list of u-uniform tilings.
https://oeis.org/A315334	Coordination sequence Gal.6.623.2 where G.u.t.v denotes the coordination sequence for a vertex of type v in tiling number t in the Galebach list of u-uniform tilings.
https://oeis.org/A315742	Coordination sequence Gal.5.302.5 where G.u.t.v denotes the coordination sequence for a vertex of type v in tiling number t in the Galebach list of u-uniform tilings.
https://oeis.org/A004165	OEIS writing backward
https://oeis.org/A83186	Sum of first n primes whose indices are primes.
https://oeis.org/A88176	Primes such that the previous two primes are a twin prime pair.
https://oeis.org/A96282	Sums of successive twin primes of order 2.
https://oeis.org/A53176	Primes p such that $2p + 1$ is composite.
https://oeis.org/A267262	Total number of OFF (white) cells after n iterations of the "Rule 111" elementary cellular automaton starting with a single ON (black) cell.

What Are the Current AI/TP TODOs/Bottlenecks?

- High-level structuring of proofs - proposing good lemmas
- Proposing new concepts, definitions and theories
- Proposing new targeted algorithms, decision procedures, tactics
- Proposing good witnesses for existential proofs
- All these problems involve synthesis of some mathematical objects
- Btw., constructing proofs is also a synthesis task
- This talk: explore learning-guided synthesis for OEIS
- Interesting research topic and tradeoff in learning/AI/proving:
- Learning direct guessing of objects (this talk) vs guidance for search procedures (ENIGMA and friends)
- Start looking also at semantics rather than just syntax of the objects

Quotes: Learning vs. Reasoning vs. Guessing

“C’est par la logique qu’on démontre, c’est par l’intuition qu’on invente.”
(It is by logic that we prove, but by intuition that we discover.)

– Henri Poincaré, Mathematical Definitions and Education.

“Hypothesen sind Netze; nur der fängt, wer auswirft.”
(Hypotheses are nets: only he who casts will catch.)

– Novalis, quoted by Popper – The Logic of Scientific Discovery

Certainly, let us learn proving, but also let us learn guessing.

– G. Polya - Mathematics and Plausible Reasoning

Galileo once said, "Mathematics is the language of Science." Hence, facing the same laws of the physical world, alien mathematics must have a good deal of similarity to ours.

– R. Hamming - Mathematics on a Distant Planet

QSynt: Semantics-Aware Synthesis of Math Objects

- Synthesize math expressions based on semantic characterizations
- i.e., not just on the syntactic descriptions (e.g. proof situations)
- Tree Neural Nets and Monte Carlo Tree Search
- Recently also various (small) *language models* with their search methods
- Invention of programs for OEIS sequences from scratch
- 100k OEIS sequences (out of 350k) solved so far: `https://www.youtube.com/watch?v=24oejR9wsXs`, `http://grid01.ciirc.cvut.cz/~thibault/qsynt.html`
- Millions of conjectures invented: 20+ different characterizations of primes
- Non-neural (Turing complete) computing and semantics collaborates with the statistical/neural learning

OEIS: ⩾ 350000 finite sequences

The OEIS is supported by the many generous donors to the OEIS Foundation.

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES ®

founded in 1964 by N. J. A. Sloane

2 3 5 7 11 | Search | Hints

(Greetings from The On-Line Encyclopedia of Integer Sequences!)

Search: **seq:2,3,5,7,11**

Displaying 1-10 of 1163 results found. page 1 2 3 4 5 6 7 8 9 10 ... 117

Sort: relevance | references | number | modified | created Format: long | short | data

A000040 The prime numbers. +30
(Formerly M0652 N0241) 10150

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271 (list; graph; refs; listen; history; text; internal format)

OFFSET 1,1

COMMENTS See A065091 for comments, formulas etc. concerning only odd primes. For all information concerning prime powers, see A000961. For contributions concerning "almost primes" see A002808.
A number p is prime if (and only if) it is greater than 1 and has no positive divisors except 1 and p.
A natural number is prime if and only if it has exactly two (positive) divisors.
A prime has exactly one proper positive divisor, 1.

Generating programs for OEIS sequences

$0, 1, 3, 6, 10, 15, 21, \ldots$

An undesirable large program:

```
if x = 0 then 0 else
if x = 1 then 1 else
if x = 2 then 3 else
if x = 3 then 6 else ...
```

Small program (Occam's Razor):

$$\sum_{i=1}^{n} i$$

Fast program (efficiency criteria):

$$\frac{n \times n + n}{2}$$

Programming language

- Constants: $0, 1, 2$
- Variables: x, y
- Arithmetic: $+, -, \times, div, mod$
- Condition : if $\ldots \leqslant 0$ then ... else ...
- $loop(f, a, b) := u_a$ where $u_0 = b$,

$$u_n = f(u_{n-1}, n)$$

- Two other loop constructs: *loop2*, a while loop

Example:

$2^{\mathbf{x}} = \prod_{y=1}^{x} 2 = loop(2 \times x, \mathbf{x}, 1)$
$\mathbf{x}! = \prod_{y=1}^{x} y = loop(y \times x, \mathbf{x}, 1)$

QSynt: synthesizing the programs/expressions

- Inductively defined set P of our *programs and subprograms*,
- and an auxiliary set F of binary functions (higher-order arguments)
- are the smallest sets such that $0, 1, 2, x, y \in P$, and if $a, b, c \in P$ and $f, g \in F$ then:

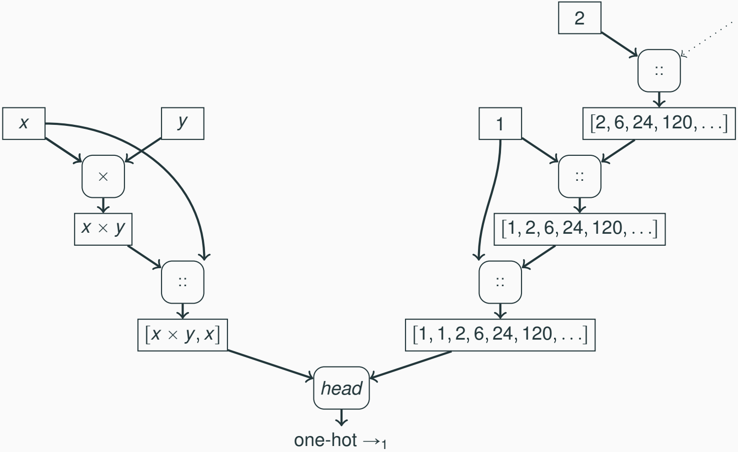
$$a + b, a - b, a \times b, a \; div \; b, a \; mod \; b, cond(a, b, c) \in P$$
$$\lambda(x, y).a \in F, \; loop(f, a, b), loop2(f, g, a, b, c), compr(f, a) \in P$$

- Programs are built in reverse polish notation
- Start from an empty stack
- Use ML to repeatedly choose the next operator to push on top of a stack
- Example: Factorial is $loop(\lambda(x, y).\; x \times y, x, 1)$, built by:

$$[\,] \rightarrow_x [x] \rightarrow_y [x, y] \rightarrow_\times [x \times y] \rightarrow_x [x \times y, x]$$
$$\rightarrow_1 [x \times y, x, 1] \rightarrow_{loop} [loop(\lambda(x, y).\; x \times y, x, 1)]$$

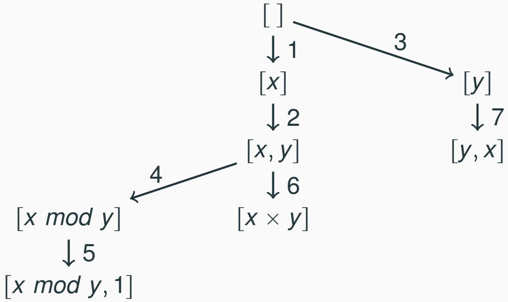
QSynt: Training of the Neural Net Guiding the Search

- The triple $((\mathit{head}([x \times y, x], [1, 1, 2, 6, 24, 120 \ldots]), \rightarrow_1)$ is a training example extracted from the program for factorial $\mathit{loop}(\lambda(x, y).\ x \times y, x, 1)$
- $\rightarrow_1$ is the action (adding 1 to the stack) required on $[x \times y, x]$ to progress towards the construction of $\mathit{loop}(\lambda(x, y).\ x \times y, x, 1)$.



QSynt program search - Monte Carlo search tree

7 iterations of the tree search gradually extending the search tree. The set of the synthesized programs after the 7th iteration is $\{1, x, y, x \times y, x\ mod\ y\}$.

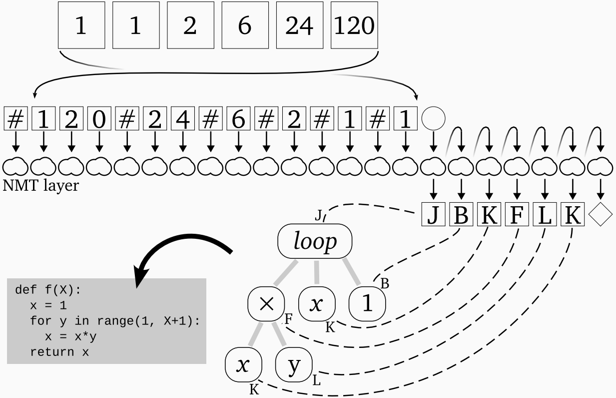


Using Language Models for Math Tasks

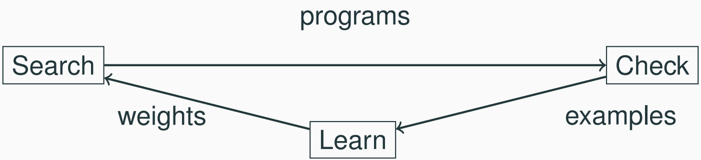
- Recurrent neural networks (RNNs) with attention (NMT)
- Transformers (BERT, GPT)
- Applied recently to symbolic/mathematical tasks:
- ... rewriting, conjecturing, translation from informal to formal
- Formulated as sequence-to-sequence translation tasks
- Efficient training and inference on GPUs, many toolkits
- Can get expensive for large LMs (LLMs) ($5M for GPT-3)
- We use small models on old HW, our total energy bill is below $1000

Encoding OEIS for Language Models

- Input sequence is a series of digits
- Separated by an additional token # at the integer boundaries
- Output program is a sequence of tokens in Polish notation
- Parsed by us to a syntax tree and translatable to Python
- Example: $a(n) = n!$



Search-Verify-Train Feedback Loop



Analogous to our Prove/Learn feedback loops in learning-guided proving (since 2006)

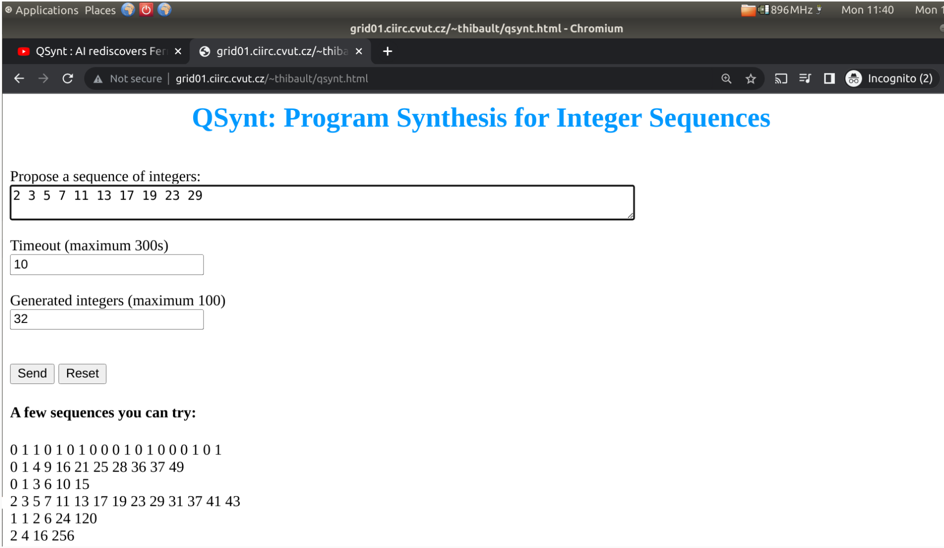
Search-Verify-Train Feedback Loop for OEIS

- **search phase**: LM synthesizes many programs for input sequences
- typically 240 candidate programs for each input using beam search
- 84M programs for OEIS in several hours on the GPU (depends on model)
- **checking phase**: the millions of programs efficiently evaluated
- resource limits used, fast indexing structures for OEIS sequences
- check if the program generates *any* OEIS sequence (hindsight replay)
- we keep the shortest (Occams's razor) and fastest program (efficiency)
- **learning phase**: LM trains to translate the “solved” OEIS sequences into the best program(s) generating them

Search-Verify-Train Feedback Loop

- The weights of the LM either trained from scratch or continuously updated
- This yields *new minds vs seasoned experts* (who have seen it all)
- We also train experts on varied selections of data, in varied ways
- Orthogonality: common in theorem proving - different experts help
- Each iteration of the self-learning loop discovers more solutions
- ... also improves/optimizes existing solutions
- The alien mathematician thus self-evolves
- Occam's razor and efficiency are used for its weak supervision
- Quite different from today's LLM approaches:
- LLMs do one-time training on everything human-invented
- Our alien instead starts from zero knowledge
- Evolves increasingly nontrivial skills, may diverge from humans
- Turing complete (unlike Go/Chess) – arbitrary complex algorithms

QSynt web interface for program invention



QSynt inventing Fermat pseudoprimes

Positive integers k such that $2^k \equiv 2 \mod k$. ($341 = 11 * 31$ is the first non-prime)

```
First 16 generated numbers (f(0),f(1),f(2),...):
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53
Generated sequence matches best with: A15919(1-75), A100726(0-59), A40(0-58)

Program found in 5.81 seconds
f(x) := 2 + compr(\x.loop(\(x,i).2*x + 2, x, 2) mod (x + 2), x)
Run the equivalent Python program here or in the window below:
```

Brython

Tutorial Demo Documentation Console Editor Gallery Resources

English

Brython version: 3.10.6

run Python Javascript Share code

```python
def f2(X):
  x = 2
  for i in range (1,X + 1):
    x = 2*x + 2
  return x

def f1(X):
  x,i = 0,0
  while i <= X:
    if f2(x) % (x + 2) <= 0:
      i = i + 1
    x = x + 1
  return x - 1

def f0(X):
  return 2 + f1(X)

for x in range(32):
  print (f0(x))

```

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
```

Lucas/Fibonacci characterization of (pseudo)primes

```
input sequence: 2,3,5,7,11,13,17,19,23,29

invented output program:
f(x) := compr(\(x,y).(loop2(\(x,y).x + y, \(x,y).x, x, 1, 2) - 1)
              mod (1 + x), x + 1) + 1

human conjecture: x is prime iff? x divides (Lucas(x) - 1)

PARI program:
? lucas(n) = fibonacci(n+1)+fibonacci(n-1)
? b(n) = (lucas(n) - 1) % n

Counterexamples (Bruckman-Lucas pseudoprimes):
? for(n=1,4000,if(b(n)==0,if(isprime(n),0,print(n))))
1
705
2465
2737
3745
```

QSynt inventing primes using Wilson's theorem

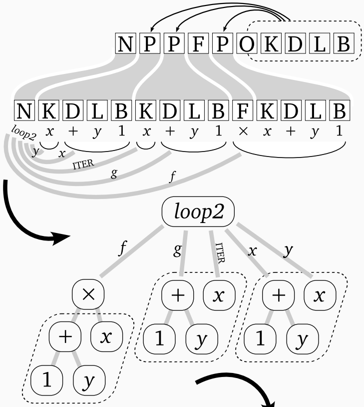
n is prime iff $(n-1)! + 1$ is divisible by n (i.e.: $(n-1)! \equiv -1 \mod n$)

```
First 32 generated numbers (f(0),f(1),f(2),...):
0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0
Generated sequence matches best with: A10051(0-100), A252233(0-29), A283991(0-24)

Program found in 5.17 seconds
f(x) := (loop(\(x,i).x * i, x, x) mod (x + 1)) mod 2
Run the equivalent Python program here or in the window below:
```

Introducing Local Macros/Definitions

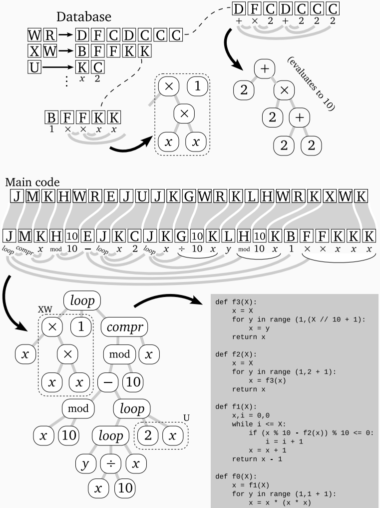
A macro/expanded version of a program invented for A1813: $a(n) = (2n)!/n!$.
1, 2, 12, 120, 1680, 30240, 665280, 17297280, 518918400, 17643225600,




```
def f(X,Y):
  x = 1+Y
  y = X
  for y in range(X+1,2*X+1):
    x = y*x
  return x
```

Introducing Global Macros/Definitions

A macro/expanded version of A14187 (cubes of palindromes).
0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1331, 10648, 35937, 85184,




```
def f3(X):
    x = X
    for y in range (1,(X // 10 + 1):
        x = y
    return x

def f2(X):
    x = X
    for y in range (1,2 + 1):
        x = f3(x)
    return x

def f1(X):
    x,i = 0,0
    while i <= X:
        if (x % 10 - f2(x)) % 10 <= 0:
            i = i + 1
        x = x + 1
    return x - 1

def f0(X):
    x = f1(X)
    for y in range (1,1 + 1):
        x = x * (x * x)
    return x
```

Five Different Self-Learning Runs

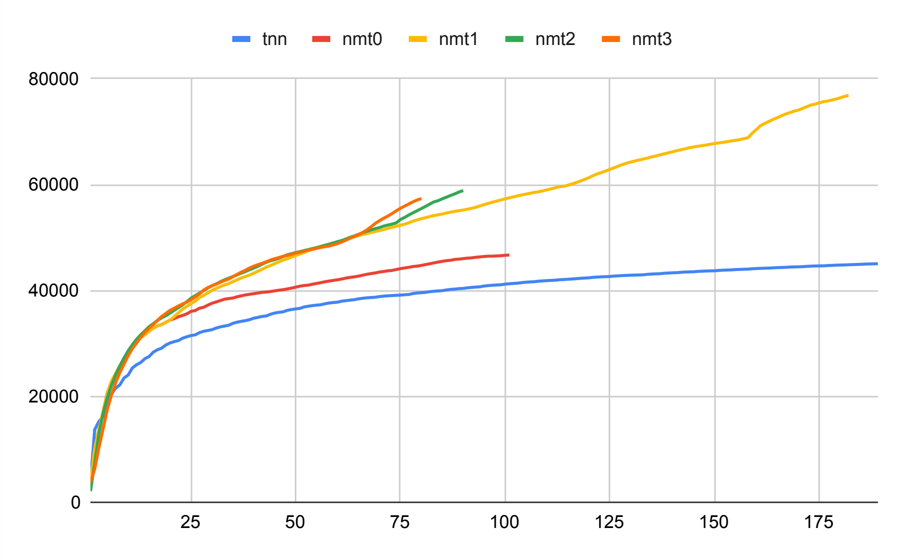


Figure: Cumulative counts of solutions.

Five Different Self-Learning Runs

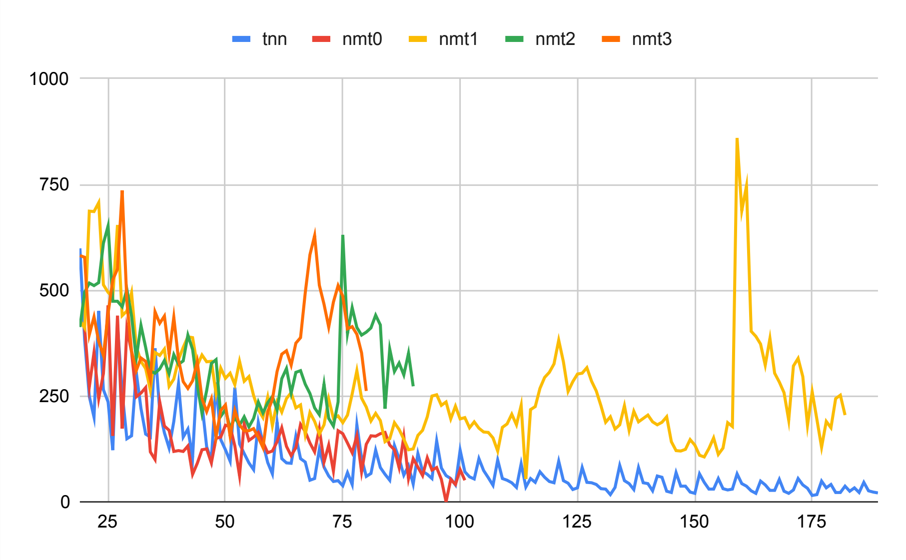


Figure: Increments of solutions.

Size Evolution

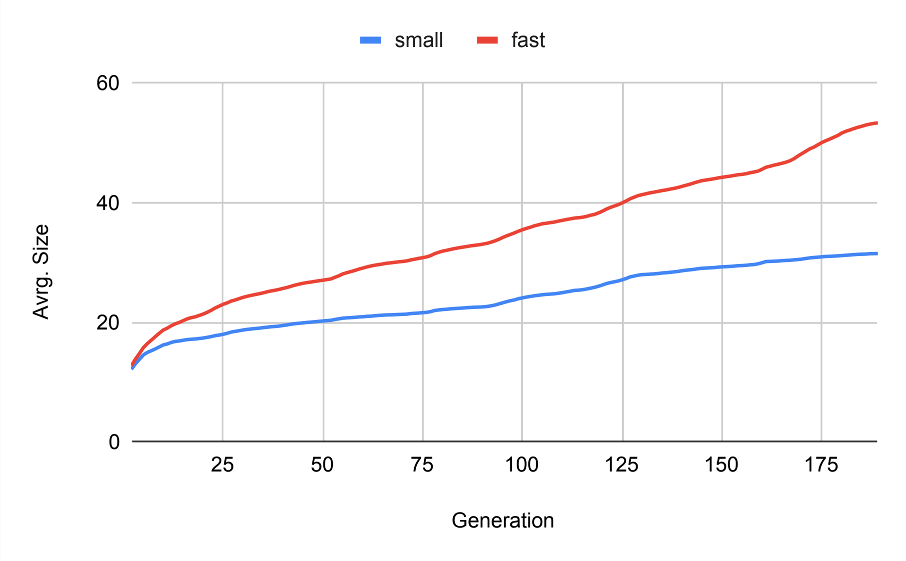


Figure: Avrg. size in iterations

Speed Evolution

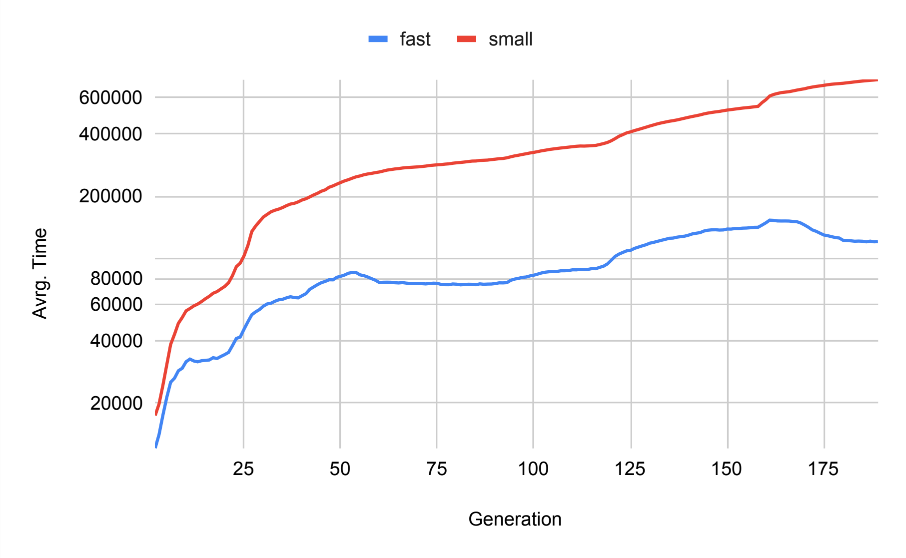


Figure: Avrg. time in iterations

Generalization of the Solutions to Larger Indices

- Are the programs correct?
- OEIS provides additional terms for some of the OEIS entries
- Among 78118 solutions, 40,577 of them have a b-file with 100 terms
- We evaluate both the small and the fast programs on them
- Here, 14,701 small and 11,056 fast programs time out.
- 90.57% of the remaining slow programs check
- 77.51% for the fast programs
- A common error is reliance on an approximation for a real number, such as π.

A Benchmark for Automated Theorem Provers

- 29687 sequences of with a fast program P and a fast program Q.
- Creation of 29687 SMT problems of the form $\forall x \in \mathbb{N}.\ f_P(x) = f_Q(x)$.
- Checked on the first 100 natural numbers.
- Can we prove that they hold on all natural numbers?
- Requires arithmetical and inductive reasoning

A Benchmark for Automated Theorem Provers

- A217, triangular numbers:

$$\sum_{i=0}^{n} i = \frac{n \times n + n}{2}$$

- A537, sum of first n cubes:

$$\sum_{i=0}^{n} i^3 = (\frac{n \times n + n}{2})^2$$

- A79, powers of 2:

$$2^x = 2^{(x \ mod \ 2)} \times (2^{(x \ div \ 2)})^2$$

- A165, double factorial of even numbers:

$$\prod_{i=1}^{n} 2i = 2^n \times n!$$

Application to Mathematics

Mathematicians are already using computer searches to produce conjectures or find mathematical objects:

- Discovery in 1995 of a more efficent formula for generating the digits of π by Simon Plouffe.
- In 2005, Hadi Kharaghani and Behruz Tayfeh-Rezaie published their construction of a Hadamard matrix of order 428.
- In 2012, Geoffrey Exoo has found an edge colorings of K35 that have no complete graphs of order 4 in the first color, and no complete graphs of order 6 in the second color proving that $R(4,6) \geqslant 36$.
- Discovery in 2023 of a chiral aperiodic monotile by David Smith.

Application to Mathematics

Find faster algorithms for generating pseudo-prime numbers.

Given a synthesized program implementing a function $f : \mathbb{Z} \times \mathbb{Z} \mapsto \mathbb{Z}$ we can construct a candidate matrix:

- $A = (a_{ij})$ for a Hadamard matrix by
  $a_{ij} =$ if $f(i, j) \leqslant 0$ then 1 else -1.
- $B = (b_{ij})$ for the adjacency matrix of a Ramsey graph by
  $b_{ij} =$ if $f(i, j) \leqslant 0$ then 1 else 0.

Solely guessing/conjecturing may not be enough to obtain new results on these open problems. A combination of statistical conjecture-making steps and deductive-style reasoning steps could be more successful.

Thank you for your attention!

`https://github.com/Anon52MI4/oeis-alien`