Finding interesting mathematical objects with ML

Adam Zsolt Wagner

Google DeepMind

AI 4 Maths Workshop
November 18, 2025

Joint work with François Charton, Jordan Ellenberg, Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Geordie Williamson

Goal of the talk

Proofs are important in mathematics, but for many problems this is not the focus.

Finding 'good' constructions is often the crucial bit.

- Counterexamples to conjectures
- Knowing the optimal constructions for large parameters let us spot patterns
- Lead to exploration, new conjectures, new theorems

"The methods for coming up with useful examples in mathematics . . . are even less clear than the methods for proving mathematical statements."
— Gil Kalai.

I am interested in simple, useful ML tools that we can give mathematicians to use, to search for such objects.

Simplest possible approach: Reinforcement Learning

Conjecture

For any graph G, we have $\lambda_1(G) + \mu(G) \geq \sqrt{n-1} + 1$.

Refuted in 2010, but smallest counterexample found has 600 vertices.

Game: for each edge, decide whether to include it in the graph of not

Reward: $\lambda_1 + \mu$ (minimize).

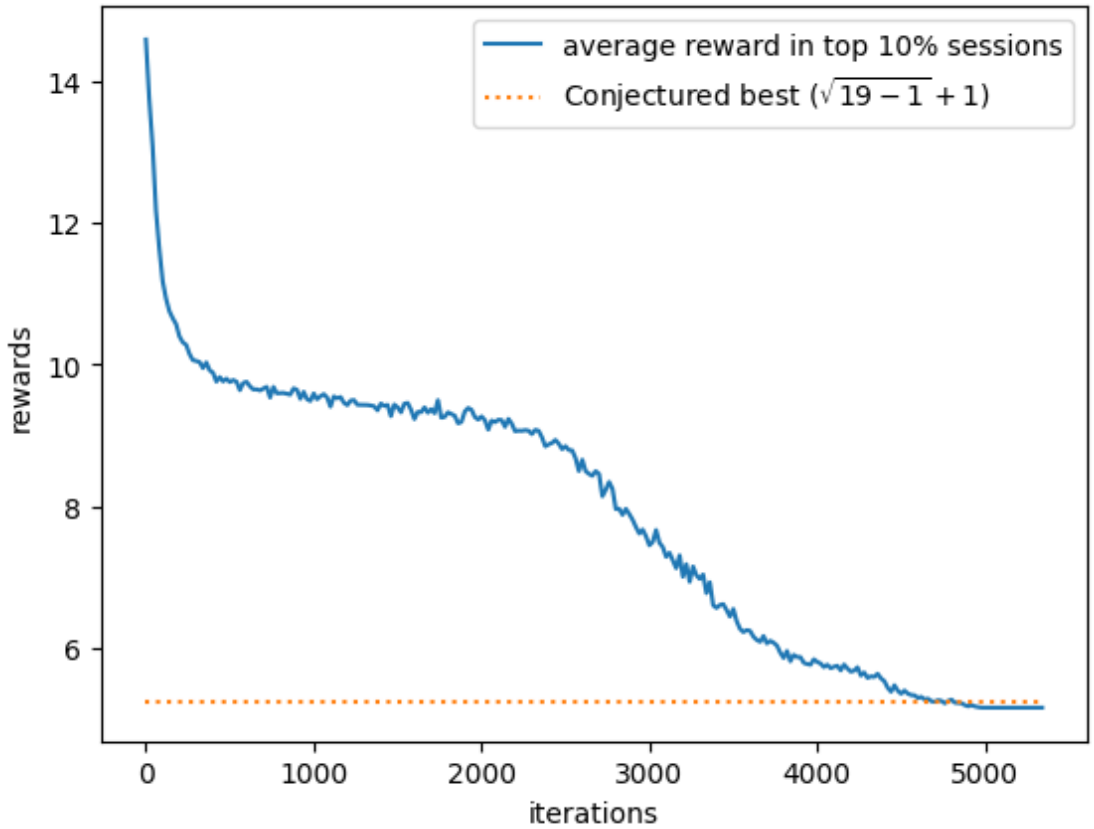
Run our simple algorithm for $n = 19$:

Constructions in combinatorics via neural networks, https://arxiv.org/abs/2104.14516

Example 1

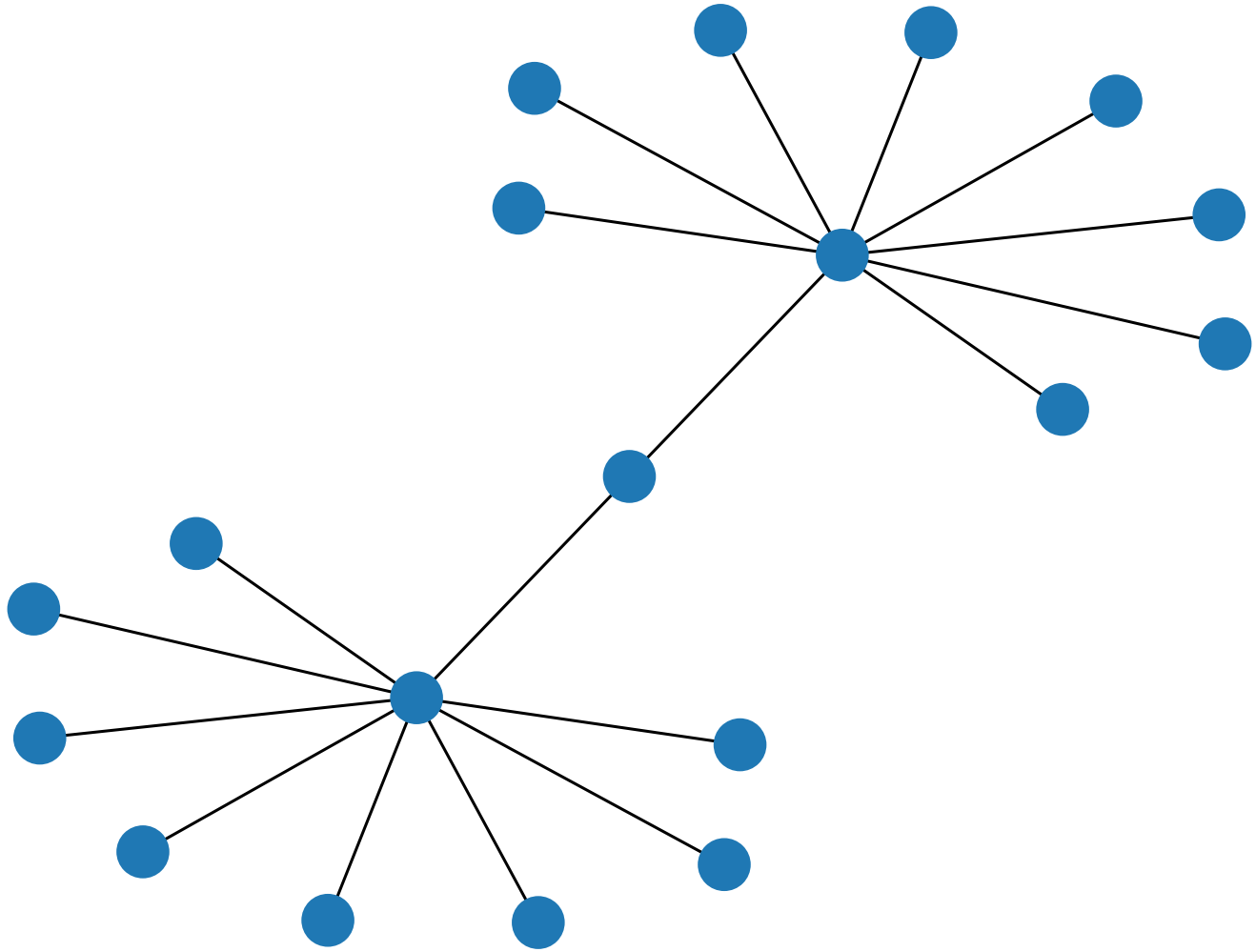
Conjecture

For any graph, $\lambda_1 + \mu \geq \sqrt{n-1} + 1$.



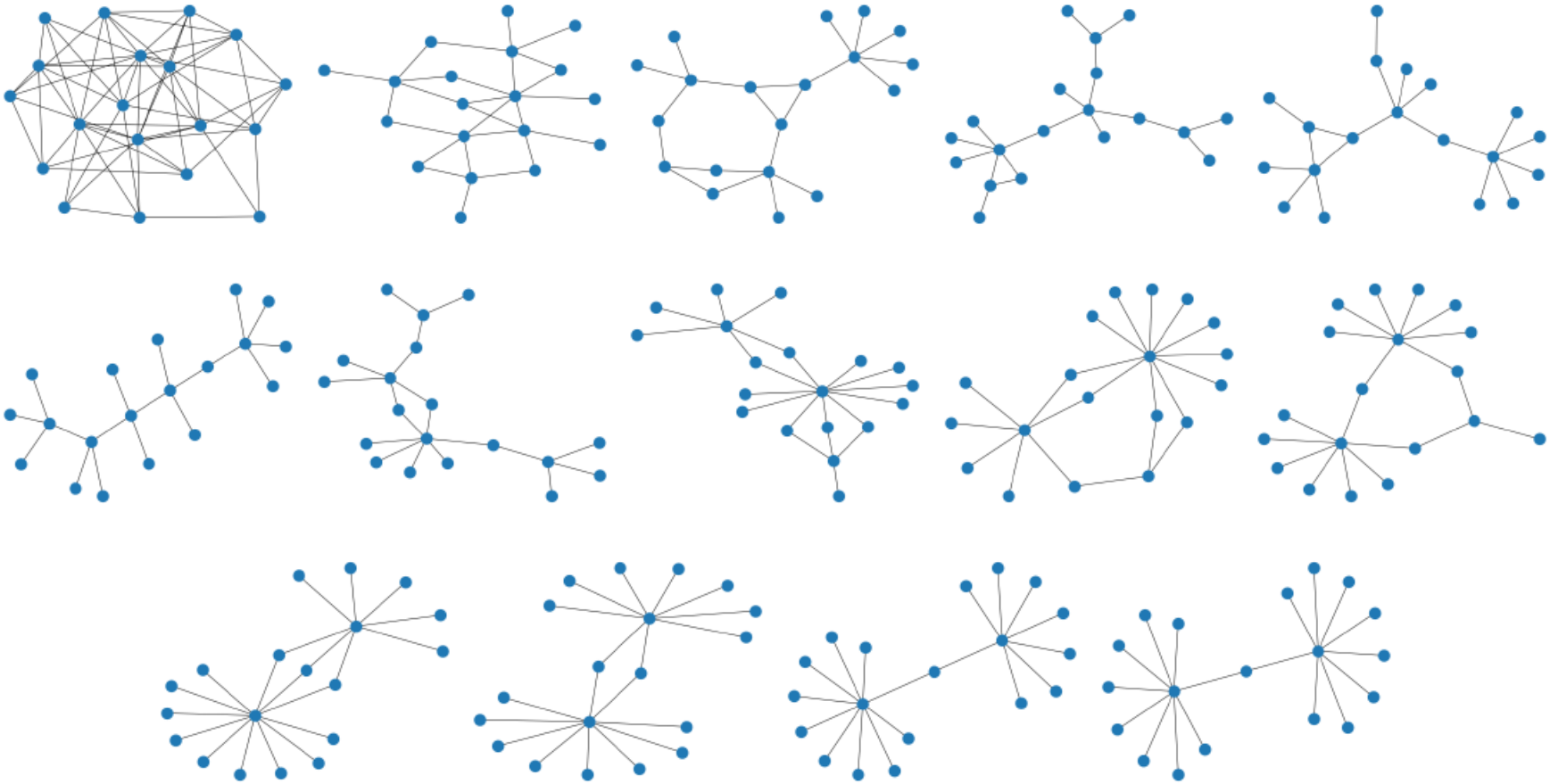
Constructions in combinatorics via neural networks, https://arxiv.org/abs/2104.14516

Example 1



Constructions in combinatorics via neural networks, https://arxiv.org/abs/2104.14516

Example 1



Pros and cons

How useful is this simple RL setup in pure maths research?

Pros:

- Simple and fun baseline method that can be thrown at a large class of problems
- Occasionally it works...

Cons:

- ...but most of the times it doesn't.
- Very slow, doesn't scale well
- Often doesn't perform better than simpler methods

It works well on a niche subfield of algebraic graph theory: Ghebleh–Al-Yakoob–Kanso–Stevanović (2024) found counterexamples to 30 out of 68 conjectures they tried.

The isosceles triangle problem

With a small change, we can make this method work a lot better in practice.

Question (Erdős)

How many points can we choose in the $N \times N$ grid, without choosing three points that satisfy $d(a, b) = d(b, c)$, i.e. without creating any isosceles triangles?

Let this maximum be $f(N)$. Barely anything is known about $f(N)$, it would be helpful to know see the best constructions for $N = 64$, say.

Charton, Ellenberg, W., Williamson PatternBoost: Constructions in Mathematics with a Little Help from AI https://arxiv.org/abs/2411.00566

The isosceles triangle problem

We create a large database of good 64×64 constructions, using standard local search methods.

We train a simple transformer model (Makemore) on these, and then generate more constructions like those in the dataset.

The model finds new good constructions much more frequently. We can feed these back into the local search method, and repeat.

Idea: alternating the local and global steps yields good results

---

Charton, Ellenberg, W., Williamson PatternBoost: Constructions in Mathematics with a Little Help from AI https://arxiv.org/abs/2411.00566

The isosceles triangle problem

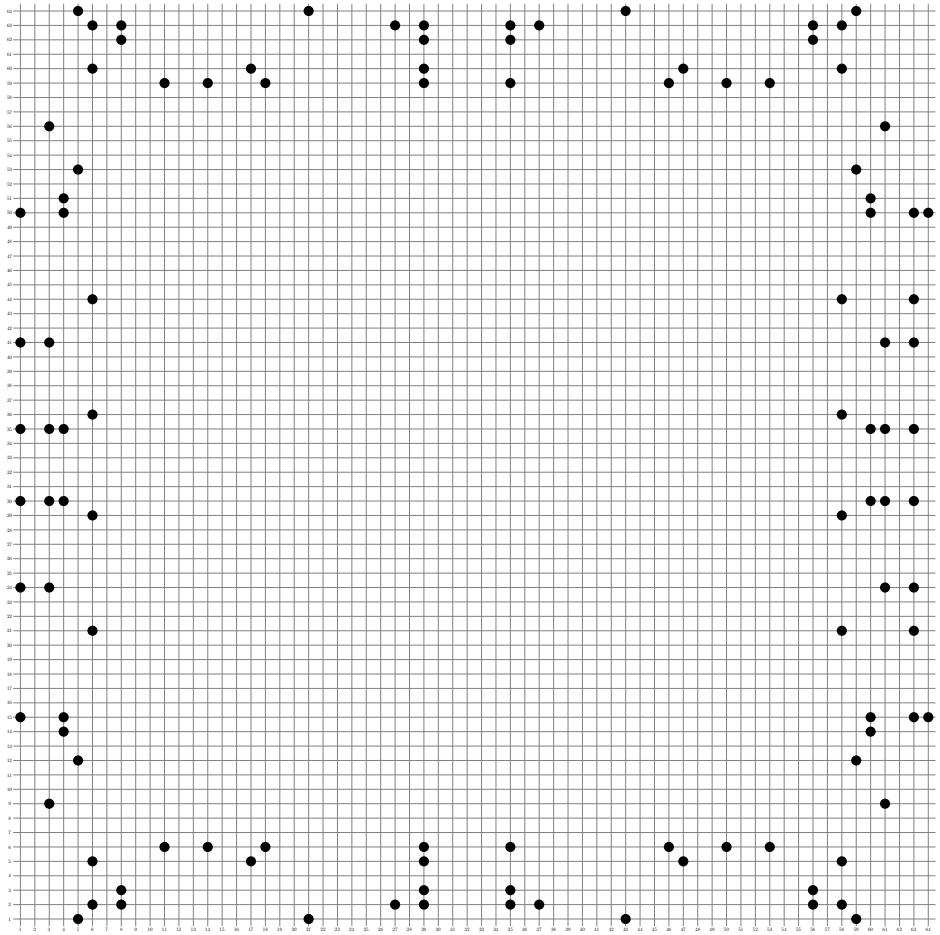


Figure: $f(64) \geq 110$

We were sure 112 was possible, but even after searching for months, we couldn't find it!

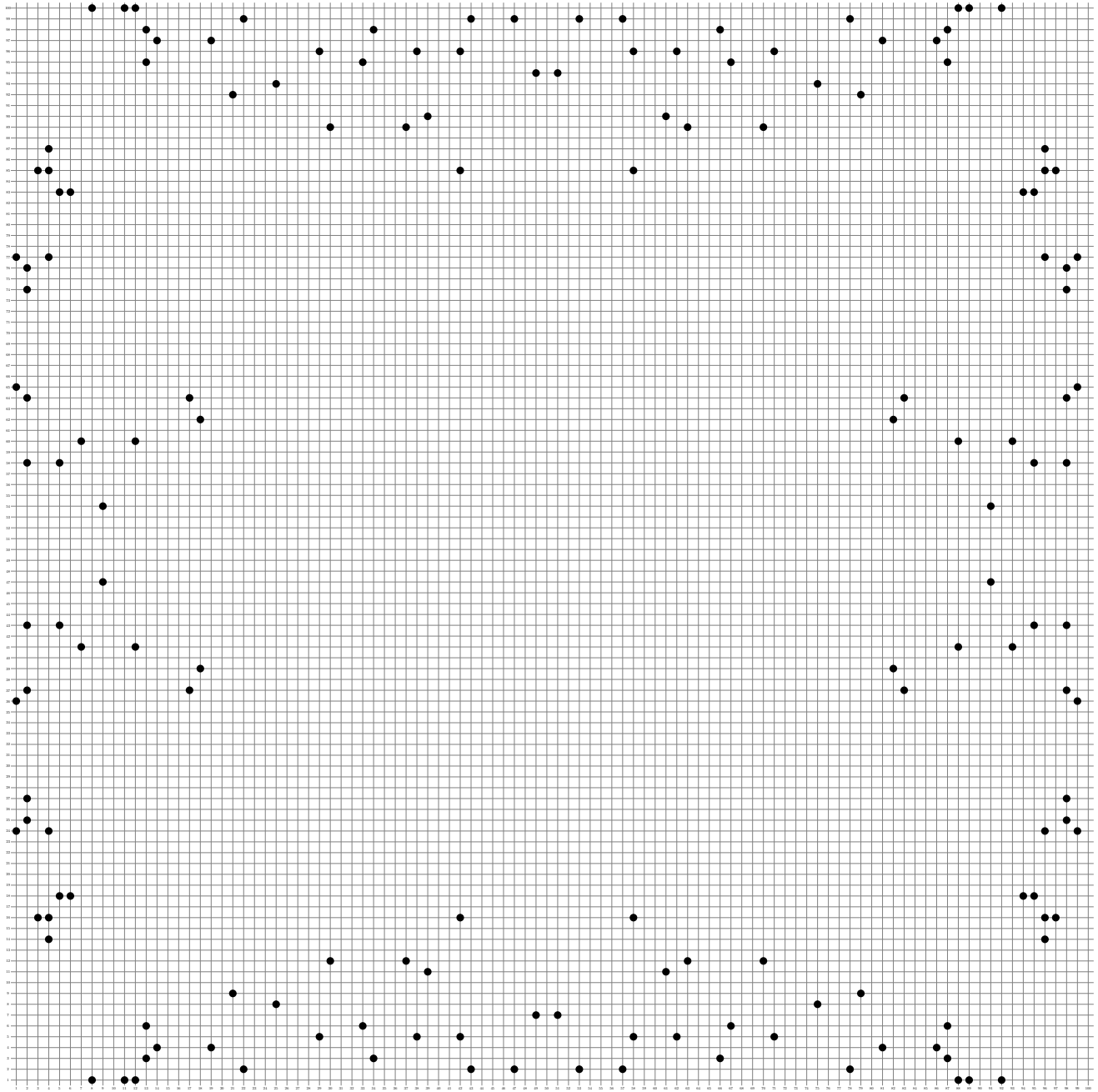


Figure: $f(100) \geq 160$

Searching in language space

Searching in language-space

Local search = if a construction is good, try other constructions that are **close** to the original one. But what does "close" mean?

- Standard local search: add/delete a few edges, change a handful of numbers slightly, etc
- Local search in language space: "close" means whatever the LLM thinks "close" means.

When is this useful?

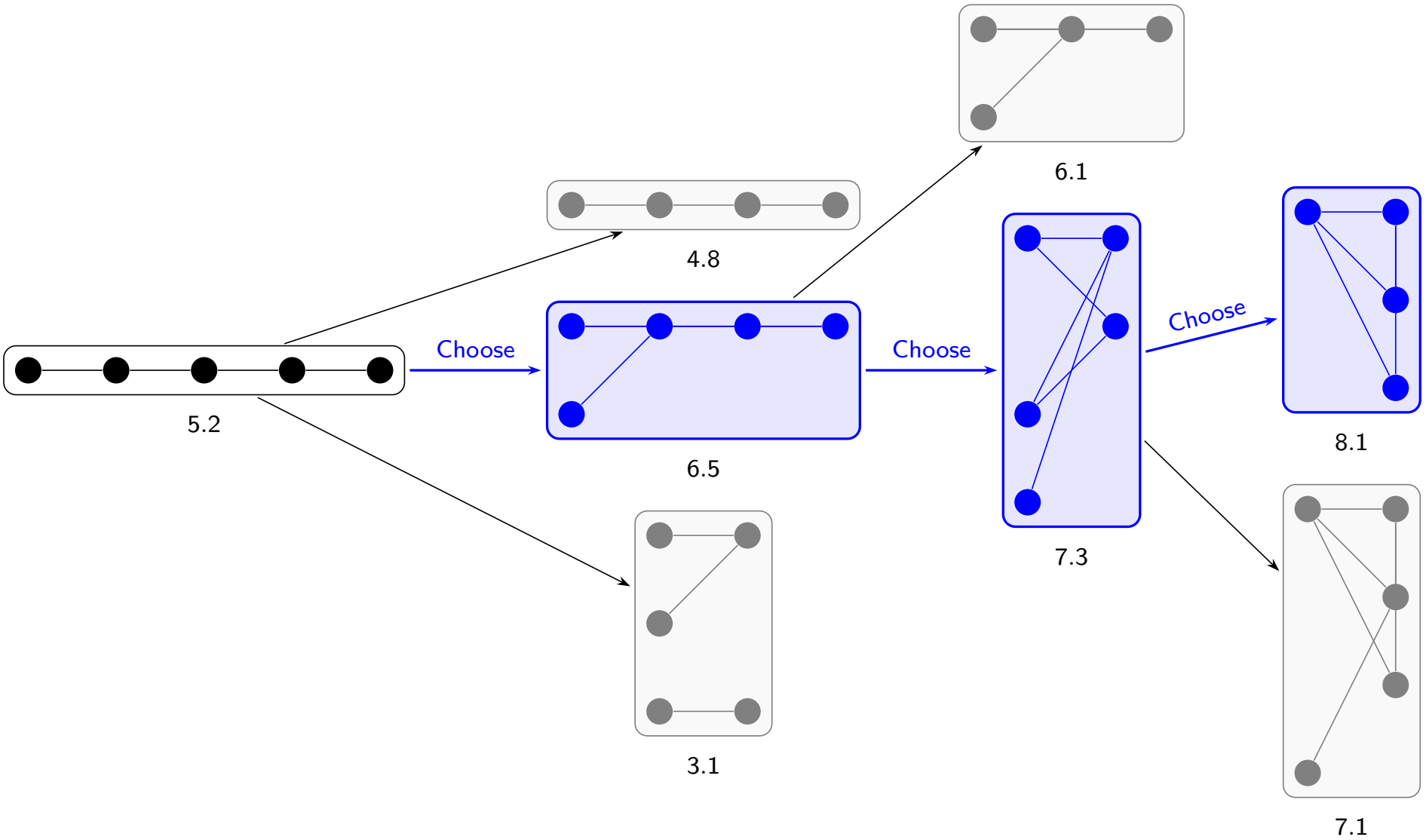
1. In the mathematics we humans care about, constructions often have a short description
2. We can use this setup to find efficient search functions that find good constructions

---

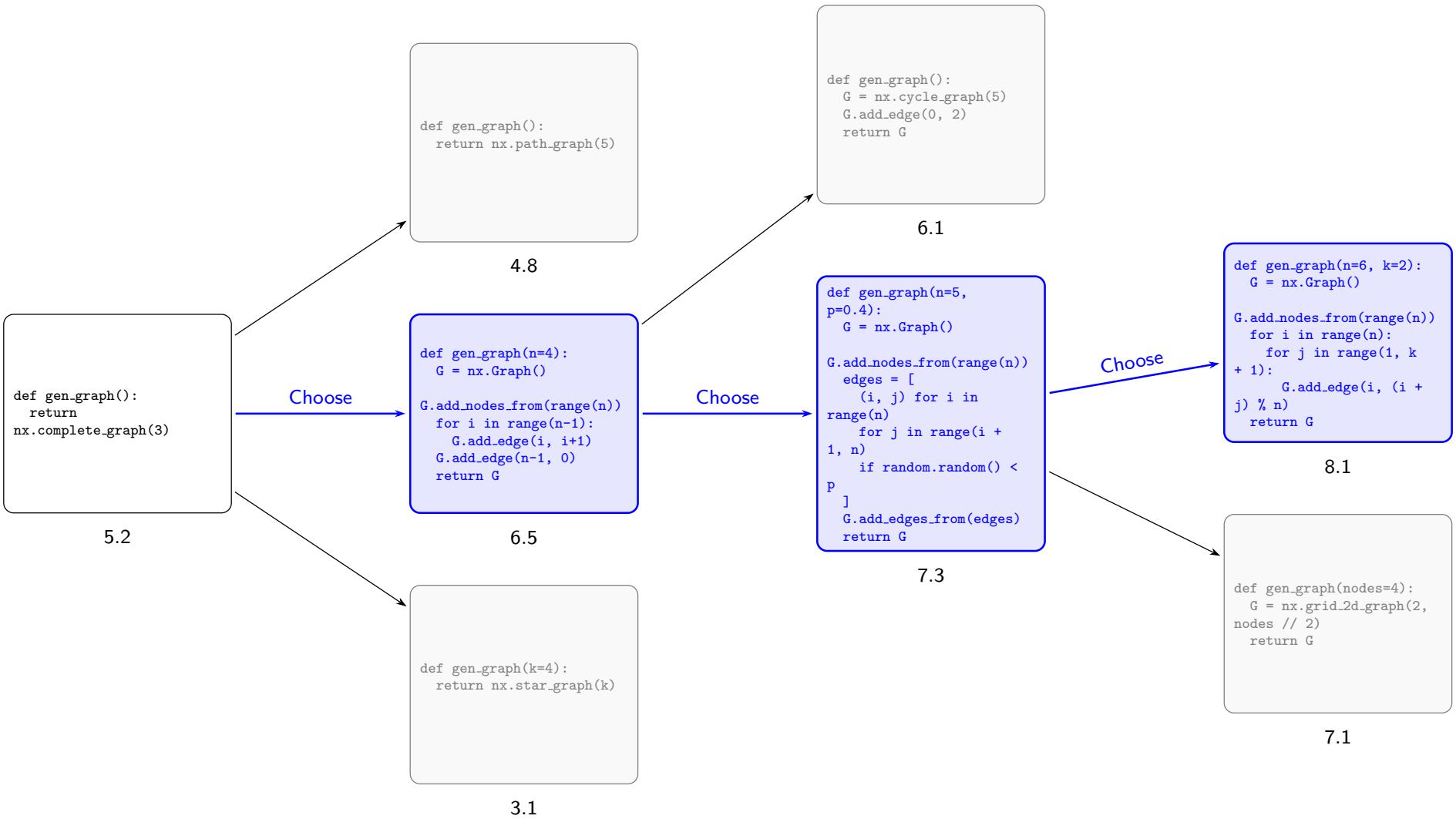
FunSearch. Mathematical discoveries from program search with large language models by Romera-Paredes et al., Nature, 2023

AlphaEvolve: A coding agent for scientific and algorithmic discovery by Novikov et al., 2025, in collaboration with Terence Tao and Javier Gomez-Serrano

Local Search Example: Graph Search



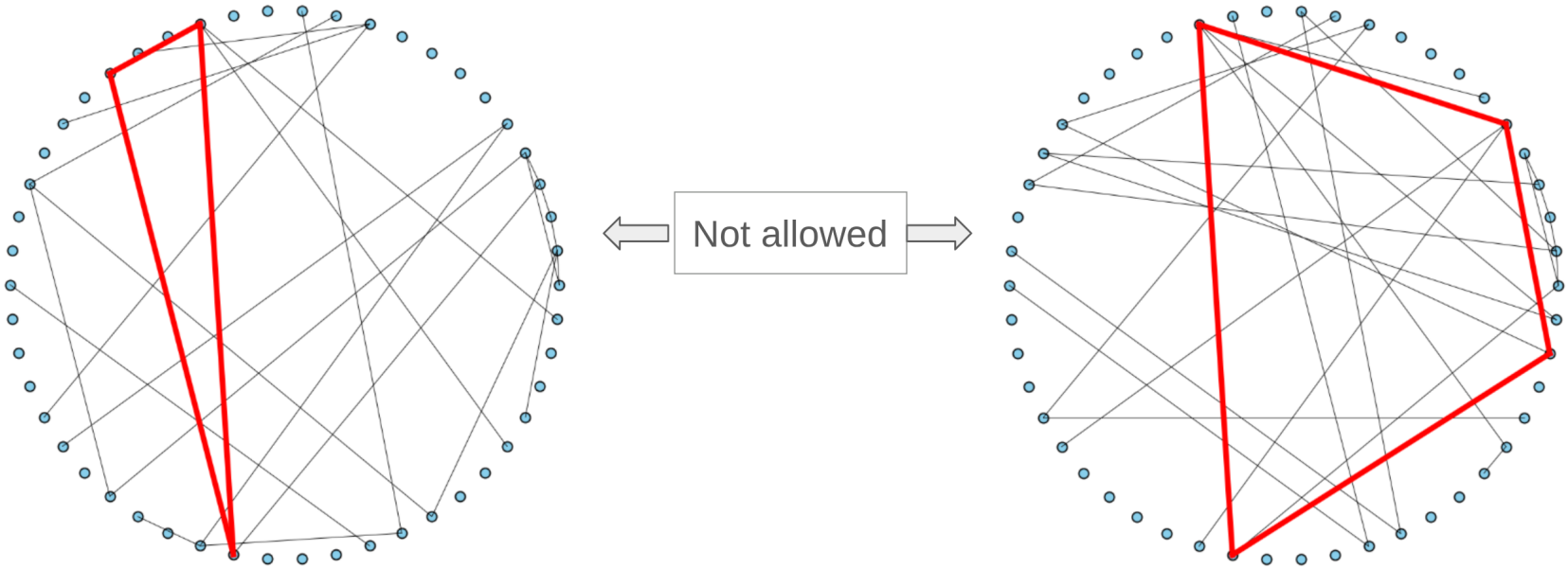
Local Search Example: Complex Program Space



FunSearch example

Problem

At most how many edges can a graph have, that has 50 vertices, and no triangles or cycles of length 4?



FunSearch example

Problem

At most how many edges can a graph have, that has 50 vertices, and no triangles or cycles of length 4?

We could try to solve this using standard methods, or RL

- Create graphs edge by edge
- Receive +1 point for each new edge included
- 100 point penalty for every triangle or 4-cycle created
- Try to maximize the score

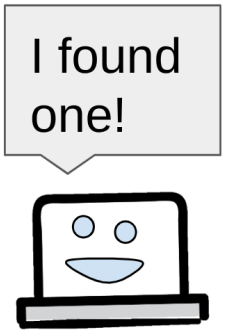
Turns out, this is difficult.

Most likely outcome: we find graphs with 160-170 edges

There is only one graph that has 175 edges, out of about 10^{300} graphs!

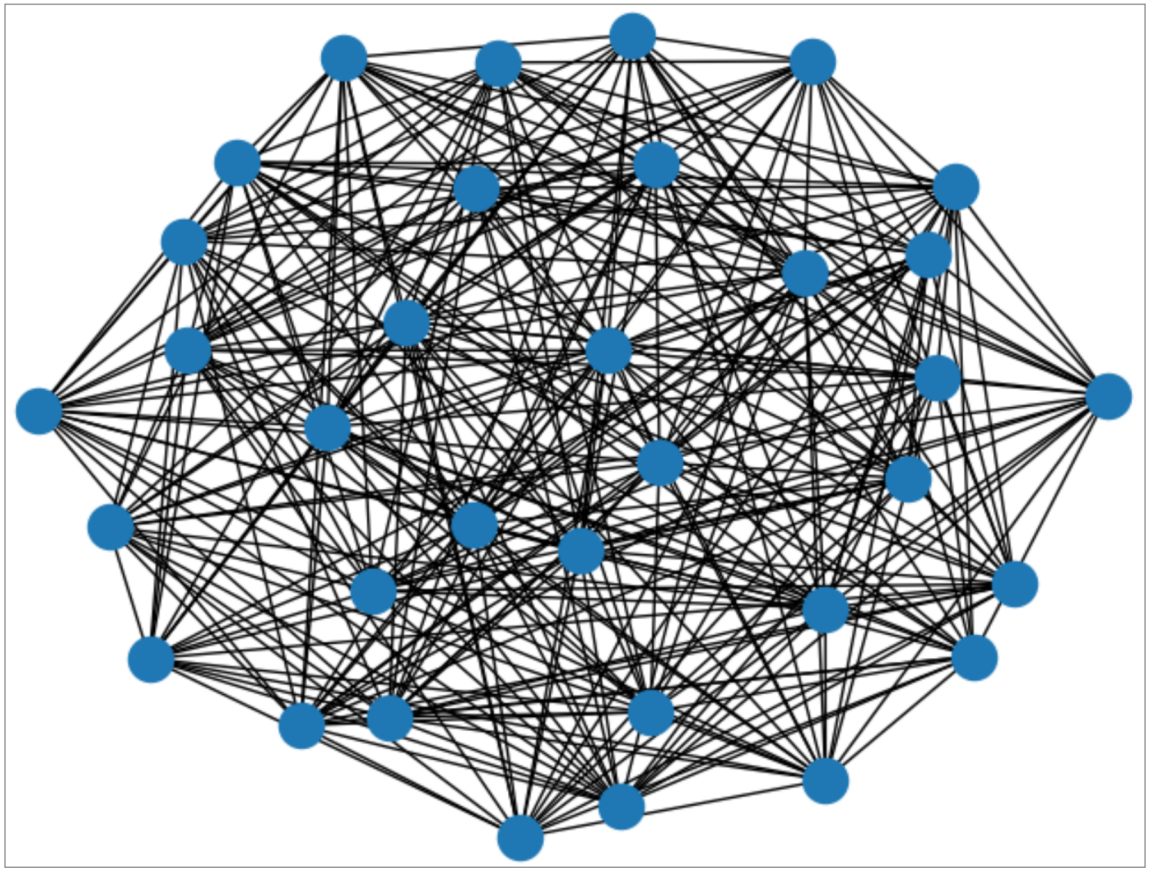
FunSearch example

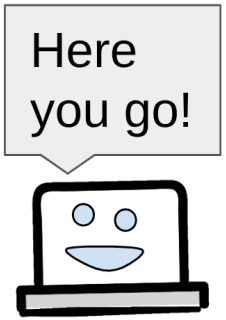
Best case scenario:



FunSearch example

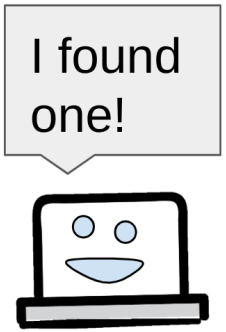
Best case scenario:





FunSearch example

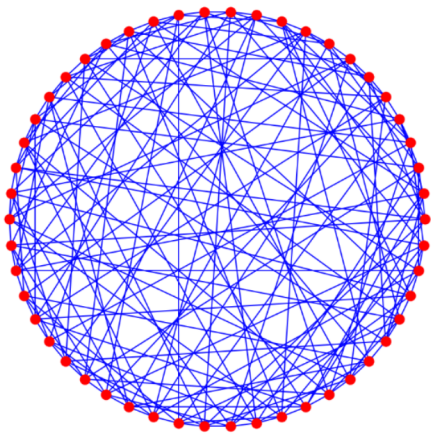
Solving with FunSearch:



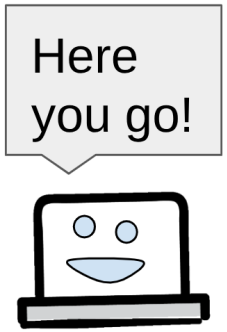
FunSearch example

Solving with FunSearch:

```python
def construct_graph():
  graph = set()
  graph.add((0, 2))
  for j in range(4, n):
    for i in range(j - 1, -1, -1):
      if count_triangles_and_squares(graph | {(i, j)}, n) == 0:
        graph.add((i, j))
  return graph
```



(This is definitely not the most informative or symmetric description, but it works!)



FunSearch example

Sometimes it figures out the problem it was asked to do

```python
import networkx as nx

def create_graph():
  g = nx.hoffman_singleton_graph()
  return g
```

FunSearch example

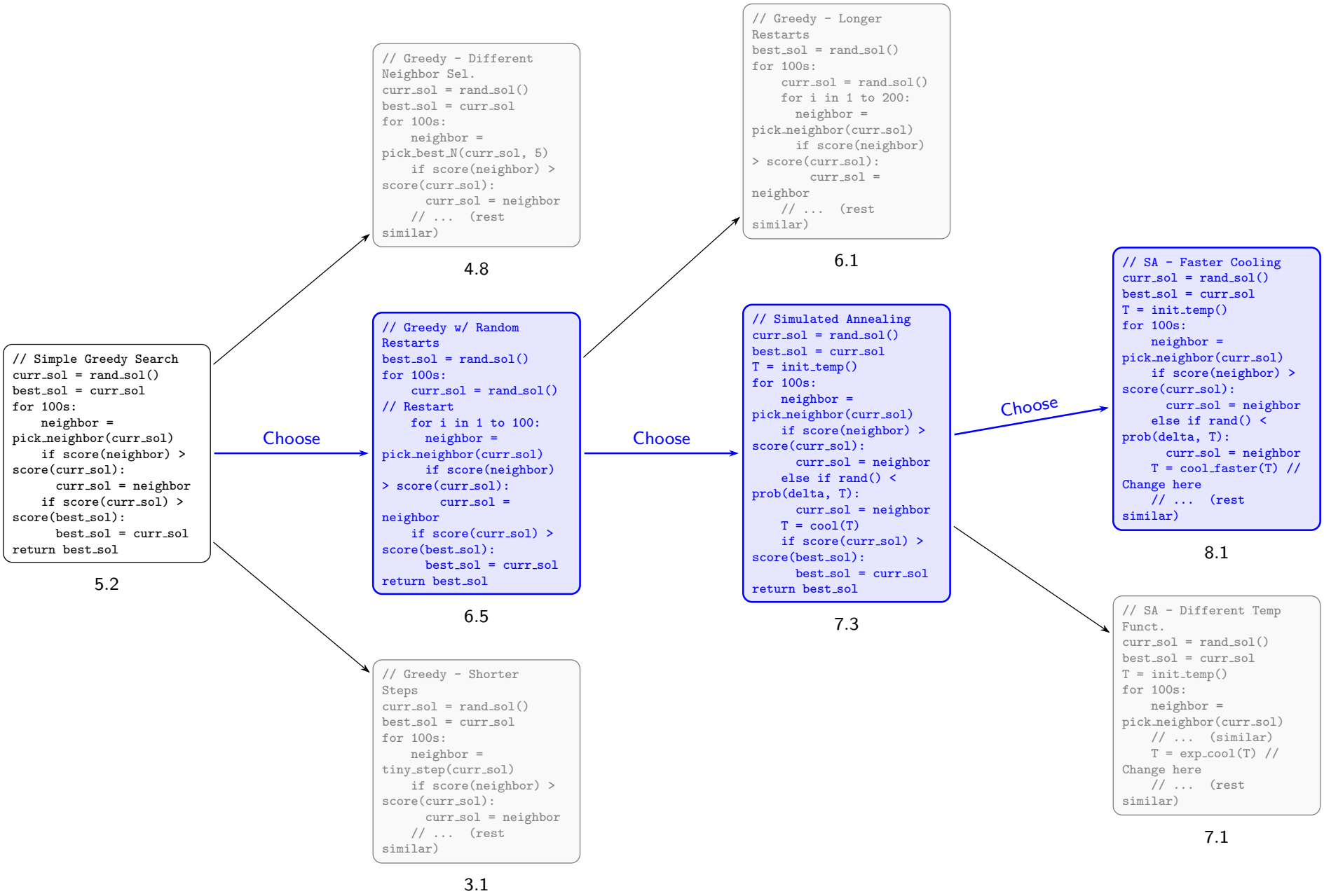
…and sometimes it arrives at a solution without ever really understanding the problem, just by hard work and many iterations

```python
def create_graph():
  g = nx.Graph()
  g.add_nodes_from(range(50))

  # Create 10 disjoint 5-cycles
  for i in range(0, 50, 5):
      for j in range(5):
          g.add_edge(i + j, i + (j + 1) % 5)


  for _ in range(50):

    all_nodes_full = True
    for node in g.nodes:
        if g.degree(node) < 7:
          all_nodes_full = False
          break
    if all_nodes_full:
      break

    nodes_with_low_degree = sorted(
        [node for node in g.nodes if g.degree(node) < 7], key=g.degree
    )

    for start_node in nodes_with_low_degree:
          for end_node in g.nodes:
            if start_node != end_node and g.degree(end_node) < 7:
                if not g.has_edge(start_node, end_node):

                    has_small_cycle = False
                    # Check for 4 cycle
                    for n1 in g.neighbors(start_node):
                        for n2 in g.neighbors(end_node):
                            if n1 != end_node and n2 != start_node and \
                              g.has_edge(n1, n2):
                                has_small_cycle = True
                                break
                        if has_small_cycle:
                            break
                    if has_small_cycle:
                         continue

                    # Check triangle
                    has_small_cycle = False
                    for neighbor in g.neighbors(start_node):
                        if g.has_edge(neighbor, end_node):
                            has_small_cycle = True
                            break
                    if has_small_cycle:
                        continue
                    g.add_edge(start_node, end_node)
                    break
  return g
```

The best of both worlds

- Previous example: find a python code that generates a good construction directly
- Examples earlier in the talk: what kind of codebase can find good constructions quickly?
- Let's combine both: we can try to find a search function that finds the best possible construction within a fixed time limit
- "Local search in the space of search functions"
- This will find the best heuristic search function for your problem
- We can expect better results (can check thousands of constructions per LLM call), but less interpretability.

Local Search Example: Search Program Space



Strange heuristics work best

A lesson I learned in the past year: for many math problems I tried, there was some strange heuristic algorithms that worked unreasonably well.

Example: What is the largest C for which one has

$$\max_{-1/2 \leq t \leq 1/2} \int_{\mathbb{R}} f(t-x) f(x)\, dx \geq C \left(\int_{-1/4}^{1/4} f(x)\, dx \right)^2$$

for all non-negative $f : \mathbb{R} \to \mathbb{R}$?

An expert spent a week optimizing this problem with clever techniques.

AlphaEvolve found that something called "cubic backtracking" worked very well and we could beat their result in half an hour

I am sure the expert would have eventually found this heuristic. But it would have been a major pain, and not worth the effort.

Evolving a chain of search functions

One more trick: when evaluating a search function, we can initialize the search at the best construction we have found so far.

The result is a chain of search heuristics, evolved automatically, that when applied one after another, yields a good construction.

This turns out to be a good black box optimizer for many math problems (but interpretability goes out the window)

AlphaEvolve examples

Let C_1 denote the largest constant for which one has

$$\max_{-1/2\le t\le 1/2} \int_{\mathbb{R}} f(t-x)f(x)\,dx \ge C_1 \left(\int_{-1/4}^{1/4} f(x)\,dx\right)^2 \qquad (1.1)$$

for all non-negative $f : \mathbb{R} \to \mathbb{R}$. This problem arises in additive combinatorics, relating to the size of Sidon sets. It is currently known that

$$1.28 \le C_1 \le 1.50992$$

1.5099 → 1.5056

Let C_1' be the best constant for which one has

$$\|f * f\|_2^2 \le C_1' \|f * f\|_1 \|f * f\|_\infty$$

for non-negative $f : \mathbb{R} \to \mathbb{R}$. It is known that

$$0.88922 \le C_1' \le 1$$

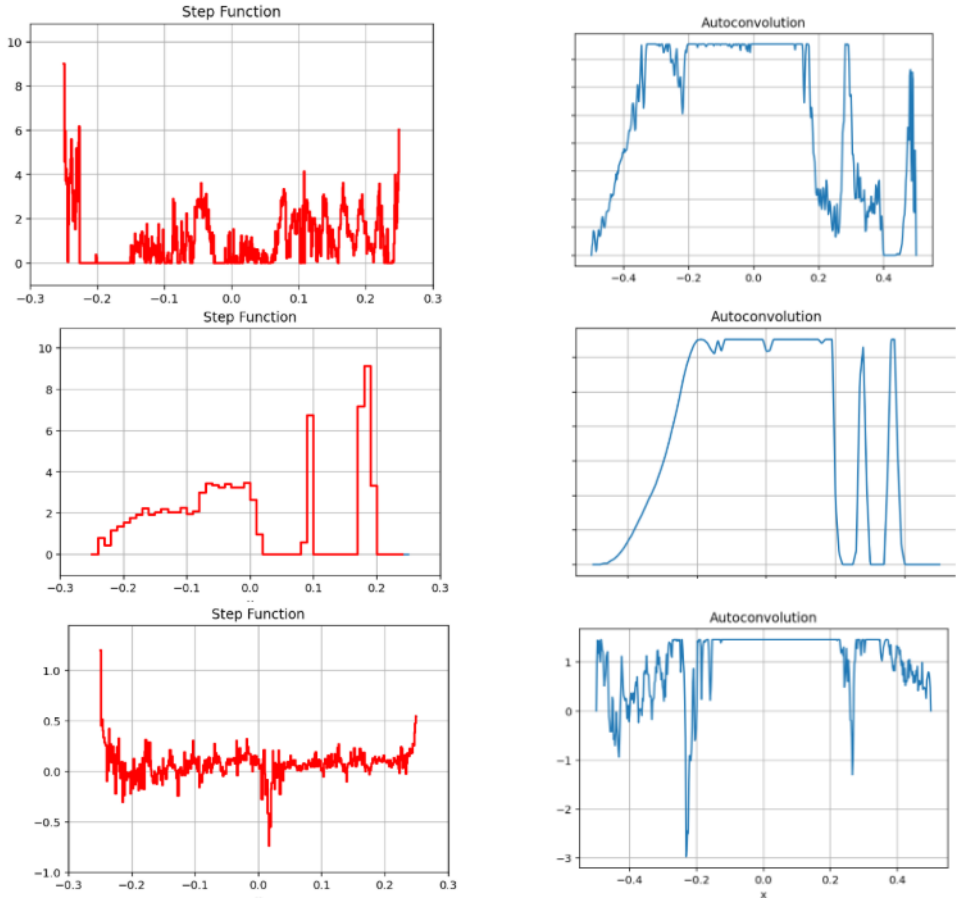
0.889 → 0.896

Now let C_1'' be the best constant for which one has

$$\max_{-1/2\le t\le 1/2} \left| \int_{\mathbb{R}} f(t-x)f(x)\,dx \right| \ge C_1'' \left(\int_{-1/4}^{1/4} f(x)\,dx\right)^2$$

for all $f : [-1/4, 1/4] \to \mathbb{R}$, which we now allow to take both negative and positive values. Then $C_1'' \le C_1$. Here, there is a better example that gives a new upper bound on C_1'', namely $C_1'' \le 1.45810$.

1.458 → 1.455



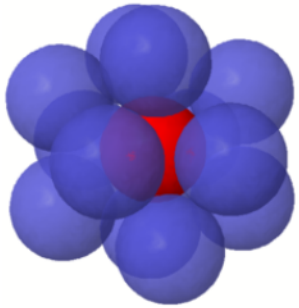
B. Georgiev, J. Gómez-Serrano, T. Tao, A. Z. Wagner: Mathematical exploration and discovery at scale, 2025

Novikov et al.: AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025

AlphaEvolve examples

Kissing numbers: how many disjoint unit spheres can touch a unit sphere simultaneously?

Dimension 11: 582 ⟶ 583



2. UNCERTAINTY PRINCIPLES

Given a function $f \in L^1(\mathbb{R})$, define the Fourier transform $\hat{f}(\xi) := \int_{\mathbb{R}} f(x)e^{-2\pi ix\xi}\,dx$ and

$$A(f) := \inf\{r > 0 : f(x) \geq 0 \text{ for all } |x| \geq r\}.$$

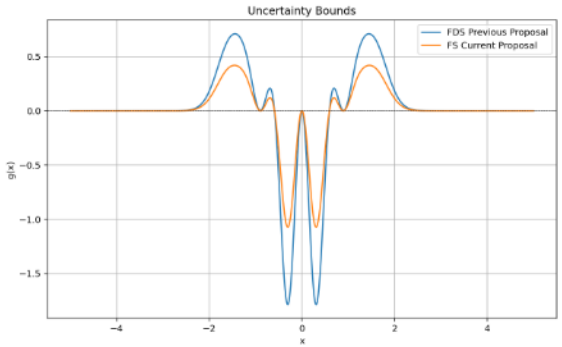
Let $C_{2.1}$ be the largest constant for which one has

$$A(f)A(\hat{f}) \geq C_{2.1}$$

for all even f with $f(0), \hat{f}(0) < 0$. It is known [53] that

$$0.2025 \leq C_{2.1} \leq 0.353.$$

0.3522 ⟶ 0.3520



B. Georgiev, J. Gómez-Serrano, T. Tao, A. Z. Wagner: Mathematical exploration and discovery at scale, 2025

Novikov et al.: AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025

The isosceles triangle problem

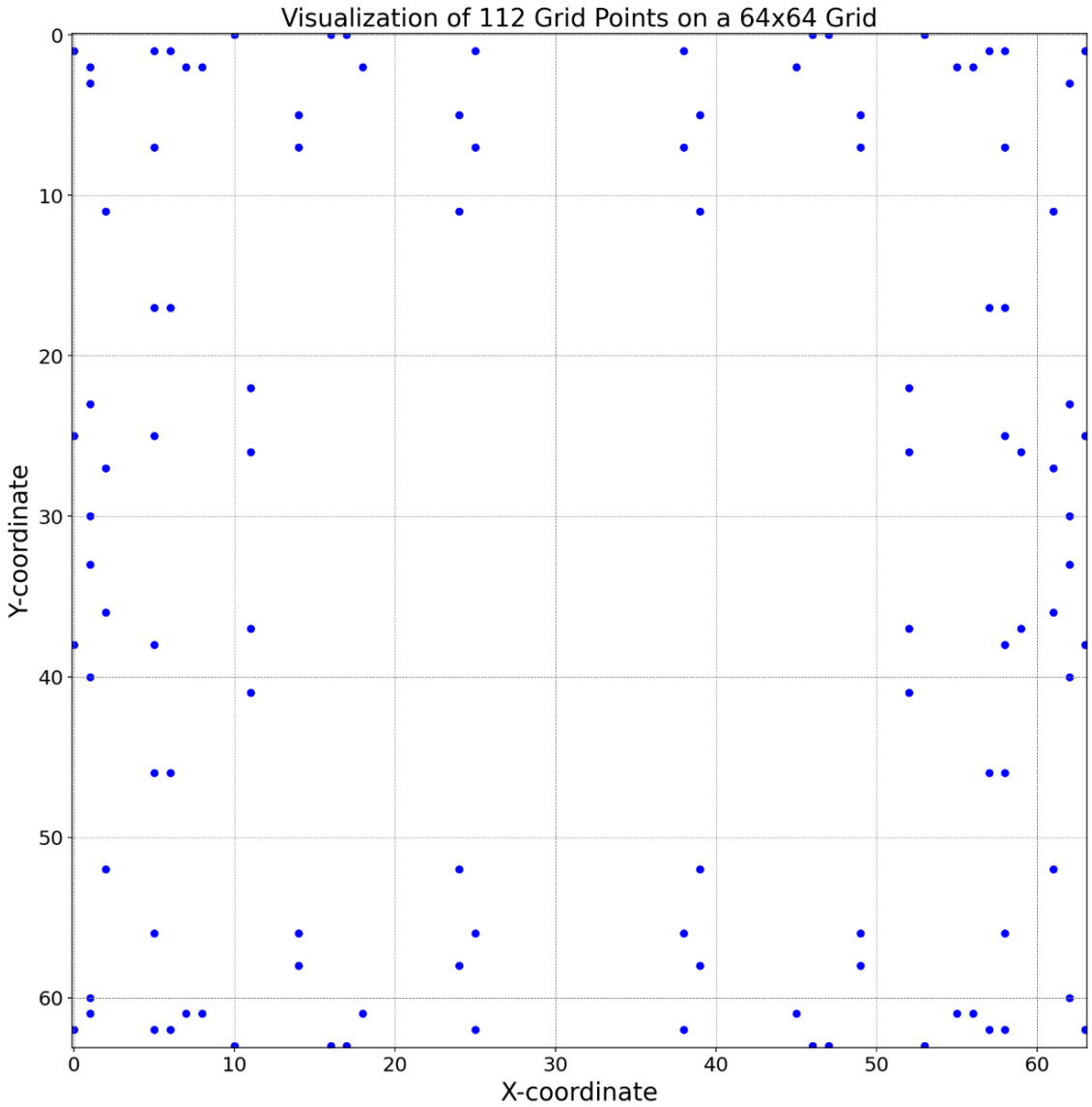
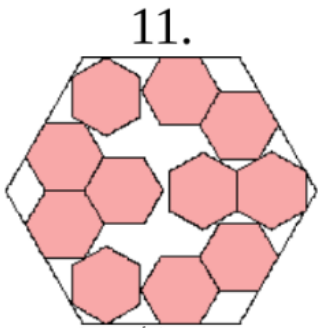


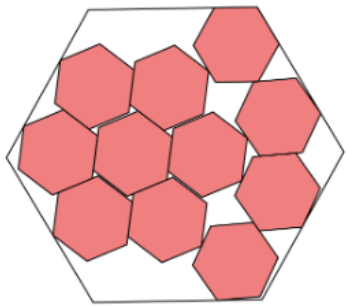
Figure: $f(64) \geq 112$!

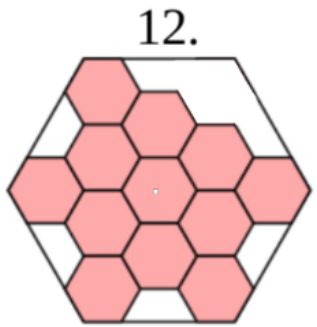
AlphaEvolve examples

What is the smallest hexagon one can fit n unit hexagons into?

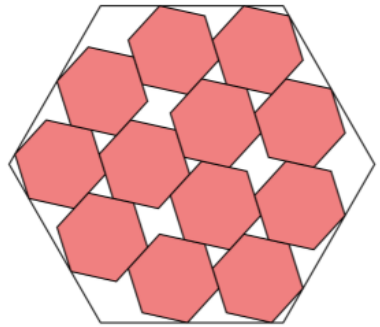


$s = 5(3 + \sqrt{3})/6 = 3.943+$
Found by Maurizio Morandi
in April 2015.





$s = 4$
Trivial.



None of these results are impossible to obtain with standard tools. They just substantially reduce the amount of effort needed.

B. Georgiev, J. Gómez-Serrano, T. Tao, A. Z. Wagner: Mathematical exploration and discovery at scale, 2025

Novikov et al.: AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025

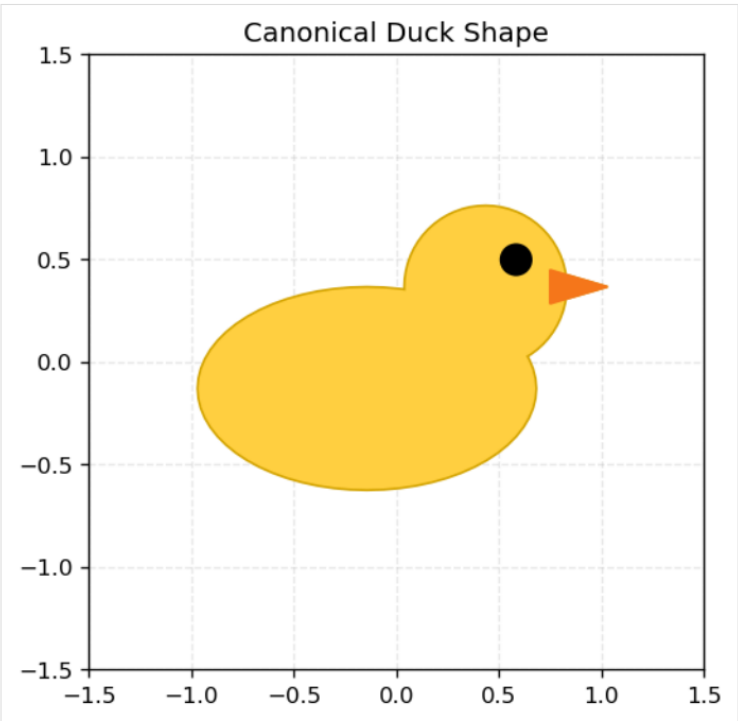
Less effort to attack a wide variety of problems

What is the smallest pond one can fit n unit rubber ducks into?

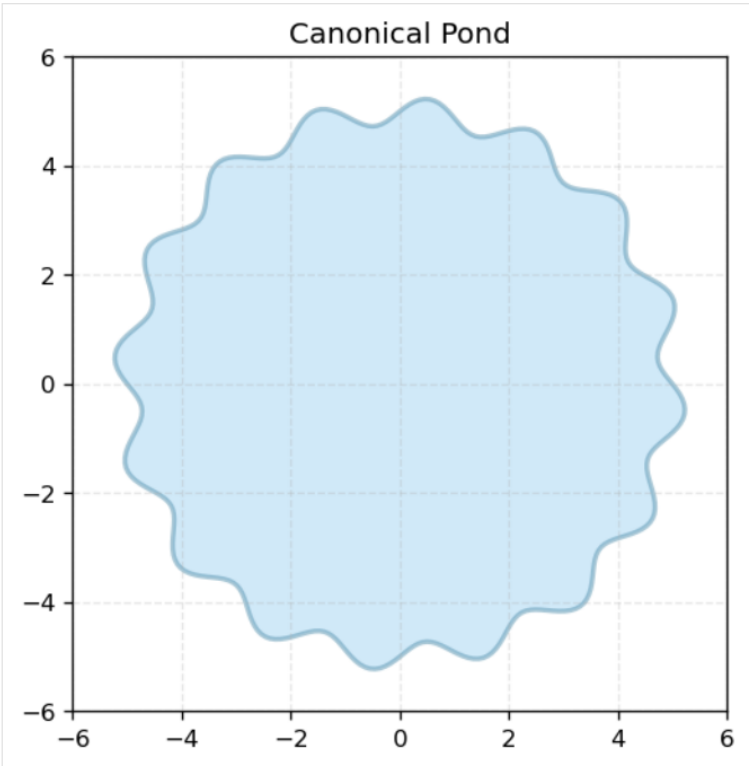
Less effort to attack a wide variety of problems

What is the smallest pond one can fit n unit rubber ducks into?

Canonical Duck

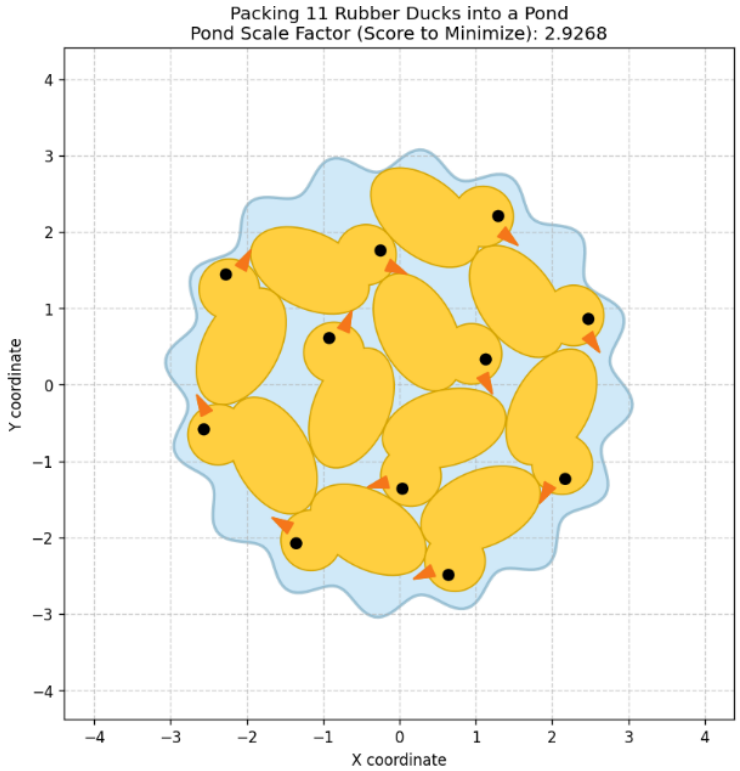


Empty Pond



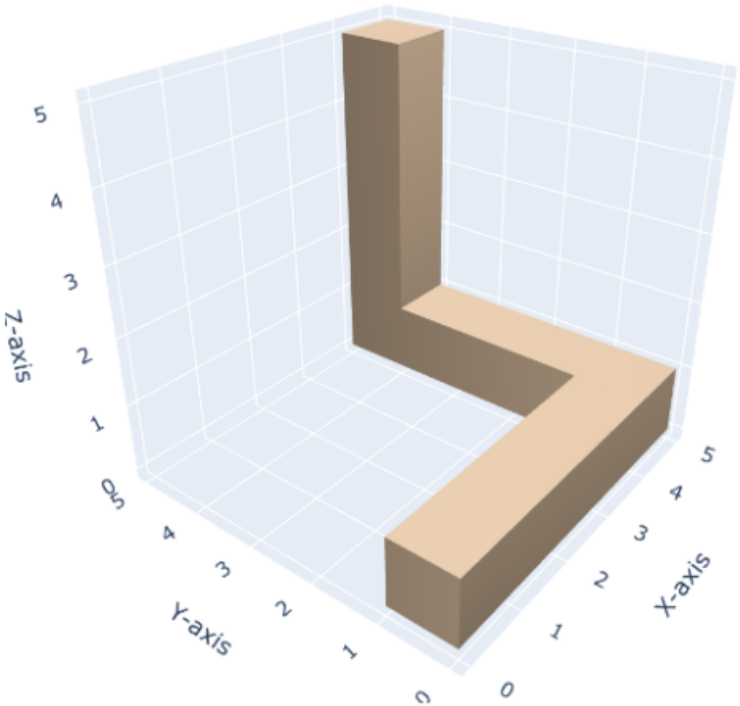
Less effort to attack a wide variety of problems

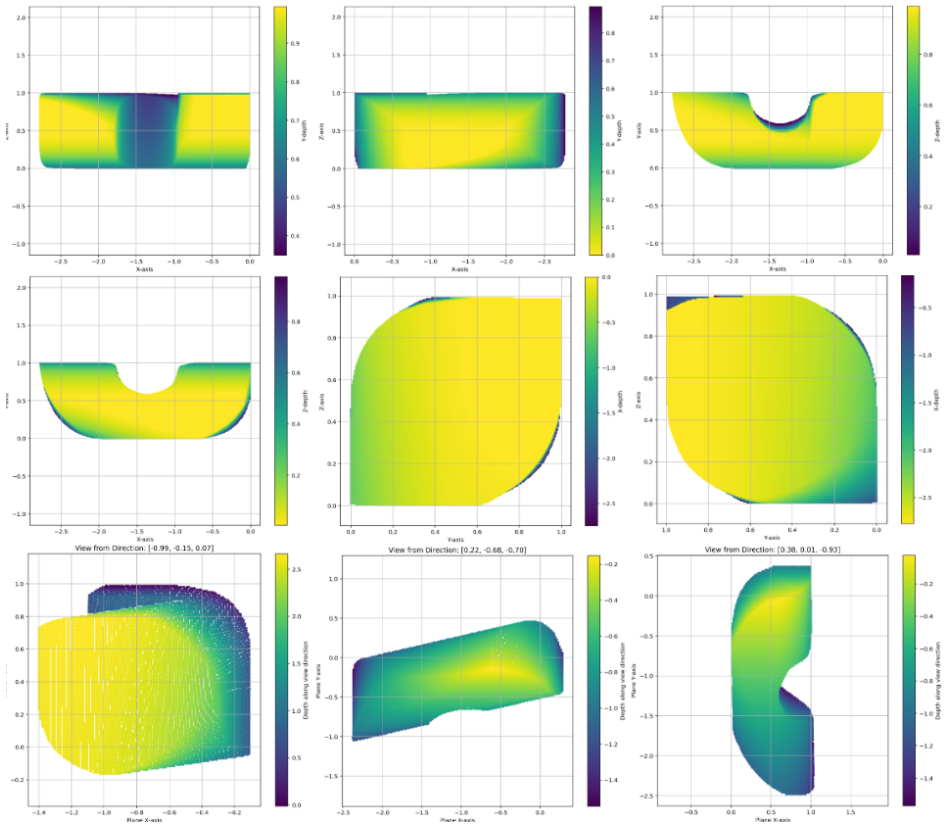
What is the smallest pond one can fit n unit rubber ducks into?



3D sofa

What is the largest sofa that can be moved through this 3D corridor?





Code with LLM calls in it

What if we let AlphaEvolve search in the space of python programs that are allowed to call LLMs and e.g. execute the code they return?

```
# Request a "quantum perturbation" function from the LLM at the start
quantum_perturbation = None
prompt_quantum = f"""
  You are an expert in theoretical physics and optimization.
  Propose a "quantum-inspired" perturbation strategy for points in 2D space within a unit square.
  The strategy should be implemented as a Python function `quantum_perturbation(current_placements, perturbat
  `current_placements` is a list of floats [x1, y1, x2, y2, ...]. `perturbation_scale` is a float.
  The function should return a new list of placements after applying the "quantum" perturbation.
  Make it conceptually crazy and mathematically plausible (even if it's just a metaphor).
  For example, consider concepts like entanglement, tunneling, or superposition.
  Ensure points are clipped to [0, 1].
  Do not print anything else besides the python code for the function.
  """
code_quantum = call_llm(prompt_quantum)
try:
    temp_globals_quantum = {'np': np, 'random': random, 'math': math, 'quantum_perturbation': None}
    exec(code_quantum, temp_globals_quantum)
    quantum_perturbation = temp_globals_quantum['quantum_perturbation']
    logging.info("Successfully loaded quantum_perturbation function.")
except Exception as e:
    logging.error('Failed to execute LLM-generated quantum_perturbation code: %s', e)
    quantum_perturbation = None # Fallback to default if generation fails

# Request a "metamorphic optimizer" function from the LLM
metamorphic_optimizer = None
prompt_metamorphic = f"""
  You are an expert in software engineering and optimization, particularly in metamorphic testing and evoluti
  Propose a "metamorphic optimizer" function for point placements.
  The function should be named `metamorphic_optimizer(initial_placements, pc, current_best_score, time_budget
  `initial_placements` is a list of floats [x1, y1, x2, y2, ...]. `pc` is `point_count`.
  `current_best_score` is the score of `initial_placements`. `time_budget` is a float for max runtime.
  The function should return `(best_placements_found, best_score_found)` within the given `time_budget`.
  Instead of point-wise perturbations, it should apply "metamorphic transformations" to the entire set of poi
```

Code with LLM calls in it

This is useful for solving logic puzzles, to some extent!

We have three guards in front of three doors. The guards are, in some order, an angel (always tells the truth), the devil (always lies), and ... . The prizes behind the doors are .... You can ask two yes/no questions ...

To evaluate how good a program is, we just play it out in all possible guard-door combinations! (The guards are also LLMs)

Human + AI is still the best

Currently, human expert + AI combination is still the best

1. For many problems, after we published a new constructions, others
   - were inspired by the construction and improved it, or
   - noted some obvious things AlphaEvolve missed, and improved the construction.

   AlphaEvolve often finds some crazy constructions, but then misses things that are obvious to us.

2. An expert using AlphaEvolve will always produce better results than a non-expert: the advice we put in the prompt matters, it guides the search.

Comparing battle plans with AlphaEvolve

Expert advice: how much does this help and how to do it best?

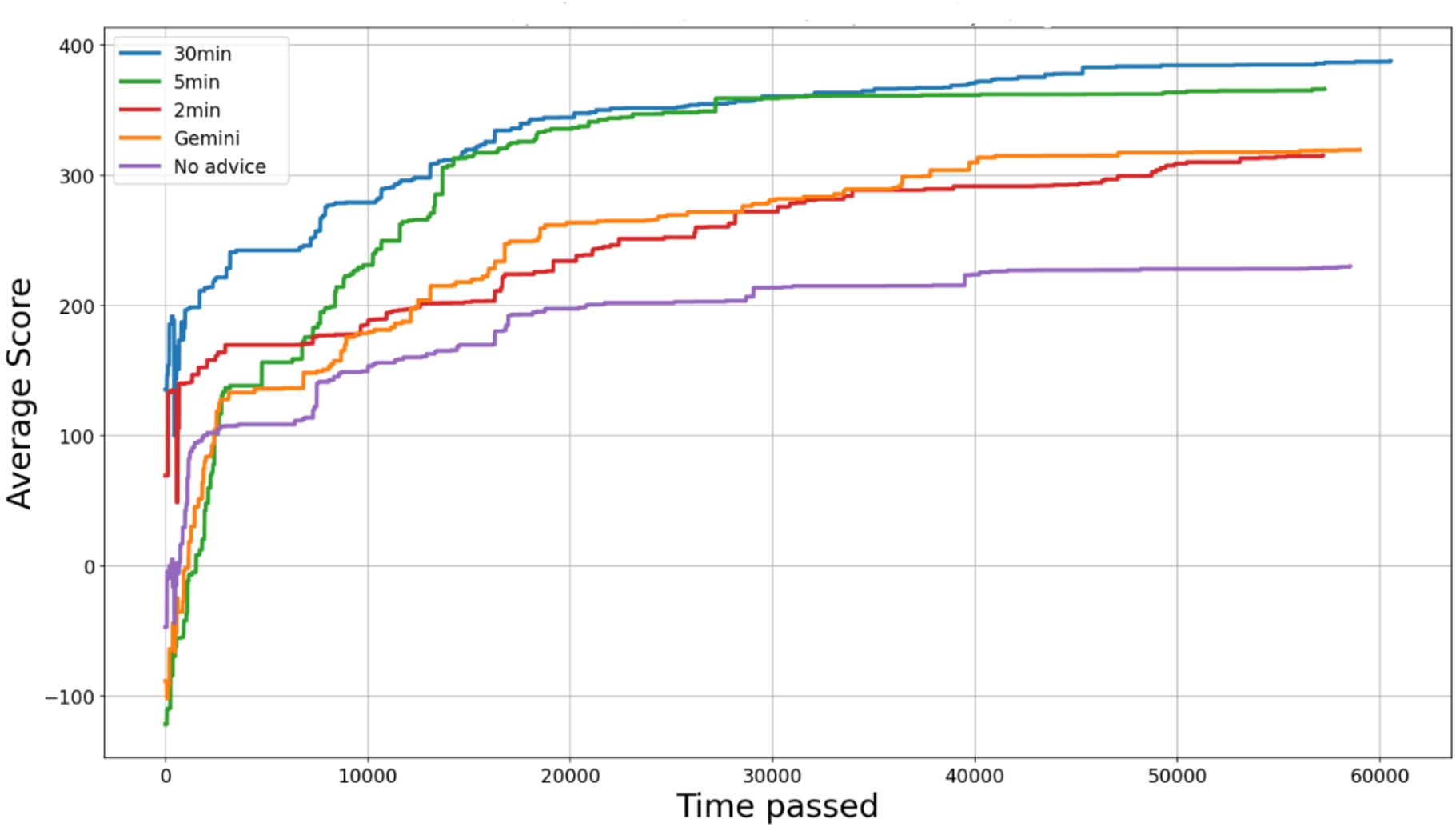
We came up with the following arbitrary problem to test this with:

Problem

Find a graph with 50 vertices, that maximises the quantity
$\#edges + 5\ (\#triangles - (\#4cycles - 5)^2)$

This problem is completely arbitrary, hopeless to solve by hand. We asked three of our colleagues to spend 2/5/30 minutes on this problem, and write down some general advice that we will give to AlphaEvolve.

Comparing battle plans with AlphaEvolve

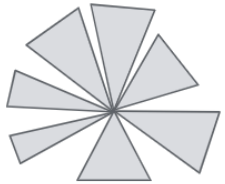


Comparing battle plans with AlphaEvolve

Each experiment retained the original advice's characteristics, but evolved them into a stronger version

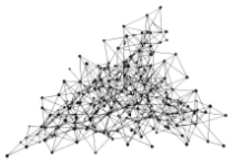
2min advice:

```
having vertex number 21 be connected to vertices 22 through 50, and
then pair up these vertices to form several triangles that are all"
connected to vertex 21 and have no other vertices in common."
```



Evolved 2min advice:

```
def build_triangle_tree(graph, root, used_nodes, degree_limit=6):

  eligible_nodes = []
  for next_node in range(root+1, 50):
      if next_node not in used_nodes and graph.degree(next_node) <
```



```
def build_fan_triangles(graph, root, used_nodes, degree_limit):
    eligible_nodes= []
    for k in range(21, 50):
        if graph.degree(k) < degree_limit and k not in used_nodes:
            eligible_nodes.append(k)
```

AlphaEvolve takes whatever idea you give to it, and tries to squeeze as much out of it as possible. This gives us an easy way to compare different strategies

What problems are a good fit for these tools?

If we put a thousand eager undergraduates in a room and give them this problem, how likely is it that they will succeed?

- AlphaEvolve is like having 1000 eager undergraduates in a room, who are excited to work on your problem
- They will read every possible paper they think is related, and try to combine the ideas in them in all sorts of crazy ways, whether they understand them or not
- They will zoom in on any idea that gets them a high score

If a problem can be solved this way, then AlphaEvolve will do well on it. If the problem needs genuine new ideas, AlphaEvolve will probably not find it.

Papers mentioned in the talk

- Wagner, A. Z. (2021). Constructions in combinatorics via neural networks. arXiv preprint arXiv:2104.14516.
- Charton, F., Ellenberg, J. S., Wagner, A. Z., & Williamson, G. (2024). Patternboost: Constructions in mathematics with a little help from AI. arXiv preprint arXiv:2411.00566.
- Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M.P., Dupont, E., Ruiz, F.J., Ellenberg, J.S., Wang, P., Fawzi, O. and Kohli, P., 2024. Mathematical discoveries from program search with large language models. Nature, 625(7995), pp.468-475.
- Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.S., Wagner, A.Z., Shirobokov, S., Kozlovskii, B., Ruiz, F.J., Mehrabian, A. and Kumar, M.P., 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. arXiv preprint arXiv:2506.13131.
- Georgiev, B., Gómez-Serrano, J., Tao, T., & Wagner, A. Z. (2025). Mathematical exploration and discovery at scale. arXiv preprint arXiv:2511.02864.