FunSearch

Evolving programs through LLMs

Bernardino Romera-Paredes

PPSN 2024

Discover new and verifiably correct algorithms outperforming SOTA results in impactful problems

Previously: AlphaTensor

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{pmatrix}$$

$$\begin{aligned}
h_1 &= a_{1,1}b_{1,3} \\
h_2 &= (a_{1,1} + a_{3,1} + a_{3,3})(b_{1,1} + b_{3,1} + b_{3,3}) \\
h_3 &= (a_{1,1} + a_{3,1} + a_{3,4})(b_{1,2} + b_{4,2} + b_{4,3}) \\
h_4 &= (a_{1,3} + a_{2,1} + a_{2,3})(b_{1,3} + b_{1,4} + b_{3,4}) \\
h_5 &= (a_{1,1} + a_{3,1})(b_{1,1} + b_{1,2} + b_{1,3} + b_{3,1} + b_{3,3} + b_{4,2} + b_{4,3}) \\
h_6 &= (a_{1,3} + a_{2,3})(b_{1,3} + b_{1,4} + b_{3,2} + b_{3,3} + b_{3,4} + b_{4,2} + b_{4,3}) \\
h_7 &= (a_{1,4} + a_{4,3} + a_{4,4})(b_{3,1} + b_{3,3} + b_{4,1}) \\
h_8 &= (a_{1,4} + a_{4,1} + a_{4,4})(b_{1,3} + b_{1,4} + b_{4,4}) \\
h_9 &= (a_{1,3} + a_{2,3} + a_{2,4})(b_{3,2} + b_{4,2} + b_{4,3}) \\
h_{10} &= (a_{1,4} + a_{4,4})(b_{1,3} + b_{1,4} + b_{3,1} + b_{3,3} + b_{4,1} + b_{4,3} + b_{4,4}) \\
h_{11} &= a_{3,3}(b_{1,1} + b_{2,2} + b_{2,3} + b_{3,1} + b_{3,2}) \\
h_{12} &= (a_{1,2} + a_{3,2} + a_{3,3})(b_{2,2} + b_{2,3} + b_{3,2}) \\
h_{13} &= a_{3,4}(b_{1,2} + b_{2,1} + b_{2,3} + b_{4,1} + b_{4,2}) \\
h_{14} &= (a_{1,2} + a_{3,2})(b_{2,1} + b_{2,2} + b_{2,3} + b_{3,2} + b_{4,1}) \\
h_{15} &= (a_{1,2} + a_{3,2} + a_{3,4})(b_{2,1} + b_{2,3} + b_{4,1}) \\
h_{16} &= a_{2,1}(b_{1,2} + b_{1,4} + b_{2,2} + b_{2,3} + b_{3,4}) \\
h_{17} &= (a_{1,2} + a_{2,1} + a_{2,2})(b_{1,2} + b_{2,2} + b_{2,3}) \\
h_{18} &= (a_{1,2} + a_{2,2})(b_{1,2} + b_{2,2} + b_{2,3} + b_{2,4} + b_{4,4}) \\
h_{19} &= a_{2,4}(b_{2,3} + b_{2,4} + b_{3,2} + b_{4,2} + b_{4,4}) \\
h_{20} &= (a_{1,2} + a_{2,3} + a_{2,4} + a_{3,2} + a_{3,3})b_{3,2} \\
h_{21} &= (a_{1,2} + a_{2,2} + a_{2,4})(b_{2,3} + b_{2,4} + b_{4,4}) \\
h_{22} &= a_{4,3}(b_{2,3} + b_{2,4} + b_{3,1} + b_{3,4} + b_{4,1}) \\
h_{23} &= (a_{1,1} + a_{1,3} + a_{1,4} + a_{2,3} + a_{2,4} + a_{3,1} + a_{3,4})(b_{4,2} + b_{4,3}) \\
h_{24} &= (a_{1,2} + a_{4,2} + a_{4,3})(b_{2,3} + b_{2,4} + b_{3,4}) \\
h_{25} &= (a_{1,2} + a_{4,2})(b_{1,1} + b_{2,1} + b_{2,3} + b_{2,4} + b_{3,4}) \\
h_{26} &= (a_{1,2} + a_{4,1} + a_{4,2})(b_{1,1} + b_{2,1} + b_{2,3}) \\
h_{27} &= a_{1,4}b_{4,3} \\
h_{28} &= (a_{1,2} + a_{2,1} + a_{2,2} + a_{3,1} + a_{3,4})b_{1,2} \\
h_{29} &= (a_{1,2} + a_{2,1} + a_{2,3} + a_{4,2} + a_{4,3})b_{3,4} \\
h_{30} &= (a_{1,2} + a_{3,1} + a_{3,3} + a_{4,1} + a_{4,2})b_{1,1} \\
h_{31} &= a_{4,1}(b_{1,1} + b_{1,4} + b_{2,1} + b_{2,3} + b_{4,4})
\end{aligned}$$

$$\begin{aligned}
h_{32} &= (a_{1,2} + a_{3,2} + a_{3,4} + a_{4,3} + a_{4,4})b_{4,1} \\
h_{33} &= (a_{1,2} + a_{2,2} + a_{2,4} + a_{4,1} + a_{4,4})b_{4,4} \\
h_{34} &= (a_{2,1} + a_{3,1} + a_{4,1})(b_{1,1} + b_{1,2} + b_{1,4}) \\
h_{35} &= (a_{1,2} + a_{2,1} + a_{2,2} + a_{3,2} + a_{3,3})(b_{2,2} + b_{2,3}) \\
h_{36} &= (a_{1,2} + a_{2,4} + a_{3,2} + a_{4,3})(b_{2,3} + b_{2,4} + b_{3,2} + b_{4,1}) \\
h_{37} &= (a_{1,2} + a_{2,1} + a_{3,3} + a_{4,2})(b_{1,1} + b_{2,2} + b_{2,3} + b_{3,4}) \\
h_{38} &= (a_{2,2} + a_{3,2} + a_{4,2})(b_{2,1} + b_{2,2} + b_{2,4}) \\
h_{39} &= a_{1,2}b_{2,3} \\
h_{40} &= a_{1,3}b_{3,3} \\
h_{41} &= (a_{1,1} + a_{1,3} + a_{1,4} + a_{2,1} + a_{2,3} + a_{4,1} + a_{4,4})(b_{1,3} + b_{1,4}) \\
h_{42} &= (a_{1,2} + a_{3,2} + a_{3,4} + a_{4,1} + a_{4,2})(b_{2,1} + b_{2,3}) \\
h_{43} &= (a_{2,4} + a_{3,4} + a_{4,4})(b_{4,1} + b_{4,2} + b_{4,4}) \\
h_{44} &= (a_{2,3} + a_{3,3} + a_{4,3})(b_{3,1} + b_{3,2} + b_{3,4}) \\
h_{45} &= (a_{1,1} + a_{1,3} + a_{1,4} + a_{3,1} + a_{3,3} + a_{4,3} + a_{4,4})(b_{3,1} + b_{3,3}) \\
h_{46} &= (a_{1,2} + a_{2,2} + a_{3,4} + a_{4,1})(b_{1,2} + b_{2,1} + b_{2,3} + b_{4,4}) \\
h_{47} &= (a_{1,2} + a_{2,2} + a_{2,4} + a_{4,2} + a_{4,3})(b_{2,3} + b_{2,4}) \\
c_{1,1} &= h_{15} + h_{26} + h_2 + h_{30} + h_{32} + h_{39} + h_{40} + h_{42} + h_{45} + h_7 \\
c_{2,1} &= h_{11} + h_{12} + h_{14} + h_{20} + h_{22} + h_{24} + h_{25} + h_{29} + h_{35} + h_{36} + h_{37} + h_{38} + h_{44} + h_{47} \\
c_{3,1} &= h_{11} + h_{12} + h_{14} + h_{15} + h_{26} + h_{30} + h_{39} + h_{42} \\
c_{4,1} &= h_{15} + h_{22} + h_{24} + h_{25} + h_{26} + h_{32} + h_{39} + h_{42} \\
c_{1,2} &= h_{12} + h_{17} + h_{20} + h_{23} + h_{27} + h_{28} + h_{35} + h_{39} + h_3 + h_9 \\
c_{2,2} &= h_{12} + h_{17} + h_{18} + h_{19} + h_{20} + h_{21} + h_{35} + h_{39} \\
c_{3,2} &= h_{12} + h_{13} + h_{14} + h_{15} + h_{17} + h_{28} + h_{35} + h_{39} \\
c_{4,2} &= h_{13} + h_{14} + h_{15} + h_{18} + h_{19} + h_{21} + h_{32} + h_{33} + h_{36} + h_{38} + h_{42} + h_{43} + h_{46} + h_{47} \\
c_{1,3} &= h_1 + h_{27} + h_{39} + h_{40} \\
c_{2,3} &= h_{16} + h_{17} + h_{18} + h_{19} + h_{21} + h_{39} + h_{40} + h_4 + h_6 + h_9 \\
c_{3,3} &= h_{11} + h_{12} + h_{13} + h_{14} + h_{15} + h_1 + h_2 + h_{39} + h_3 + h_5 \\
c_{4,3} &= h_{10} + h_{22} + h_{24} + h_{25} + h_{26} + h_{27} + h_{31} + h_{39} + h_7 + h_8 \\
c_{1,4} &= h_1 + h_{21} + h_{24} + h_{29} + h_{33} + h_{39} + h_{41} + h_{47} + h_4 + h_8 \\
c_{2,4} &= h_{16} + h_{17} + h_{18} + h_{21} + h_{24} + h_{29} + h_{39} + h_{47} \\
c_{3,4} &= h_{16} + h_{17} + h_{18} + h_{25} + h_{26} + h_{28} + h_{30} + h_{31} + h_{34} + h_{35} + h_{37} + h_{38} + h_{42} + h_{46} \\
c_{4,4} &= h_{21} + h_{24} + h_{25} + h_{26} + h_{31} + h_{33} + h_{39} + h_{47}
\end{aligned}$$

Why searching in the program/function space?

Generality

- **No Structure**

  Does not rely on existing structure for any specific problem.

- **Universal Interface**

  Can be applied to any fundamental problems with solutions in programming language space

Interpretability

- **Code Output**

  Output is a computer program which is *interpretable* plus much *faster* and *safer* to execute than a black box neural net.

Scalability

- **Concise Representation**

  Scale to much larger instances of a problem as opposed to traditional search because search space scales better with problem dimensions.

Article | Open access | Published: 14 December 2023

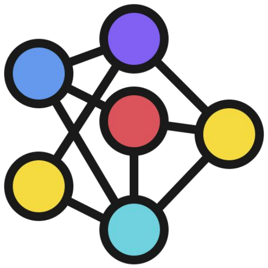
Mathematical discoveries from program search with large language models

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli & Alhussein Fawzi

Nature **625**, 468–475 (2024) | Cite this article

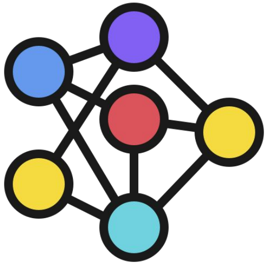
Can LLMs find and code algorithms?

Write a very efficient program in Python that finds largest independent sets in a sparse graph



Can LLMs find and code algorithms?

Write a very efficient program in Python that finds largest independent sets in a sparse graph



Python

```python
def find_largest_independent_sets(graph):
  """Finds the largest independent sets in a sparse graph.

  Args:
    graph: A sparse graph.

  Returns:
    A list of the largest independent sets in the graph.
  """

  # Initialize the list of independent sets.
  independent_sets = []

  # Iterate through the vertices of the graph.
  for vertex in graph.vertices:

    # Create a new independent set containing the current vertex.
    independent_set = set([vertex])

    # Iterate through the neighbors of the current vertex.
    for neighbor in graph.neighbors(vertex):

      # If the neighbor is already in the independent set, remove it.
      if neighbor in independent_set:
        independent_set.remove(neighbor)

    # Add the independent set to the list of independent sets.
    independent_sets.append(independent_set)

  # Return the list of independent sets.
  return independent_sets
```

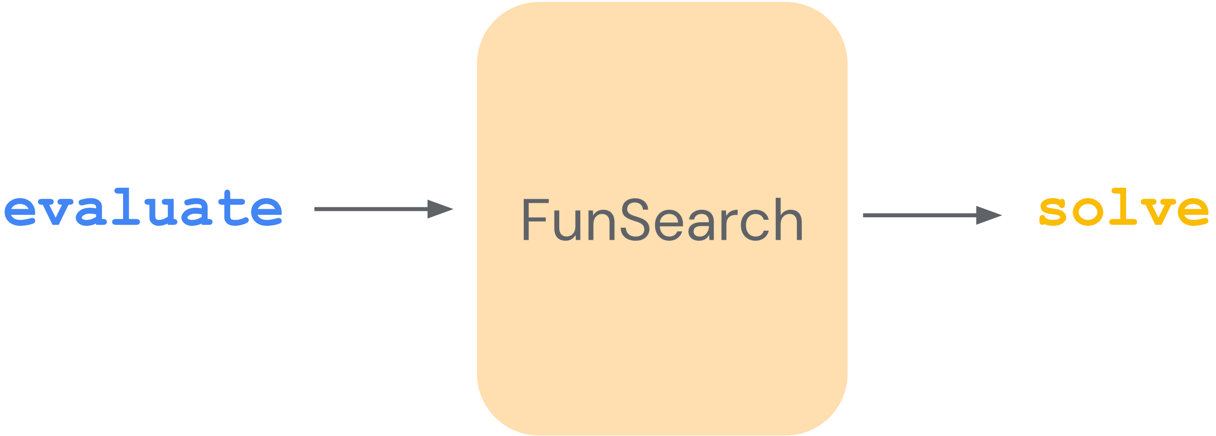
Use code with caution. Learn more

This program works by first initializing a list of independent sets. Then, it iterates through the vertices of the graph. For each vertex, it creates a new independent set containing the current vertex. Then, it iterates through the neighbors of the current vertex. If any of the neighbors are already in the independent set, it removes them. Finally, it adds the independent set to the list of independent sets. The program then returns the list of independent sets.

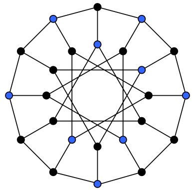
This program is efficient because it only iterates through the vertices and neighbors of the graph once. It also uses a set to store the independent sets, which is a fast data structure for storing and retrieving elements.

LLM+Evaluator: towards solving problems in NP "in practice"

Many problems admit a fast `evaluator`, but `solving` the problem is hard.



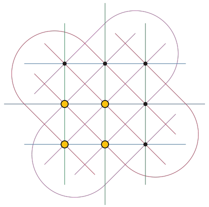
Our main results



Largest independent set

Find largest independent set in a graph

⟶ NP-hard problem



Particular focus on a structured graph (*cap-set graph*), with high mathematical significance

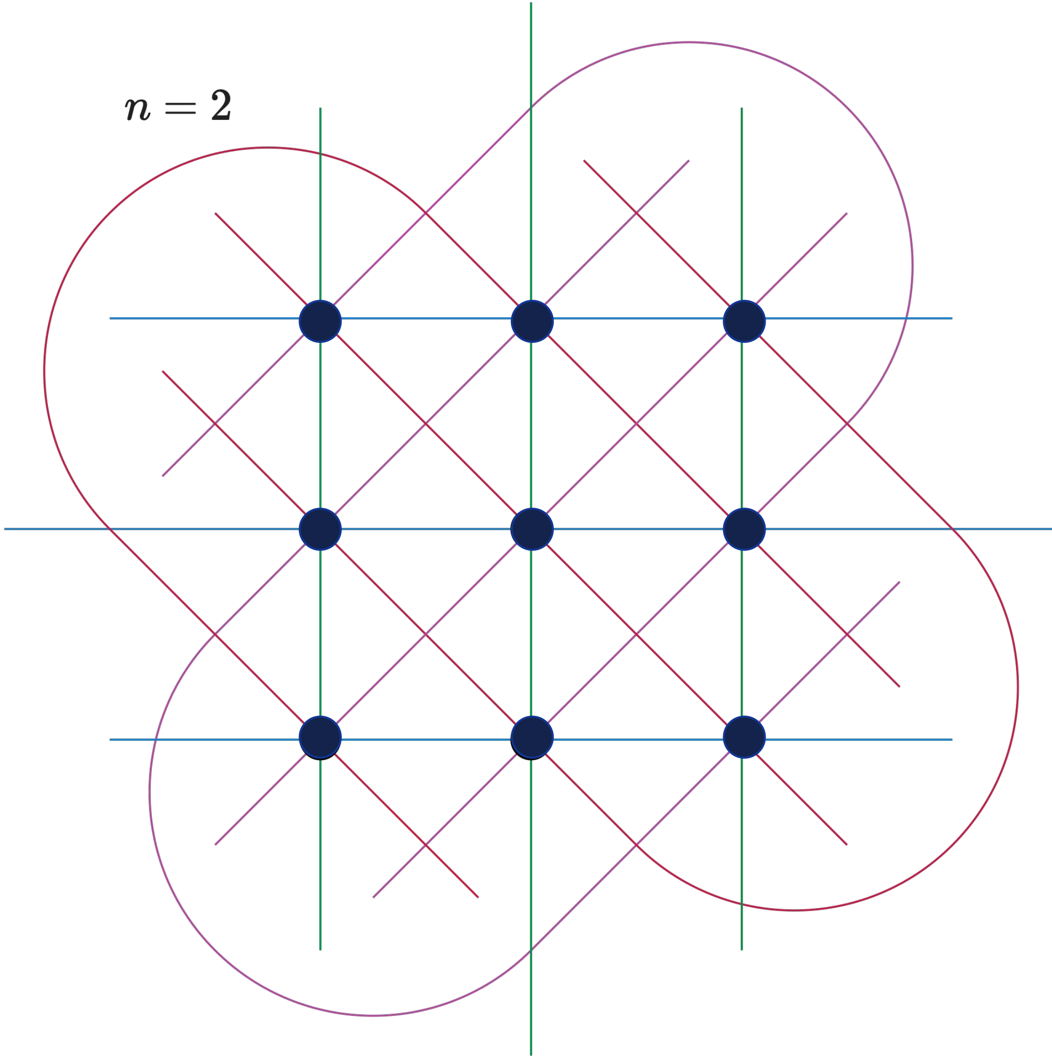
“Perhaps my favourite open question”

Terence Tao

FunSearch finds constructions that improve over existing state-of-the-art

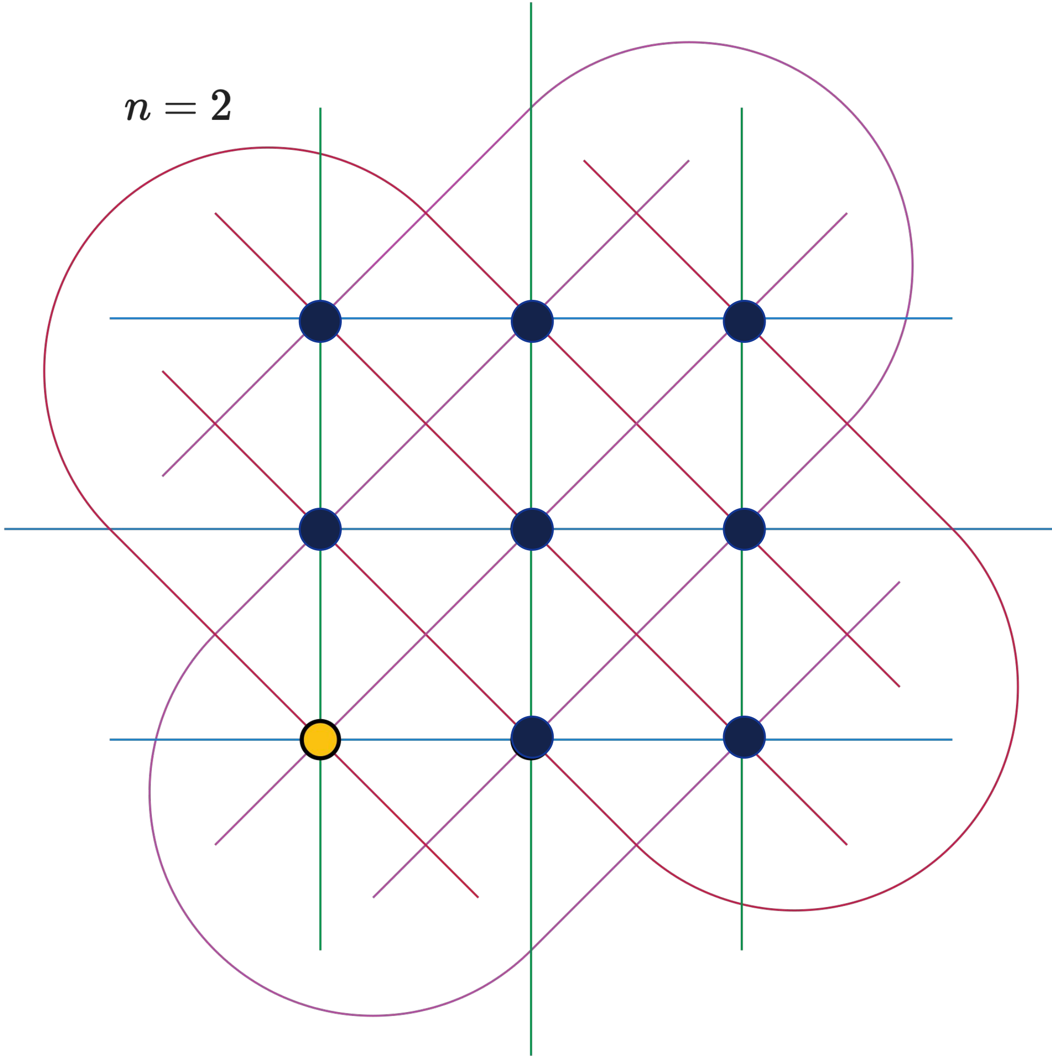
Cap set in 2 dimensions

What is the largest possible set of vectors in $\mathbb{F}_3^n$ such that no three lie on a line?



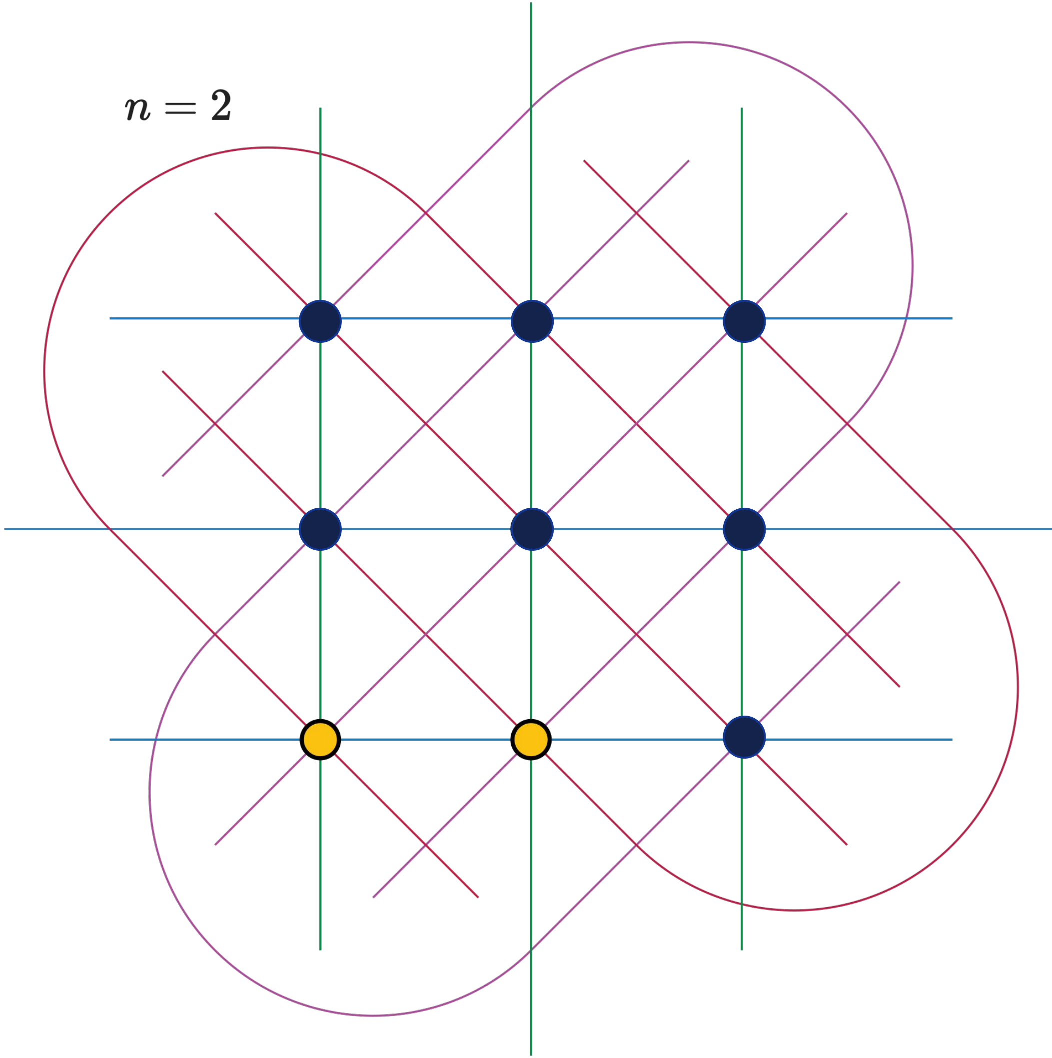
Cap set in 2 dimensions

What is the largest possible set of vectors in $\mathbb{F}_3^n$ such that no three lie on a line?



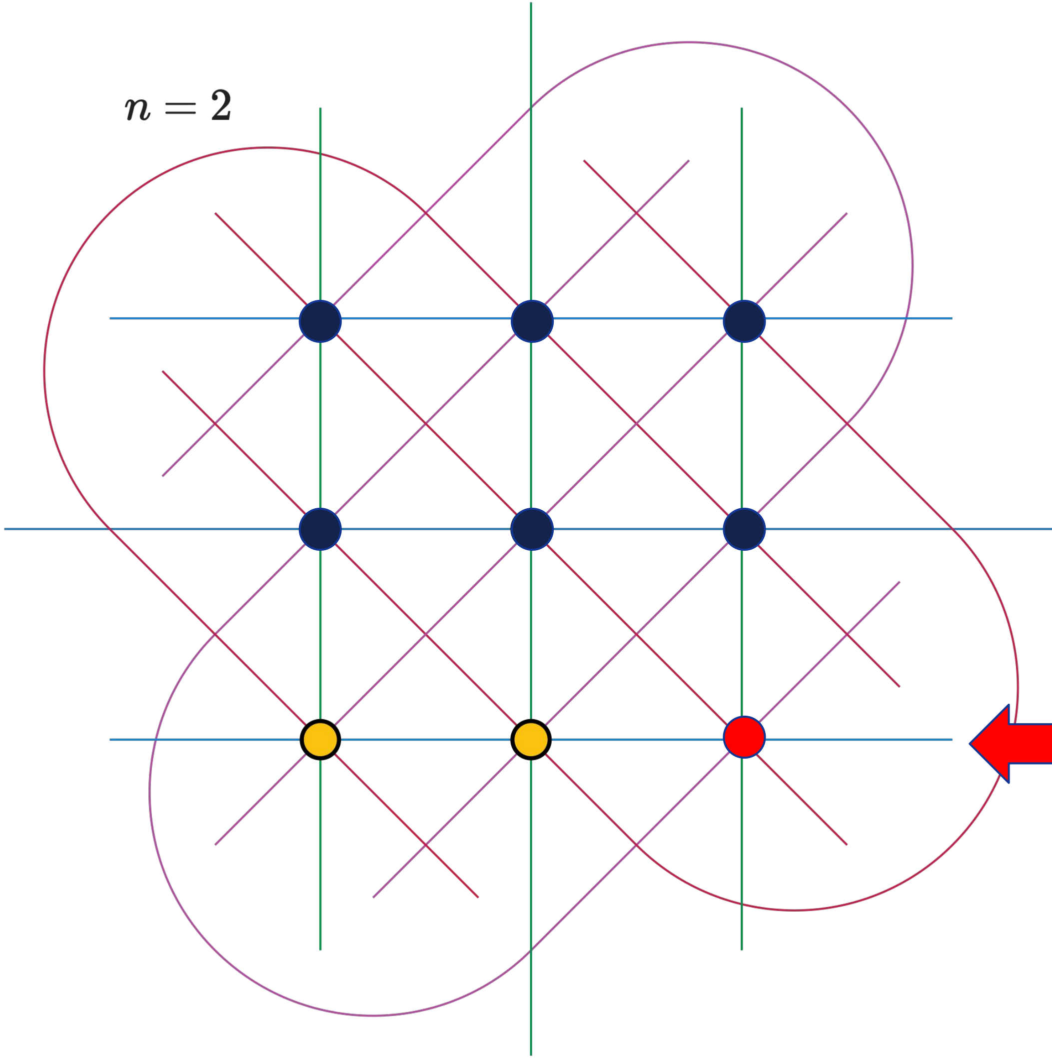
Cap set in 2 dimensions

What is the largest possible set of vectors in $\mathbb{F}_3^n$ such that no three lie on a line?



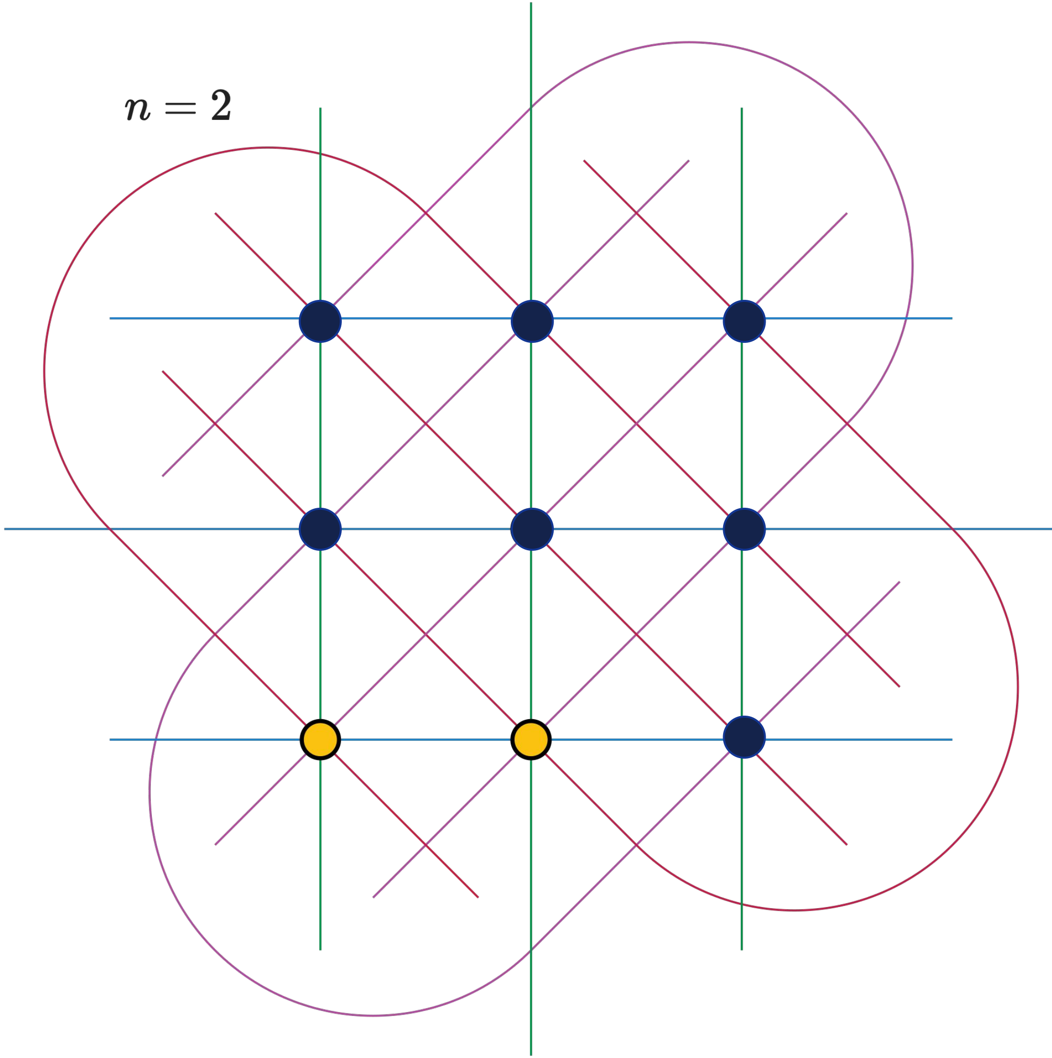
Cap set in 2 dimensions

What is the largest possible set of vectors in $\mathbb{F}_3^n$ such that no three lie on a line?



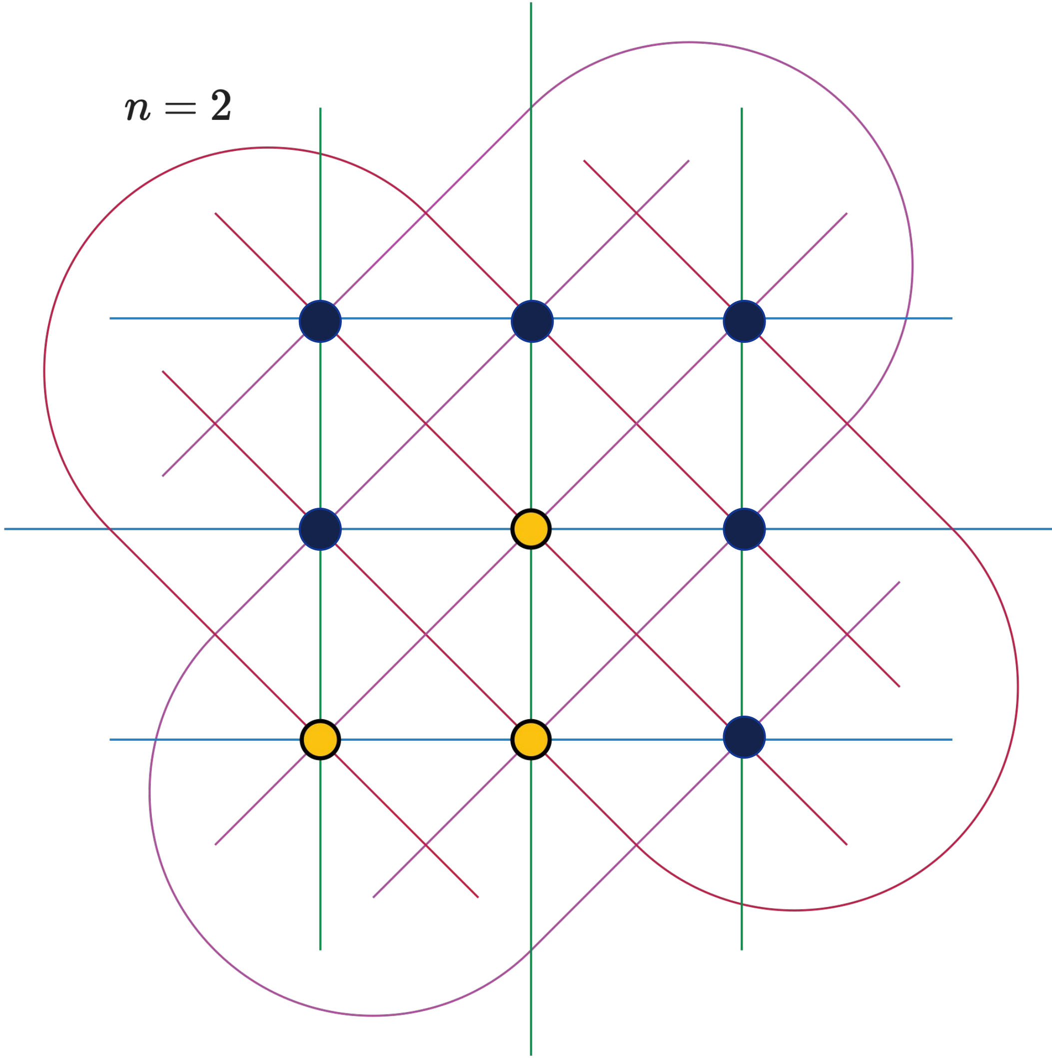
Cap set in 2 dimensions

What is the largest possible set of vectors in $\mathbb{F}_3^n$ such that no three lie on a line?



Cap set in 2 dimensions

What is the largest possible set of vectors in $\mathbb{F}_3^n$ such that no three lie on a line?



Cap set in 2 dimensions

What is the largest possible set of vectors in $\mathbb{F}_3^n$ such that no three lie on a line?

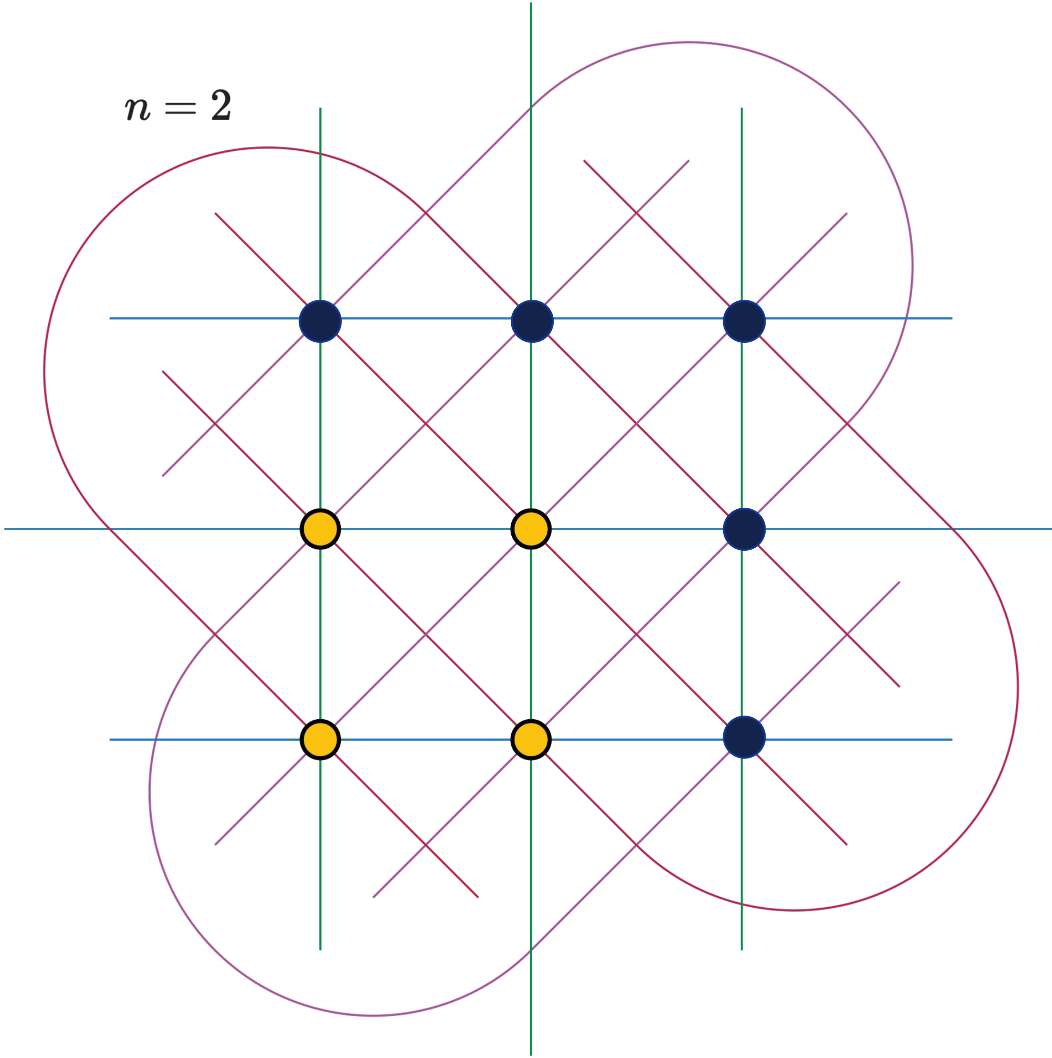
Cap set:

[(0, 0),

(1, 0),

(0, 1),

(1, 2),]



Two properties of a cap set candidate

- We can verify its correctness ✅
  - ➞ Is the set of points a cap set, i.e. are there not more than 3 points in a line?
- We can measure how good it is 🌟
  - ➞ How many elements are in the cap set?

⟶ We can write an efficient `evaluate` function

Largest independent set results

Dimension *n*	1	2	3	4	5	6	7	8	...	n ⟶ ∞
#nodes in hypergraph										
#edges in hypergraph										
Best known construction										
FunSearch construction										

Largest independent set results

Dimension *n*	1	2	3	4	5	6	7	8	...	n ⟶ ∞
#nodes in hypergraph	3									
#edges in hypergraph	3									
Best known construction	2									
FunSearch construction	**2**									

Largest independent set results

Dimension *n*	1	2	3	4	5	6	7	8	...	$n \rightarrow \infty$
#nodes in hypergraph	3	9								
#edges in hypergraph	3	36								
Best known construction	2	4								
FunSearch construction	**2**	**4**								

Largest independent set results

Dimension *n*	1	2	3	4	5	6	7	8	...	$n \longrightarrow \infty$
#nodes in hypergraph	3	9	27							
#edges in hypergraph	3	36	351							
Best known construction	2	4	9							
FunSearch construction	**2**	**4**	**9**							

Largest independent set results

Dimension *n*	1	2	3	4	5	6	7	8	...	$n \to \infty$
#nodes in hypergraph	3	9	27	81						
#edges in hypergraph	3	36	351	3240						
Best known construction	2	4	9	20						
FunSearch construction	**2**	**4**	**9**	**20**						

Largest independent set results

Dimension *n*	1	2	3	4	5	6	7	8	...	$n \rightarrow \infty$
#nodes in hypergraph	3	9	27	81	243					
#edges in hypergraph	3	36	351	3240	2.9×10^4					
Best known construction	2	4	9	20	45					
FunSearch construction	**2**	**4**	**9**	**20**	**45**					

Largest independent set results

Dimension *n*	1	2	3	4	5	6	7	8	...	$n \to \infty$
#nodes in hypergraph	3	9	27	81	243	729				
#edges in hypergraph	3	36	351	3240	2.9×10^4	2.7×10^5				
Best known construction	2	4	9	20	45	112				
FunSearch construction	**2**	**4**	**9**	**20**	**45**	**112**				

Largest independent set results

Dimension *n*	1	2	3	4	5	6	7	8	...	$n \longrightarrow \infty$
#nodes in hypergraph	3	9	27	81	243	729	2187			
#edges in hypergraph	3	36	351	3240	2.9×10^4	2.7×10^5	2.4×10^6			
Best known construction	2	4	9	20	45	112	236			
FunSearch construction	**2**	**4**	**9**	**20**	**45**	**112**	**236**			

Largest independent set results

Search space size ~3^{3900}

Dimension n	1	2	3	4	5	6	7	8	...	$n \longrightarrow \infty$
#nodes in hypergraph	3	9	27	81	243	729	2187	6561		
#edges in hypergraph	3	36	351	3240	2.9×10^4	2.7×10^5	2.4×10^6	2.1×10^7		
Best known construction	2	4	9	20	45	112	236	496		
FunSearch construction	**2**	**4**	**9**	**20**	**45**	**112**	**236**	**512**		

FunSearch finds constructions that improve over existing state-of-the-art

Largest independent set results

Search space size ~3^{3900}

Dimension *n*	1	2	3	4	5	6	7	8	...	$n \to \infty$
#nodes in hypergraph	3	9	27	81	243	729	2187	6561	...	3^n
#edges in hypergraph	3	36	351	3240	2.9×10^4	2.7×10^5	2.4×10^6	2.1×10^7	...	$\binom{3^n}{2}$
Best known construction	2	4	9	20	45	112	236	496	...	2.2180^n
FunSearch construction	**2**	**4**	**9**	**20**	**45**	**112**	**236**	**512**	...	$\mathbf{2.2202^n}$

FunSearch finds constructions that improve over existing state-of-the-art

Largest independent set results

Search space size ~3^{3900}

Dimension *n*	1	2	3	4	5	6	7	8	...	$n \to \infty$
#nodes in hypergraph	3	9	27	81	243	729	2187	6561	...	3^n
#edges in hypergraph	3	36	351	3240	2.9×10^4	2.7×10^5	2.4×10^6	2.1×10^7	...	$\binom{3^n}{2}$
Best known construction	2	4	9	20	45	112	236	496	...	2.2180^n
FunSearch construction	**2**	**4**	**9**	**20**	**45**	**112**	**236**	**512**	...	$\mathbf{2.2202^n}$

FunSearch finds constructions that improve over existing state-of-the-art

FunSearch also improves over existing state-of-the-art on other problems in maths:

- **Shannon capacity of cycle graphs**
- **Corners problem**

Solution space vs program space

Raw list of nodes

```
[1 1 1 1 1 1 1 1] [2 1 1 1 1 1 1 1] [0 0 1 0 1 0 2 1] [0 1 0 0 2 0 1 2] [1 0 0 0 0 1 1 1] [2 0 0 0 0 1 1 1] [0 0 1 1 0 1 2 1] [1 0 0 1 1 2 0 1]
[1 1 1 1 1 1 1 2] [2 1 1 1 1 1 1 2] [0 0 1 0 1 0 2 2] [0 1 0 0 2 0 2 2] [1 0 0 0 0 1 1 2] [2 0 0 0 0 1 1 2] [0 0 1 1 0 1 2 2] [1 0 0 1 1 2 0 2]
[1 1 1 1 1 1 2 1] [2 1 1 1 1 1 2 1] [0 0 1 0 1 1 0 1] [0 1 0 0 2 1 0 2] [1 0 0 0 0 1 2 1] [2 0 0 0 0 1 2 1] [0 0 1 1 0 2 2 1] [1 0 0 1 1 2 1 0]
[1 1 1 1 1 2 1 1] [2 1 1 1 1 2 1 1] [0 0 1 0 1 1 0 2] [0 1 0 0 2 1 1 0] [1 0 0 0 0 1 2 2] [2 0 0 0 0 1 2 2] [0 0 1 1 0 2 2 2] [1 0 0 1 1 2 2 0]
[1 1 1 1 2 1 1 1] [2 1 1 1 2 1 1 1] [0 0 1 0 1 1 2 0] [0 1 0 0 2 1 2 0] [1 0 0 0 0 2 1 1] [2 0 0 0 0 2 1 1] [0 0 1 2 0 1 2 1] [1 0 0 1 2 2 0 1]
[1 1 1 1 2 1 1 2] [2 1 1 1 2 1 1 2] [0 0 1 0 1 2 0 1] [0 1 0 0 2 2 0 2] [1 0 0 0 0 2 1 2] [2 0 0 0 0 2 1 2] [0 0 1 2 0 1 2 2] [1 0 0 1 2 2 0 2]
[1 1 1 1 2 1 2 1] [2 1 1 1 2 1 2 1] [0 0 1 0 1 2 0 2] [0 1 0 0 2 2 1 0] [1 0 0 0 0 2 2 1] [2 0 0 0 0 2 2 1] [0 0 1 2 0 2 2 1] [1 0 0 1 2 2 1 0]
[1 1 1 1 2 2 1 1] [2 1 1 1 2 2 1 1] [0 0 1 0 1 2 2 0] [0 1 0 0 2 2 2 0] [1 0 0 0 0 2 2 2] [2 0 0 0 0 2 2 2] [0 0 1 2 0 2 2 2] [1 0 0 1 2 2 2 0]
[1 1 1 2 1 1 1 1] [2 1 1 2 1 1 1 1] [0 0 1 0 2 0 2 1] [0 1 0 1 1 0 0 2] [1 0 0 1 0 0 1 1] [2 0 0 1 0 0 1 1] [0 0 2 1 0 1 1 1] [1 0 0 2 1 1 0 1]
[1 1 1 2 1 2 1 1] [2 1 1 2 1 2 1 1] [0 0 1 0 2 0 2 2] [0 1 0 1 1 0 1 0] [1 0 0 1 0 0 1 2] [2 0 0 1 0 0 1 2] [0 0 2 1 0 1 1 2] [1 0 0 2 1 1 0 2]
[1 1 1 2 1 2 1 2] [2 1 1 2 1 2 1 2] [0 0 1 0 2 1 0 1] [0 1 0 1 1 0 2 0] [1 0 0 1 0 0 2 1] [2 0 0 1 0 0 2 1] [0 0 2 1 0 2 1 1] [1 0 0 2 1 1 1 0]
[1 1 1 2 1 2 2 1] [2 1 1 2 1 2 2 1] [0 0 1 0 2 1 0 2] [0 1 0 1 2 0 0 2] [1 0 0 1 0 0 2 2] [2 0 0 1 0 0 2 2] [0 0 2 1 0 2 1 2] [1 0 0 2 1 1 2 0]
[1 1 1 2 2 1 1 1] [2 1 1 2 2 1 1 1] [0 0 1 0 2 1 2 0] [0 1 0 1 2 0 1 0] [1 0 0 2 0 0 1 1] [2 0 0 2 0 0 1 1] [0 0 2 2 0 1 1 1] [1 0 0 2 2 1 0 1]
[1 1 1 2 2 2 1 1] [2 1 1 2 2 2 1 1] [0 0 1 0 2 2 0 1] [0 1 0 1 2 0 2 0] [1 0 0 2 0 0 1 2] [2 0 0 2 0 0 1 2] [0 0 2 2 0 1 1 2] [1 0 0 2 2 1 0 2]
[1 1 1 2 2 2 1 2] [2 1 1 2 2 2 1 2] [0 0 1 0 2 2 0 2] [0 1 0 2 1 0 0 2] [1 0 0 2 0 0 2 1] [2 0 0 2 0 0 2 1] [0 0 2 2 0 2 1 1] [1 0 0 2 2 1 1 0]
[1 1 1 2 2 2 2 1] [2 1 1 2 2 2 2 1] [0 0 1 0 2 2 2 0] [0 1 0 2 1 0 1 0] [1 0 0 2 0 0 2 2] [2 0 0 2 0 0 2 2] [0 0 2 2 0 2 1 2] [1 0 0 2 2 1 2 0]
[1 1 2 1 1 1 1 1] [2 1 2 1 1 1 1 1] [0 0 1 1 1 0 0 1] [0 1 0 2 1 0 2 0] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 1 2] [1 0 1 1 1 2 0 0]
[1 1 2 1 1 1 2 1] [2 1 2 1 1 1 2 1] [0 0 1 1 1 0 0 2] [0 1 0 2 2 0 0 2] [1 0 1 0 0 0 2 2] [2 0 1 0 0 0 2 2] [0 1 0 1 0 1 2 2] [1 0 1 1 2 2 0 0]
[1 1 2 1 1 1 2 2] [2 1 2 1 1 1 2 2] [0 0 1 1 1 0 2 0] [0 1 0 2 2 0 1 0] [1 0 1 0 0 1 2 0] [2 0 1 0 0 1 2 0] [0 1 0 1 0 2 1 2] [1 0 1 2 1 1 0 0]
[1 1 2 1 1 2 2 1] [2 1 2 1 1 2 2 1] [0 0 1 1 2 0 0 1] [0 1 0 2 2 0 2 0] [1 0 1 0 0 2 2 0] [2 0 1 0 0 2 2 0] [0 1 0 1 0 2 2 2] [1 0 1 2 2 1 0 0]
[1 1 2 1 2 1 1 1] [2 1 2 1 2 1 1 1] [0 0 1 1 2 0 0 2] [0 1 1 0 1 0 0 2] [1 0 1 1 0 0 2 0] [2 0 1 1 0 0 2 0] [0 1 0 2 0 1 1 2] [1 0 2 1 1 2 0 0]
[1 1 2 1 2 1 2 1] [2 1 2 1 2 1 2 1] [0 0 1 1 2 0 2 0] [0 1 1 0 1 0 2 0] [1 0 1 2 0 0 2 0] [2 0 1 2 0 0 2 0] [0 1 0 2 0 1 2 2] [1 0 2 1 2 2 0 0]
[1 1 2 1 2 1 2 2] [2 1 2 1 2 1 2 2] [0 0 1 2 1 0 0 1] [0 1 1 0 2 0 0 2] [1 0 2 0 0 0 1 1] [2 0 2 0 0 0 1 1] [0 1 0 2 0 2 1 2] [1 0 2 2 1 1 0 0]
[1 1 2 1 2 2 2 1] [2 1 2 1 2 2 2 1] [0 0 1 2 1 0 0 2] [0 1 1 0 2 0 2 0] [1 0 2 0 0 0 1 2] [2 0 2 0 0 0 1 2] [0 1 0 2 0 2 2 2] [1 0 2 2 2 1 0 0]
[1 1 2 2 1 1 2 1] [2 1 2 2 1 1 2 1] [0 0 1 2 1 0 2 0] [0 1 2 0 1 0 0 2] [1 0 2 0 0 1 1 0] [2 0 2 0 0 1 1 0] [0 1 1 1 0 1 0 2] [1 1 0 1 1 2 0 0]
[1 1 2 2 1 2 1 1] [2 1 2 2 1 2 1 1] [0 0 1 2 2 0 0 1] [0 1 2 0 1 0 1 0] [1 0 2 0 0 2 1 0] [2 0 2 0 0 2 1 0] [0 1 1 1 0 1 2 0] [1 1 0 1 2 2 0 0]
[1 1 2 2 1 2 2 1] [2 1 2 2 1 2 2 1] [0 0 1 2 2 0 0 2] [0 1 2 0 2 0 0 2] [1 0 2 1 0 0 1 0] [2 0 2 1 0 0 1 0] [0 1 1 1 0 2 0 2] [1 1 0 2 1 1 0 0]
[1 1 2 2 1 2 2 2] [2 1 2 2 1 2 2 2] [0 0 1 2 2 0 2 0] [0 1 2 0 2 0 1 0] [1 0 2 2 0 0 1 0] [2 0 2 2 0 0 1 0] [0 1 1 1 0 2 2 0] [1 1 0 2 2 1 0 0]
[1 1 2 2 2 1 2 1] [2 1 2 2 2 1 2 1] [0 0 2 0 1 0 1 1] [0 2 0 0 1 0 1 1] [1 1 0 0 0 0 1 2] [2 1 0 0 0 0 1 2] [0 1 1 2 0 1 0 2] [1 2 0 1 1 2 0 0]
[1 1 2 2 2 2 1 1] [2 1 2 2 2 2 1 1] [0 0 2 0 1 0 1 2] [0 2 0 0 1 0 2 1] [1 1 0 0 0 0 2 2] [2 1 0 0 0 0 2 2] [0 1 1 2 0 1 2 0] [1 2 0 1 2 2 0 0]
[1 1 2 2 2 2 2 1] [2 1 2 2 2 2 2 1] [0 0 2 0 1 1 0 1] [0 2 0 0 1 1 0 1] [1 1 0 0 0 1 0 2] [2 1 0 0 0 1 0 2] [0 1 1 2 0 2 0 2] [1 2 0 2 1 1 0 0]
[1 1 2 2 2 2 2 2] [2 1 2 2 2 2 2 2] [0 0 2 0 1 1 0 2] [0 2 0 0 1 1 1 0] [1 1 0 0 0 2 0 2] [2 1 0 0 0 2 0 2] [0 1 1 2 0 2 2 0] [1 2 0 2 2 1 0 0]
[1 2 1 1 1 1 1 1] [2 2 1 1 1 1 1 1] [0 0 2 0 1 1 1 0] [0 2 0 0 1 1 2 0] [1 1 0 1 0 0 0 2] [2 1 0 1 0 0 0 2] [0 1 2 1 0 1 0 2] [2 0 0 1 1 2 0 1]
[1 2 1 1 1 1 1 2] [2 2 1 1 1 1 1 2] [0 0 2 0 1 2 0 1] [0 2 0 0 1 2 0 1] [1 1 0 2 0 0 0 2] [2 1 0 2 0 0 0 2] [0 1 2 1 0 1 1 0] [2 0 0 1 1 2 0 2]
[1 2 1 1 1 1 2 2] [2 2 1 1 1 1 2 2] [0 0 2 0 1 2 0 2] [0 2 0 0 1 2 1 0] [1 1 1 0 0 0 0 2] [2 1 1 0 0 0 0 2] [0 1 2 1 0 2 0 2] [2 0 0 1 1 2 1 0]
[1 2 1 1 1 2 1 2] [2 2 1 1 1 2 1 2] [0 0 2 0 1 2 1 0] [0 2 0 0 1 2 2 0] [1 1 1 0 0 0 2 0] [2 1 1 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 2 0]
[1 2 1 1 2 1 1 1] [2 2 1 1 2 1 1 1] [0 0 2 0 2 0 1 1] [0 2 0 0 2 0 1 1] [1 1 1 0 0 1 0 0] [2 1 1 0 0 1 0 0] [0 1 2 2 0 1 0 2] [2 0 0 1 2 2 0 1]
[1 2 1 1 2 1 1 2] [2 2 1 1 2 1 1 2] [0 0 2 0 2 0 1 2] [0 2 0 0 2 0 2 1] [1 1 1 0 0 2 0 0] [2 1 1 0 0 2 0 0] [0 1 2 2 0 1 1 0] [2 0 0 1 2 2 0 2]
[1 2 1 1 2 1 2 2] [2 2 1 1 2 1 2 2] [0 0 2 0 2 1 0 1] [0 2 0 0 2 1 0 1] [1 1 1 1 0 0 0 0] [2 1 1 1 0 0 0 0] [0 1 2 2 0 2 0 2] [2 0 0 1 2 2 1 0]
[1 2 1 1 2 2 1 2] [2 2 1 1 2 2 1 2] [0 0 2 0 2 1 0 2] [0 2 0 0 2 1 1 0] [1 1 1 2 0 0 0 0] [2 1 1 2 0 0 0 0] [0 1 2 2 0 2 1 0] [2 0 0 1 2 2 2 0]
[1 2 1 2 1 1 1 2] [2 2 1 2 1 1 1 2] [0 0 2 0 2 1 1 0] [0 2 0 0 2 1 2 0] [1 1 2 0 0 0 0 2] [2 1 2 0 0 0 0 2] [0 2 0 1 0 1 1 1] [2 0 0 2 1 1 0 1]
[1 2 1 2 1 2 1 1] [2 2 1 2 1 2 1 1] [0 0 2 0 2 2 0 1] [0 2 0 0 2 2 0 1] [1 1 2 0 0 0 1 0] [2 1 2 0 0 0 1 0] [0 2 0 1 0 1 2 1] [2 0 0 2 1 1 0 2]
[1 2 1 2 1 2 1 2] [2 2 1 2 1 2 1 2] [0 0 2 0 2 2 0 2] [0 2 0 0 2 2 1 0] [1 1 2 0 0 1 0 0] [2 1 2 0 0 1 0 0] [0 2 0 1 0 2 1 1] [2 0 0 2 1 1 1 0]
[1 2 1 2 1 2 2 2] [2 2 1 2 1 2 2 2] [0 0 2 0 2 2 1 0] [0 2 0 0 2 2 2 0] [1 1 2 0 0 2 0 0] [2 1 2 0 0 2 0 0] [0 2 0 1 0 2 2 1] [2 0 0 2 1 1 2 0]
[1 2 1 2 2 1 1 2] [2 2 1 2 2 1 1 2] [0 0 2 1 1 0 0 1] [0 2 0 1 1 0 0 1] [1 1 2 1 0 0 0 0] [2 1 2 1 0 0 0 0] [0 2 0 2 0 1 1 1] [2 0 0 2 2 1 0 1]
[1 2 1 2 2 1 1 1] [2 2 1 2 2 1 1 1] [0 0 2 1 1 0 0 2] [0 2 0 1 1 0 1 0] [1 1 2 2 0 0 0 0] [2 1 2 2 0 0 0 0] [0 2 0 2 0 1 2 1] [2 0 0 2 2 1 0 2]
[1 2 1 2 2 2 1 2] [2 2 1 2 2 2 1 2] [0 0 2 1 1 0 1 0] [0 2 0 1 1 0 2 0] [1 2 0 0 0 0 1 1] [2 2 0 0 0 0 1 1] [0 2 0 2 0 2 1 1] [2 0 0 2 2 1 1 0]
[1 2 1 2 2 2 2 2] [2 2 1 2 2 2 2 2] [0 0 2 1 2 0 0 1] [0 2 0 1 2 0 0 1] [1 2 0 0 0 0 2 1] [2 2 0 0 0 0 2 1] [0 2 0 2 0 2 2 1] [2 0 0 2 2 1 2 0]
[1 2 2 1 1 1 1 2] [2 2 2 1 1 1 1 2] [0 0 2 1 2 0 0 2] [0 2 0 1 2 0 1 0] [1 2 0 0 0 1 0 1] [2 2 0 0 0 1 0 1] [0 2 1 1 0 1 0 1] [2 0 1 1 1 2 0 0]
[1 2 2 1 1 1 2 1] [2 2 2 1 1 1 2 1] [0 0 2 1 2 0 1 0] [0 2 0 1 2 0 2 0] [1 2 0 0 0 2 0 1] [2 2 0 0 0 2 0 1] [0 2 1 1 0 1 2 0] [2 0 1 1 2 2 0 0]
[1 2 2 1 1 1 2 2] [2 2 2 1 1 1 2 2] [0 0 2 2 1 0 0 1] [0 2 0 2 1 0 0 1] [1 2 0 1 0 0 0 1] [2 2 0 1 0 0 0 1] [0 2 1 1 0 2 0 1] [2 0 1 2 1 1 0 0]
[1 2 2 1 1 2 2 2] [2 2 2 1 1 2 2 2] [0 0 2 2 1 0 0 2] [0 2 0 2 1 0 1 0] [1 2 0 2 0 0 0 1] [2 2 0 2 0 0 0 1] [0 2 1 1 0 2 2 0] [2 0 1 2 2 1 0 0]
[1 2 2 1 2 1 1 2] [2 2 2 1 2 1 1 2] [0 0 2 2 1 0 1 0] [0 2 0 2 1 0 2 0] [1 2 1 0 0 0 0 1] [2 2 1 0 0 0 0 1] [0 2 1 2 0 1 0 1] [2 0 2 1 1 2 0 0]
[1 2 2 1 2 1 2 1] [2 2 2 1 2 1 2 1] [0 0 2 2 2 0 0 1] [0 2 0 2 2 0 0 1] [1 2 1 0 0 0 2 0] [2 2 1 0 0 0 2 0] [0 2 1 2 0 1 2 0] [2 0 2 1 2 2 0 0]
[1 2 2 1 2 1 2 2] [2 2 2 1 2 1 2 2] [0 0 2 2 2 0 0 2] [0 2 0 2 2 0 1 0] [1 2 1 0 0 1 0 0] [2 2 1 0 0 1 0 0] [0 2 1 2 0 2 0 1] [2 0 2 2 1 1 0 0]
[1 2 2 1 2 2 2 2] [2 2 2 1 2 2 2 2] [0 0 2 2 2 0 1 0] [0 2 0 2 2 0 2 0] [1 2 1 0 0 2 0 0] [2 2 1 0 0 2 0 0] [0 2 1 2 0 2 2 0] [2 0 2 2 2 1 0 0]
[1 2 2 2 1 1 2 2] [2 2 2 2 1 1 2 2] [0 1 0 0 1 0 1 2] [0 2 1 0 1 0 0 1] [1 2 1 1 0 0 0 0] [2 2 1 1 0 0 0 0] [0 2 2 1 0 1 0 1] [2 1 0 1 1 2 0 0]
[1 2 2 2 1 2 1 2] [2 2 2 2 1 2 1 2] [0 1 0 0 1 0 2 2] [0 2 1 0 1 0 2 0] [1 2 1 2 0 0 0 0] [2 2 1 2 0 0 0 0] [0 2 2 1 0 1 1 0] [2 1 0 1 2 2 0 0]
[1 2 2 2 1 2 2 1] [2 2 2 2 1 2 2 1] [0 1 0 0 1 1 0 2] [0 2 1 0 2 0 0 1] [1 2 2 0 0 0 0 1] [2 2 2 0 0 0 0 1] [0 2 2 1 0 2 0 1] [2 1 0 2 1 1 0 0]
[1 2 2 2 1 2 2 2] [2 2 2 2 1 2 2 2] [0 1 0 0 1 1 1 0] [0 2 1 0 2 0 2 0] [1 2 2 0 0 0 1 0] [2 2 2 0 0 0 1 0] [0 2 2 1 0 2 1 0] [2 1 0 2 2 1 0 0]
[1 2 2 2 2 1 2 2] [2 2 2 2 2 1 2 2] [0 1 0 0 1 1 2 0] [0 2 2 0 1 0 0 1] [1 2 2 0 0 1 0 0] [2 2 2 0 0 1 0 0] [0 2 2 2 0 1 0 1] [2 2 0 1 1 2 0 0]
[1 2 2 2 2 2 1 2] [2 2 2 2 2 2 1 2] [0 1 0 0 1 2 0 2] [0 2 2 0 1 0 1 0] [1 2 2 0 0 2 0 0] [2 2 2 0 0 2 0 0] [0 2 2 2 0 1 1 0] [2 2 0 1 2 2 0 0]
[1 2 2 2 2 2 2 1] [2 2 2 2 2 2 2 1] [0 1 0 0 1 2 1 0] [0 2 2 0 2 0 0 1] [1 2 2 1 0 0 0 0] [2 2 2 1 0 0 0 0] [0 2 2 2 0 2 0 1] [2 2 0 2 1 1 0 0]
[1 2 2 2 2 2 2 2] [2 2 2 2 2 2 2 2] [0 1 0 0 1 2 2 0] [0 2 2 0 2 0 1 0] [1 2 2 2 0 0 0 0] [2 2 2 2 0 0 0 0] [0 2 2 2 0 2 1 0] [2 2 0 2 2 1 0 0]
```

Solution space vs program space

Raw list of nodes

```
[1 1 1 1 1 1 1 1] [2 1 1 1 1 1 1 1] [0 0 1 0 1 0 2 1] [0 1 0 0 2 0 1 2] [1 0 0 0 0 1 1 1] [2 0 0 0 0 1 1 1] [0 0 1 1 0 1 2 1] [1 0 0 1 1 2 0 1]
[1 1 1 1 1 1 1 2] [2 1 1 1 1 1 1 2] [0 0 1 0 1 0 2 2] [0 1 0 0 2 0 2 2] [1 0 0 0 0 1 1 2] [2 0 0 0 0 1 1 2] [0 0 1 1 0 1 2 2] [1 0 0 1 1 2 0 2]
[1 1 1 1 1 1 2 1] [2 1 1 1 1 1 2 1] [0 0 1 0 1 1 0 1] [0 1 0 0 2 1 0 2] [1 0 0 0 0 1 2 1] [2 0 0 0 0 1 2 1] [0 0 1 1 0 2 2 1] [1 0 0 1 1 2 1 0]
[1 1 1 1 1 2 1 1] [2 1 1 1 1 2 1 1] [0 0 1 0 1 1 0 2] [0 1 0 0 2 1 1 0] [1 0 0 0 0 1 2 2] [2 0 0 0 0 1 2 2] [0 0 1 1 0 2 2 2] [1 0 0 1 1 2 2 0]
[1 1 1 1 2 1 1 1] [2 1 1 1 2 1 1 1] [0 0 1 0 1 1 2 0] [0 1 0 0 2 1 2 0] [1 0 0 0 0 2 1 1] [2 0 0 0 0 2 1 1] [0 0 1 2 0 1 2 1] [1 0 0 1 2 2 0 1]
[1 1 1 1 2 1 1 2] [2 1 1 1 2 1 1 2] [0 0 1 0 1 2 0 1] [0 1 0 0 2 2 0 2] [1 0 0 0 0 2 1 2] [2 0 0 0 0 2 1 2] [0 0 1 2 0 1 2 2] [1 0 0 1 2 2 0 2]
[1 1 1 1 2 1 2 1] [2 1 1 1 2 1 2 1] [0 0 1 0 1 2 0 2] [0 1 0 0 2 2 1 0] [1 0 0 0 0 2 2 1] [2 0 0 0 0 2 2 1] [0 0 1 2 0 2 2 1] [1 0 0 1 2 2 1 0]
[1 1 1 1 2 2 1 1] [2 1 1 1 2 2 1 1] [0 0 1 0 1 2 2 0] [0 1 0 0 2 2 2 0] [1 0 0 0 0 2 2 2] [2 0 0 0 0 2 2 2] [0 0 1 2 0 2 2 2] [1 0 0 1 2 2 2 0]
[1 1 1 2 1 1 1 1] [2 1 1 2 1 1 1 1] [0 0 1 0 2 0 2 1] [0 1 0 1 1 0 0 2] [1 0 0 1 0 0 1 1] [2 0 0 1 0 0 1 1] [0 0 2 1 0 1 1 1] [1 0 0 2 1 1 0 1]
[1 1 1 2 1 2 1 1] [2 1 1 2 1 2 1 1] [0 0 1 0 2 0 2 2] [0 1 0 1 1 0 1 0] [1 0 0 1 0 0 1 2] [2 0 0 1 0 0 1 2] [0 0 2 1 0 1 1 2] [1 0 0 2 1 1 0 2]
[1 1 1 2 1 2 1 2] [2 1 1 2 1 2 1 2] [0 0 1 0 2 1 0 1] [0 1 0 1 1 0 2 0] [1 0 0 1 0 0 2 1] [2 0 0 1 0 0 2 1] [0 0 2 1 0 2 1 1] [1 0 0 2 1 1 1 0]
[1 1 1 2 1 2 2 1] [2 1 1 2 1 2 2 1] [0 0 1 0 2 1 0 2] [0 1 0 1 2 0 0 2] [1 0 0 1 0 0 2 2] [2 0 0 1 0 0 2 2] [0 0 2 1 0 2 1 2] [1 0 0 2 1 1 2 0]
[1 1 1 2 2 1 1 1] [2 1 1 2 2 1 1 1] [0 0 1 0 2 1 2 0] [0 1 0 1 2 0 1 0] [1 0 0 2 0 0 1 1] [2 0 0 2 0 0 1 1] [0 0 2 2 0 1 1 1] [1 0 0 2 2 1 0 1]
[1 1 1 2 2 2 1 1] [2 1 1 2 2 2 1 1] [0 0 1 0 2 2 0 1] [0 1 0 1 2 0 2 0] [1 0 0 2 0 0 1 2] [2 0 0 2 0 0 1 2] [0 0 2 2 0 1 1 2] [1 0 0 2 2 1 0 2]
[1 1 1 2 2 2 1 2] [2 1 1 2 2 2 1 2] [0 0 1 0 2 2 0 2] [0 1 0 2 1 0 0 2] [1 0 0 2 0 0 2 1] [2 0 0 2 0 0 2 1] [0 0 2 2 0 2 1 1] [1 0 0 2 2 1 1 0]
[1 1 1 2 2 2 2 1] [2 1 1 2 2 2 2 1] [0 0 1 0 2 2 2 0] [0 1 0 2 1 0 1 0] [1 0 0 2 0 0 2 2] [2 0 0 2 0 0 2 2] [0 0 2 2 0 2 1 2] [1 0 0 2 2 1 2 0]
[1 1 2 1 1 1 1 1] [2 1 2 1 1 1 1 1] [0 0 1 1 1 0 0 1] [0 1 0 2 1 0 2 0] [1 0 1 0 0 0 2 1] [2 0 1 0 0 0 2 1] [0 1 0 1 0 1 1 2] [1 0 1 1 1 2 0 0]
[1 1 2 1 1 1 2 1] [2 1 2 1 1 1 2 1] [0 0 1 1 1 0 0 2] [0 1 0 2 2 0 0 2] [1 0 1 0 0 0 2 2] [2 0 1 0 0 0 2 2] [0 1 0 1 0 1 2 2] [1 0 1 1 2 2 0 0]
[1 1 2 1 1 1 2 2] [2 1 2 1 1 1 2 2] [0 0 1 1 1 0 2 0] [0 1 0 2 2 0 1 0] [1 0 1 0 0 1 2 0] [2 0 1 0 0 1 2 0] [0 1 0 1 0 2 1 2] [1 0 1 2 1 1 0 0]
[1 1 2 1 1 2 2 1] [2 1 2 1 1 2 2 1] [0 0 1 1 2 0 0 1] [0 1 0 2 2 0 2 0] [1 0 1 0 0 2 2 0] [2 0 1 0 0 2 2 0] [0 1 0 1 0 2 2 2] [1 0 1 2 2 1 0 0]
[1 1 2 1 2 1 1 1] [2 1 2 1 2 1 1 1] [0 0 1 1 2 0 0 2] [0 1 1 0 1 0 0 2] [1 0 1 1 0 0 2 0] [2 0 1 1 0 0 2 0] [0 1 0 2 0 1 1 2] [1 0 2 1 1 2 0 0]
[1 1 2 1 2 1 2 1] [2 1 2 1 2 1 2 1] [0 0 1 1 2 0 2 0] [0 1 1 0 1 0 2 0] [1 0 1 2 0 0 2 0] [2 0 1 2 0 0 2 0] [0 1 0 2 0 1 2 2] [1 0 2 1 2 2 0 0]
[1 1 2 1 2 1 2 2] [2 1 2 1 2 1 2 2] [0 0 1 2 1 0 0 1] [0 1 1 0 2 0 0 2] [1 0 2 0 0 0 1 1] [2 0 2 0 0 0 1 1] [0 1 0 2 0 2 1 2] [1 0 2 2 1 1 0 0]
[1 1 2 1 2 2 2 1] [2 1 2 1 2 2 2 1] [0 0 1 2 1 0 0 2] [0 1 1 0 2 0 2 0] [1 0 2 0 0 0 1 2] [2 0 2 0 0 0 1 2] [0 1 0 2 0 2 2 2] [1 0 2 2 2 1 0 0]
[1 1 2 2 1 1 2 1] [2 1 2 2 1 1 2 1] [0 0 1 2 1 0 2 0] [0 1 2 0 1 0 0 2] [1 0 2 0 0 1 1 0] [2 0 2 0 0 1 1 0] [0 1 1 1 0 1 0 2] [1 1 0 1 1 2 0 0]
[1 1 2 2 1 2 1 1] [2 1 2 2 1 2 1 1] [0 0 1 2 2 0 0 1] [0 1 2 0 1 0 1 0] [1 0 2 0 0 2 1 0] [2 0 2 0 0 2 1 0] [0 1 1 1 0 1 2 0] [1 1 0 1 2 2 0 0]
[1 1 2 2 1 2 2 1] [2 1 2 2 1 2 2 1] [0 0 1 2 2 0 0 2] [0 1 2 0 2 0 0 2] [1 0 2 1 0 0 1 0] [2 0 2 1 0 0 1 0] [0 1 1 1 0 2 0 2] [1 1 0 2 1 1 0 0]
[1 1 2 2 1 2 2 2] [2 1 2 2 1 2 2 2] [0 0 1 2 2 0 2 0] [0 1 2 0 2 0 1 0] [1 0 2 2 0 0 1 0] [2 0 2 2 0 0 1 0] [0 1 1 1 0 2 2 0] [1 1 0 2 2 1 0 0]
[1 1 2 2 2 1 2 1] [2 1 2 2 2 1 2 1] [0 0 2 0 1 0 1 1] [0 2 0 0 1 0 1 1] [1 1 0 0 0 0 1 2] [2 1 0 0 0 0 1 2] [0 1 1 2 0 1 0 2] [1 2 0 1 1 2 0 0]
[1 1 2 2 2 2 1 1] [2 1 2 2 2 2 1 1] [0 0 2 0 1 0 1 2] [0 2 0 0 1 0 2 1] [1 1 0 0 0 0 2 2] [2 1 0 0 0 0 2 2] [0 1 1 2 0 1 2 0] [1 2 0 1 2 2 0 0]
[1 1 2 2 2 2 2 1] [2 1 2 2 2 2 2 1] [0 0 2 0 1 1 0 1] [0 2 0 0 1 1 0 1] [1 1 0 0 0 1 0 2] [2 1 0 0 0 1 0 2] [0 1 1 2 0 2 0 2] [1 2 0 2 1 1 0 0]
[1 1 2 2 2 2 2 2] [2 1 2 2 2 2 2 2] [0 0 2 0 1 1 0 2] [0 2 0 0 1 1 1 0] [1 1 0 0 0 2 0 2] [2 1 0 0 0 2 0 2] [0 1 1 2 0 2 2 0] [1 2 0 2 2 1 0 0]
[1 2 1 1 1 1 1 1] [2 2 1 1 1 1 1 1] [0 0 2 0 1 1 1 0] [0 2 0 0 1 1 2 0] [1 1 0 1 0 0 0 2] [2 1 0 1 0 0 0 2] [0 1 2 1 0 1 0 2] [2 0 0 1 1 2 0 1]
[1 2 1 1 1 1 1 2] [2 2 1 1 1 1 1 2] [0 0 2 0 1 2 0 1] [0 2 0 0 1 2 0 1] [1 1 0 2 0 0 0 2] [2 1 0 2 0 0 0 2] [0 1 2 1 0 1 1 0] [2 0 0 1 1 2 0 2]
[1 2 1 1 1 1 2 2] [2 2 1 1 1 1 2 2] [0 0 2 0 1 2 0 2] [0 2 0 0 1 2 1 0] [1 1 1 0 0 0 0 2] [2 1 1 0 0 0 0 2] [0 1 2 1 0 2 0 2] [2 0 0 1 1 2 1 0]
[1 2 1 1 1 2 1 2] [2 2 1 1 1 2 1 2] [0 0 2 0 1 2 1 0] [0 2 0 0 1 2 2 0] [1 1 1 0 0 0 2 0] [2 1 1 0 0 0 2 0] [0 1 2 1 0 2 1 0] [2 0 0 1 1 2 2 0]
[1 2 1 1 2 1 1 1] [2 2 1 1 2 1 1 1] [0 0 2 0 2 0 1 1] [0 2 0 0 2 0 1 1] [1 1 1 0 0 1 0 0] [2 1 1 0 0 1 0 0] [0 1 2 2 0 1 0 2] [2 0 0 1 2 2 0 1]
[1 2 1 1 2 1 1 2] [2 2 1 1 2 1 1 2] [0 0 2 0 2 0 1 2] [0 2 0 0 2 0 2 1] [1 1 1 0 0 2 0 0] [2 1 1 0 0 2 0 0] [0 1 2 2 0 1 1 0] [2 0 0 1 2 2 0 2]
[1 2 1 1 2 1 2 2] [2 2 1 1 2 1 2 2] [0 0 2 0 2 1 0 1] [0 2 0 0 2 1 0 1] [1 1 1 1 0 0 0 0] [2 1 1 1 0 0 0 0] [0 1 2 2 0 2 0 2] [2 0 0 1 2 2 1 0]
[1 2 1 1 2 2 1 2] [2 2 1 1 2 2 1 2] [0 0 2 0 2 1 0 2] [0 2 0 0 2 1 1 0] [1 1 1 2 0 0 0 0] [2 1 1 2 0 0 0 0] [0 1 2 2 0 2 1 0] [2 0 0 1 2 2 2 0]
[1 2 1 2 1 1 1 2] [2 2 1 2 1 1 1 2] [0 0 2 0 2 1 1 0] [0 2 0 0 2 1 2 0] [1 1 2 0 0 0 0 2] [2 1 2 0 0 0 0 2] [0 2 0 1 0 1 1 1] [2 0 0 2 1 1 0 1]
[1 2 1 2 1 2 1 1] [2 2 1 2 1 2 1 1] [0 0 2 0 2 2 0 1] [0 2 0 0 2 2 0 1] [1 1 2 0 0 0 1 0] [2 1 2 0 0 0 1 0] [0 2 0 1 0 1 2 1] [2 0 0 2 1 1 0 2]
[1 2 1 2 1 2 1 2] [2 2 1 2 1 2 1 2] [0 0 2 0 2 2 0 2] [0 2 0 0 2 2 1 0] [1 1 2 0 0 1 0 0] [2 1 2 0 0 1 0 0] [0 2 0 1 0 2 1 1] [2 0 0 2 1 1 1 0]
[1 2 1 2 1 2 2 2] [2 2 1 2 1 2 2 2] [0 0 2 0 2 2 1 0] [0 2 0 0 2 2 2 0] [1 1 2 0 0 2 0 0] [2 1 2 0 0 2 0 0] [0 2 0 1 0 2 2 1] [2 0 0 2 1 1 2 0]
[1 2 1 2 2 1 1 2] [2 2 1 2 2 1 1 2] [0 0 2 1 1 0 0 1] [0 2 0 1 1 0 0 1] [1 1 2 1 0 0 0 0] [2 1 2 1 0 0 0 0] [0 2 0 2 0 1 1 1] [2 0 0 2 2 1 0 1]
[1 2 1 2 2 2 1 1] [2 2 1 2 2 2 1 1] [0 0 2 1 1 0 0 2] [0 2 0 1 1 0 1 0] [1 1 2 2 0 0 0 0] [2 1 2 2 0 0 0 0] [0 2 0 2 0 1 2 1] [2 0 0 2 2 1 0 2]
[1 2 1 2 2 2 1 2] [2 2 1 2 2 2 1 2] [0 0 2 1 1 0 1 0] [0 2 0 1 1 0 2 0] [1 2 0 0 0 0 1 1] [2 2 0 0 0 0 1 1] [0 2 0 2 0 2 1 1] [2 0 0 2 2 1 1 0]
[1 2 1 2 2 2 2 2] [2 2 1 2 2 2 2 2] [0 0 2 1 2 0 0 1] [0 2 0 1 2 0 0 1] [1 2 0 0 0 0 2 1] [2 2 0 0 0 0 2 1] [0 2 0 2 0 2 2 1] [2 0 0 2 2 1 2 0]
[1 2 2 1 1 1 1 2] [2 2 2 1 1 1 1 2] [0 0 2 1 2 0 0 2] [0 2 0 1 2 0 1 0] [1 2 0 0 0 1 0 1] [2 2 0 0 0 1 0 1] [0 2 1 1 0 1 0 1] [2 0 1 1 1 2 0 0]
[1 2 2 1 1 1 2 1] [2 2 2 1 1 1 2 1] [0 0 2 1 2 0 1 0] [0 2 0 1 2 0 2 0] [1 2 0 0 0 2 0 1] [2 2 0 0 0 2 0 1] [0 2 1 1 0 1 2 0] [2 0 1 1 2 2 0 0]
[1 2 2 1 1 1 2 2] [2 2 2 1 1 1 2 2] [0 0 2 2 1 0 0 1] [0 2 0 2 1 0 0 1] [1 2 0 1 0 0 0 1] [2 2 0 1 0 0 0 1] [0 2 1 1 0 2 0 1] [2 0 1 2 1 1 0 0]
[1 2 2 1 1 2 2 2] [2 2 2 1 1 2 2 2] [0 0 2 2 1 0 0 2] [0 2 0 2 1 0 1 0] [1 2 0 2 0 0 0 1] [2 2 0 2 0 0 0 1] [0 2 1 1 0 2 2 0] [2 0 1 2 2 1 0 0]
[1 2 2 1 2 1 1 2] [2 2 2 1 2 1 1 2] [0 0 2 2 1 0 1 0] [0 2 0 2 1 0 2 0] [1 2 1 0 0 0 0 1] [2 2 1 0 0 0 0 1] [0 2 1 2 0 1 0 1] [2 0 2 1 1 2 0 0]
[1 2 2 1 2 1 2 1] [2 2 2 1 2 1 2 1] [0 0 2 2 2 0 0 1] [0 2 0 2 2 0 0 1] [1 2 1 0 0 0 2 0] [2 2 1 0 0 0 2 0] [0 2 1 2 0 1 2 0] [2 0 2 1 2 2 0 0]
[1 2 2 1 2 1 2 2] [2 2 2 1 2 1 2 2] [0 0 2 2 2 0 0 2] [0 2 0 2 2 0 1 0] [1 2 1 0 0 1 0 0] [2 2 1 0 0 1 0 0] [0 2 1 2 0 2 0 1] [2 0 2 2 1 1 0 0]
[1 2 2 1 2 2 2 2] [2 2 2 1 2 2 2 2] [0 0 2 2 2 0 1 0] [0 2 0 2 2 0 2 0] [1 2 1 0 0 2 0 0] [2 2 1 0 0 2 0 0] [0 2 1 2 0 2 2 0] [2 0 2 2 2 1 0 0]
[1 2 2 2 1 1 2 2] [2 2 2 2 1 1 2 2] [0 1 0 0 1 0 1 2] [0 2 1 0 1 0 0 1] [1 2 1 1 0 0 0 0] [2 2 1 1 0 0 0 0] [0 2 2 1 0 1 0 1] [2 1 0 1 1 2 0 0]
[1 2 2 2 1 2 1 2] [2 2 2 2 1 2 1 2] [0 1 0 0 1 0 2 2] [0 2 1 0 1 0 2 0] [1 2 1 2 0 0 0 0] [2 2 1 2 0 0 0 0] [0 2 2 1 0 1 1 0] [2 1 0 1 2 2 0 0]
[1 2 2 2 1 2 2 1] [2 2 2 2 1 2 2 1] [0 1 0 0 1 1 0 2] [0 2 1 0 2 0 0 1] [1 2 2 0 0 0 0 1] [2 2 2 0 0 0 0 1] [0 2 2 1 0 2 0 1] [2 1 0 2 1 1 0 0]
[1 2 2 2 1 2 2 2] [2 2 2 2 1 2 2 2] [0 1 0 0 1 1 1 0] [0 2 1 0 2 0 2 0] [1 2 2 0 0 0 1 0] [2 2 2 0 0 0 1 0] [0 2 2 1 0 2 1 0] [2 1 0 2 2 1 0 0]
[1 2 2 2 2 1 2 2] [2 2 2 2 2 1 2 2] [0 1 0 0 1 1 2 0] [0 2 2 0 1 0 0 1] [1 2 2 0 0 1 0 0] [2 2 2 0 0 1 0 0] [0 2 2 2 0 1 0 1] [2 2 0 1 1 2 0 0]
[1 2 2 2 2 2 1 2] [2 2 2 2 2 2 1 2] [0 1 0 0 1 2 0 2] [0 2 2 0 1 0 1 0] [1 2 2 0 0 2 0 0] [2 2 2 0 0 2 0 0] [0 2 2 2 0 1 1 0] [2 2 0 1 2 2 0 0]
[1 2 2 2 2 2 2 1] [2 2 2 2 2 2 2 1] [0 1 0 0 1 2 1 0] [0 2 2 0 2 0 0 1] [1 2 2 1 0 0 0 0] [2 2 2 1 0 0 0 0] [0 2 2 2 0 2 0 1] [2 2 0 2 1 1 0 0]
[1 2 2 2 2 2 2 2] [2 2 2 2 2 2 2 2] [0 1 0 0 1 2 2 0] [0 2 2 0 2 0 1 0] [1 2 2 2 0 0 0 0] [2 2 2 2 0 0 0 0] [0 2 2 2 0 2 1 0] [2 2 0 2 2 1 0 0]
```

Program that outputs a list of nodes

```python
def get_capset(n: int) -> CapSet:
  """Returns a 512-cap in AG(8, 3)."""
  V = np.array(list(itertools.product(range(3), repeat=n)), dtype=np.int32)
  reflections = lambda v: sum(1 for i in range(1, n // 2) if v[i] == v[-i])

  # First we list 128 weight-8 vectors with >= 2 reflections.
  weight8_points = [v for v in V
                    if np.count_nonzero(v) == 8  # Weight is 8.
                    and reflections(v) >= 2]  # At least 2 reflections.

  # Then we list 256 weight-4 vectors with allowed support and <= 1 reflections.
  allowed_supports = [
      (0, 1, 2, 3), (0, 1, 2, 5), (0, 1, 2, 7), (0, 1, 2, 6), (0, 1, 3, 7),
      (0, 1, 6, 7), (0, 3, 6, 7), (0, 5, 6, 7), (0, 1, 5, 7), (1, 3, 4, 6),
      (1, 4, 5, 6), (0, 2, 3, 6), (2, 3, 4, 7), (2, 4, 5, 7), (0, 2, 6, 7),
      (0, 2, 5, 6), (1, 2, 4, 7), (1, 2, 4, 6), (1, 3, 4, 7), (1, 4, 6, 7),
      (1, 4, 5, 7), (2, 3, 4, 6), (2, 4, 6, 7), (2, 4, 5, 6),
  ]
  weight4_points = [
      v for v in V
      if np.count_nonzero(v) == 4  # Weight is 4.
      and tuple(i for i in range(n) if v[i] != 0) in allowed_supports
      and reflections(v) <= 1]  # At most 1 reflection.

  # Finally we add 128 weight-5 vectors with <= 1 reflections.
  allowed_zeros = [(0, 4, 7), (0, 2, 4), (0, 1, 4), (0, 4, 6),
                   (1, 2, 6), (2, 6, 7), (1, 2, 7), (1, 6, 7)]
  weight5_points = [
      v for v in V
      if np.count_nonzero(v) == 5  # Weight is 4.
      and tuple(i for i in range(n) if v[i] == 0) in allowed_zeros
      and reflections(v) <= 1  # At most 1 reflection.
      and (v[1] * v[7]) % 3 != 1 and (v[2] * v[6]) % 3 != 1]  # Mod conditions.

  return weight8_points + weight4_points + weight5_points
```

Solution space vs program space

Raw list of nodes

Program that outputs a list of nodes

```
def get_capset(n: int) -> CapSet:
  """Returns a 512-cap in AG(8, 3)."""
  V = np.array(list(itertools.product(range(3), repeat=n)), dtype=np.int32)
  reflections = lambda v: sum(1 for i in range(1, n // 2) if v[i] == v[-i])

  # First we list 128 weight-8 vectors with >= 2 reflections.
  weight8_points = [v for v in V
```

“The program supplied by the LLM is far conceptually richer than a mere list of vectors. **I am learning something** — e.g. this idea of classifying by number of reflections is novel.”

Jordan Ellenberg, author of a breakthrough in this area and author of NYT bestseller “How Not to be Wrong: The Power of Mathematical Thinking”

```
      and tuple(i for i in range(n) if v[i] != 0) in allowed_supports
      and reflections(v) <= 1]  # At most 1 reflection.

  # Finally we add 128 weight-5 vectors with <= 1 reflections.
  allowed_zeros = [(0, 4, 7), (0, 2, 4), (0, 1, 4), (0, 4, 6),
                   (1, 2, 6), (2, 6, 7), (1, 2, 7), (1, 6, 7)]
  weight5_points = [
      v for v in V
      if np.count_nonzero(v) == 5  # Weight is 4.
      and tuple(i for i in range(n) if v[i] == 0) in allowed_zeros
      and reflections(v) <= 1  # At most 1 reflection.
      and (v[1] * v[7]) % 3 != 1 and (v[2] * v[6]) % 3 != 1]  # Mod conditions.

  return weight8_points + weight4_points + weight5_points
```

Actionable interpretability

```
def priority(el: tuple[int, ...], n: int, w: int) -> float:
  score = 0.0
  for i in range(n):
    if el[i] == 1:
      score -= 0.9 ** ( i % 4 )
    if el[i] == 2:
      score -= 0.98 ** (30 - ( i % 4 ))
    if el[i] == 1 and el[i - 4] == 1:
      score -= 0.98 ** (30 - ( i % 4 ))
    if el[i] == 2 and el[i - 4] != 0:
      score -= 0.98 ** (30 - ( i % 4 ))
    if el[i] == 2 and el[i - 4] == 1 and el[i - 8] == 2:
      score -= 0.98 ** (30 - ( i % 4 ))
      score -= 6.3
    if el[i] == 2 and el[i - 4] == 2 and el[i - 8] == 1:
      score -= 0.98 ** (30 - ( i % 4 ))
    if el[i] == 2 and el[i - 4] == 1 and el[i - 8] == 1:
      score -= 6.3
    if el[i] == 2 and el[i - 4] == 0 and el[i - 8] == 2:
      score -= 6.3
    if el[i] == 1 and el[i - 4] == 1 and el[i - 8] == 0:
      score -= 2.2
  return score
```

The function treats tuple of coordinates (i, i+4, i+8) together

New symmetry of the problem

Restrict search space to find symmetric solutions

Why is searching in function space working surprisingly well?

Why is searching in function space working surprisingly well?

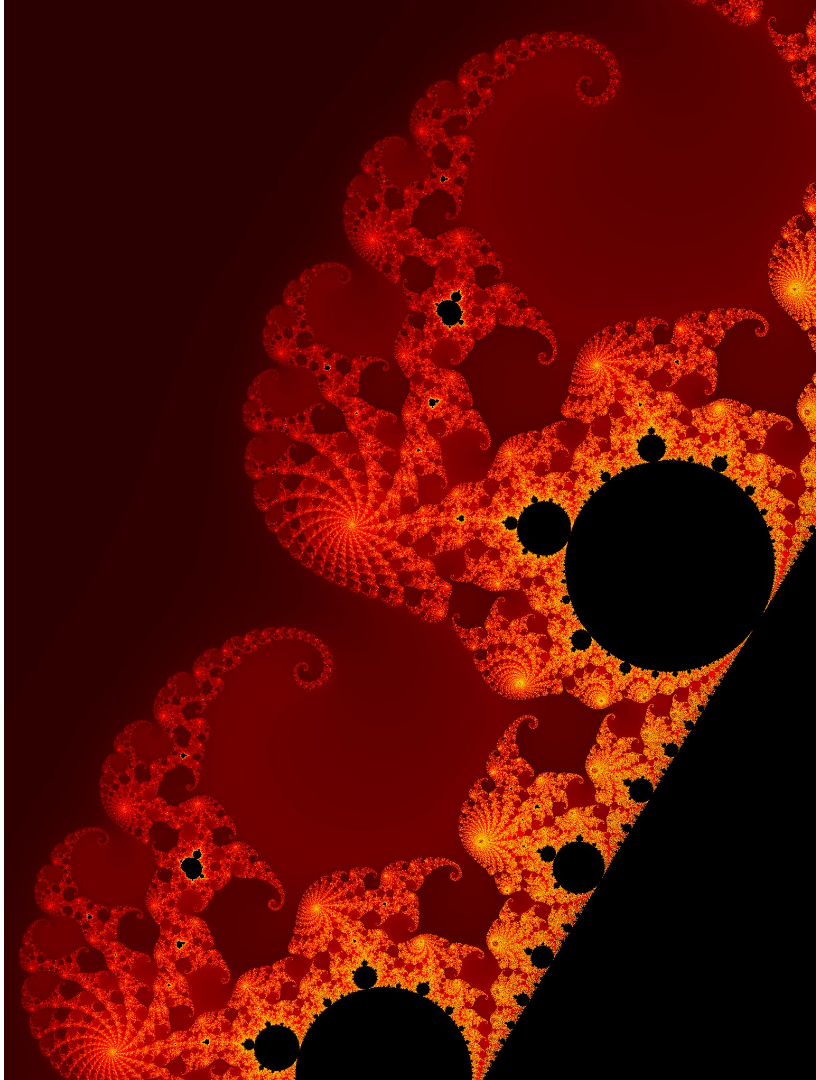
Our hypothesis: Most problems we care about are structured → solutions have small *Kolmogorov complexity*.

Kolmogorov complexity (KC): KC(y) = length of the shortest computer program that outputs y

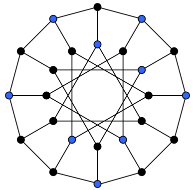
KC("abababababababab") << KC("kasjiovmoisoeimpsl")

FunSearch's implicit prior is encouraging solutions with concise functional description.

By searching in the program space, we are implicitly looking for objects with small KC.



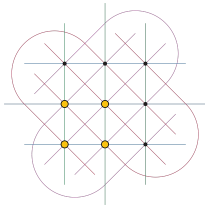
Our main results



Largest independent set

Find largest independent set in a graph

⟶ NP-hard problem



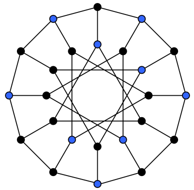
Particular focus on a structured graph (*cap-set graph*), with high mathematical significance

“Perhaps my favourite open question”

Terence Tao

FunSearch finds constructions that improve over existing state-of-the-art

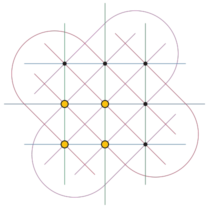
Our main results



Largest independent set

Find largest independent set in a graph

⟶ NP-hard problem



Particular focus on a structured graph (*cap-set graph*), with high mathematical significance

"Perhaps my favourite open question"

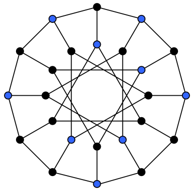
Terence Tao

FunSearch finds constructions that improve over existing state-of-the-art

Online bin-packing problem

What is the most resource efficient way to pack items onto bins?

Our main results

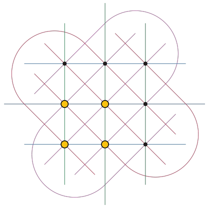


Largest independent set

Find largest independent set in a graph

⟶ NP-hard problem

Particular focus on a structured graph (*cap-set graph*), with high mathematical significance



"Perhaps my favourite open question"

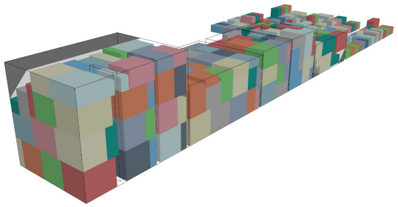
Terence Tao

FunSearch finds constructions that improve over existing state-of-the-art

Online bin-packing problem

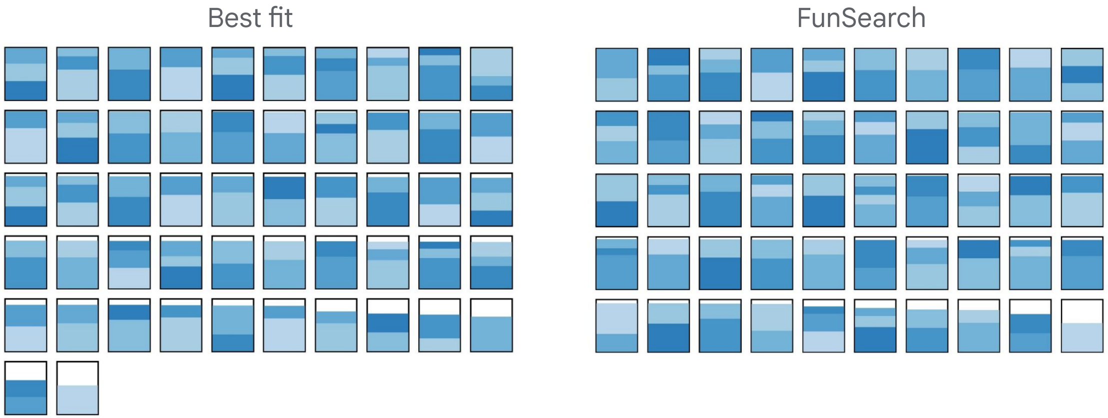
What is the most resource efficient way to pack items onto bins?

It sits at the core of many real-world problems, from loading containers with items to allocating compute jobs in data centers to minimize costs.

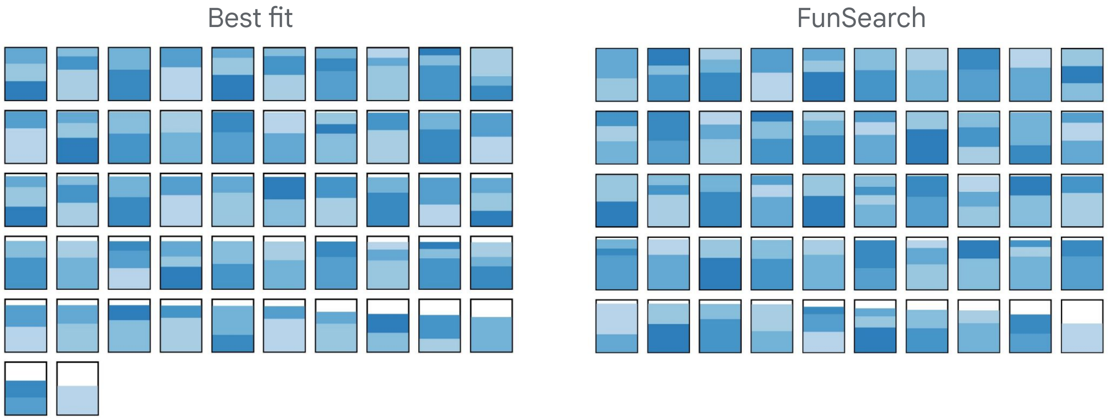


FunSearch delivers automatically tailored programs that outperformed established heuristics

FunSearch for online bin packing

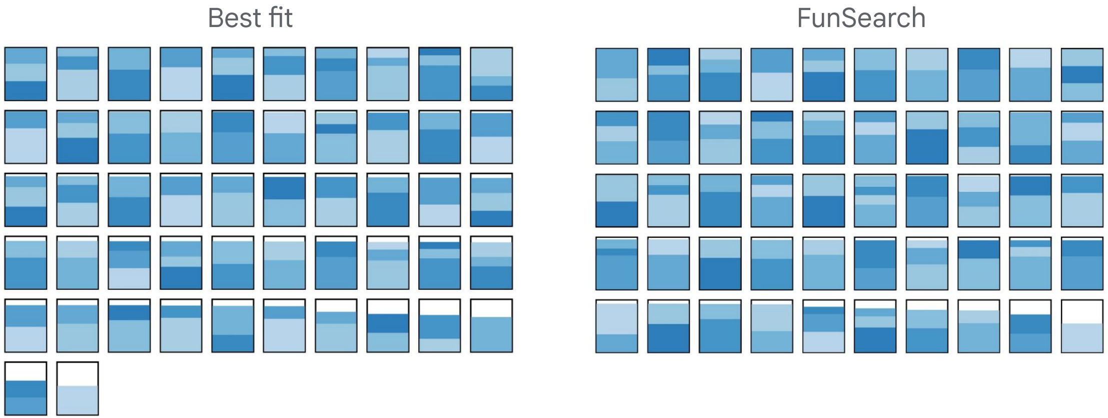


FunSearch for online bin packing




```python
def heuristic(item: float, bins: np.ndarray) -> np.ndarray:
  """Online bin packing heuristic discovered with FunSearch."""
  score = 1000 * np.ones(bins.shape)
  # Penalize bins with large capacities.
  score -= bins * (bins-item)
  # Extract index of bin with best fit.
  index = np.argmin(bins)
  # Scale score of best fit bin by item size.
  score[index] *= item
  # Penalize best fit bin if fit is not tight.
  score[index] -= (bins[index] - item)**4
  return score
```

FunSearch for online bin packing




```python
def heuristic(item: float, bins: np.ndarray) -> np.ndarray:
  """Online bin packing heuristic discovered with FunSearch."""
  score = 1000 * np.ones(bins.shape)
  # Penalize bins with large capacities.
  score -= bins * (bins-item)
  # Extract index of bin with best fit.
  index = np.argmin(bins)
  # Scale score of best fit bin by item size.
  score[index] *= item
  # Penalize best fit bin if fit is not tight.
  score[index] -= (bins[index] - item)**4
  return score
```

	First fit	Best fit	**FunSearch**
OR1	6.42%	5.81%	**5.30%**
OR2	6.45%	6.06%	**4.19%**
OR3	5.74%	5.37%	**3.11%**
OR4	5.23%	4.94%	**2.47%**
Weibull 5K	4.23%	3.98%	**0.68%**
Weibull 10K	4.20%	3.90%	**0.32%**
Weibull 100K	4.00%	3.79%	**0.03%**

FunSearch for tailoring programs

This showcases how FunSearch can be used to **automatically** produce programs / strategies that are **adapted** to a specific use case

Unlike neural networks - based approaches, the output of FunSearch is code:

- It is easier to deploy (no need for specialized hardware)
- It is easier to debug
- More understandable
- More predictable
- More scalable

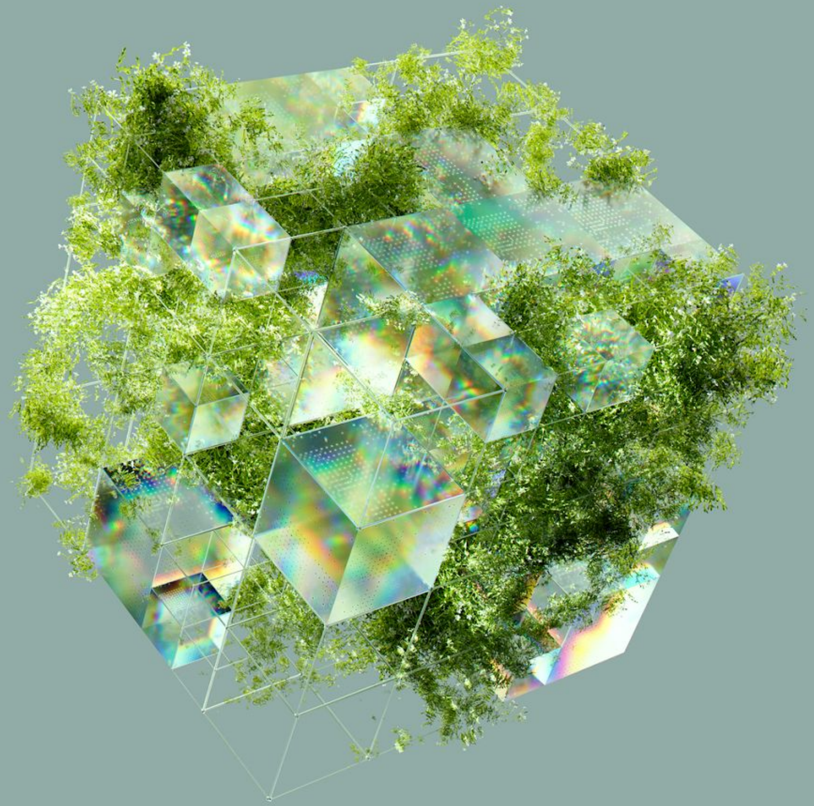
FunSearch principles

LLMs by themselves cannot solve complex problems

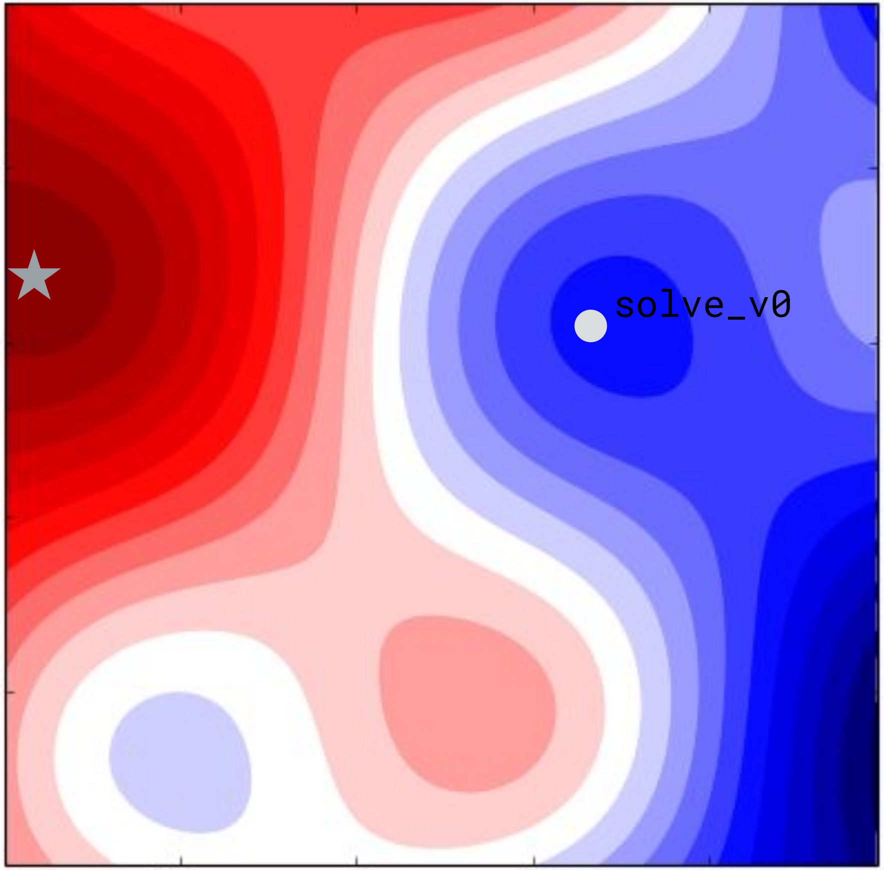
- They often come up with plausible but wrong outputs
- But we can couple it with Python runtime to provide grounding

We need to do search in the space of functions

- Search based on evolutionary algorithms

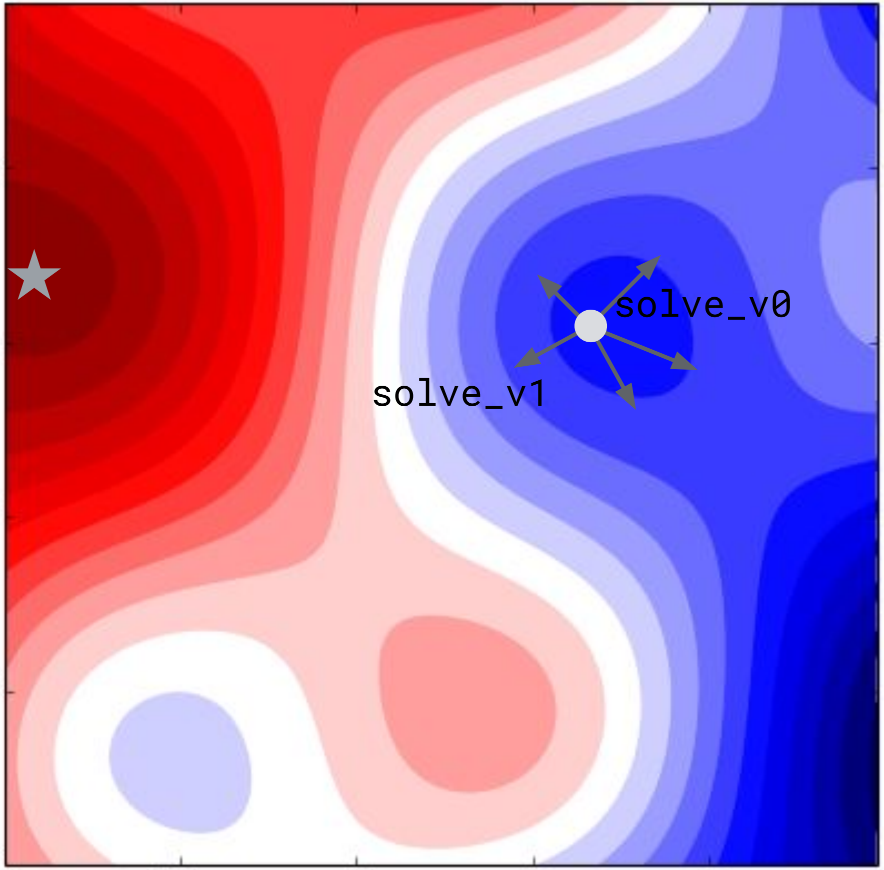


Score map of functions




```python
def solve_v0(n):
    # Trivial implementation
    return [(0,) * n]
```

Score map of functions

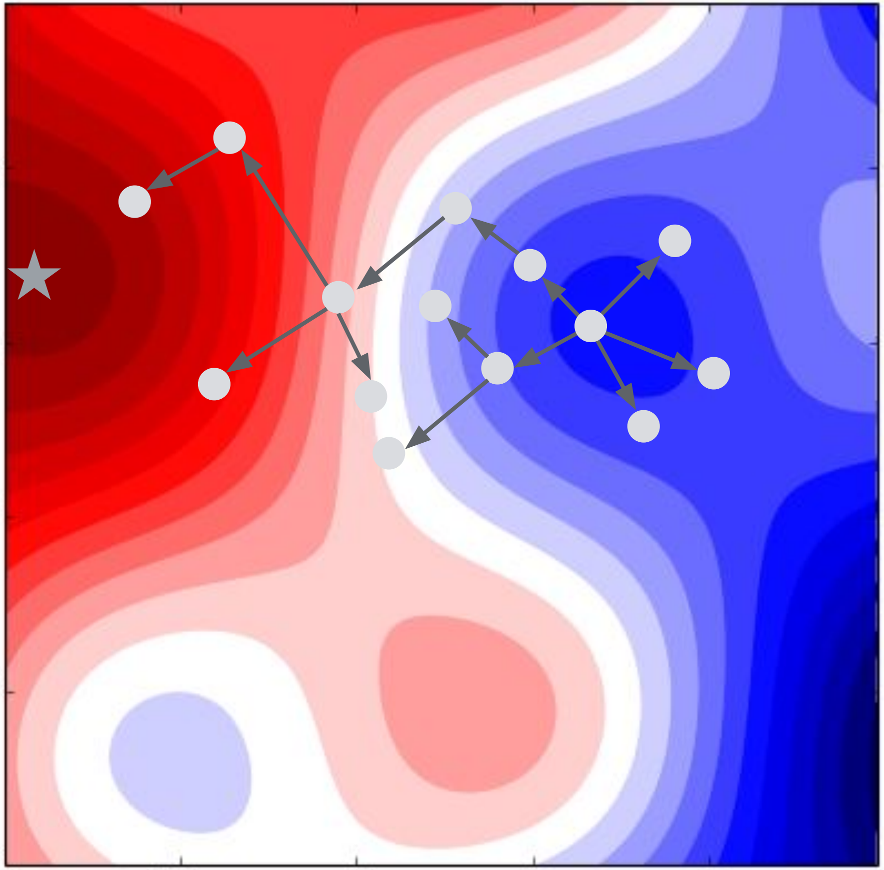


Sample a large number of **mutations** from the LLM

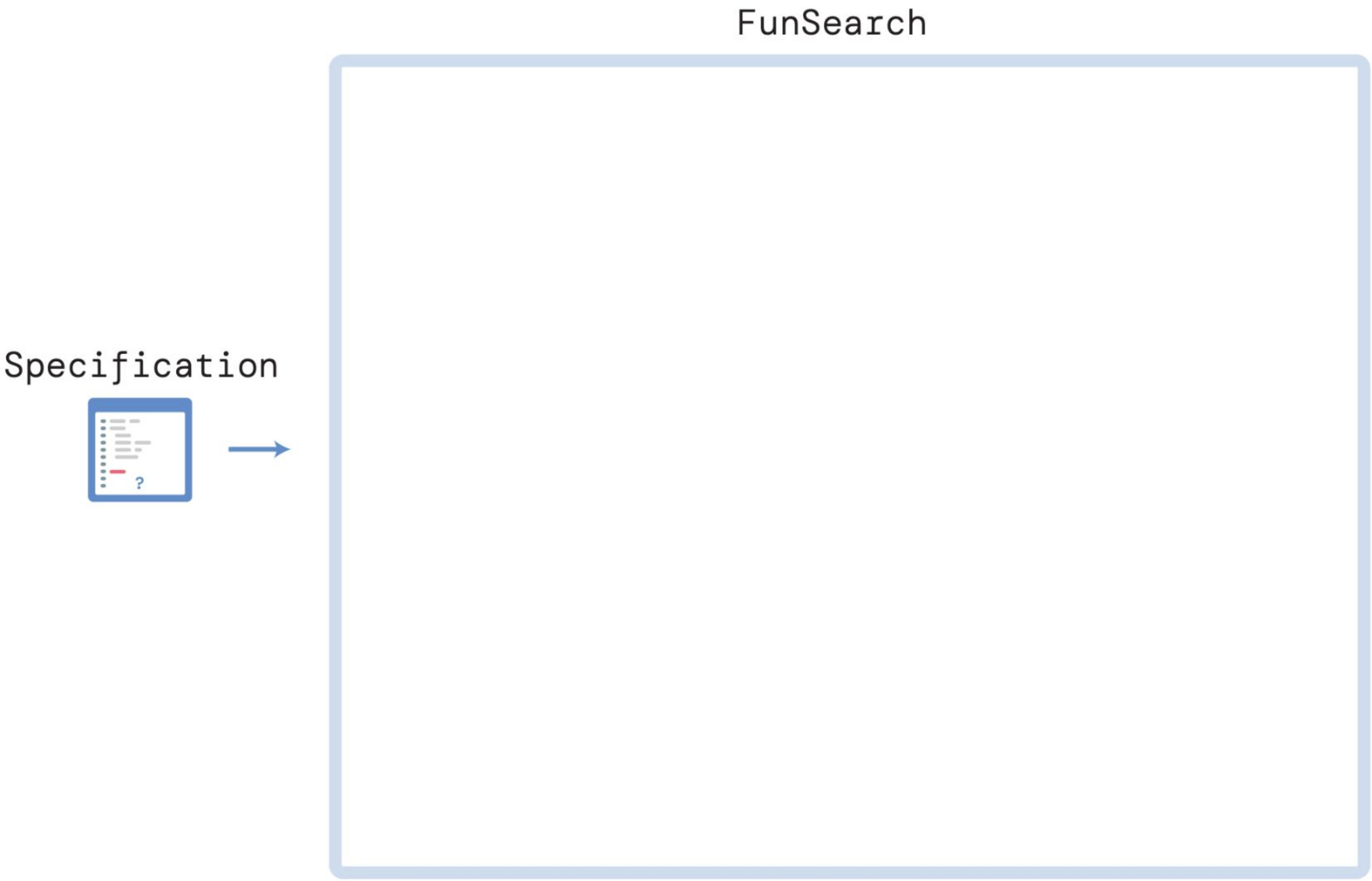
```
def solve_v0(n):
    # Trivial implementation
    return [(0,) * n]

def solve_v1(n):
    # Improve over `solve_v0`
    (To be completed by LLM)
```

Very unlikely to reach

Score map of functions

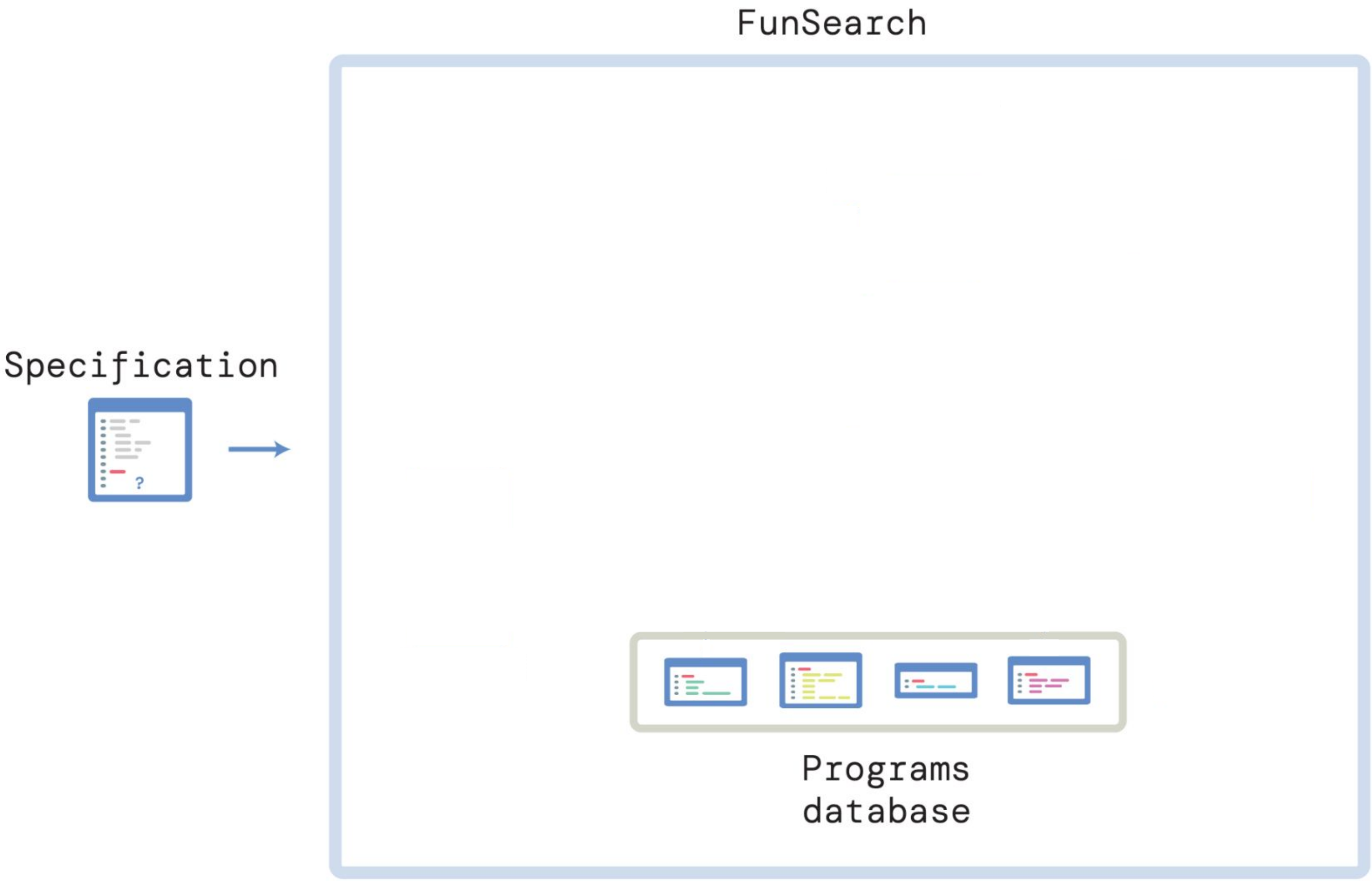


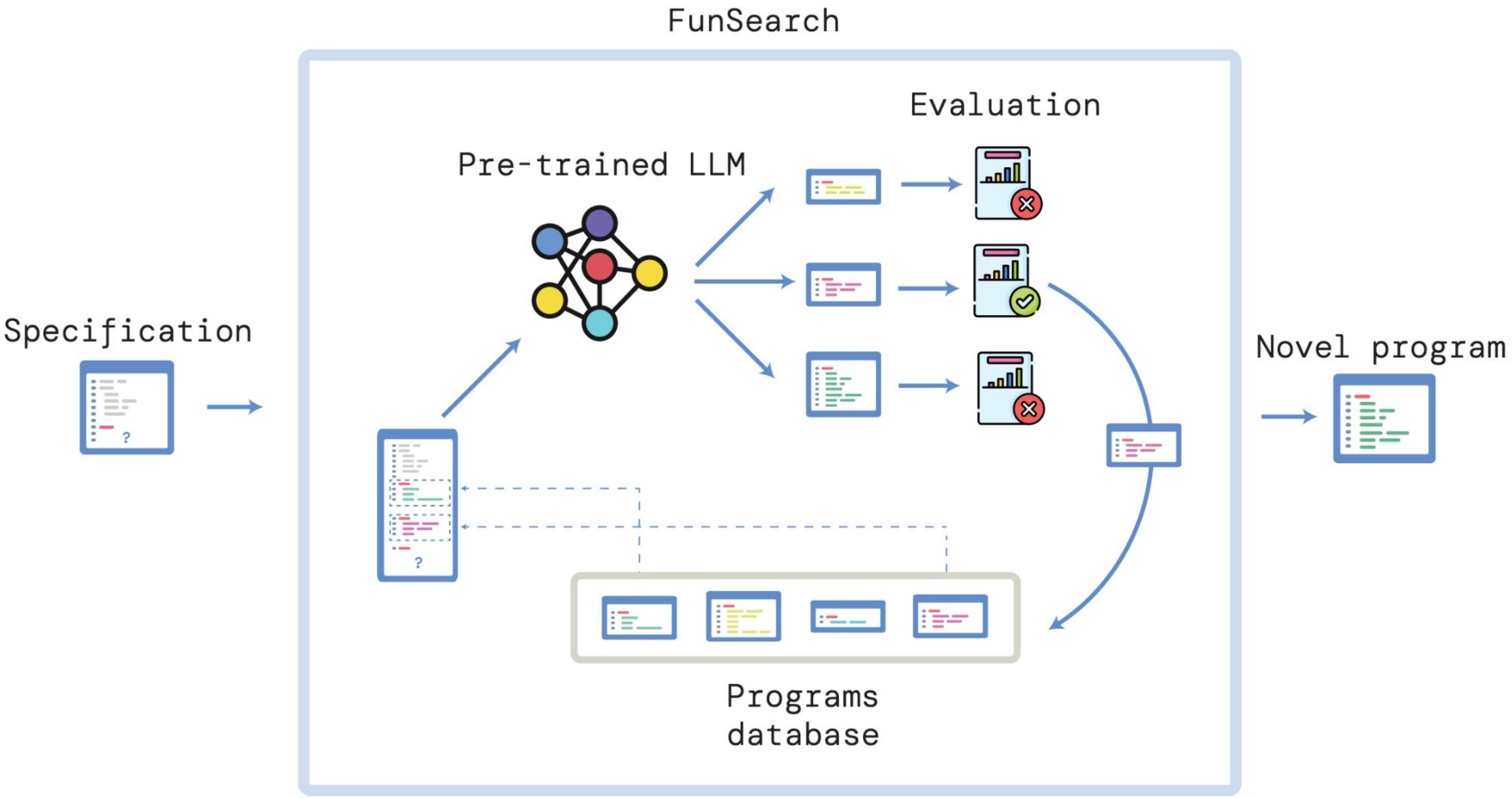
Chaining mutations


FunSearch
Specification

Specification

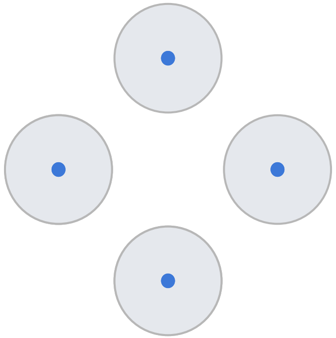
```
@funsearch.evolve
def get_selection_score(state):
  return 0.0

@funsearch.run
def evaluate(params):
  load_dataset(params)
  … # Greedy algorithm begins
  score = get_selection_score(state)
  … # Greedy algorithm ends
  eval_score = compute_eval_score(score)
  return eval_score
```


FunSearch
Specification
?
Programs database


FunSearch
Specification
Pre-trained LLM
Evaluation
Novel program
Programs database

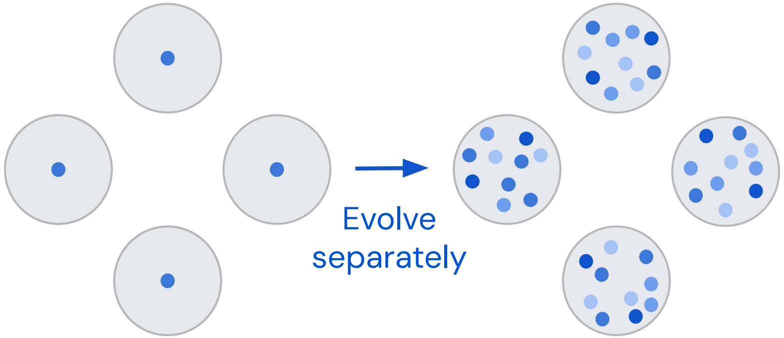
Maintaining diversity with islands 🏝️

Instead of evolving a single population, evolve several populations separately on different "islands" and occasionally allow "migration" between islands.



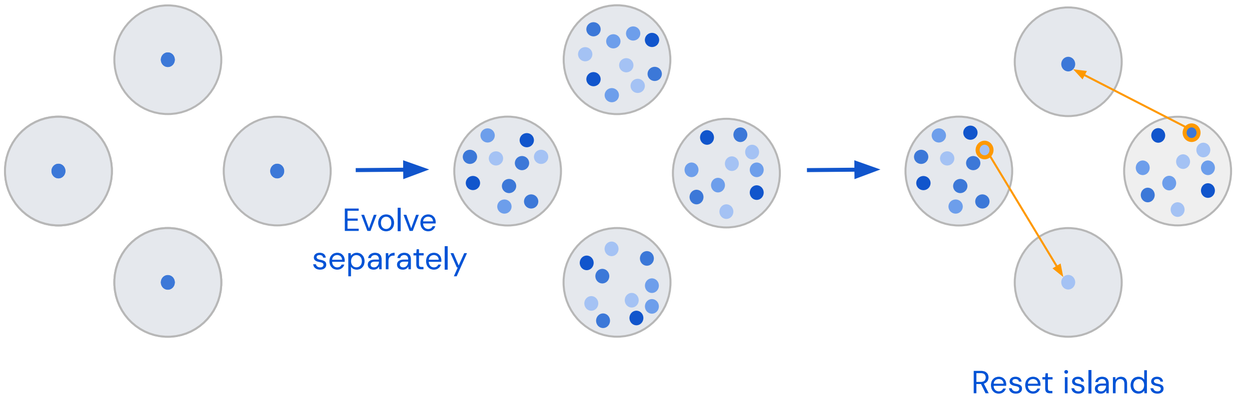
Maintaining diversity with islands 🏝️

Instead of evolving a single population, evolve several populations separately on different "islands" and occasionally allow "migration" between islands.



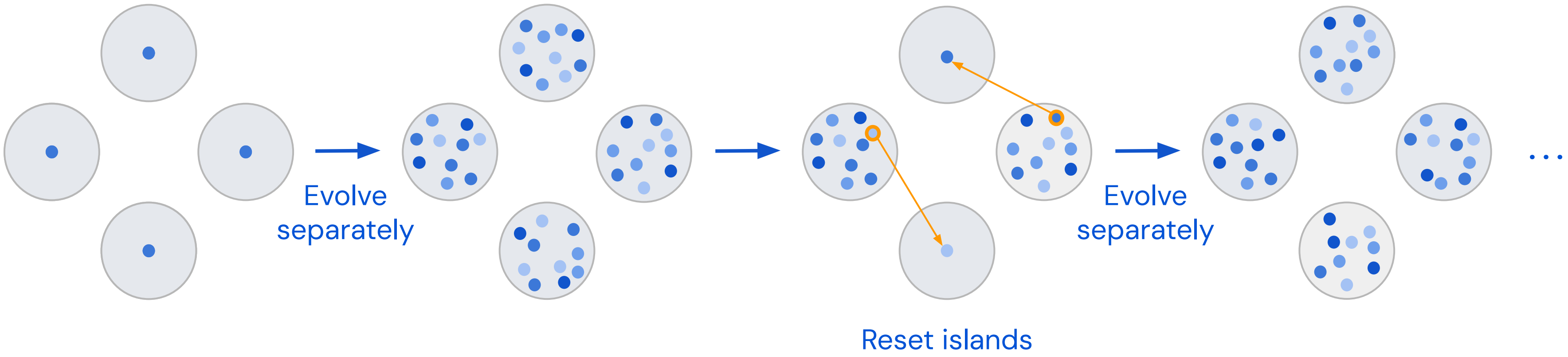
Maintaining diversity with islands 🏝️

Instead of evolving a single population, evolve several populations separately on different “islands” and occasionally allow “migration” between islands.



Maintaining diversity with islands 🏝️

Instead of evolving a single population, evolve several populations separately on different “islands” and occasionally allow “migration” between islands.



Some more details

Pretrained LLMs

- Trade-off between capabilities and speed: Codey (Palm 2)
- No gradients were computed in the making of these experiments

Distributed system

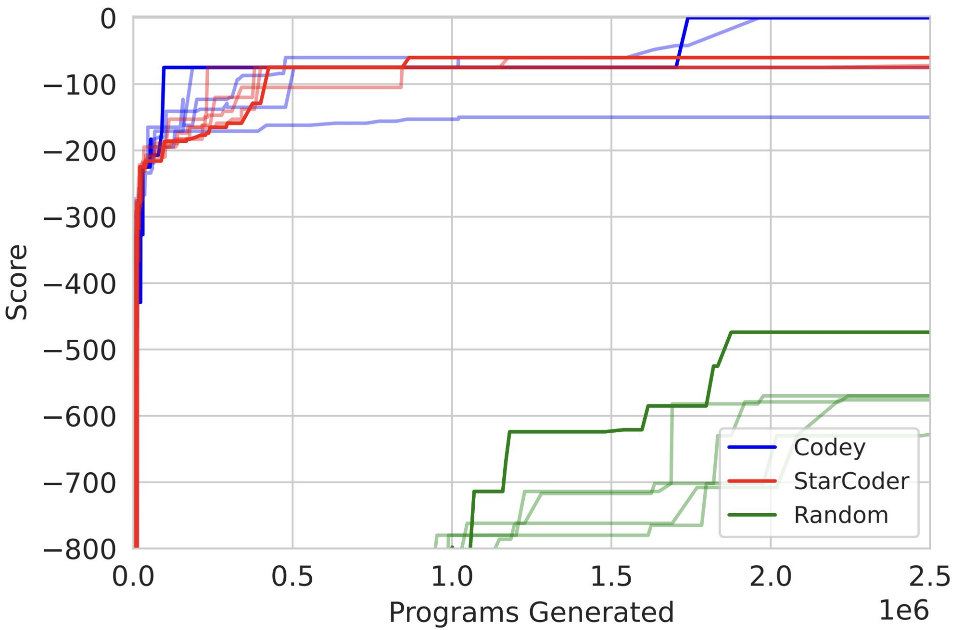
We asynchronously connect 15 LLM samplers to 100s evaluators and a central programs database.

Skeleton

```python
def solve(n):
  """Builds a cap set using `priority` function."""
  # Precompute all priority scores.
  elements = utils_capset.get_all_elements(n)
  scores = [priority(el, n) for el in elements]
  # Sort elements according to the scores.
  elements = elements[np.argsort(scores, kind='stable')[::-1]]

  # Build `capset` greedily, using scores for prioritization.
  capset = []
  for element in elements:
    if utils_capset.can_be_added(element, capset):
      capset.append(element)
  return capset

@funsearch.evolve
def priority(element, n):
  """Returns the priority with which we want to add `element`
to the cap set."""
  return 0.0
```

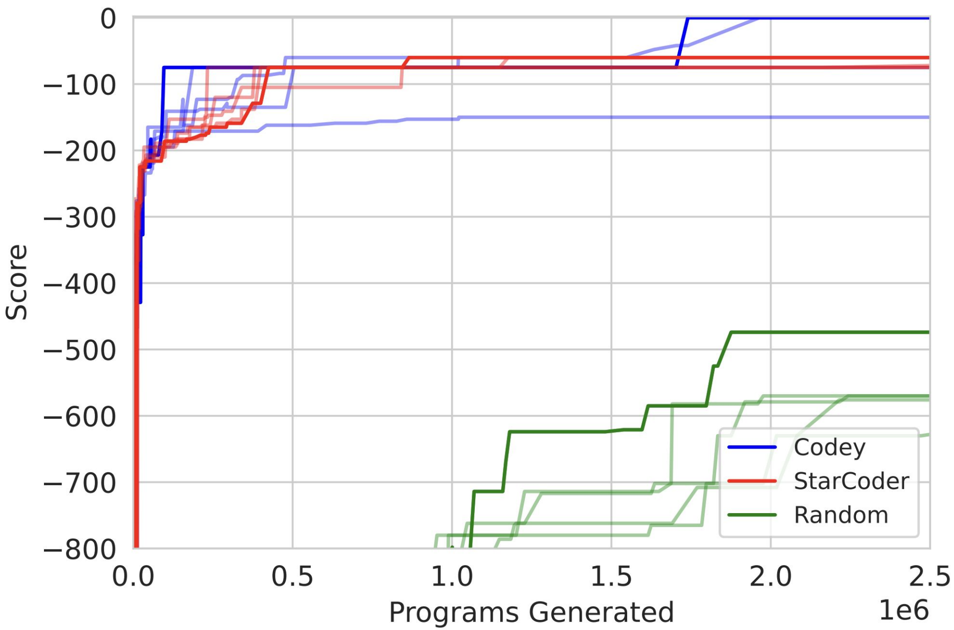
Ablation experiments

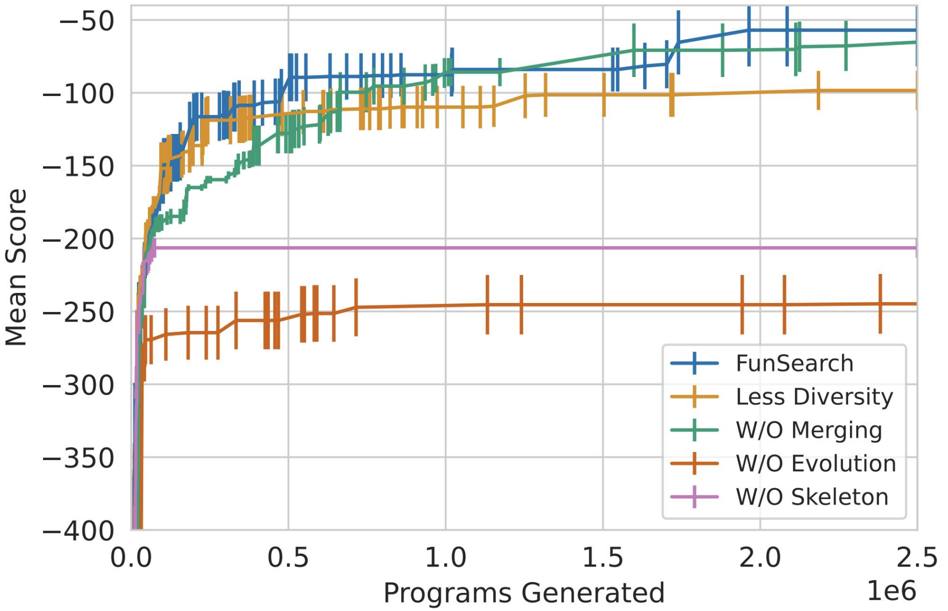
Using admissible sets (asymptotic cap set)



Ablation experiments

Using admissible sets (asymptotic cap set)



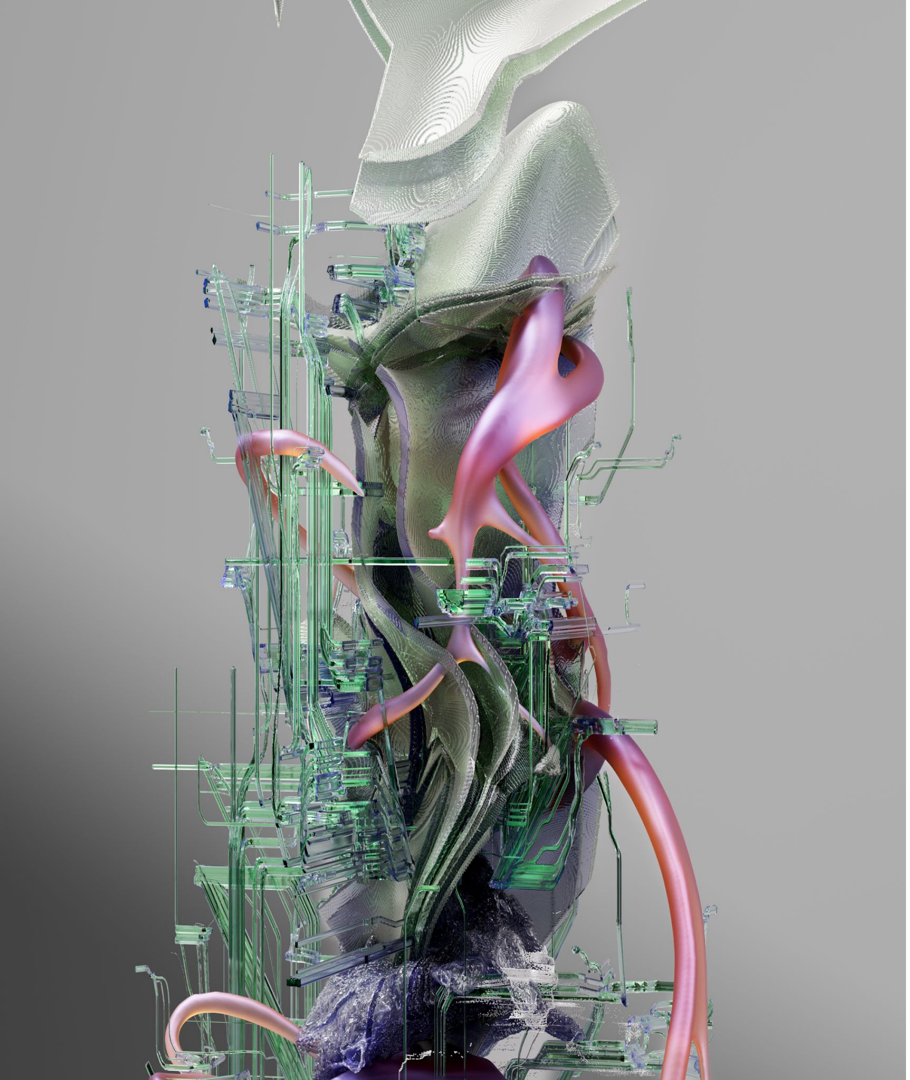


For which problems is FunSearch useful?

1. Efficient evaluator is available

2. Smooth scoring feedback → it is possible to gradually improve the solution

3. Prior information about the problem → good skeleton



This is just the beginning

In this work we have striven for **simplicity**, creating a strong and simple base that we can build on in the future.

This, together with LLMs getting more powerful, faster and cheaper make us highly confide

FunSearch Team

Bernardino Romera Paredes

Amin Barekatain

Alexander Novikov

Matej Balog

Pawan Mudigonda

Emilien Dupont

Francisco Ruiz

Alhussein Fawzi

Jordan Ellenberg

Pengming Wang

Omar Fawzi

Pushmeet Kohli

FunSearch Team

Bernardino Romera Paredes

Amin Barekatain

Alexander Novikov

Matej Balog

Pawan Mudigonda

Emilien Dupont

Francisco Ruiz

Alhussein Fawzi

Jordan Ellenberg

Pengming Wang

Omar Fawzi

Pushmeet Kohli

Thank you