

FUNSEARCH: PROGRAM SEARCH USING LLMS



Omar FAWZI
QInfo, LIP, ENS Lyon

Collaboration with Google DeepMind




nature

[Explore content](#) ▾ [About the journal](#) ▾ [Publish with us](#) ▾

[nature](#) > [articles](#) > [article](#)

Article | [Open access](#) | Published: 14 December 2023

Mathematical discoveries from program search with large language models

[Bernardino Romera-Paredes](#) , [Mohammadamin Barekatin](#), [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](#)

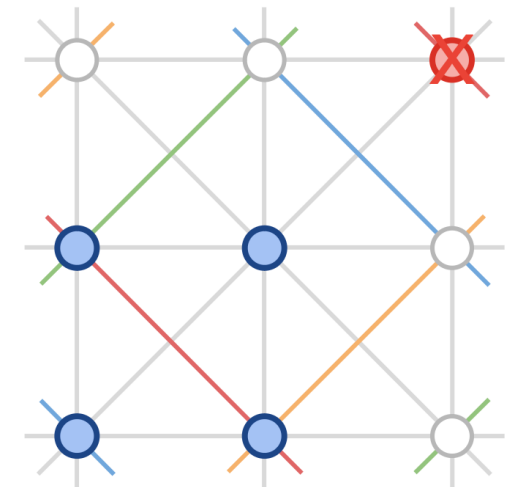
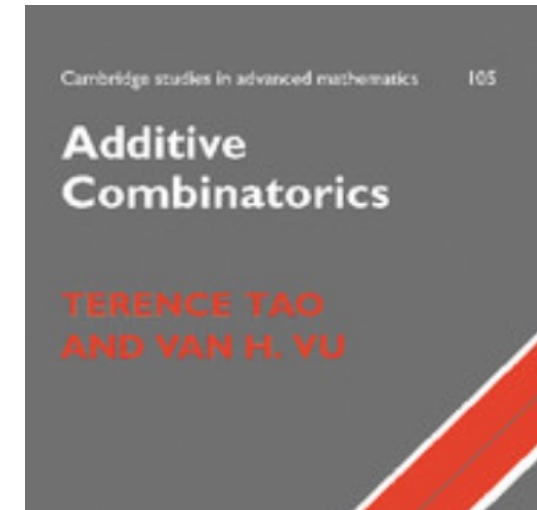
The cap set problem

Additive combinatorics

- ▶ A group G , e.g., $G = \mathbb{Z}$, $G = \mathbb{Z}_N$
- ▶ Objective: construct a large set $A \subset G$ with no arithmetic progressions

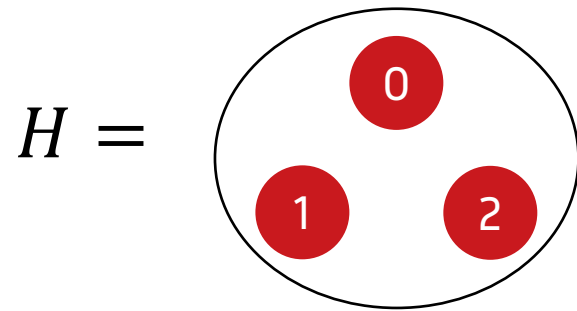
The cap set problem

- ▶ A cap set: A subset $A \subset \mathbb{Z}_3^n$ with no arithmetic progressions of length 3
- ▶ Main question: as $n \rightarrow \infty$: $|A| \sim c^n$ how large can c be?
 - Lower bound: Trivial $c \geq 2$, important work: $c \geq 2.217$
[Edel, *Design, Codes and Cryptography*, 2002]
 - Upper bound: Trivial $c \leq 3$, first improvement: $c \leq 2.75$
[Ellenberg, Gijswijt, *Annals of Mathematics*, 2017]
- ▶ Also of interest for finite n
 - $n = 4$ card game SET
 - Open for $n = 8$ [Encyclopedia of Integer Sequences, <https://oeis.org/A090245CT>]
- ▶ Relation to other problems:
 - Algorithms for matrix multiplication
 - Circuit complexity



Cap set for $n=2$

Cap sets are independent sets of a simple hypergraph



Cap set = Independent set in $H^{\boxtimes n}$

Dimension n	1	2	3	4	8	...	$n \rightarrow \infty$
#nodes	3	9	27	81	6561	...	3^n
#edges	1	12	117	1080	$\sim 7,000,000$		$\sim 9^n$
Prev. best construction	2	4	9	20	496	...	2.217^n

As a computational problem

- Maximum independent set NP-hard in general
- Hypergraph is of exponential size in n
- Any generic algorithm: doubly exponential in n

Approach we take here:

- Describe an independent set by a **program** instead of a list of vertices
- Search for code describing a large indep set

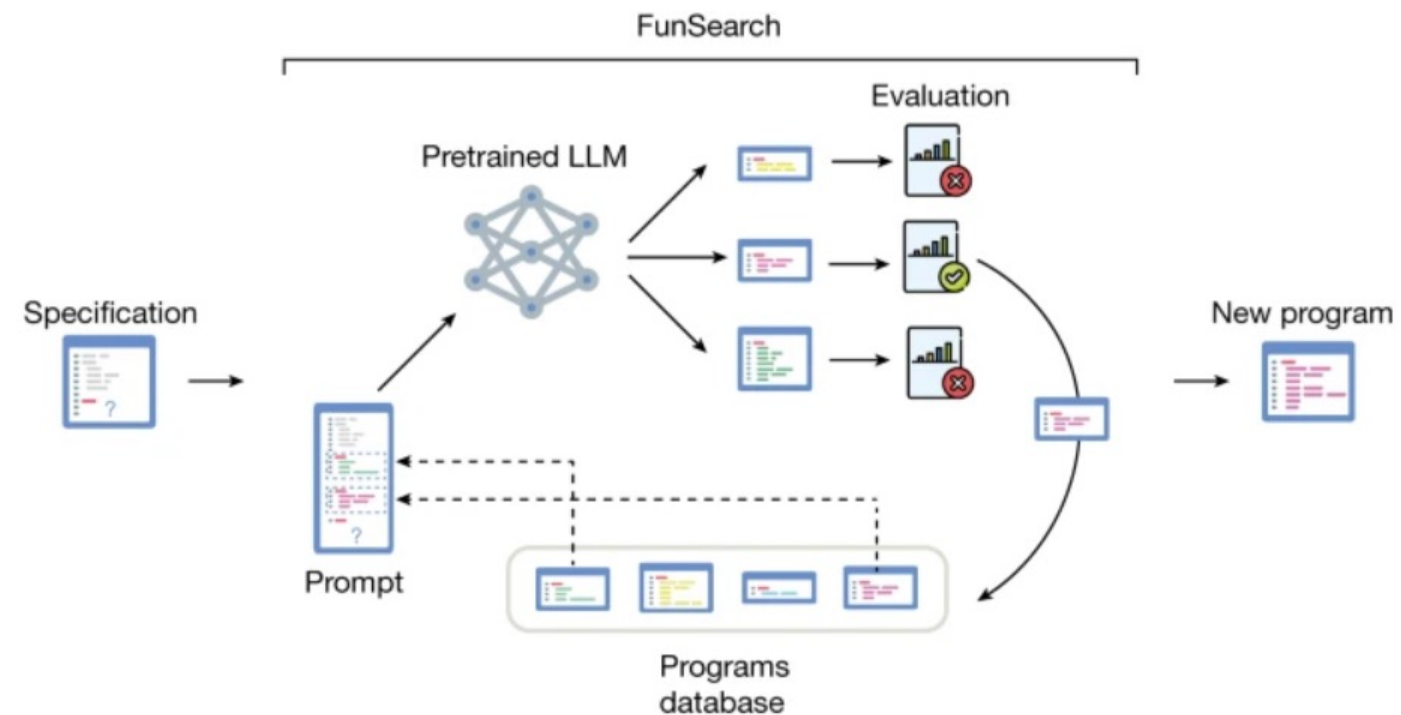
FunSearch (search in the **function** space)

- ▶ Asking LLM directly does not work (we do not trust it)

FunSearch combines

- ▶ Trusted & fast verification (problem is in NP)
- ▶ Large language model (generates meaningful programs)

Fig. 1: Overview of FunSearch.



FunSearch applied to the cap set problem

Choosing a good skeleton is crucial

- ▶ We do not search for a program outputs the list directly
- ▶ Instead, define a greedy algorithm: adding vertices in order defined by priority
- ▶ Search for priority function
- ▶ evaluate will output the size of the independent set generated by greedy algorithm with priority

Why?

- ▶ Boilerplate code factored out
- ▶ Guaranteed to give a cap set

a

```
"""Finds large cap sets."""
import numpy as np
import utils_capset

# Function to be executed by FunSearch.
def main(n):
    """Runs `solve` on `n`-dimensional cap set and
    ↪ evaluates the output."""
    solution = solve(n)
    return evaluate(solution, n)

def evaluate(candidate_set, n):
    """Returns size of candidate_set if it is a cap
    ↪ set, None otherwise."""
    if utils_capset.is_capset(candidate_set, n):
        return len(candidate_set)
    else:
        return None

def solve(n):
    """Builds a cap set of dimension `n` 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

# Function to be evolved by FunSearch.
def priority(element, n):
    """Returns the priority with which we want to add
    ↪ `element` to the cap set."""
    return 0.0
```

Results

Dimension n	1	2	3	4	8	...	$n \rightarrow \infty$
#nodes	3	9	27	81	6561	...	3^n
#edges	1	12	117	1080	$\sim 7,000,000$		$\sim 9^n$
Prev. best construction	2	4	9	20	496	...	2.217^n
Using FunSearch	2	4	9	20	<u>512</u>		<u>2.220^n</u>

Based on finding a finite construction ("admissible set") that implies the asymptotic bound

Why working well here?

- Problem is very structured. Can hope for a structured solution
- Scales better than traditional search methods, e.g., SAT solvers

Raw search space vs function space

Raw list of vertices

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

Program that outputs list of vertices (simplified by hand)

```
78 def get_capset(n: int) -> CapSet:
79     """Returns a 512-cap in AG(8, 3)."""
80     V = np.array(list(itertools.product(range(3), repeat=n)), dtype=np.int32)
81     reflections = lambda v: sum(1 for i in range(1, n // 2) if v[i] == v[-i])
82
83     # First we list 128 weight-8 vectors with >= 2 reflections.
84     weight8_points = [v for v in V
85                       if np.count_nonzero(v) == 8 # Weight is 8.
86                          and reflections(v) >= 2] # At least 2 reflections.
87
88     # Then we list 256 weight-4 vectors with allowed support and <= 1 reflections.
89     allowed_supports = [
90         (0, 1, 2, 3), (0, 1, 2, 5), (0, 1, 2, 7), (0, 1, 2, 6), (0, 1, 3, 7),
91         (0, 1, 6, 7), (0, 3, 6, 7), (0, 5, 6, 7), (0, 1, 5, 7), (1, 3, 4, 6),
92         (1, 4, 5, 6), (0, 2, 3, 6), (2, 3, 4, 7), (2, 4, 5, 7), (0, 2, 6, 7),
93         (0, 2, 5, 6), (1, 2, 4, 7), (1, 2, 4, 6), (1, 3, 4, 7), (1, 4, 6, 7),
94         (1, 4, 5, 7), (2, 3, 4, 6), (2, 4, 6, 7), (2, 4, 5, 6),
95     ]
96     weight4_points = [
97         v for v in V
98         if np.count_nonzero(v) == 4 # Weight is 4.
99            and tuple(i for i in range(n) if v[i] != 0) in allowed_supports
100            and reflections(v) <= 1] # At most 1 reflection.
101
102     # Finally we add 128 weight-5 vectors with <= 1 reflections.
103     allowed_zeros = [(0, 4, 7), (0, 2, 4), (0, 1, 4), (0, 4, 6),
104                     (1, 2, 6), (2, 6, 7), (1, 2, 7), (1, 6, 7)]
105     weight5_points = [
106         v for v in V
107         if np.count_nonzero(v) == 5 # Weight is 4.
108            and tuple(i for i in range(n) if v[i] == 0) in allowed_zeros
109            and reflections(v) <= 1 # At most 1 reflection.
110            and (v[1] * v[7]) % 3 != 1 and (v[2] * v[6]) % 3 != 1 # Mod conditions.
111
112     return weight8_points + weight4_points + weight5_points
```

Programs can be more insightful

Actionable interpretability

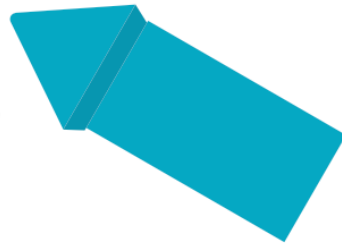
```
def priority(el: tuple[int, ...], n: int, w: int) -> float:
    score = 0.0
    for i in range(n):
        if el[i] == 1:
            score -= 0.9 ** (i % 4)
        if el[i] == 2:
            score -= 0.98 ** (30 - (i % 4))
        if el[i] == 1 and el[i - 4] == 1:
            score -= 0.98 ** (30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



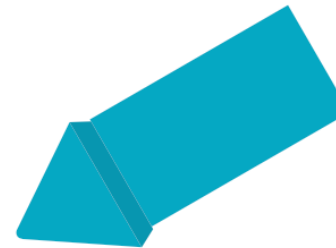
The function treats tuple of coordinates (i, i+4, i+8) together



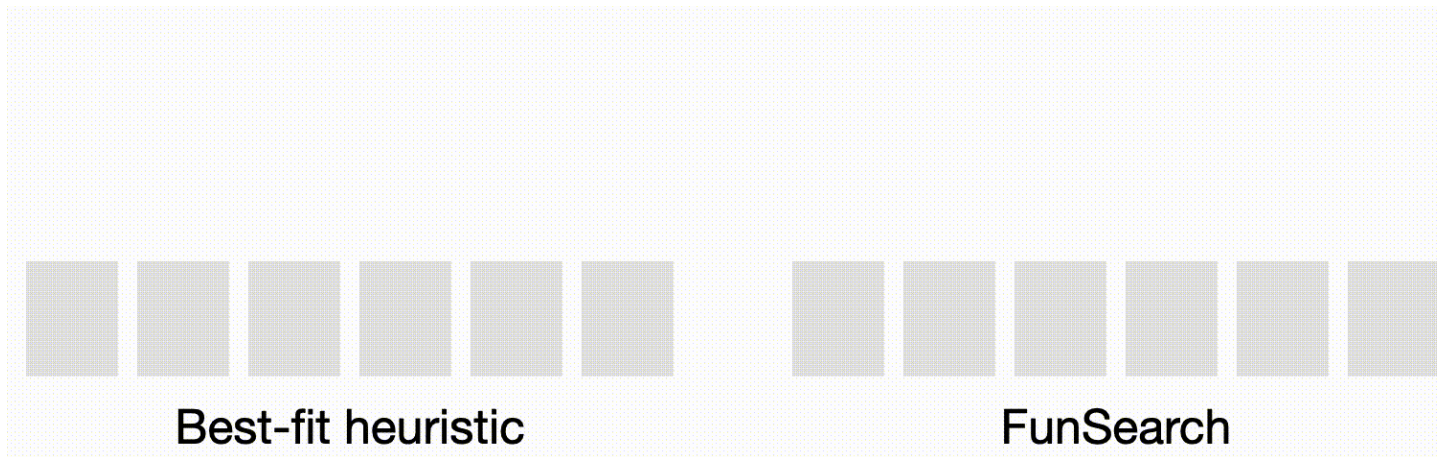
Relevant symmetry of the problem



Restrict search space to find symmetric solutions



Another application: online bin packing



```
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
```

	OR1	OR2	OR3	OR4	Weibull 5k	Weibull 10k	Weibull 100k
First Fit	6.42%	6.45%	5.74%	5.23%	4.23%	4.20%	4.00%
Best Fit	5.81%	6.06%	5.37%	4.94%	3.98%	3.90%	3.79%
<i>FunSearch</i>	5.30%	4.19%	3.11%	2.47%	0.68%	0.32%	0.03%

Summary

Search paradigm

Meaningful creativity of LLM

+

Formal verification skeleton

What characteristics of the problem were important?

- Fast evaluation with signal of performance
- Search is difficult
- Searching for “short” programs

In 2026, this approach is applied very widely

- AlphaEvolve helped in advancing hundreds of maths problems
- Scaling this approach has computational/engineering cost

FunSearch to find Sum-of-Squares proofs (work with Mostafa Taheri)

Polynomial optimization

- Many problems can be formulated as: compute $\min_{x \in R^d} P(x)$ or prove $P(x) \geq \gamma \forall x \in R^d$.
- Combinatorial optimization, control and robotics, quantum information,...
- If $P(x) - \gamma = \sum_i R_i(x)^2$, then R_i provide a proof that $\min_{x \in R^d} P(x) \geq \gamma$.
- Searching for an SoS is a convex problem (semidefinite program):

$$P(x) - \gamma = \vec{b}^t Q \vec{b}$$

where \vec{b} is a vector of monomials and $Q \geq 0$

Example

$$P(x) = (x_1 + x_2)^2 + (x_3 + x_4)^2 + (x_4 + x_5)^2$$

$$\vec{b} = (x_1, x_2, x_3, x_4, x_5)$$

$$Q = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Q has many zeros

→ Find \vec{b} to reduce size of convex problem

Example application: Entanglement witness

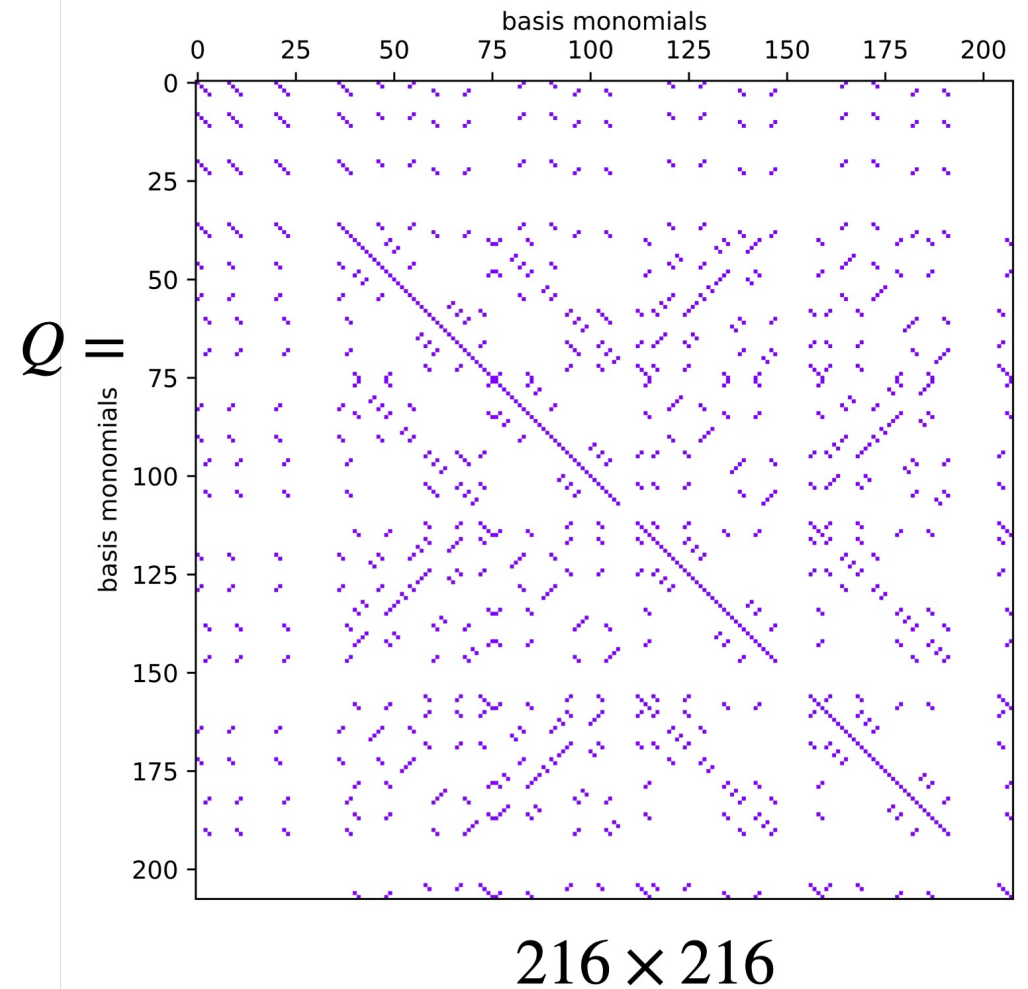
W is an entanglement witness if it is positive on product states:

$$P(x, y) = \langle x | \otimes \langle y | W | x \rangle \otimes | y \rangle \geq 0$$

If $\text{tr}(W\rho) < 0$, then ρ is entangled

Example:

$$\begin{aligned} W = & (|22\rangle - |00\rangle)(\langle 22| - \langle 00|) + (|22\rangle - |11\rangle)(\langle 22| - \langle 11|) \\ & + (|33\rangle - |01\rangle)(\langle 33| - \langle 01|) + (|33\rangle - |10\rangle)(\langle 33| - \langle 10|) \\ & + |23\rangle\langle 23| + |32\rangle\langle 32| - |22\rangle\langle 22| - |33\rangle\langle 33| \end{aligned}$$



Search problem: Good basis sets

Given P find \vec{b} or family $\{\vec{b}_1, \vec{b}_2, \dots\}$ such that

$$P(x) - \gamma = \sum_i \vec{b}_i^t Q_i \vec{b}_i$$

so that the number of variables in Q_i is small

Gain from a family $\{\vec{b}_1, \vec{b}_2, \dots\} \cong \frac{|\#monomials|^2}{\sum_i |\vec{b}_i|^2}$

Example

$$P(x) = (x_1 + x_2)^2 + (x_3 + x_4)^2 + (x_4 + x_5)^2$$

$$\vec{b}_1 = (x_1, x_2), \vec{b}_2 = (x_3, x_4), \vec{b}_3 = (x_4, x_5)$$

$$Q_1 = Q_2 = Q_3 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Number of variables: 9

Results (preliminary)

- Use same approach: search for code describing $\{\vec{b}_1, \vec{b}_2, \dots\}$
- For entanglement witness W : found a more concise sum-of-squares decomposition
- Another related family:

$$X(\alpha) := I_d \otimes I_d + \alpha d |\phi\rangle\langle\phi|, \quad X_k(\alpha) := |\phi_k\rangle\langle\phi_k| \otimes X(\alpha).$$

$$P(x, y) := \langle xy | X_2(\alpha) | xy \rangle \quad \max_{\gamma} \text{ s.t. } P - \gamma \in \text{SOS}$$

α	Full SDP			FunSearch			Fine-tuned FunSearch		
	Gain	Value	time(s)	Gain	Value	time(s)	Gain	Value	time(s)
-0,6	1	-0,14	85	30,11	-0,477	0,08	16	-0,14	0,36
-0,7	1	-0,18	71	30,11	-0,566	0,08	16	-0,18	0,37
-0,8	1	-0,22	87	30,11	-0,655	0,08	16	-0,22	0,37
-0,9	1	-0,26	73	30,11	-0,74	0,08	16	-0,26	0,37
-1,0	1	-0,30	74	40,65	-1,22	0,08	16	-0,30	0,36

- With open language models that run on local computers and open source ShinkaEvolve framework

**Please reach out if you are interested
in similar problems**

