Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model

ICML 2024

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, Qingfu Zhang

Stanford CS/MS&E 331

Motivation

Heuristics underpin many optimization pipelines

- Yet they're manually crafted and domain-specific

Automating their discovery is a core AI challenge

- Design spaces are vast and nonlinear

LLMs enable data-driven heuristic generation

This paper: evolves natural language "**thoughts**" & **code** jointly

- Aligns reasoning quality with executable performance
- Lower compute than prior work
  - E.g., FunSearch [Romera-Paredes et al., Nature'24]

High-level approach

Goal: co-evolve natural-language thoughts and code

Search mechanism: Maintain population of heuristics

- Use LLMs + genetic operators (crossover, mutation) for exploration
- Evaluate on problem instances; select top performers

Prompting: Guide reasoning over prior thoughts and codes

- Acts as fine-grained in-context learning during evolution

Comparison to prior work:

- EoH evolves both thoughts and code; FunSearch evolves code only

Example: Online bin packing

Task: Pack items of varying sizes into fewest bins

Items arrive sequentially

- Must be packed into a bin immediately
- No knowledge of future arrivals

Each bin has fixed capacity (experiments: $C = 100$)

Example: Online bin packing

Heuristic representation:

- Natural language "thought":

  The heuristic incorporates a weighted average of the utilization ratio, dynamic adjustment, and an exponentially decaying factor, with different parameter settings to minimize the number of used bins.

- **Code:**

```
def heuristic(item, bins):
    """
    item: scalar item size
    bins: 1D np.array of remaining capacities
    returns: per-bin scores (higher is better)
    …
```

- **Fitness:** 0.0196

Example: Online bin packing

Fitness metric:

- **Test instances** [Romera-Paredes et al., Nature'24]**:**
  Five Weibull test instances, each with 5000 items
- ℓb = lower bound on opt bin count [Martello & Toth '90]
- n = number of bins used by heuristic
- $\text{Fitness} = \text{avg}\left(\frac{\ell b}{n}\right)$ across the test instances

Algorithm overview

1. **Initialization:**
   - Generate N initial heuristics
   - Use *Initialization Prompt* to produce thoughts + code.
2. **Heuristic Generation:**
   - Apply 5 *Evolution Prompts* in parallel ($5N$ new heuristics)
     i. Select parent heuristic(s) to form prompt
     ii. LLM generates new thought and code
     iii. Evaluate fitness on test instances
     iv. Add feasible heuristics to population
3. **Population Management:**
   - Retain top N heuristics by fitness; return to Step 1

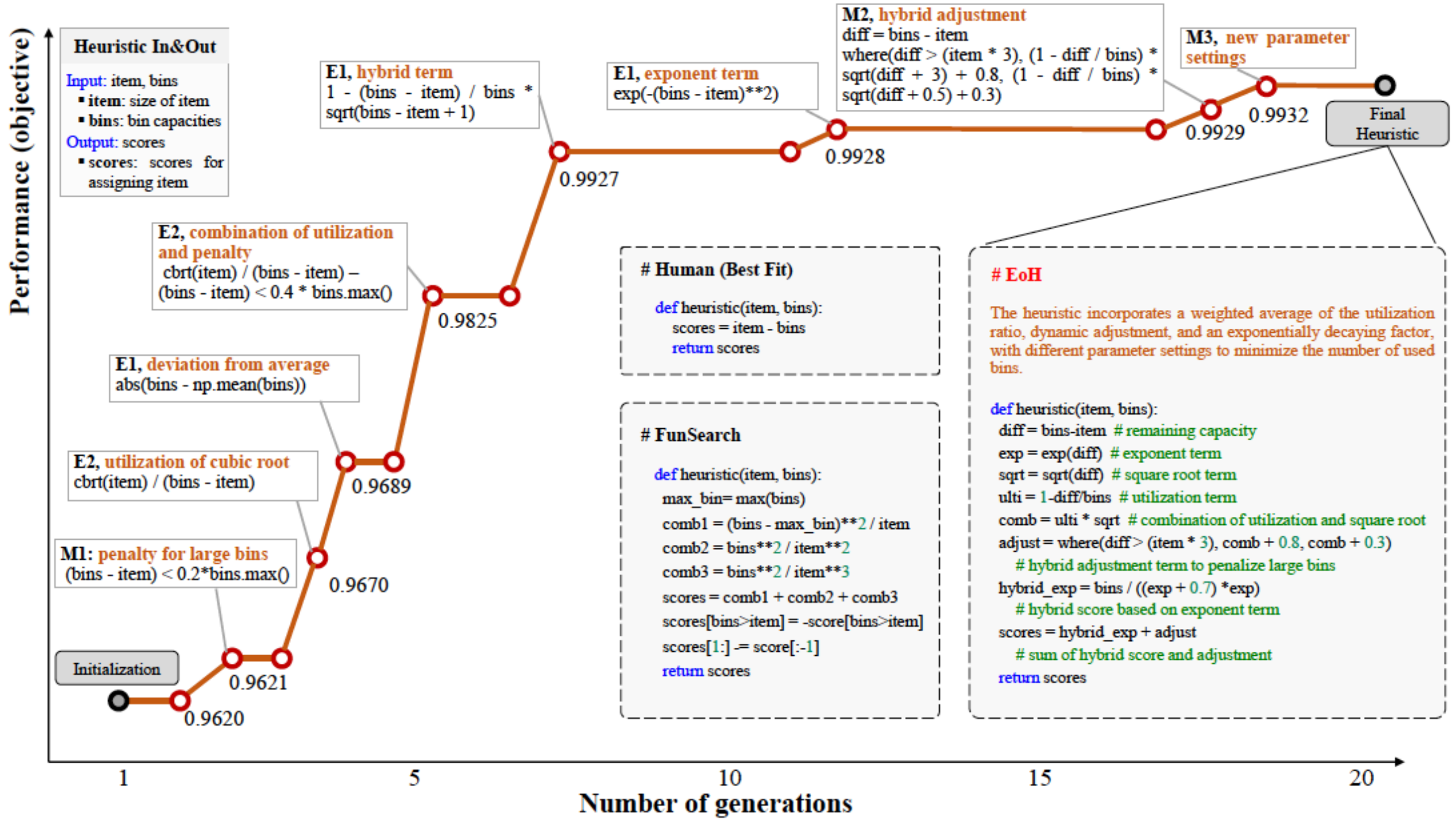
Example: Online bin packing

Initialization prompt:

- I need help designing a new heuristic that scores a set of bins to assign an item.
- In each step, the item will be assigned to the bin with the maximum score.
- If the rest capacity of a bin equals the maximum capacity, it will not be used.
- The final goal is to minimize the number of used bins.
- Firstly, describe your new heuristic and main steps in one sentence.
- Next, implement it in Python as a function named 'score'.

Evolution prompts

- **E1 – Diverse Exploration:**
  - Generate entirely new heuristic ideas from scratch
- **E2 – Shared-Idea Variants:**
  - Create new heuristics based on high-performing “themes”
- **M1 – Edit:**
  - Modify an existing heuristic
- **M2 – Parameter Tuning:**
  - Fine-tune numeric settings or thresholds in code
- **M3 – Simplification:**
  - Prune unnecessary components or redundant logic


Performance (objective)
Heuristic In&Out
Input: item, bins
item: size of item
bins: bin capacities
Output: scores
scores: scores for assigning item
Initialization
0.9620
M1: penalty for large bins
(bins - item) < 0.2*bins.max()
0.9621
E2, utilization of cubic root
cbrt(item) / (bins - item)
0.9670
E1, deviation from average
abs(bins - np.mean(bins))
0.9689
E2, combination of utilization and penalty
cbrt(item) / (bins - item) – (bins - item) < 0.4 * bins.max()
0.9825
E1, hybrid term
1 - (bins - item) / bins * sqrt(bins - item + 1)
0.9927
E1, exponent term
exp(-(bins - item)**2)
0.9928
M2, hybrid adjustment
diff = bins - item
where(diff > (item * 3), (1 - diff / bins) * sqrt(diff + 3) + 0.8, (1 - diff / bins) * sqrt(diff + 0.5) + 0.3)
0.9929
M3, new parameter settings
0.9932
Final Heuristic
# Human (Best Fit)
def heuristic(item, bins):
scores = item - bins
return scores
# FunSearch
def heuristic(item, bins):
max_bin= max(bins)
comb1 = (bins - max_bin)**2 / item
comb2 = bins**2 / item**2
comb3 = bins**2 / item**3
scores = comb1 + comb2 + comb3
scores[bins>item] = -score[bins>item]
scores[1:] -= score[:-1]
return scores
EoH
The heuristic incorporates a weighted average of the utilization ratio, dynamic adjustment, and an exponentially decaying factor, with different parameter settings to minimize the number of used bins.
def heuristic(item, bins):
diff = bins-item # remaining capacity
exp = exp(diff) # exponent term
sqrt = sqrt(diff) # square root term
ulti = 1-diff/bins # utilization term
comb = ulti * sqrt # combination of utilization and square root
adjust = where(diff > (item * 3), comb + 0.8, comb + 0.3)
# hybrid adjustment term to penalize large bins
hybrid_exp = bins / ((exp + 0.7) *exp)
# hybrid score based on exponent term
scores = hybrid_exp + adjust
# sum of hybrid score and adjustment
return scores
1
5
10
15
20
Number of generations

Experimental setup

Baselines:

- First Fit: place item in first bin that fits
- Best Fit: place item in bin w/ least available space
- Published FunSearch heuristic as-is
  - Requires ~1 million queries, as reported by Romera-Paredes et al. [Nature'24]

Problem sizes: 1000–10,000 items

Capacities: C = 100 and C = 500

Each setting: 5 randomly generated instances

Bin packing results

	1k_C100	5k_C100	10k_C100	1k_C500	5k_C500	10k_C500
First Fit	5.32%	4.40%	4.44%	4.97%	4.27%	4.28%
Best Fit	4.87%	4.08%	4.09%	4.50%	3.91%	3.95%
FunSearch	3.78%	**0.80%**	**0.33%**	6.75%	1.47%	0.74%
EoH (ours)	**2.24%**	**0.80%**	0.61%	**2.13%**	**0.78%**	**0.61%**

Metric: average gap to lower bound [Martello & Toth '90]

Additional problems (see paper)

- **Traveling Salesman Problem** (TSP)
- **Permutation Flow-Shop Scheduling** (FSSP)
  - n jobs must be processed on m machines
  - Jobs must follow same processing order
  - **Input:** $n \times m$ processing-time matrix
  - **Output:** permutation over machines for each job to follow
  - **Goal:** minimize makespan (total processing time)
- **Guided Local Search** for TSP and FSSP
  - Local search moves like swap cities/jobs, relocate cities/jobs
  - When local search stalls, reshape the costs to escape local minima
  - EoH learns a cost-update heuristic

Takeaways

Core idea: For automated heuristic design, co-evolve

1. Natural-language thoughts, and
2. Executable code for heuristic design

Integrates reasoning & implementation, unlike code-only

Outperforms FunSearch with far fewer LLM queries

Generalization:

Maintains good performance across unseen distributions