Interpretability of LLM-evolved heuristics

Julien Herrmann [1]
Guillaume Pallez [2]

[1]CNRS, IRIT, Toulouse
[2]Inria Rennes

18th Scheduling for large-scale systems workshop
École de Technologie Supérieure, Montréal, Québec, Canada

July 8th 2025

FunSearch published in Nature by Google DeepMind

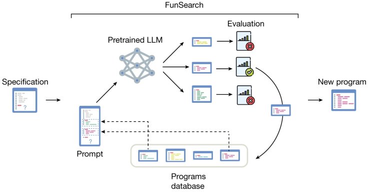

Article | Open access | Published: 14 December 2023

Mathematical discoveries from program search with large language models

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. 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

Cited by 485

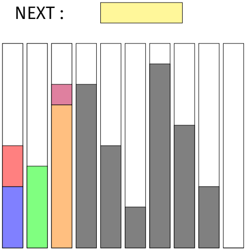


- Target 3 combinatorial problems : Cap sets, Admissible sets, and Online Bin Packing

> ***Paper take-away :***
> Through a combination of LLM+metaheuristic, possible to generate **interesting** heuristics for optimization problems that **human did not think of before**.

Online Bin Packing

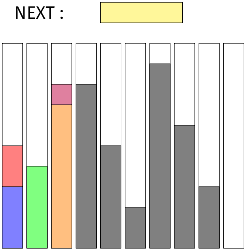
Priority heuristic :	
In	Object size + array of bins
Out	Bins priority



Then object scheduled in bin with maximum priority.

Online Bin Packing

Priority heuristic :	
In	Object size + array of bins
Out	Bins priority



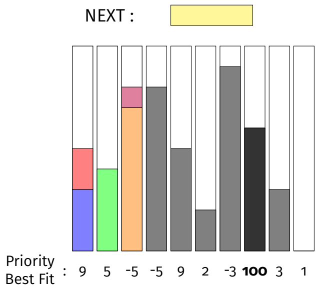
Then object scheduled in bin with maximum priority.

Nature paper :

- LLM (Codey) generates priority heuristic
- A genetic meta-heuristic (island-based) select the priority heuristics
- Specificity : job unknown (online) *but* follow probability distribution

Online Bin Packing

Priority heuristic :	
In	Object size + array of bins
Out	Bins priority



Then object scheduled in bin with maximum priority.

Nature paper :

- LLM (Codey) generates priority heuristic
- A genetic meta-heuristic (island-based) select the priority heuristics
- Specificity : job unknown (online) *but* follow probability distribution

Nature Results

- Trained on 5 instances of the OR-library
- Evaluated on the 20 others
- ≈ 10.000 inferences to the LLM
- BestFit as the baseline

Uniform[20, 100], **Bins : 150**

```
def priority_c12(item: float,
        bins: np.ndarray) -> np.ndarray:
    def s(bin, item):
        if bin - item <= 2:
            return 4
        elif (bin - item) <= 3:
            return 3
        elif (bin - item) <= 5:
            return 2
        elif (bin - item) <= 7:
            return 1
        elif (bin - item) <= 9:
            return 0.9
        elif (bin - item) <= 12:
            return 0.95
        elif (bin - item) <= 15:
            return 0.97
        elif (bin - item) <= 18:
            return 0.98
        elif (bin - item) <= 20:
            return 0.98
        elif (bin - item) <= 21:
            return 0.98
        else:
            return 0.99
    return np.array([s(bin, item)
        for bin in bins])
```

Weibull(45, 3), **Bins : 100**

```
def priority_c14(item: float, bins: np.ndarray)
        -> np.ndarray:
    score = (bins - max(bins))**2 / item + bins
        **2 / item**2 + bins**2 / item**3
    score[bins > item] *= -1
    score[1:] -= score[:-1]
    return score
```

Weibull(45, 3), **Bins : 100**

```
def priority_c13(item: float, bins: np.ndarray)
        -> np.ndarray:
    score = 1.56 * bins - item - 4 * np.log(
        bins) + 0.16
    score[score > item] = item * 0.56
    return -score
```

Discussion

The effectiveness of FunSearch in discovering new knowledge for hard problems might seem intriguing. We believe that the LLM used within FunSearch does not use much context about the problem; the LLM should instead be seen as a source of diverse (syntactically correct) programs with occasionally interesting ideas. When further constrained to operate on the

Uniform Distribution : U([20,100]); capacity 150

```
def priority_c12(item: float, bins: np.ndarray) -> np.
        ndarray:
    def s(bin, item):
        if bin - item <= 2:
            return 4
        elif (bin - item) <= 3:
            return 3
        elif (bin - item) <= 5:
            return 2
        elif (bin - item) <= 7:
            return 1
        elif (bin - item) <= 9:
            return 0.9
        elif (bin - item) <= 12:
            return 0.95
        elif (bin - item) <= 15:
            return 0.97
        elif (bin - item) <= 18:
            return 0.98
        elif (bin - item) <= 20:
            return 0.98
        elif (bin - item) <= 21:
            return 0.98
        else:
            return 0.99
    return np.array([s(bin, item) for bin in bins])
```

Figure – c12 heuristic in Nature

Uniform Distribution : U([20,100]); capacity 150

```python
def priority_c12(item: float, bins: np.ndarray) -> np.ndarray:
    def s(bin, item):
        if bin - item <= 2:
            return 4
        elif (bin - item) <= 3:
            return 3
        elif (bin - item) <= 5:
            return 2
        elif (bin - item) <= 7:
            return 1
        elif (bin - item) <= 9:
            return 0.9
        elif (bin - item) <= 12:
            return 0.95
        elif (bin - item) <= 15:
            return 0.97
        elif (bin - item) <= 18:
            return 0.98
        elif (bin - item) <= 20:
            return 0.98
        elif (bin - item) <= 21:
            return 0.98
        else:
            return 0.99
    return np.array([s(bin, item) for bin in bins])
```

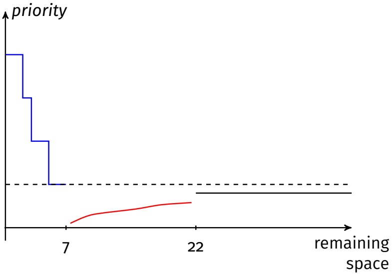


Figure – c12 heuristic in Nature

Uniform Distribution : U([20,100]); capacity 150

```python
def priority_c12(item: float, bins: np.ndarray) -> np.ndarray:
    def s(bin, item):
        if bin - item <= 2:
            return 4
        elif (bin - item) <= 3:
            return 3
        elif (bin - item) <= 5:
            return 2
        elif (bin - item) <= 7:
            return 1
        elif (bin - item) <= 9:
            return 0.9
        elif (bin - item) <= 12:
            return 0.95
        elif (bin - item) <= 15:
            return 0.97
        elif (bin - item) <= 18:
            return 0.98
        elif (bin - item) <= 20:
            return 0.98
        elif (bin - item) <= 21:
            return 0.98
        else:
            return 0.99
    return np.array([s(bin, item) for bin in bins])
```

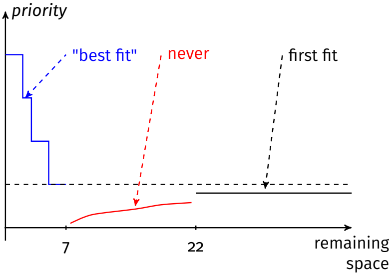


Figure – c12 heuristic in Nature

Uniform Distribution : U([20,100]); capacity 150

```python
def priority_c12(item: float, bins: np.ndarray) -> np.ndarray:
    def s(bin, item):
        if bin - item <= 2:
            return 4
        elif (bin - item) <= 3:
            return 3
        elif (bin - item) <= 5:
            return 2
        elif (bin - item) <= 7:
            return 1
        elif (bin - item) <= 9:
            return 0.9
        elif (bin - item) <= 12:
            return 0.95
        elif (bin - item) <= 15:
            return 0.97
        elif (bin - item) <= 18:
            return 0.98
        elif (bin - item) <= 20:
            return 0.98
        elif (bin - item) <= 21:
            return 0.98
        else:
            return 0.99
    return np.array([s(bin, item) for bin in bins])
```

Figure – c12 heuristic in Nature

```python
def priority_c12_simplified(item: float, bins: np.ndarray) -> np.ndarray:
    def s(bin, item):
        if (bin - item) <= 7:
            return capacity - bin
        elif (bin - item) <= 21:
            return 0
        else:
            return 1
    return np.array([s(bin, item) for bin in bins])
```

Figure – Simplified c12 heuristic

Uniform Distribution : U([20,100]); capacity 150

```
def priority_c12(item: float, bins: np.ndarray) -> np.ndarray:
    def s(bin, item):
        if bin - item <= 2:
            return 4
        elif (bin - item) <= 3:
            return 3
        elif (bin - item) <= 5:
            return 2
        elif (bin - item) <= 7:
            return 1
        elif (bin - item) <= 9:
            return 0.9
        elif (bin - item) <= 12:
            return 0.95
        elif (bin - item) <= 15:
            return 0.97
        elif (bin - item) <= 18:
            return 0.98
        elif (bin - item) <= 20:
            return 0.98
        elif (bin - item) <= 21:
            return 0.98
        else:
            return 0.99
    return np.array([s(bin, item) for bin in bins])
```

Figure – c12 heuristic in Nature

```
def priority_c12_simplified(item: float, bins: np.ndarray) -> np.ndarray:
    def s(bin, item):
        if (bin - item) <= 7:
            return capacity - bin
        elif (bin - item) <= 21:
            return 0
        else:
            return 1
    return np.array([s(bin, item) for bin in bins])
```

Figure – Simplified c12 heuristic

```
def priority_c12_parameterized(item: float, bins: np.ndarray) -> np.ndarray:
    def s(bin, item):
        if (bin - item) <= prio_a:
            return capacity - bin
        elif (bin - item) <= prio_b:
            return 0
        else:
            return 1
    return np.array([s(bin, item) for bin in bins])
```

Figure – Parameterized c12 heuristic

Uniform Distribution : U([20,100]); capacity 150

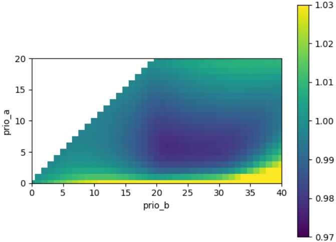


Figure – Performance of parameterized c12 for various prio_a and prio_b on the same uniform distribution c12 has been evolved compared to Bestfit

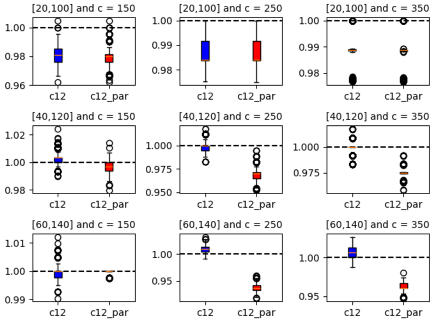


Figure – Performance of c12 and parameterized c12 for various uniform distribution

- On Uniform([x,y]) :

$$\boxed{\begin{array}{l} prio\_a = x/3 \\ prio\_b = x+1 \end{array}}$$

Weibull Distribution : $k = 45$; $\lambda = 3$; capacity 100

```python
def priority_c14(item: float, bins: np.ndarray) -> np.
        ndarray:
    score = (bins - max(bins))**2 / item + bins**2 / item**2
            + bins**2 / item**3
    score[bins > item] *= -1
    score[1:] -= score[:-1]
    return score
```

Figure – c14 heuristic in Nature

Weibull Distribution : $k = 45$; $\lambda = 3$; capacity 100

```
1  def priority_c14(item: float, bins: np.ndarray) -> np.
       ndarray:
2      score = (bins - max(bins))**2 / item + bins**2 / item**2
           + bins**2 / item**3
3      score[bins > item] *= -1
4      score[1:] -= score[:-1]
5      return score
```

Figure – c14 heuristic in Nature

Line 2 :

$$score(b) = \frac{(b-c)^2}{s} + \frac{b^2}{s^2} + \frac{b^2}{s^3}$$

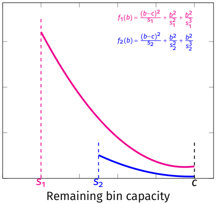


Figure – $b \mapsto score_{line2}(b)$ for $c = 100$, $s_1 = 20$, $s_2 = 50$

Weibull Distribution : $k = 45$; $\lambda = 3$; capacity 100

```
1   def priority_c14(item: float, bins: np.ndarray) -> np.
        ndarray:
2       score = (bins - max(bins))**2 / item + bins**2 / item**2
            + bins**2 / item**3
3       score[bins > item] *= -1
4       score[1:] -= score[:-1]
5       return score
```

Figure – c14 heuristic in Nature

Line 2 :

$$score(b) = \frac{(b-c)^2}{s} + \frac{b^2}{s^2} + \frac{b^2}{s^3}$$

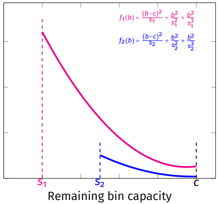


Figure – $b \mapsto score_{line2}(b)$ for $c = 100$, $s_1 = 20$, $s_2 = 50$

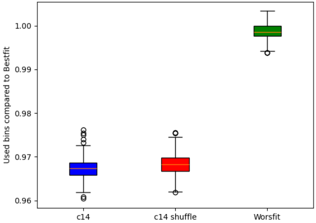


Figure – c14 and c14_shuffle performance compared to Bestfit over 1000 instances of the Weibull distribution

Weibull Distribution : $k = 45$; $\lambda = 3$; capacity 100

- Since f is non-increasing, the emptiest the bin is and the fullest the previous bin, the higher its priority will be

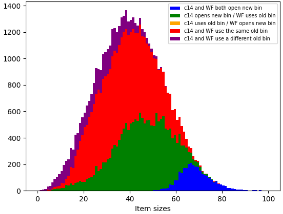


Figure – Comparing the behavior of c14 and Worstfit on the Weibull distribution (50k items)

Weibull Distribution : $k = 45$; $\lambda = 3$; capacity 100

- Since f is non-increasing, the emptiest the bin is and the fullest the previous bin, the higher its priority will be

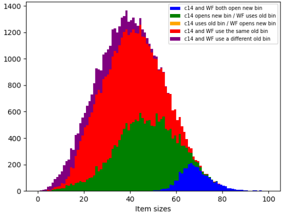


Figure – Comparing the behavior of c14 and Worstfit on the Weibull distribution (50k items)

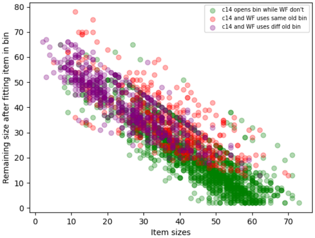


Figure – Remaining size in the bin used by WF after fitting the item in the red, green, and purple cases

c_14 heuristic approximate behavior :

- It puts the item in a perfect-fitting bin if it exists
- Otherwise, it puts the item in the worst fitting open bin leaving more than 21 remaining space if it exists
- Otherwise, it opens a new bin

Weibull Distribution : $k = 45$; $\lambda = 3$; capacity 100

```python
def priority_c14simplified(item: float, bins: np.ndarray) -> np.ndarray:
    m = max(bins)
    return [-1 if x == m else 1 if x == item else -2 if  x-item < 20 else -1/(x-item) for x in bins]
```

Figure – Simplified c14 heuristic

```python
def priority_c14parameterized(item: float, bins: np.ndarray) -> np.ndarray:
    m = max(bins)
    return [-1 if x == m else 1 if x == item else -2 if  x-item < prio_a else -1/(x-item) for x in bins]
```

Figure – Parameterized c14 heuristic

- On Weibull(3.0, 45.0), items of size < 20 correspond to 8.4% of the dataset
- On Weibull(a,b) :

$$prio\_a = b * weibull\_min.ppf(0.084, a)$$

Weibull Distribution : $k = 45$; $\lambda = 3$; capacity 100

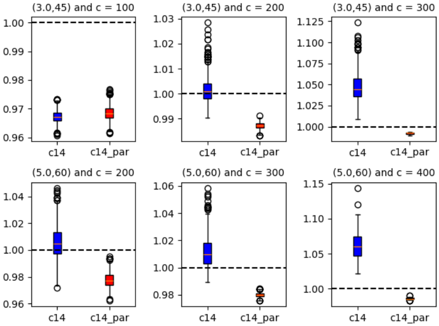


Figure – Boxplots showing the heuristics performance compared to Bestfit over 100 instances of 5000 items from various Weibull distributions and capacity

Bibliography

- 485 papers citing "Mathematical discoveries"
- 45 of them mentioning "bin packing"
- $\approx$ 10 proposing a novel LLM-evolved heuristic and comparing to FunSearch
- 5 give the best heuristic they found for Bin Packing in their appendix
- 1 has a critical perspective on the conclusion of "Mathematical discoveries"

Another example : Evolution of Heuristics

Evolution of Heuristics: Towards Efficient Automatic Algorithm Large Language Model

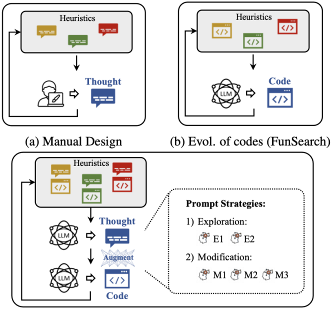
Cité 127 fois

Fei Liu [1] Xialiang Tong [2] Mingxuan Yuan [2] Xi Lin [1] Fu Luo [3] Zhenkun Wang [3] Zhichao Lu [1] Qingfu Zhang [1]

- Using Chain-of-thoughts to represent the heuristics
- ≈ 2.000 inferences to the LLM

```python
def priority_eoh(item, bins):
    diff = bins-item
    exp = np.exp(diff)
    sqrt = np.sqrt(diff)
    ulti = 1-diff/bins
    comb = ulti * sqrt
    adjust = np.where(diff > (item * 3), comb + 0.8, comb + 0.3)
    hybrid_exp = bins / ((exp + 0.7) *exp)
    scores = hybrid_exp + adjust
    return scores
```

$$score(b) = \frac{b}{(e^{b-s}+0.7)e^{b-s}} + (1 - \frac{b-s}{b})\sqrt{b-s} + (0.5 \text{ if } b > 4s)$$

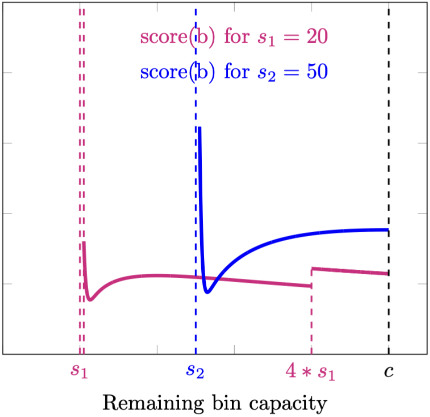


(a) Manual Design (b) Evol. of codes (FunSearch)

(c) Evolution of both thoughts and codes (EoH, ours)

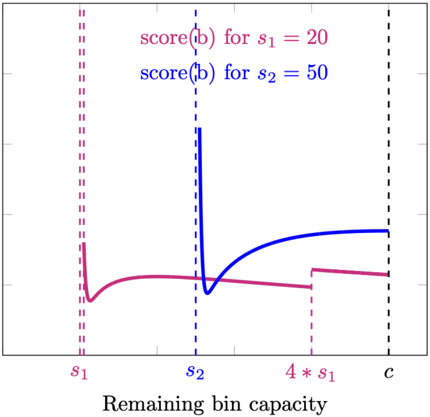
Evolution of Heuristics

$$score(b) = \frac{b}{(e^{b-s} + 0.7)e^{b-s}} + (1 - \frac{b-s}{b})\sqrt{b-s} + (0.5 \text{ if } b > 4s)$$



Evolution of Heuristics

$$score(b) = \frac{b}{(e^{b-s}+0.7)e^{b-s}} + (1-\frac{b-s}{b})\sqrt{b-s} + (0.5 \text{ if } b > 4s)$$

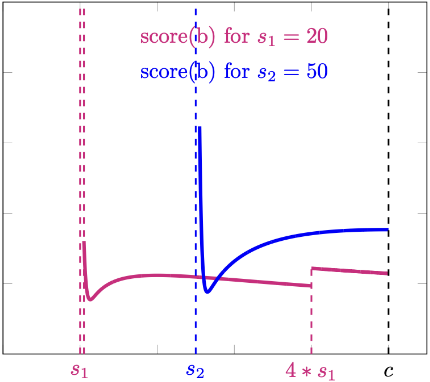


→ *abbf*-heuristic(s,B) :

if	exists a bin i, s.t. $B[i] - s < prio\_a$, then best fit
else if	exists a bin i, s.t. $B[i] - s > prio\_b$, then **best fit** in those bins
else	new bin

Evolution of Heuristics

$$score(b) = \frac{b}{(e^{b-s}+0.7)e^{b-s}} + (1-\frac{b-s}{b})\sqrt{b-s} + (0.5 \text{ if } b > 4s)$$



3 main types of heuristics :

→ *abbf*-heuristic(s,B) :

if	exists a bin i, s.t. $B[i] - s < prio\_a$, then best fit
else if	exists a bin i, s.t. $B[i] - s > prio\_b$, then **best fit** in those bins
else	new bin

→ *abwf*-heuristic(s,B) :

if	exists a bin i, s.t. $B[i] - s < prio\_a$, then best fit
else if	exists a bin i, s.t. $B[i] - s > prio\_b$, then **worst fit** in those bins
else	new bin

→ *abff*-heuristic(s,B) :

if	exists a bin i, s.t. $B[i] - s < prio\_a$, then best fit
else if	exists a bin i, s.t. $B[i] - s > prio\_b$, then **first fit** in those bins
else	new bin

abX heuristics performance

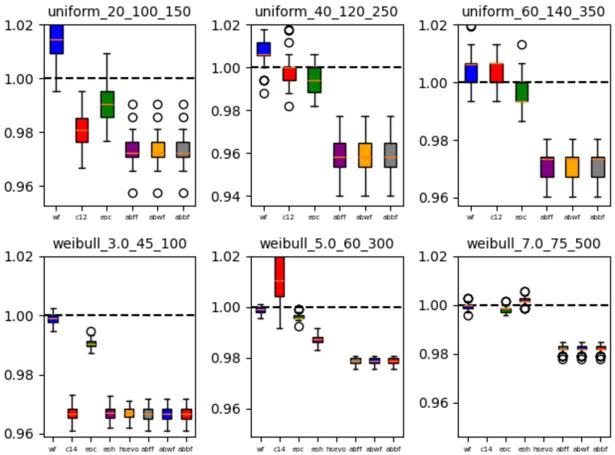


Figure – abwf, abff, and abbf performance compared to Bestfit over 100 instances of 5000 items from various distributions and capacity

Conclusion

- LLM-evolved are indeed (not that easily ?) **interpretable** and **generalizable**
- Interpretation still need human expertise and understanding of the problem
- All the LLM-evolved heuristic ideas can be summarized with **one line** heuristics in Python that the LLM did not find
- The genetic meta-heuristic can be replaced with **2 for-loop** ?
- New ideas or **obscure implementation** of old ideas ?
  ↪ Human expert needed to answer the question
- Would we accept/submit the same paper with the same heuristics that we can't explain ?
  ↪ How much AI is part of the acceptance ?
  ↪ Need of methodology ?