APPENDIX

A. More Experiments

1) SA Cooling rate: We perform an ablation study on the SA cooling rate and compare the results obtained by SA and the Hill Climbing(HC) strategy [39] on manually designed prompts, which allow LLMs to improve the current heuristic to some degree. The results are presented in Table X.

TABLE X
COMPARISON OF DIFFERENT COOLING RATES USING MANUALLY DESIGNED PROMPTS.

Cooling rate	Run1	Run2	Run3	Run4	Run 5	Deviation	Average
0.1	6.51	6.47	6.66	6.63	6.46	0.09	**6.55**
0.3	6.59	6.43	6.48	6.51	7.14	0.26	6.63
0.5	6.63	6.90	7.19	7.21	7.09	0.22	7.00
0.7	6.56	6.59	6.56	6.63	6.63	0.03	6.59
0.9	6.63	7.13	7.21	7.21	7.17	0.22	7.07
HC	6.69	6.69	6.53	6.55	6.51	0.08	6.59

2) RR Ratio: Additionally, we perform an ablation study on the ratio r when applying the SA cooling rate 0.1. The results are shown in Table XI.

TABLE XI
COMPARISON OF DIFFERENT COOLING RATES USING MANUALLY DESIGNED PROMPTS.

RR ratio	Run1	Run2	Run3	Run4	Run 5	Deviation	Average
0.1	6.48	6.69	6.58	6.48	6.46	0.09	6.54
0.2	6.46	6.46	6.76	6.52	6.50	0.11	6.54
0.3	6.71	6.88	6.48	6.58	6.48	0.15	6.63
0.4	6.61	6.59	6.56	6.63	6.62	0.02	6.60
0.5	6.58	6.44	6.53	6.56	6.38	0.08	**6.50**

3) Heuristics description in LHNS: Given that heuristics in LHNS consist of both code and its description, we conduct ablation experiments to evaluate the importance of each component on {problem}. Specifically, we design two experiment settings: the original LHNS with three proposed frameworks (LHNS), and LHNS without code description with three proposed frameworks (LHNS d). The experiments are conducted three runs on 64 instances of TSP50 with each 1000 iterations.

TABLE XII
COMPARISON BETWEEN COMPONENTS WITH AND WITHOUT CODE DESCRIPTION.

	Run1	Run2	Run3	Deviation	Average
LHNS-VNS	6.43	6.28	6.56	0.14	**6.42**
LHNS-VNS d	6.91	7.11	7.21	0.15	7.08
LHNS-ILS	6.44	6.46	6.41	0.03	**6.44**
LHNS-ILS d	7.09	7.19	6.48	0.38	6.92
LHNS-TS	6.47	6.41	6.46	0.03	**6.45**
LHNS-TS d	7.21	7.21	7.15	0.03	7.19

4) Results on TSP-GLS: In this subsection, we exhibit the results of experiments on TSP-GLS [11] of the TSPLib. As shown in Table XIII, the results demonstrate the highly competitive performance of our proposed methods.

B. Method Prompts

1) Prompts on LHNS-VNS: We exhibit the detailed prompt engineering used in the recreation process of LHNS-VNS for the TSP in a white-box setting. The prompt is divided into six main components:

- Task Description: This part provides the LLMs with the basic settings and the final goal of the target task.
- Common Strategy-Specific Prompts: These prompts instruct the LLMs on how to connect the expected outputs.
- RR Strategy Prompts: These guide the LLM through the recreate operation.
- Expected Outputs: This section asks the LLM to describe the recreated heuristic and implement its code. The name, input, and output of the code block are explicitly defined for easy identification by the EoH framework.
- Notes: Additional instructions for efficiency and robustness are provided here, such as specifying the type of inputs and outputs and discouraging the inclusion of unnecessary explanations.

TABLE XIII
RESULTS ON TSPLIB INSTANCES. THE GAP (%) TO THE BEST-KNOWN SOLUTION FROM TSPLIB.

Instance	FunSearch	EoH	LHNS-VNS	LHNS-ILS	LHNS-TS	LHNS-best
eil51	0.700	0.818	0.674	0.674	0.674	0.674
berlin52	3.894	0.031	0.031	3.894	0.031	0.031
st70	0.313	0.313	0.313	0.313	0.313	0.313
eil76	1.641	2.512	1.184	1.184	1.925	1.184
pr76	1.506	0.000	0.000	1.506	0.000	0.000
rat99	3.935	0.681	0.681	0.732	0.681	0.681
kroA100	2.998	0.016	0.058	2.998	0.016	0.016
kroB100	0.577	0.254	0.545	0.577	0.000	0.000
kroC100	5.582	0.498	2.001	5.582	0.008	0.008
kroD100	6.499	0.317	2.099	6.641	0.001	0.001
kroE100	4.641	0.024	1.596	4.641	0.174	0.174
rd100	9.314	0.093	0.776	7.295	0.005	0.005
eil101	3.149	2.113	1.782	2.584	2.171	1.782
lin105	2.606	0.028	2.162	2.606	0.028	0.028
pr107	0.614	0.614	0.353	0.614	0.000	0.000
pr124	2.441	0.096	0.976	2.441	0.001	0.001
bier127	1.789	0.385	0.468	1.789	0.185	0.185
ch130	5.719	0.852	1.703	2.689	0.984	0.984
pr136	6.304	0.000	0.123	6.304	0.319	0.123
pr144	4.194	0.000	0.740	4.194	0.000	0.000
ch150	1.282	0.562	0.376	1.202	0.385	0.376
kroA150	7.101	0.201	3.526	7.101	0.003	0.003
kroB150	5.546	0.675	0.902	5.546	0.861	0.861
pr152	1.895	0.644	1.702	1.895	0.002	0.002
u159	5.952	0.000	0.568	5.952	0.000	0.000
rat195	0.982	1.247	0.777	1.562	1.183	0.777
d198	1.085	0.939	0.477	0.668	0.469	0.469
kroA200	0.907	0.677	0.837	0.907	0.131	0.131
kroB200	5.357	1.321	2.010	5.357	0.299	0.299
ts225	0.002	0.002	0.002	0.002	0.002	0.002
tsp225	0.710	0.486	0.164	0.000	0.265	0.000
pr226	1.271	0.380	0.929	1.271	0.358	0.358
gil262	1.777	1.910	0.732	1.720	0.429	0.429
pr264	4.024	0.370	2.619	4.024	0.493	0.493
a280	3.593	0.430	0.704	0.562	1.865	0.562
pr299	3.852	2.227	1.015	3.852	1.186	1.015
Average	3.160	0.603	0.989	2.802	0.429	**0.332**

- Seed Heuristics: These are usually the current heuristics after the ruin operation, provided during a specific iteration to guide the LLM in recreating them.

In Figure 3, we provide the recreate strategy prompt used in LHNS-VNS. Note that the strategy of LHNS-VNS is to control the size of the search neighborhood by adjusting the number of deleted lines in the code. The RR prompt is utilized across all three frameworks.

2) Prompts on LHNS-ILS: We exhibit the detailed prompt engineering used in the recreation process of LHNS-ILS for OBPP in a white-box setting. We adopt the same six components outlined in the prompt engineering section for OBPP. In Figure 4, we provide two illustrative recreation prompts used in LHNS-ILS, each with different purposes. The perturbation prompt is exclusively used in LHNS-ILS.

3) Prompts on LHNS-TS: We detail the prompt engineering involved in the recreation process of LHNS-TS for ASP in a white-box setting. Following the six components outlined in the prompt engineering section for OBPP, Figure 5 presents two illustrative recreation prompts used in LHNS-TS, each serving a different purpose. Notably, the merge prompt is uniquely utilized in LHNS-TS.

C. Designed Heuristics

We present the most effective heuristics developed by LHNS. As shown from Figure6 to Figure8 for the white-box setting.

D. White-box and Black-box setting

This subsection we present the setting for the white-box and the black-box.

1) White-box setting:

a) TSP: Figure9 and Figure10 shows the initial heuristic seed and the task prompt for TSP.

b) Online Bin Packing Problem: Figure11 and Figure12 shows the initial heuristic seed and the task prompt for ASP.

Fig. 3. Example of prompt engineering used in LHNS-VNS for the TSP in a white-box setting.

Prompt for Recreate of RR

Given a set of nodes with their coordinates, you need to find the shortest route that visits each node once and returns to the starting node. The task can be solved step-by-step by starting from the current node and iteratively choosing the next node. Help me design a novel algorithm that is different from the algorithms in literature to select the next node in each step.

I have one algorithm with its code as follows.
Algorithm description:
Partial Code:
...

{number_of_deleted_lines} lines have been removed from the provided code. Please review the code, add the necessary lines to get a better result.
Firstly, describe your new heuristic and main steps in one sentence.
Next, implement it in Python as a function named 'select_next_node'. This function should accept four inputs: 'current_node', 'destination_node', 'univisited_nodes', 'distance_matrix'. The function should return one output: 'next_node'.
'current_node', 'destination_node', 'next_node', and 'unvisited_nodes' are node IDs. 'distance_matrix' is the distance matrix of nodes.

The 'distance_matrix' is a Numpy array.

Fig. 4. Example of prompt engineering used in LHNS-ILS for the OBPP in a white-box setting.

Prompt for Recreate of RR

I need help designing a novel score function that scoring 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.

I have one algorithm with its code as follows.
Algorithm description:
Partial Code:
...
{number_of_deleted_lines} lines have been removed from the provided code. Please review the code, add the necessary lines to get a better result.
Firstly, describe your new heuristic and main steps in one sentence.
Next, implement it in Python as a function named 'score'. This function should accept two inputs: 'item', 'bins'. The function should return one output: 'scores'.
'item' and 'bins' are the size of current item and the rest capacities of feasible bins, which are larger than the item size. The output named 'scores' is the scores for the bins for assignment.

Note that 'item' is of type int, while 'bins' and 'scores' are both Numpy arrays. The novel function should be sufficiently complex in order to achieve better performance. It is important to ensure self-consistency.Do not give additional explanations.Do not give additional explanations.

Prompt for Recreate of Perturbation

I need help designing a novel score function that scoring 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.

I have one algorithm with its code as follows.
Algorithm description:
Code:
...
Modify the provided algorithm to improve its performance, where you can determine the degree of modification needed.
Firstly, describe your new heuristic and main steps in one sentence.
Next, implement it in Python as a function named 'score'. This function should accept two inputs: 'item', 'bins'. The function should return one output: 'scores'.
'item' and 'bins' are the size of current item and the rest capacities of feasible bins, which are larger than the item size. The output named 'scores' is the scores for the bins for assignment.

Note that 'item' is of type int, while 'bins' and 'scores' are both Numpy arrays. The novel function should be sufficiently complex in order to achieve better performance. It is important to ensure self-consistency.Do not give additional explanations.Do not give additional explanations.

Fig. 5. Example of prompt engineering used in LHNS-TS for the ASP in a white-box setting.

Prompt for Recreate of RR

I need help developing a novel scoring function to evaluate vectors for potential inclusion in a set. This involves iteratively scoring the priority of adding a vector 'el' to the set based on analysis (like bitwise), with the objective of maximizing the set's size.

I have one algorithm with its code as follows.
Algorithm description:
Partial Code:
...
{number_of_deleted_lines} lines have been removed from the provided code. Please review the code, add the necessary lines to get a better result.
Firstly, describe your new heuristic and main steps in one sentence.
Next, implement it in Python as a function named 'priority'. This function should accept three inputs: 'el', 'n', 'w'. The function should return one output: 'score'.
'el' is the candidate vector for the admissible set. 'n' is the number of dimensions and the length of a vector. 'w' is the weight of each vector. 'score' is the priority of 'el'.

'el' is a int tuple, 'n' is int, 'w' is int, 'score' is a float.Do not give additional explanations.

Prompt for Recreate of Merge

I need help developing a novel scoring function to evaluate vectors for potential inclusion in a set. This involves iteratively scoring the priority of adding a vector 'el' to the set based on analysis (like bitwise), with the objective of maximizing the set's size.

I have algorithm A with its code, which inserts key lines from algorithm B's code, inserted just before the 'return' statement.
Algorithm A description:
Partial Code:
...
Algorithm B description:
...
Please review the given code, integrating two algorithm descriptions provided to rearrange it to get a better result.
Firstly, describe your new heuristic and main steps in one sentence.
Next, implement it in Python as a function named 'priority'. This function should accept three inputs: 'el', 'n', 'w'. The function should return one output: 'score'.
'el' is the candidate vector for the admissible set. 'n' is the number of dimension and the length of a vector. 'w' is the weight of each vector. 'score' is the priority of 'el'.

'el' is a int tuple, 'n' is int, 'w' is int, 'score' is a float.Do not give additional explanations.

c) Admissible Set Problem: Figure13 and Figure14 shows the initial heuristic seed and the task prompt for ASP.

2) *Black-box setting:*

a) TSP: Figure15 and Figure16 shows the initial heuristic seed and the task prompt for TSP.

b) Online Bin Packing Problem: Figure17 and Figure18 shows the initial heuristic seed and the task prompt for OBPP.

c) Admissible Set Problem: Figure19 and Figure20 shows the initial heuristic seed and the task prompt for ASP.

Fig. 6. Heuristic found by LHNS on TSP.

Heuristic designed by LHNS

<Heuristic Description>
The new algorithm selects the next node based on the product of
the inverse distance to the current node, the square root of the distance to
the destination node, the total distance to unvisited nodes, and the
reciprocal of the distance to the closest unvisited node, then chooses the node
with the highest product as the next node.

<Code>

```python
import numpy as np

def select_next_node(current_node, destination_node, unvisited_nodes, distance_matrix):
    distances_to_destination_node = np.sqrt(distance_matrix[destination_node, unvisited_nodes])

    closest_unvisited_node = unvisited_nodes[np.argmin(distance_matrix[current_node, unvisited_nodes])]

    distances_to_unvisited_nodes = np.sum(distance_matrix[unvisited_nodes[:, np.newaxis], unvisited_nodes
        ], axis=1)

    inverse_distances_to_current_node = 1 / distance_matrix[current_node, unvisited_nodes]

    product_values = inverse_distances_to_current_node * distances_to_destination_node *
        distances_to_unvisited_nodes * (1 / distance_matrix[current_node, closest_unvisited_node])

    next_node = unvisited_nodes[np.argmax(product_values)]

    return next_node
```

Fig. 7. Heuristic found by LHNS on OBPP.

Heuristic designed by LHNS

<Heuristic Description>
The new algorithm dynamically calculates penalties based on multiple factors such as under-utilization, over-reliance, deviation from optimal capacity, and refining the probabilistic element for bin selection optimization, aiming to minimize the number of used bins.
The main steps are as follows:
1. Calculate penalty for under-utilization of bins based on exponential function.
2. Calculate penalty for over-reliance on certain bins using logarithmic function.
3. Compute penalty for deviation from optimal bin capacity (half of item size).
4. Determine penalty for over-utilization of bins with a hyperbolic tangent function.
5. Calculate penalty for deviation from average bin capacity.
6. Combine all penalties to generate the final scores for bin assignment.

<Code>

```python
import numpy as np

def score(item, bins):

    penalty_under_capacity = np.exp(-np.abs(bins - item))  # Penalty for under-utilization of bins

    reliance_factor = 0.5
    over_utilization_factor = 0.3

    penalty_reliance = reliance_factor * np.log(bins)  # Penalty for over-reliance on certain bins

    distance_from_optimal = np.abs(bins - np.ceil(item / 2))  # Penalty for deviation from optimal bin
        capacity (half of item size)

    penalty_over_utilization = over_utilization_factor * np.tanh(bins - item)  # Penalty for over-
        utilization of bins

    penalty_deviation = np.abs(bins - np.mean(bins))  # Penalty for deviation from average bin capacity

    prob = np.random.rand(bins.shape[0])  # Adding a probabilistic element for bin selection optimization

    scores = penalty_under_capacity + penalty_reliance + prob - distance_from_optimal -
        penalty_over_utilization - penalty_deviation

    return scores
```

Fig. 8. Heuristic found by LHNS on ASP.

Heuristic designed by LHNS

<Heuristic Description>

```
New algorithm gives a bonus for vectors where the maximum element is a
multiple of 10, a penalty for vectors with repeated elements, and
considers the sum of squared distances between consecutive elements to
prioritize vectors with elements closer to each other and a higher
sum of elements while penalizing longer vectors with elements
further apart and a smaller maximum element.
```

<Code>

```python
def priority(el: tuple, n: int, w: int) -> float:

    max_penalty = max(el) % 10  # Penalty for maximum element not being a multiple of 10
    diff_sum = sum((el[i] - el[i-1])**2 for i in range(1, len(el)))  # Sum of squared distances between
        consecutive elements

    factor = 1.0  # Adjust this factor as needed
    penalty = len(el) * max(el)  # Penalty for longer and larger vectors
    desc_penalty = sum(1 for i in range(1, len(el)) if el[i] < el[i-1])  # Penalty for consecutive
        elements in descending order
    repeat_penalty = len(el) - len(set(el))  # Penalty for repeated elements

    score = factor * (sum(el) + diff_sum + w) - penalty + max_penalty + desc_penalty + repeat_penalty

    return score
```

Fig. 9. Initial heuristic on TSP in a white-box setting.

Inital heuristic of TSP

<Heuristic Description>

```
The new algorithm selects the next node by first calculating the
distance of all unvisited nodes to the current node, then selecting
the node with the shortest distance as the next node.
```

<Code>

```python
import numpy as np

def select_next_node(current_node, destination_node, unvisited_nodes, distance_matrix):
    distances = distance_matrix[current_node, unvisited_nodes]
    next_node = unvisited_nodes[np.argmin(distances)]
    return next_node
```

Fig. 10. Prompt setting of TSP in a white-box setting.

```
prompt_task = "Given a set of nodes with their coordinates,
you need to find the shortest route that visits each node once and
returns to the starting node. The task can be solved step-by-step
by starting from the current node and iteratively choosing the
next node. Help me design a novel algorithm that is different
from the algorithms in literature to select the next node in each step."

prompt_func_name = "select_next_node"

prompt_func_inputs = ["current_node","destination_node","univisited_nodes"
,"distance_matrix"]

prompt_func_outputs = ["next_node"]

prompt_inout_inf = "'current_node', 'destination_node', 'next_node',
    and 'unvisited_nodes' are node IDs. 'distance_matrix' is the distance matrix of nodes."

prompt_other_inf = "All are Numpy arrays."
```

Fig. 11. Initial heuristic on OBPP in a white-box setting.

```
Inital heuristic of TSP

<Heuristic Description>
The new algorithm selects the bin with the maximum rest capacity as
the greedy score function for assigning an item, optimizing the
assignment process by directly choosing the bin with the highest remaining capacity.

<Code>
import numpy as np

def score(item, bins):
    scores = bins / np.max(bins)
    return scores
```

Fig. 12. Prompt setting of OBPP in a white-box setting.

```
prompt_task = "I need help designing a novel score function that
    scoring 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."

prompt_func_name = "score"

prompt_func_inputs = ['item', 'bins']

prompt_func_outputs = ['scores']

prompt_inout_inf = "'item' and 'bins' are the size of current
item and the rest capacities of feasible bins, which are larger
than the item size. The output named 'scores' is the scores for
the bins for assignment. "

prompt_other_inf = "Note that 'item' is of type int, while
'bins' and 'scores' are both Numpy arrays. The novel function
should be sufficiently complex in order to achieve better performance.
It is important to ensure self-consistency."
```

Fig. 13. Initial heuristic on ASP in a white-box setting.

```
Inital heuristic of TSP

<Heuristic Description>
The algorithm is a greedy approach that calculates the score
function based on the weighted sum of the absolute values of the
elements in the vector 'el', and then divides by the number of
elements in the vector to prioritize vectors with smaller average
absolute values for adding to the set in order to maximize its size.

<Code>
def priority(el, n, w):
    score = sum(abs(i) for i in el) / n
    return score
```

Fig. 14. Prompt setting of ASP in a white-box setting.

```
prompt_task = "I need help developing a novel scoring function to
evaluate vectors for potential inclusion in a set. This involves
iteratively scoring the priority of adding a vector 'el' to the set
based on analysis (like bitwise), with the objective of maximizing
the set's size."

prompt_func_name = "priority"

prompt_func_inputs = ["el", "n", "w"]

prompt_func_outputs = ["score"]

prompt_inout_inf = "`el` is the candidate vector for the admissible
set. `n` is the number of dimensions and the length of a vector.
`w` is the weight of each vector. 'score' is the priority of `el`."

prompt_other_inf = "'el' is a int tuple, 'n' is int, 'w' is int, 'score' is a float."
```

Fig. 15. Initial heuristic on TSP in a black-box setting.

```
Inital heuristic of TSP

<Heuristic Description>
None.

<Code>
import numpy as np

def heuristics(node1, node2, nodes , edge_attrs):
    edge_attrs1 = edge_attrs[node1, nodes]
    node = nodes[np.argmin(edge_attrs1)]
    return node
```

Fig. 16. Prompt setting of TSP in a black-box setting.

```
prompt_task = "Solving a black-box graph combinatorial optimization problem via " + "'heuristics'."

prompt_func_name = "heuristics"

prompt_func_inputs = ["node1", "node2", "nodes", "edge_attrs"]

prompt_func_outputs = ["node"]

prompt_inout_inf = "'node1', 'node2', 'node', and 'nodes' are node
IDs. 'edge_attrs' is the edge attributes. All are Numpy arrays."

prompt_other_inf = "Please analyze the given code and function
description to identify the specific problem it addresses. Based on
your analysis to design improved heuristics."
```

Fig. 17. Initial heuristic on OBPP in a black-box setting.

Inital heuristic of TSP

<Heuristic Description>
None.

<Code>

```
import numpy as np

def heuristics(node_attr, node_attrs1):
    node_attrs1 = node_attrs1 / np.max(node_attrs1)
    return node_attrs1
```

Fig. 18. Prompt setting of OBPP in a black-box setting.

```
prompt_task = "Solving a black-box combinatorial optimization problem via " + "'heuristics'."

prompt_func_name = "heuristics"

prompt_func_inputs = ['node', 'nodes']

prompt_func_outputs = ['node_attrs']

prompt_inout_inf = "Note that 'node' is of type int, while 'nodes'
and 'node_attrs' are both Numpy arrays."

prompt_other_inf = "Please analyze the given code and function
description to identify the specific problem it addresses. Based on
your analysis to design improved heuristics."
```

Fig. 19. Initial heuristic on ASP in a black-box setting.

Inital heuristic of TSP

<Heuristic Description>
None.

<Code>

```
def heuristics(vector, number1, number2):
    vector_attr = sum(abs(i) for i in vector) / number1
    return vector_attr
```

Fig. 20. Prompt setting of ASP in a black-box setting.

```
prompt_task = "Solving a black-box combinatorial optimization problem via " +
"'heuristics'."

prompt_func_name = "heuristics"

prompt_func_inputs = ["vector", "number1", "number2"]

prompt_func_outputs = ["vector_attr"]

prompt_inout_inf = "'vector' is a int tuple, 'number1' is int,
'number2' is int, 'vector_attr' is a float"

prompt_other_inf = "Please analyze the given code and function
description to identify the specific problem it addresses. Based on
your analysis to design improved heuristics."
```