Learning Heuristic Selection Policies with Transformers for Selection Hyper-Heuristics

Mironshoh Sobirov, Doniyor Erkinov, Mustafa Mısır
Division of Natural and Applied Sciences
Duke Kunshan University
Kunshan, Jiangsu, China
{mironshoh.sobirov, doniyor.erkinov, mustafa.misir}@dukekunshan.edu.cn

Aldy Gunawan
School of Computing and Information Systems
Singapore Management University
Singapore
aldygunawan@smu.edu.sg

***Abstract*—The present study introduces a Transformer-based approach for learning heuristic selection policies in Selection Hyper-heuristics (SHHs) on combinatorial optimization. The iterative selection of low-level heuristics (LLHs) is modeled as a sequence prediction task. Two attention-based architectures are trained to imitate high-quality heuristic sequences produced by expert SHHs on the 1-dimensional bin packing problem (1D-BPP) within the HyFlex SHH framework. The models include a standard Transformer and the inverted Transformer (iTransformer). iTransformer uses self-attention across the feature (variable) dimension rather than time steps, with temporal information embedded in each token. Both models learn exclusively from successful decision trajectories without incorporating problem-specific features, thereby preserving the domain-agnostic principle central to SHH design. Evaluation is conducted across all 12 1D-BPP instances from HyFlex under a fixed computational budget. The iTransformer consistently outperforms Long Short-Term Memory (LSTM) and Temporal Convolutional Networks (TCN), as well as the standard Transformer variant across all tested instances. These results demonstrate that architectural choices in attention mechanisms significantly influence policy learning for heuristic selection. The findings establish the iTransformer as a particularly effective model for capturing sequential decision patterns in discrete optimization contexts.**

***Index Terms*—selection hyper-heuristics, transformers, neural networks, heuristic selection, combinatorial optimization, bin packing**

I. Introduction

Combinatorial optimization focuses on identifying an optimal solution within a finite, discrete set of feasible candidates for a given objective function assessing solutions' quality [1]. This field deals with numerous computationally challenging problems of varying domains such as logistics, manufacturing, and scheduling. To exemplify, representative combinatorial optimization problems (COPs) include bin packing (BP), permutation flowshop scheduling (FSP), nurse rostering (NSP), and the traveling salesman problem (TSP). These problems share the characteristic of requiring discrete decision variables and typically exhibit combinatorial explosion in their search spaces. This motivates the development of exact approaches, approximation algorithmsm and specialized (meta-)heuristic solutions, depending on the nature of the target problem.

COPs are often NP-hard, meaning no known algorithm can guarantee an optimal solution in polynomial time for all instances. Thus, there have been immense efforts for developing efficient algorithms that can quickly deliver feasible and high-quality solutions. However, these algorithms are typically problem-specific and lack transferability across different problems. Hyper-heuristics (HHs) address this limitation by operating at a higher level [2]. Rather than directly searching the solution space, HHs explore the space of heuristic operators, i.e. low-level heuristics (LLHs). A domain barrier separates the problem-independent HH layer from the problem-specific LLHs. This separation enforces problem independence for generality [3]. Unlike Meta-Heuristics (MHs) [4], [5], which often require domain expertise to adapt to new problems, HHs autonomously manage heuristic selection and application. This design reduces reliance on manual algorithm redesign when problem characteristics change, supporting broader applicability across combinatorial optimization domains.

The present study focuses on Selection HHs (SHHs) [2], [6]. At each decision step, an SHH chooses a LLH from a predefined set $\mathcal{H} = \{h_1, \ldots, h_k\}$. LLHs operate directly on the solution space $\mathcal{S}$ to modify or improve candidate solutions, ranging from simple move operators to sophisticated procedures capable of generating complete solutions. SHHs incorporate two core domain-independent mechanisms : (1) a heuristic selection mechanism that chooses the next LLH based solely on solution quality feedback $\rho : \mathcal{S} \rightarrow \mathbb{R}$ and selection history, without accessing problem-specific features of $\mathcal{S}$; and (2) a move acceptance criterion that evaluates the resulting candidate solution and decides whether to accept it. Formally, given an initial solution $s_0 \in \mathcal{S}$ and computational budget B, an SHH generates a solution trajectory $\{s_t\}_{t=0}^{T}$ with $T \leq B$ where $s_{t+1} = h_t(s_t)$ if accepted, otherwise $s_{t+1} = s_t$. This 2-stage process enables SHHs to navigate the search space while preserving problem independence.

Machine learning (ML) and combinatorial optimization (CO) represent closely interconnected research fields. Some studies apply ML techniques to solve combinatorial optimization problems [7], [8], while others leverage CO to model learning tasks within or for ML [9]. This synergy has motivated extensive integration of ML with MHs [10] and HHs [11] to enhance algorithmic performance through learning and adaptation capabilities. Learning in HHs occurs in two ways. Offline learning trains models prior to optimization, such as using Neural Networks (NN) to learn heuristic selection policies [12]. Online learning adapts algorithm behavior while an algo-

ritm operates, as in Reinforcement Learning (RL) approaches [13]. Deep RL [14] has further expanded adaptive optimization methods following advances in Deep Learning (DL). This paper adopts an offline learning approach. To be specific, a SHH setup accommodates Transformers to guide heuristic selection decisions. The methodology follows a prior work [12] that models heuristic selection as a sequence prediction task, where a sequence represents an ordered series of LLHs applied to improve solutions. For formally describing this sequence prediction task, let $\Pi^{(j)} = \langle h_{i_0}^{(j)}, h_{i_1}^{(j)}, \ldots, h_{i_{L_j-1}}^{(j)} \rangle$ denote a high-quality heuristic sequence of length L_j produced by an expert SHH on instance j. Given a dataset $\mathcal{D} = \{\Pi^{(1)}, \ldots, \Pi^{(N)}\}$ of such sequences, the objective is to learn a mapping from prefix sequences to the next heuristic:

$$\hat{h}_t = \arg\max_{h \in \mathcal{H}} \; p(h \mid h_{i_0}, \ldots, h_{i_{t-1}})$$

That study identified high-quality LLH sequences produced by expert SHHs on the 1D bin packing problem (1D-BPP), then trained LSTM and Temporal Convolutional Network (TCN) models to reproduce these sequences. Building on this foundation, we leverage the effectiveness of learning LLH sequences capturing temporal dependencies between consecutive heuristic applications. The present work applies Transformers to the same sequence prediction task, motivated by their strong performance on sequential data. Transformer-based models capture long-range dependencies across the full input sequence via self-attention, making them well-suited for modeling heuristic selection histories in which the relevance of previously applied LLHs is not strictly bounded by their temporal proximity. Evaluation is reported on the same 1D-BPP setup in [12]. For our study, two variants of Transformers are examined: a standard Transformer and the inverted Transformer (iTransformer) [15], which applies self-attention across the feature dimension. Experimental results on all 12 bin packing instances under HyFlex show that the iTransformer consistently outperforms both the vanilla Transformer and the LSTM and TCN baselines, delivering superior solution quality.

The remainder of the paper is organized as follows. Section II introduces the target problem of 1D bin packing and the HyFlex setup used to carry out the experiments. Section III presents the Transformer-based methodology for learning heuristic selection policies. Section IV reports empirical results and analysis. Section V concludes with key findings and discusses about future research directions.

II. Problem Description and Evaluation Environment

This section introduces the 1-dimensional bin packing problem (1D-BPP) under the HyFlex framework [16], a well-known experimental environment for Selection Hyper-heuristics (SHHs). HyFlex provides a well-established, domain-independent setup that enforces a strict domain barrier between problem-specific components and the SHH layer, enabling domain-agnostic comprehensive evaluation of SHHs across multiple combinatorial optimization problems (COPs). While its extended variant has 9 COPs, since this study focuses on a single problem, HyFlex use is mainly for convenience.

Going back to the problem, the 1D-BPP requires packing a set of items with varying sizes into the minimum number of fixed capacity bins. Formally, given items with sizes s_i and bin capacity C, the objective is to minimize the number of bins used while ensuring $\sum_{i \in B_j} s_i \leq C$ for each bin B_j. Within the HyFlex framework, however, the objective function differs.

$$f = 1 - \frac{1}{n} \sum_{i=1}^{n} \left(\frac{\text{fullness}_i}{C} \right)^2,$$

where n denotes the number of bins used, C is the bin capacity, and fullness_i represents the total size of items assigned to bin i. This formulation rewards compact packing by penalizing bins with low utilization through a squared term. Thus, this fitness function provides fine-grained feedback to SHHs, allowing more informed selection and acceptance decisions.

Under HyFlex, the 1D-BPP comes with 8 low-level heuristics (LLHs) of 4 types:

- **Mutation** – perform random yet controlled perturbations, primarily to escape local optima and enhance exploration
  - LLH_0: Swap random items between bins (m times)
  - LLH_3: Repack the least-filled bin
  - LLH_5: Split a bin into two bins of balanced load
- **Ruin and Recreate** – partially destroy the current solution and rebuild it greedily to enable larger exploration steps
  - LLH_1: Destroy X most-filled bins, repack with best-fit
  - LLH_2: Destroy X least-filled bins, repach with best-fit
- **Local Search** – perform iterative checks within the solution's neighborhood to identify higher-quality nearby solutions for achieving exploitation
  - LLH_4: Swap an item from the least-filled bin to improve utilization
  - LLH_6: Perform l iterations of random swaps followed by local improvement
- **Crossover** – combine two solutions to generate new ones
  - LLH_7: Exon shuffling crossover that exchanges segments of bin assignments

HyFlex has direct access to 20 SHHs regarding the Cross-domain Heuristic Search Challenge 2011 (CHeSC'11). Each SHH comes with distinct designs and capabilities, offering diverse behavioral patterns for sequence models to learn. These SHHs were executed on the 1D-BPP instances to generate LLH sequences used for model training (Table I).

III. Methodology

Following the setup from [12], the same 20 Selection Hyper-heuristics (SHHs) are utilized to generate LLHs sequences as training data. Each SHH is run 20 times on 12 instances of 1-D Bin Packing where each run is capped at 100,000 iterations. The fitness values and LLHs applied at each iteration are recorded. Upon obtaining these sequences, top 30 per-instance LLH sequences are determined, giving $30 \times 12 = 360$ sequences to feed the NN models. Next, we extract subsequences of

TABLE I
THE 20 SHHS FROM CHESC 2011 USED TO GENERATE LLH SEQUENCE TRAJECTORIES FOR MODEL TRAINING.

#	SHH
1	Adaptive Hyper-Heuristic (GIHH)
2	Variable Neighborhood Search-based Hyper-Heuristic
3	ML Hyper-Heuristic
4	Pearl Hunter
5	Evolutionary Programming Hyper-heuristic (EPH)
6	HAHA
7	Non-Adaptive Hyper-Heuristic
8	Iterated Search Driven by Evolutionary Algorithm Hyper-Heuristic
9	KSATS-HH
10	Hybrid Adaptive Evolutionary Algorithm Hyper-Heuristic
11	Ant Colony Optimization Hyper-Heuristic
12	Genetic Hive Hyper-Heuristic
13	Dynamic Iterated Local Search
14	SA-ILS
15	XCJ
16	AVEG-Nep
17	Generic Iterative Simulated-Annealing Search
18	SelfSearch
19	elomari2011self
20	Ant-Q Hyper Heuristic

length 32 as input data, while single LLHs that immediately follow these subsequences serve as labels.

1) Long Short Term Memory (LSTM): Long Short-Term Memory (LSTM) is a class of recurrent neural networks (RNN) that models sequential data by maintaining an explicit memory state [17]. Using gated mechanisms, LSTMs regulate information flow across time steps, thereby mitigating the vanishing gradient problem that is common in standard RNN architectures. Our study implements an LSTM model comprised of 2 layers, each with a hidden size of 64. A previous study [12] showed that LSTM-based selection outperform a non-learning one, making it a strong baseline.

2) Temporal Convolutional Networks: Temporal Convolutional Networks (TCNs) are convolution-based architectures designed for sequence modeling tasks, offering an alternative to recurrent neural networks by replacing recurrent computation with 1D causal convolutions, thus enabling stable gradient propagation and parallel computation [18]. In our implementation, the TCN baseline operates on input sequences of length 32 and consists of five temporal convolutional blocks with a hidden channel width of 64 and a kernel size of 3. Similarly to LSTM, TCN has been successfully applied for the same LLH sequence prediction task [12].

Both LSTM and TCN models were retrained on identical training data under the same experimental setup for fairness.

3) Vanilla Transformer: The Transformer is a sequence-to-sequence neural network architecture. Structurally, it consists of an encoder and a decoder. The encoder maps a sequence of embeddings $(x_1, ..., x_n)$ to a different sequence of embeddings $\boldsymbol{z} = (z_1, ..., z_n)$. Next, the decoder takes $\boldsymbol{z}$ and generates probabilities for the next token in the sequence.

The Transformer differs from prior sequence-to-sequence models in that it relies entirely on a self-attention mechanism to learn embeddings for input and output sequences. Attention is a function that maps key, query, and value vectors to an output vector. More precisely, the output vector is a weighted sum of value vectors, weights to which are assigned based on a particular compatibility function of keys and queries. Equation 1 is an example of the attention function that uses scaled dot-product as the mentioned compatibility function.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{1}$$

where Q, K, and V are the query, key, and value matrices, and d_k is the column dimension of the query and key matrices.

It has been found that linearly projecting query, key, and value matrices into h different, learned matrices is beneficial for the model's performance. This mechanism is called multi-head attention. The mathematical formulation reads as follows:

$$\text{MultiHead} = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where} \quad \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V),$$

where W_i^Q, W_i^K, W_i^V, and W_i^O are the projection matrices for query, key, value, and output matrices, respectively.

In this work, the vanilla Transformer baseline comprises two encoder layers, each with 4 attention heads and a model dimension of 64, yielding approximately 0.56M trainable parameters. Each encoder layer consists of a multi-head self-attention mechanism followed by a position-wise feed-forward network that captures global dependencies across input sequences without recurrence or convolution.

4) iTransformer: iTransformer is an encoder-based Transformer variant proposed for multivariate time-series forecasting [15]. Its key idea is to *invert* the standard tokenization used in many Transformer forecasters. Specifically, instead of treating each time step as a token, i.e., embedding $X_{t,:}$ and applying attention over time, iTransformer treats each *feature channel (variate)* as a token by embedding the entire lookback series $X_{:,n}$ of the n-th channel into a single representation, and applies self-attention across these channel tokens.

To formalize this inverted representation, let the historical input be denoted by $X \in \mathbb{R}^{T\times N}$, where T is the lookback length and N is the number of feature channels. Rather than forming tokens from temporal slices $X_{t,:}$, iTransformer constructs tokens along the feature dimension. For each channel $n \in \{1, \ldots, N\}$, the entire lookback sequence $X_{:,n} \in \mathbb{R}^T$ is embedded into a single token

$$h_n^0 = \text{Embedding}(X_{:,n}),$$

where $\text{Embedding} : \mathbb{R}^T \to \mathbb{R}^D$ is implemented as a multi-layer perceptron (MLP). Collecting all channel-wise embeddings yields the initial token matrix

$$H^0 = [h_1^0; \ldots; h_N^0] \in \mathbb{R}^{N\times D}.$$

Subsequent Transformer encoder blocks operate on H^0, applying self-attention across feature-channel tokens to capture inter-channel interactions, while the temporal order within each channel is preserved through the channel-wise embedding and feed-forward processing. Despite the inverted representation, iTransformer retains standard Transformer components

Algorithm 1 Neural Network-based SHH Framework

Require: Model $\mathcal{M}$, sequence length L, iteration limit $iterLimit$=100,000, retry limit K
Ensure: Fitness trace $\{fitness_t\}$

$llhArray \leftarrow [\,]$, $fitness \leftarrow \infty$, $iter \leftarrow 0$, $S \leftarrow 1$
while $iter < iterLimit$ **do**
  $iter \leftarrow iter + 1$
  if $|llhArray| < L$ **then**
    Append SAMPLELLH() to $llhArray$
  else
    $llh \leftarrow \mathcal{M}(llhArray)_S$ {Apply the LLH with the Sth highest score}
    $newFitness \leftarrow$ APPLY(llh)
    if $newFitness < fitness$ **or** RAND() < 0.1 **then**
      $fitness \leftarrow newFitness$; Append llh to $llhArray$; $S \leftarrow 1$
    else
      $S \leftarrow S + 1$
      if $S \geq K$ **then**
        Append SAMPLELLH() to $llhArray$; $S \leftarrow 1$
      end if
    end if
  end if
  Record $fitness_t \leftarrow fitness$
end while

such as layer normalization, self-attention, and feed-forward networks, but yet with different roles. Instead of operating along the temporal dimension, each standard Transformer component is applied across *feature-channel dimension* that enables the model to capture inter-variable dependencies.

a) Encoder architecture: In vanilla Transformer, *layer normalization* is applied to the multivariate representation of a single time step, implicitly mixing feature channels. Yet, iTransformer uses layer normalization independently to each feature-channel token, normalizing its series representation:

$$\mathrm{LN}(H) = \left\{ \frac{h_n - \mu(h_n)}{\sqrt{\sigma^2(h_n) + \epsilon}} \;\middle|\; n = 1, \ldots, N \right\},$$

where $h_n \in \mathbb{R}^D$ denotes the token corresponding to the n-th feature channel. The intention is to reduce cross-channel interference during normalization and has been empirically validated in prior work on non-stationary time series [19].

Like in the standard Transformer, a position-wise **Feed-Forward Network (FFN)** is applied to each token. In the inverted version, the FFN operates on the series representation of each feature channel to extract temporal characteristics from the lookback sequence encoded in each token.

In iTransformer, self-attention is applied across feature-channel tokens, rather than across time steps, unlike vanilla Transformers. Given the token matrix $H \in \mathbb{R}^{N \times D}$, linear projections are used to generate the queries, keys, and values: $Q, K, V \in \mathbb{R}^{N \times d_k}$. The attention scores are then computed as

$$A = \frac{QK^\top}{\sqrt{d_k}} \in \mathbb{R}^{N \times N},$$

where $A_{i,j}$ reflects the interaction between the i-th and j-th feature channels. Self-attention in iTransformer primarily captures inter-channel relationships, while temporal dependencies are encoded within each token prior to the attention operation.

b) Use in our setting.: While iTransformer was proposed for multivariate forecasting, its inverted tokenization provides a structurally distinct attention-based sequence model. In SHH, we can construct an input matrix $X \in \mathbb{R}^{T \times N}$ from a lookback window of heuristic-history features, e.g., LLH indicator channels and optional auxiliary signals. In addition, by treating each feature channel $X_{:,n}$ as a token and applying attention across channels the proposed model can effectively capture complex interactions between different LLH features. This feature-wise attention is expected to outperform temporal attention for LLH histories, as the relevant patterns and indicators of future LLH performance are more likely to be encoded in the relationships between features at each time step. Furthermore, attaching a task-specific output head such as softmax over LLHs, for next-LLH prediction allows to learn to map the feature interactions to LLH selections directly.

The iTransformer model here follows an encoder-only architecture composed of 4 inverted Transformer layers. Each layer then applies channel-token multi-head self-attention with eight attention heads and a model dimension of 256, followed by a position-wise feed-forward network with hidden dimension 512. Residual connections and layer normalization are applied around each sub-layer. Under this configuration, the model operates on input sequences of length 32, with a vocabulary size of 8, and contains approximately 2.1M trainable parameters.

Table II details the architecture specifications.

A. Neural Networks in Selection Hyper-Heuristics

Once the models are trained, they are integrated into the typical SHH framework as detailed in Algorithm 1. Following the solution initialization, we apply LLHs sampled from a discrete uniform distribution until LLHs list reaches the length of 32. After that, we change the heuristic selection mechanism and choose LLhs using the NN models. As an addition to the implementation in the compared study, the variable S is incorporated to dictate which LLH should be chosen.For example, if $S = 3$, the LLH with the 3rd highest score, based on the output layer of the NN models, is utilized. Then, under the acceptance criterion, the selected LLH is immediately accepted if it improves the solution. Otherwise, it is accepted with 10% probability. If both conditions are not met, S is incremented once per iteration until it does not exceed the number of LLHs. In that case, a random LLH is applied. In particular a ranked fallback prevents a rejection loop, which is inherent to applying a fixed-neural policy under a move acceptance criterion. Without the S mechanism, an unchanged input context would cause the model to re-nominate the top-ranked LLH after every rejection. Incrementing S addresses the issue of indefinite re-nomination of the top-ranked LLH

TABLE II
MODEL ARCHITECTURES AND THEIR HYPERPARAMETER-VALUES.

Model	Sequence Length	Number of Layers	Hidden Units / Channels	Parameters
iTransformer	32	4	Model dimensions 256, FFN hidden size 512, 8 heads	2,117,121
Transformer	32	2	Model dimensions 64, 4 heads	563,400
TCN	32	5	64 channels per layer, kernel size 3 (dilated)	114,504
LSTM	32	2	64 hidden units per layer	52,744

by descending through the model's ranked outputs, exploring next-best learned preferences before restoring to random sampling. This workflow is analogous to top-k decoding in sequence generation for Large Language Models (LLMs) [20], where committing to the greedy top-1 prediction is avoided in favor of exploring the broader learned probability mass.

IV. COMPUTATIONAL ANALYSIS

Experiments were conducted on the same set of 12 Bin Packing problem (1D-BPP) instances used in the compared study [12] following their setup. For each instance, all methods were executed 10 times independently, with 100.000 iterations per run as the stopping criterion. Table III reports the results in terms of fitness values under the same experimental setting, with identical heuristic space, move-acceptance criteria, and other protocols. While all models built offer competitive performance, iTransformer delivers the best performance but for the best of 10-runs and per-instance 10-run averages.

Figure 2 presents the fitness value changes over 100.000 iterations for all 12 1D-BPP instances. All demonstrate rapid initial improvement followed by a slower refinement phase, indicating consistent convergence behavior across architectures.

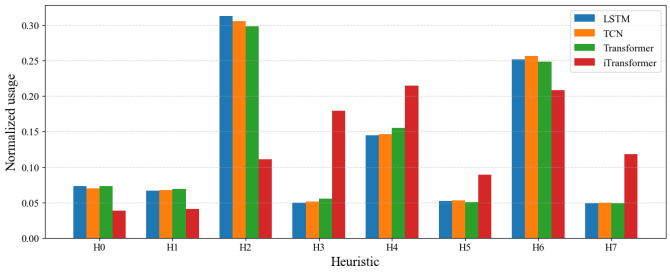

Fig. 1. Selection frequencies of LLHs by each tested model

Furthermore, according to Figure 1, the LSTM, TCN and transformers models built exhibit similar LLH selection patterns, with a dominant reliance on the heuristics *H2* (repack the lowest filled container) and *H6* (swap a piece from the lowest filled container with another piece). In contrast, iTransformer demonstrates a different LLH selection behavior, characterized by reduced reliance on *H2* LLH and increased emphasis on heuristics *H3* (split a bin into two halves) and *H7* (swap random pieces in different bins l times). iTransformer's alternative heuristic selection strategy yields the best performance.

Across all the 1D-BPP instances, iTransformer consistently reaches the best median performance with limited variability. Concerning Transformer and TCN, they exhibit inconsistent performance across instances, performing competitively on some while underperforming on others. The LSTM-based model achieves competitive ranks while generally occupying an intermediate position, suggesting a balance between performance consistency and instance sensitivity.

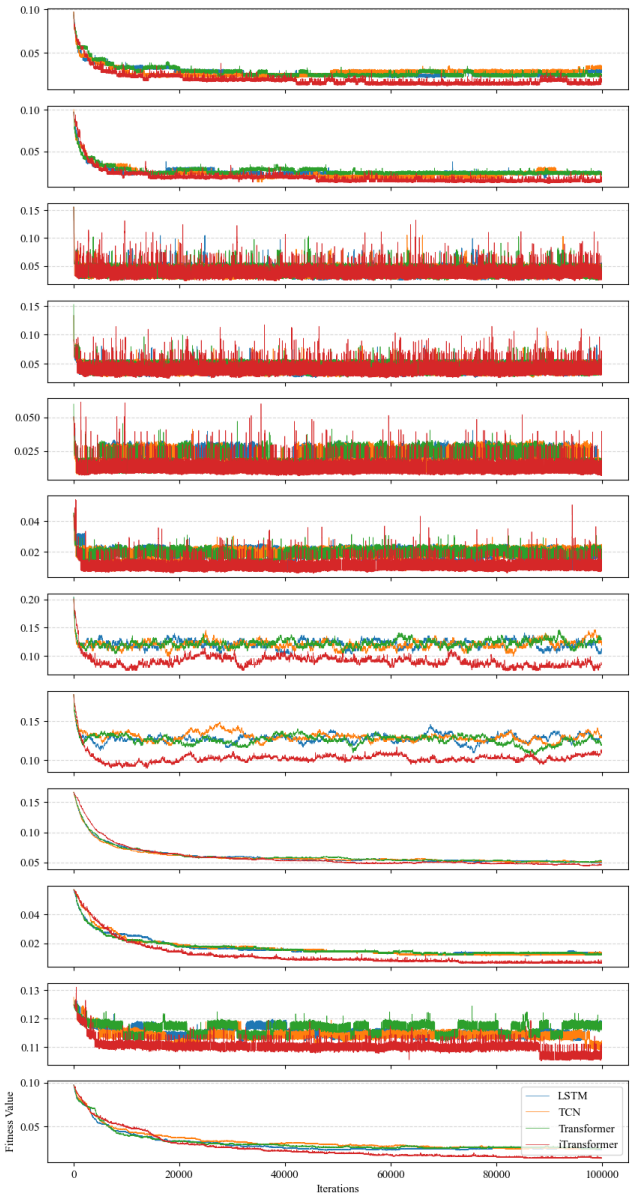

Fig. 2. Single-run fitness trace of the tested models on all the instances.

TABLE III
FITNESS RESULTS (MIN / AVG / STD) ACROSS 12 INSTANCES. THE BEST AVG VALUES ACROSS DIFFERENT ARCHITECTURES ARE HIGHLIGHTED (BLUE AND RED REPRESENT THE BEST AND THE 2ND BEST VALUES, RESPECTIVELY).

Ins	Transformer			iTransformer			TCN			LSTM		
	Min	Avg	Std	Min	Avg	Std	Min	Avg	Std	Min	Avg	Std
0	0.01716	0.01925	0.00170	0.01128	**0.01162**	0.00035	0.01668	0.01992	0.00163	0.01632	**0.01751**	0.00156
1	0.01719	0.01910	0.00165	0.00844	**0.01034**	0.00154	0.01686	0.01729	0.00045	0.01634	**0.01720**	0.00073
2	0.02387	**0.02453**	0.00036	0.02292	**0.02342**	0.00056	0.02388	0.02455	0.00035	0.02379	0.02409	0.00041
3	0.02609	0.02622	0.00016	0.02337	**0.02452**	0.00061	0.02572	0.02618	0.00029	0.02493	**0.02570**	0.00049
4	0.00724	0.00762	0.00029	0.00644	**0.00665**	0.00017	0.00772	0.00794	0.00016	0.00737	**0.00757**	0.00018
5	0.00746	0.00776	0.00019	0.00609	**0.00639**	0.00023	0.00726	**0.00758**	0.00021	0.00751	0.00769	0.00013
6	0.09458	0.09826	0.00315	0.06473	**0.07089**	0.00345	0.09441	**0.09856**	0.00356	0.09463	0.09915	0.00390
7	0.10675	**0.11169**	0.00328	0.08178	**0.08479**	0.00252	0.10706	0.11289	0.00303	0.10448	0.11333	0.00473
8	0.04741	0.04933	0.00133	0.04319	**0.04487**	0.00093	0.04733	0.04863	0.00110	0.04328	**0.04560**	0.00147
9	0.00938	0.01065	0.00080	0.00516	**0.00565**	0.00051	0.00937	0.01047	0.00068	0.00838	**0.00988**	0.00091
10	0.10880	0.10964	0.00110	0.10447	**0.10536**	0.00125	0.10870	0.10924	0.00028	0.10832	**0.10861**	0.00021
11	0.02142	0.02245	0.00088	0.01112	**0.01202**	0.00074	0.02129	0.02235	0.00105	0.01838	**0.02061**	0.00116

As it is shown in Table III, iTransformer outperforms other tested models across all 12 instances. The statistical significance is further confirmed by the Wilcoxon signed-rank test ($p < 0.001$) when comparing per-instance averages across the 12 instances for each model pair. Then, the tested models' convergence behaviors are evaluated. The standard Transformer, LSTM, and TCN models demonstrate more greedy or reactive behavior, converging quickly at earlier iterations whereas iTransformer has longer-horizon reasoning, achieving better solutions at the end.

V. CONCLUSION

This research explores Transformers as heuristic selectors in Selection Hyper-heuristics (SHHs). Their performance is compared to existing successful architectures accommodated, i.e. Long Short-Term Memory (LSTM) and Temporal Convolutional Networks (TCNs) on 1-D Bin Packing. This study essentially utilizes one standard and one specialized Transformer, iTransformer, variants as alternative to LSTM and TCN. The empirical results revealed that the iTransformer model can provide significantly better solution than other tested models, under a revised move-acceptance criterion.

Future work will expand the instance set beyond the 12 bin packing instances. Additional Transformer variants will be evaluated, and an instance-aware sequence selection mechanism. The setup will also be extended to 8 other combinatorial optimization problems in HyFlex.

REFERENCES

[1] A. Schrijver *et al.*, *Combinatorial optimization: polyhedra and efficiency*. Springer, 2003, vol. 24, no. 2.

[2] M. Mısır, "Hyper-heuristics: autonomous problem solvers," *Automated design of machine learning and search algorithms*, pp. 109–131, 2021.

[3] M. Mısır, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe, "An investigation on the generality level of selection hyper-heuristics under different empirical conditions," *Applied Soft Computing*, vol. 13, no. 7, pp. 3335–3353, 2013.

[4] R. Martí, M. Sevaux, and K. Sörensen, "Fifty years of metaheuristics," *European Journal of Operational Research*, vol. 321, no. 2, pp. 345–362, 2025.

[5] K. Sörensen, M. Sevaux, and F. Glover, "A history of metaheuristics," in *Handbook of heuristics*. Springer, 2025, pp. 991–1010.

[6] M. G. Epitropakis and E. K. Burke, "Hyper-heuristics," in *Handbook of Heuristics*. Springer, 2025, pp. 687–743.

[7] X. Wu, D. Wang, L. Wen, Y. Xiao, C. Wu, Y. Wu, C. Yu, D. L. Maskell, and Y. Zhou, "Neural combinatorial optimization algorithms for solving vehicle routing problems: A comprehensive survey with perspectives," *arXiv preprint arXiv:2406.00415*, 2024.

[8] F. Da Ros, M. Soprano, L. Di Gaspero, and K. Roitero, "Large language models for combinatorial optimization: A systematic review," *arXiv preprint arXiv:2507.03637*, 2025.

[9] M. Schiffer, H. Hoppe, Y. Su, L. Bouvier, and A. Parmentier, "Combinatorial optimization augmented machine learning," *arXiv preprint arXiv:2601.10583*, 2026.

[10] R. Zhang, J. Wang, C. Liu, K. Su, H. Ishibuchi, and Y. Jin, "Synergistic integration of metaheuristics and machine learning: latest advances and emerging trends," *Artificial Intelligence Review*, vol. 58, no. 9, p. 268, 2025.

[11] T. Dokeroglu, T. Kucukyilmaz, and E.-G. Talbi, "Hyper-heuristics: A survey and taxonomy," *Computers & Industrial Engineering*, vol. 187, p. 109815, 2024.

[12] J. Li, Y. Lin, and M. Mısır, "Neural network based heuristic selection for selection hyper-heuristics." 2023 IEEE Congress on Evolutionary Computation (CEC), 2023, pp. 1–7.

[13] C. Li, X. Wei, J. Wang, S. Wang, and S. Zhang, "A review of reinforcement learning based hyper-heuristics," *PeerJ Computer Science*, vol. 10, p. e2141, 2024.

[14] D. V. A. Nguyen, A. Gunawan, M. Misir, L. K. Hui, and P. Vansteenwegen, "Deep reinforcement learning for solving the stochastic e-waste collection problem," *European Journal of Operational Research*, 2025.

[15] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, "itransformer: Inverted transformers are effective for time series forecasting," *arXiv preprint arXiv:2310.06625*, 2023.

[16] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. J. Parkes, S. Petrovic *et al.*, "Hyflex: A benchmark framework for cross-domain heuristic search," in *Evolutionary Computation in Combinatorial Optimization: 12th European Conference, EvoCOP 2012, Málaga, Spain, April 11-13, 2012. Proceedings 12*. Springer, 2012, pp. 136–147.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[19] Y. Liu, H. Wu, J. Wang, and M. Long, "Non-stationary transformers: Exploring the stationarity in time series forecasting," *Advances in neural information processing systems*, vol. 35, pp. 9881–9893, 2022.

[20] G. Noarov, S. Mallick, T. Wang, S. Joshi, Y. Sun, Y. Xie, M. Yu, and E. Dobriban, "Foundations of top-k decoding for language models," *arXiv preprint arXiv:2505.19371*, 2025.