Neural Improvement Heuristics for Graph Combinatorial Optimization Problems

Andoni I. Garmendia, Josu Ceberio, *Member, IEEE*, and Alexander Mendiburu, *Member, IEEE*

Abstract— Recent advances in graph neural network (GNN) architectures and increased computation power have revolutionized the field of combinatorial optimization (CO). Among the proposed models for CO problems, neural improvement (NI) models have been particularly successful. However, the existing NI approaches are limited in their applicability to problems where crucial information is encoded in the edges, as they only consider node features and nodewise positional encodings (PEs). To overcome this limitation, we introduce a novel NI model capable of handling graph-based problems where information is encoded in the nodes, edges, or both. The presented model serves as a fundamental component for hill-climbing-based algorithms that guide the selection of neighborhood operations for each iteration. Conducted experiments demonstrate that the proposed model can recommend neighborhood operations that outperform conventional versions for the preference ranking problem (PRP) with a performance in the 99th percentile. We also extend the proposal to two well-known problems: the traveling salesman problem and the graph partitioning problem (GPP), recommending operations in the 98th and 97th percentile, respectively.

Index Terms— Combinatorial optimization (CO), graph neural networks (GNNs), hill-climbing (HC), preference ranking, reinforcement learning.

I. INTRODUCTION

COMBINATORIAL optimization problems (COPs) are present in a broad range of real-world applications, such as logistics, manufacturing, or biology [1], [2]. Due to the *NP-hard* nature of most COPs, finding the optimal solution applying exact methods becomes intractable as the size of the problem grows [3]. As a result, in the past few decades, heuristic and meta-heuristic methods have arisen as an alternative to approximate *hard* COPs in a reasonable amount of time. Initial works in the field proposed constructive heuristic methods that iteratively build a candidate solution. In general, constructive methods are developed ad hoc for the problem at hand, based on criteria and rules provided by expert knowledge. Later on, the constructive proposals were outperformed by meta-heuristic algorithms that introduced general purpose and easy-to-apply optimization paradigms.

Although meta-heuristics have become the main tool to adopt, contrary to constructive heuristics, they are evaluation-intensive algorithms, i.e., they need to exhaustively evaluate thousands or even millions of candidate solutions before arriving at a decision, making them impractical for scenarios with limited budget or online-streaming optimization [4].

Algorithms based on neural networks (NNs) play a crucial role in this regard. In recent years, deep learning (DL) techniques have exhibited remarkable performance across various machine learning tasks, drawing the attention of researchers from diverse domains, including optimization. As outlined in different reviews [5], [6], [7], DL-based approaches have been proposed as standalone solvers, parameter and/or operation selection methods, or as a component of more powerful hybrid algorithms. Our focus in this work is on standalone (end-to-end) models, highlighting their capabilities and the avenues for further research aimed at enhancing their performance. Once trained, these models can rapidly make decisions, such as determining the next change required. The first works in the topic proposed models, known as *constructive methods*, which generate a unique solution incrementally by iteratively adding an item to a partial solution until it is completed [8], [9], [10]. Conversely, later papers have introduced *improvement methods* that take a candidate solution and suggest a modification to improve it [11], [12], [13]. In fact, this improvement process can be repeated iteratively, using the modified solution as the new input of the model. The reported results, although less competitive than the state-of-the-art meta-heuristics for the most trending problems [14], [15], have captured the attention of the optimization research community as they were unimaginable some years ago. In fact, many of these proposals have outperformed the classical constructive heuristics.

However, looking at the progress of the research, we realize that the majority of the works have mainly illustrated their contributions on the travelling salesman problem (TSP) [18] and other similar routing problems. Particularly, most models work on the idea that when considering the graph representation of the COPs, the information is embedded nodewise [9], [10]. However, there are problems such as preference ranking problem (PRP) [16] or graph partitioning problem (GPP) [17], where the relevant information of the problem is edgewise or even both nodewise and edgewise. In these scenarios, nodewise proposals do not use all the available (and meaningful) information.

TABLE I
ANALYSIS OF THE MOST RELEVANT NI WORKS IN THE LITERATURE

Ref	Problems	Architecture	Dynamic input	Functionality and output
[11]	ES, JS, VRP	LSTM	Prior and posterior node information	Region picking and Rule selector modules
[12]	CVRP	Attn Net	Prior and posterior node information	Operator selection from a pool of operators
[24]	TSP	GCN, RNN	Recurrent Neural Networks	Select where to apply k-opt
[23]	CVRP	Attn Net	Only considers parts of the solution	Large Neighborhood Search: repair and destroy operators
[25]	JS, CVRP	GNN	Boolean edge features	A model that accepts local search moves and selects the operator and when to perturb
[13]	TSP, CVRP	Transformer	Positional Encoding	Node-pair modification
[26]	TSP, VRP, CVRP	Transformer	Cyclic Positional Encoding	Node-pair modification

In line with the idea that future generation algorithms will come from the combination of meta-heuristic algorithms alongside machine learning models [7], we propose a new optimization framework which can replace and improve the traditional local-search-based methods by incorporating neural improvement (NI) models. Specifically, our contribution is twofold: 1) we present an NI model to solve problems where the information is stored in the nodes, in the edges, or in both of them and 2) we show that the model can be used alone or can be incorporated as a building block, for example, for hill-climbing-based algorithms, to efficiently guide the selection of neighborhood operations.

To demonstrate the versatility and efficacy of the proposed framework, we conducted experiments across various optimization problems, including PRP, TSP, and GPP. The NI model, trained on node and edge features, demonstrated outstanding performance across all the three problems, with exceptional results for PRP. It consistently recommended the best or near-best neighbors for each problem and outperformed traditional methods in all the cases.

The rest of this article is organized as follows. Section II introduces the most prominent works tackling the development of NN models for CO, both in a constructive and improvement manner. With illustrative purposes, we present the PRP in Section III and propose an NI model in Section IV. A set of experiments are carried out in Section V and the generalization of the model to other problems is discussed in Section VI. Finally, Section VII concludes this article.

II. RELATED WORK

Although NNs have been used since the decade of the 80s to solve COPs in the form of Hopfield networks [19], it is only recently [6], [7] that advancements in computation power and the development of sophisticated architectures have enabled more efficient and increasingly competitive applications. As mentioned previously, the NN-based optimization methods can be divided into two main groups according to their strategy.

A. Neural Constructive Methods

Most of the DL-based works develop policies to learn a constructive heuristic. These methods start from an empty solution and iteratively add an item to the solution until it is completed. In one of the earliest works in the neural combinatorial

optimization (CO) paradigm, Bello et al. [8] used a pointer network model [20] to parameterize a policy that constructs a solution, item by item, for the TSP. Motivated by the results in [8], and mainly focusing on the TSP, DL practitioners have successfully implemented different architectures such as graph NNs (GNNs) [21], [22] or attention-based networks [9], [10].

Since the performance of the baseline models is still far from optimality (mostly in instances with more than a few hundred nodes), they are usually enhanced with supplementary algorithms, such as active search [8], sampling [9], or beam search [20], which augment the solution diversity at the cost of increasing the computational time. As will be seen in the following, improvement methods offer a more efficient alternative, directly learning the transition from the current solution to a better one.

B. NI Methods

The NI methods depart from a given solution and iteratively propose a (set of) modification(s) to improve it until the solution cannot be further improved. The NI methods use the learned policy to navigate intelligently across the different neighborhoods.

To that end, the architectures previously used for constructive methods have been reused for implementing improvement methods. Chen and Tian [11] use long short-term memory (LSTM) to parameterize two models: the first model outputs a score or probability for each region of the solution to be rewritten, while a second model selects the rule that modifies that region. Lu et al. [12] use the attention-based model to select a local operator among a pool of operators to solve the capacitated vehicle routing problem (VRP). Using also an attention network, Hottung and Tierney [23] propose a neural large neighborhood search that suggests new solutions destroying and repairing parts of the current solution.

Closer to our proposal, but limited to routing problems, Wu et al. [13] train a policy that selects the node-pair to apply a local operator, e.g., 2-opt. Similarly, da Costa et al. [24] generalize the prior work to select k-opt operators. Falkner et al. [25] propose an improvement method to tackle the job scheduling problem which learns how to control the local search in three aspects: acceptance of the solution, neighborhood selection, and perturbations. We have summarized the characteristics of the most relevant NI works in Table I.

Improvement methods do not only incorporate the stationary instance data but also need to consider the present solution.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

IRAZUSTA GARMENDIA et al.: NI HEURISTICS FOR GRAPH COPs

3

In fact, the difficulty of encoding the solution information into a latent space is a major challenge for most of the combinatorial problems.

As an example, there are various ways of representing solutions in routing problems. Each node (or city) can maintain a set of features that indicate the relative position in the current solution, such as the location and distance to the previously and subsequently visited nodes [12]. However, this technique does not consider the whole solution as one, as it only contemplates consecutive pairs of nodes in the solution. A common strategy in this case is to incorporate positional encodings (PEs), which capture the sequence of the visited cities (nodes) in a given solution [9]. Recently, Ma et al. [26] proposed a *cyclic PE* that captures the circularity and symmetry of the routing problem, making it more suitable for representing solutions than the conventional PE.

Nevertheless, in some graph problems the essential information is codified solely in the edges, and thus, prior methods that focus on node embeddings [12], [26] are not capable of properly encoding the relative information.

Even though there are few works that consider edge weights to encode problem-specific features [27], [28], they focus on creating a heatmap of probabilities for each edge to belong to the optimal solution and use it to construct (or sample) a (set of) solution(s). In this work, similar to [27] and [28], we use nodes and edges to represent graph data. However, what sets our approach apart is its ability to encode both instance and solution information, and then use this encoded information to propose a local improving move. In addition, when compared with [13], we provide some guidelines for generalizing to different graph-based problems, by considering both the node and edge features. Furthermore, unlike [13], we do not rely on PEs to embed current solution information, as we naturally embed it in the edge features. Finally, we propose a standalone NI model and demonstrate how to combine it with classical local search techniques such as multistart hill-climbing (HC), tabu search (TS), or iterative local search.

In Section III, we will present an optimization problem that illustrates the need to develop new NI models that also consider edge features.

III. PREFERENCE RANKING PROBLEM

Ranking items based on preferences or opinions is, in general, a straightforward task if the number of alternatives to rank is relatively small. Nevertheless, as the number of alternatives/items increases, it becomes harder to get full rankings that are consistent with the pairwise item preferences. Think of ranking 50 players in a tournament using their paired comparisons from the best performing player to the worst. Obtaining the ranking that agrees with most of the pairwise comparisons is not trivial. This task is known as the PRP [16]. Formally, given a preference matrix $B = [b_{ij}]_{N \times N}$ where entries of the matrix b_{ij} represent the preference of item i against item j, the aim is to find the simultaneous permutation ω of rows and columns of B so that the sum of entries in the upper triangle of the matrix is maximized [see (1)].

Note that row i in B describes the preference vector of item i over the rest of $N - 1$ items, while column i denotes
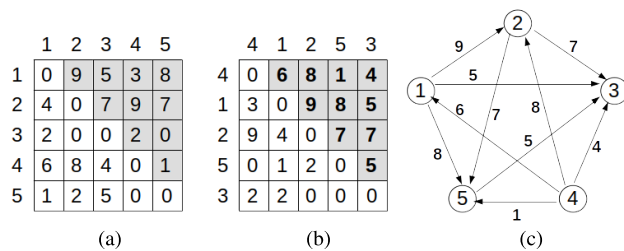

Fig. 1. Example of a PRP instance of size $N = 5$. (a) Preference matrix of size $N = 5$ ordered as the identity permutation $\omega_e = (1\ 2\ 3\ 4\ 5)$. Entries of the matrix contributing to the objective function are highlighted in grey, and the sum of the entries in the upper diagonal gives the objective value, which is 51. (b) For this instance, the optimal solution is given by the permutation $\omega_{\text{opt}} = (4\ 1\ 2\ 5\ 3)$ with an objective value of 60. (c) Equivalent graph representation of the optimal solution. Edge weights denote the preference given by the optimal solution.

the preference of the rest of the items over item i. Thus, to maximize the upper triangle of the matrix, preferred items must precede in the ranking [see Fig. 1(a)]

$$f(\omega) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} b_{\omega(i)\omega(j)}. \tag{1}$$

In addition to the matrix representation [Fig. 1(a)], the problem can be formulated as a complete bidirected graph where nodes represent the set of items to be ranked and the weighted edges denote the preference between items. A pair of nodes i and j have two connecting edges (i, j) and (j, i), with weights b_{ij} and b_{ji} that form the previously mentioned preference matrix B. A solution (permutation) to the PRP can also be represented as an acyclic tournament on the graph, where the node (item) ranked first has only outgoing edges, the second in the ranking has 1 incoming edge, the rest are outgoing, and so on until the last ranked node, which only has incoming edges [see Fig. 1(b) and (c)].

Ranking from pairwise comparisons is a ubiquitous problem in modern machine learning research. It has attracted the attention of the community due to its applicability in various research areas, including, without being limited to: machine translation [29], economics [30], corruption perception [31] or any other task requiring a ranking of items, such as sport tournaments, web search, resource allocation, and cybersecurity [32], [33], [34].

IV. METHOD

The idea of solving a graph problem iteratively with an NI model can be formulated as a Markov decision process (MDP), where a policy π is responsible for selecting an action a at each step t based on a given state s_t of the problem. The main entities of the MDP in this work can be described as follows.

1) *State:* A state s_t represents the information of the environment at step t. In this case, the state gathers data from two information sources: 1) stationary data, i.e., the instance to be solved and 2) dynamic data, that is, the current solution ω_t for the problem at step t.
2) *Action:* At every step, the learnt policy selects an action a_t, which involves a pair of items of the current solution that, according to the policy, need to be
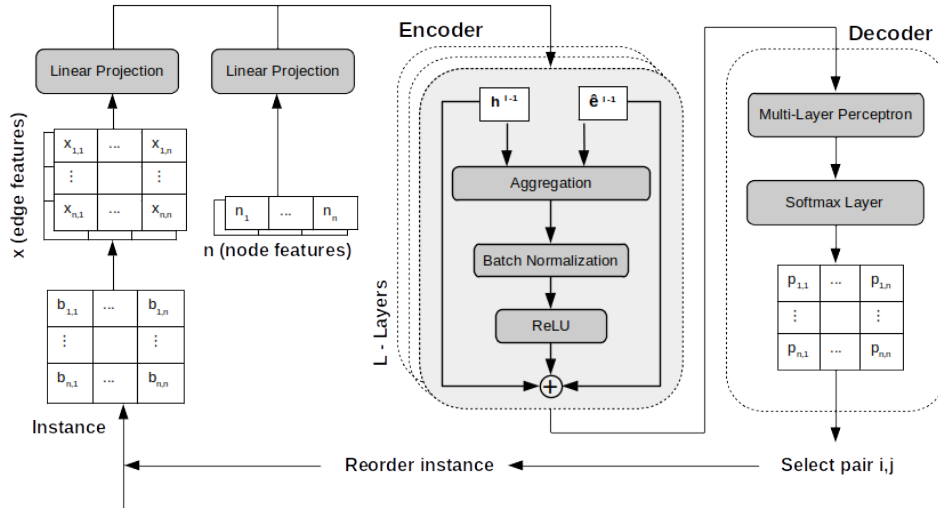
Fig. 2. High-level design of the NI model architecture. Node and edge features are linearly projected and fed to the encoder (GNN). Edge embeddings computed through L NN layers are passed to the decoder (MLP) and this outputs the edge probabilities, which will be later used to select the pair of nodes in which an operator is applied.

modified. Once selected, an operator will be applied, modifying the current solution. Note that one of the items, both, or more may be involved in the modification, depending on the operator.

3) *Reward:* The transition between states s_t and s_{t+1} is derived from an operator applied to a pair of items given by a_t. The reward function (RF) represents the improvement of the solution quality across states. Different function designs can be used, as will be explained in Section IV-B.

In what follows, a detailed description of the NI model is provided. Even though the design is general for any graph-based problem, for the sake of clarity, we provide illustrative examples based on the PRP. Extensions to other problems will be discussed later in Section VI.

A. NI Model

We will parameterize the policy π as an NN model with trainable parameters θ. Considering the case study presented in Section III, the model architecture needs to meet some requirements: 1) it needs to codify graph structure data; 2) it needs to be invariant to input permutations; 3) it needs to invariant to input size changes; and 4) it needs to consider the solution information. Considering that we opted to use a GNN encoder, capable of gathering both the node and edge features, and a multilayer perceptron (MLP) decoder, a simple and fast architecture that interprets the embedded features and generates a probability distribution over the set of possible actions. Fig. 2 presents the general architecture of the model. Apart from the presented model, we have analyzed two different encoder and decoder architectures and tested various hyperparameters. The outcomes of these experiments are presented in Appendix A.

1) Encoder: Given a fully connected graph with N nodes, there are $N \times N$ edges or node pairs, and each edge (i, j) has a weight b_{ij} that represents the relative information of node i with respect to node j. Note that only $N \times (N - 1)$ edges

need to be considered since edge (i, i) does not provide any useful information. As previously noted, the policy considers both instance information (stationary) and a candidate solution at time step t (dynamic). For this purpose, we will use a bidimensional feature vector $\mathbf{x}_{ij} \in \mathbb{R}^2$ for each edge (i, j). The first dimension represents whether node i precedes node j in the solution. If this is the case, it is set to b_{ij} (edge weight) or it is set to 0 otherwise. Similarly, the second dimension denotes the opposite, that is, whether node j precedes node i.

For the PRP, nodes do not reflect any problem-specific information, and thus all the nodes are initialized with the same value. In fact, following a similar strategy to that proposed by Kwon et al. [35], we use the identity vector as node features, $\mathbf{n} \in \mathbb{R}^N$. Even though all the nodes are initiated with the same value, their participation is required, as they spread edge features across the graph in the encoding: node i will gather information from edges (i, k) and (k, i), $k = 1, \ldots, N$.

Node and edge features will be linearly projected to produce d-dimensional node $\mathbf{h}_i \in \mathbb{R}^{1 \times d}$ and edge $\mathbf{e}_{ij} \in \mathbb{R}^{1 \times d}$ embeddings

$$\mathbf{h}_i = n_i * V_h + U_h \qquad (2)$$
$$\mathbf{e}_{ij} = \mathbf{x}_{ij} * V_e + U_e \qquad (3)$$

where $V_e \in \mathbb{R}^{2 \times d}$, and V_h, U_e, and $U_h \in \mathbb{R}^{1 \times d}$ are learnable parameters.

The encoding process consists of L GNN layers (denoted by the superscript l) that perform a sequential message-passing between nodes and their connecting edges. This enables the GNN layers to learn rich representations of the graph structure and capture complex relationships, such as precedence of items in the solution (see left part of Fig. 2). Equations (4) and (5) define the message-passing in each layer, where W_1^l, W_2^l, W_3^l, W_4^l, and $W_5^l \in \mathbb{R}^{d \times d}$ are also learnable parameters, BN denotes the batch normalization layer, σ is the sigmoid function, and

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

IRAZUSTA GARMENDIA et al.: NI HEURISTICS FOR GRAPH COPs

5

$\odot$ is the Hadamard product

$$\mathbf{h}_i^{l+1} = \mathbf{h}_i^l + \text{ReLU}\left(\text{BN}\left(W_1^l \mathbf{h}_i^l + \sum_{j=1}^{N}\left(\sigma\left(\mathbf{e}_{ij}^l\right) \odot W_2^l \mathbf{h}_j^l\right)\right)\right) \tag{4}$$

$$\mathbf{e}_{ij}^{l+1} = \mathbf{e}_{ij}^l + \text{ReLU}\left(\text{BN}\left(W_3^l \mathbf{e}_{ij}^l + W_4^l \mathbf{h}_i^l + W_5^l \mathbf{h}_j^l\right)\right). \tag{5}$$

The output of the encoder, which is fed to the decoder, consists of the edge embeddings of the last layer $\mathbf{e}_{ij}^L$.

2) Decoder: Edge embeddings are fed to the decoder, an MLP that converts the edge embeddings into logits $\mathbf{u_{ij}}$ in a format that can be used to select the next operator

$$\mathbf{u_{ij}} = \begin{cases} \text{MLP}\left(e_{ij}\right), & \text{if } i \neq j \\ -\infty, & \text{otherwise.} \end{cases} \tag{6}$$

The logits are then normalized using the Softmax function to produce a matrix $\mathbf{p} \in \mathbb{R}^{N \times N}$ which gives the probability of modifying the pair of items (i, j) in the candidate solution.

B. Learning

The improvement policy will be learned using the REIN-FORCE algorithm [36]. Given a state $s_t = (B, \boldsymbol{\omega}_t)$ which includes an instance B and a candidate solution $\boldsymbol{\omega}_t$ at step t, the model gives a probability distribution $p_\theta(a_t|s_t)$ for all the possible pairs of items to be modified. After performing an operation $O(\boldsymbol{\omega}_t|a_t)$ with the selected pair, a new solution $\boldsymbol{\omega}_{t+1}$ is obtained. The training is performed minimizing the following loss function:

$$\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(s, \boldsymbol{\omega}_t)}\left[-R_t \log p_\theta(s, \boldsymbol{\omega}_t)\right] \tag{7}$$

by gradient descent, where $R_t = \sum_{i=0}^{T-1} \gamma^i (r_{t+i} - r_{t+i-1})$ corresponds to the sum of cumulative rewards r_i with a decay factor γ in an episode of length T.

This artifact is a key piece of the model. It has been conceived to avoid myopic behaviors, permitting short-term and long-term strategies, as a different number of operations are allowed (T) before evaluating the quality of the movement sequence. In addition, the decay factor offers the practitioner a way to weight every movement, paying, for example, more attention to the earliest movements.

1) Reward Functions: Different RFs have been proposed in the literature for obtaining r. Lu et al. [12] use a reward function (RF1) that takes the objective value of the initial solution as the baseline, and, for each subsequent action, the reward at step t is defined as the difference between $f(\boldsymbol{\omega}_t)$ and the baseline. The drawback of this function is that rewards may get larger and larger, and moves that worsen the sequence can be given positive reward (as they are better than the baseline).

Alternatively, the most common approach in recent works [13], [26] is to define the reward (RF2) as $r_t = \max[f(\boldsymbol{\omega}_{t+1}), f(\boldsymbol{\omega}_t^*)] - f(\boldsymbol{\omega}_t^*)$, where $f(\boldsymbol{\omega}_t^*)$ is the objective value of the best solution found until time t. Note that this alternative yields only nonnegative rewards, and all the actions that do not improve the solution receive an equal reward $r_t = 0$. In our case, we propose a simple but effective reward function (RF3) $r_t = f(\boldsymbol{\omega}_{t+1}) - f(\boldsymbol{\omega}_t)$, which defines the
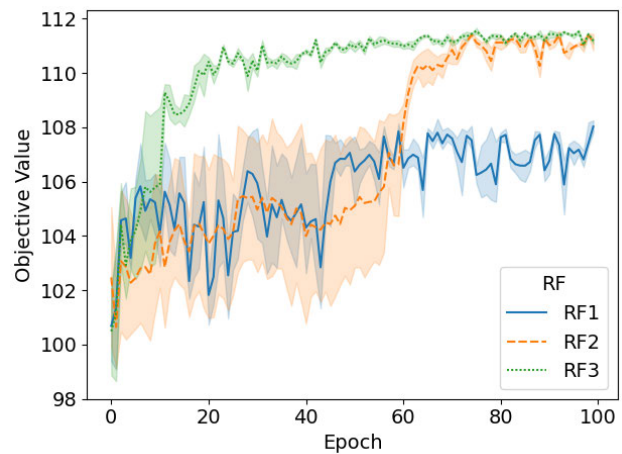

Fig. 3. Comparison of the three RFs RF1, RF2, and RF3. Model has been trained for 100 epochs, and at each epoch a model checkpoint is saved and used to solve a test benchmark of 512 randomly generated PRP instances of size 20. Obtained objective values are saved to form the curves.

reward as the improvement of the objective value between steps t and $t + 1$, and also considers negative values. RF3 RF yields a faster convergence with less variability, as can be seen in the comparison of the mentioned RFs depicted in Fig. 3.

2) Automated Curriculum Learning: Curriculum learning is a training strategy that involves controlling the difficulty of the samples throughout the process, where the difficulty level is manually increased [37]. In this context, difficulty is measured by calculating the percentage of moves that worsens the objective value in relation to all the possible moves. However, in our approach, we do not use a manual curriculum learning strategy like the one used by Ma et al. [26]. Instead, we use an iterative approach where the model receives the solution that has been modified in the previous iteration. As the model improves and suggests better moves for the current instances during training, the number of improving moves decreases, leading naturally to increase the difficulty level. This iterative approach enables an automated curriculum learning, which is valid for prior optimization problems and does not require neither any problem knowledge nor external intervention.

However, learning is performed with a batch of instances and not all of them reach a local optimum in the same number of steps. Thus, we save the best average reward obtained by the model, and we consider the algorithm to be stuck when it does not improve the best average reward for $K_{\max}$ iterations. A large value of $K_{\max}$ should give the model more chances to visit higher quality solutions, but can also introduce undesired computational overheads.

3) Operator: The model is flexible, allowing the practitioner to define the operator that best fits the problem at hand. In the particular case of the PRP, there are several operators that could be applied to perform the pairwise modification, such as *insert*, *swap*, *adjacent-swap*, and *reverse* operators.[1] Previous works [38] have demonstrated that the *insert* operator

---

[1] Given an edge (i, j), denoting the items in positions i and j in the solution: the **insert** operator consists of removing the item at the position i and placing it at position j, the **swap** exchanges the items at both the positions, the **adjacent-swap** is a swap that only considers adjacent positions, and the **reverse** operator reverses the subpermutation between positions i and j.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

6

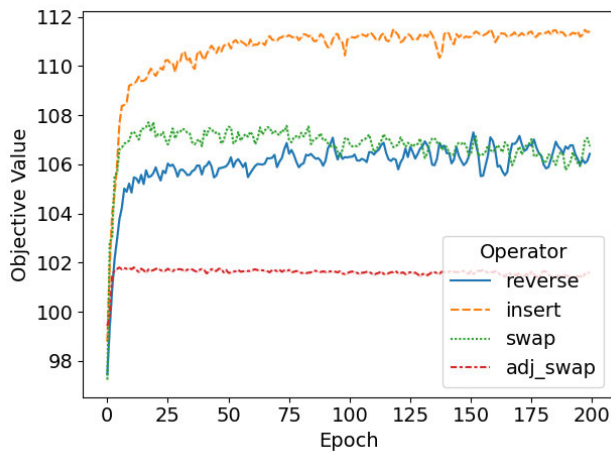IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS

Fig. 4. Training curves using different pairwise operators. Each operator is used in a training procedure of 200 epochs executed five times with a different random seed. The model output is masked so that, for each operator, only feasible pairs can be selected.

Algorithm 1 Training Algorithm for the NI Model

Require: total number of epochs n_{epochs}, stopping criterion K_{max}, operator O, episode length T, learning rate α, and discount factor γ.

1:   Initialize the policy π_θ with random weights θ
2: **for** $epoch = \{1, \ldots, n_{epochs}\}$ **do**
3:      State $s_0 \leftarrow$ RNDMGEN., step $t \leftarrow 0$, count $k \leftarrow 0$
4:      $r_{best} \leftarrow$ COMPUTEREWARD(s_0)
5:      **while** $k \leq K_{max}$ **do**
6:         $t \leftarrow t + 1$
7:         $p_\theta \leftarrow \pi_\theta(s_{t-1})$
8:         $a_t \leftarrow$ SAMPLEACTION(p_θ)
9:         $s_t \leftarrow O(s_{t-1}, a_t)$
10:        $r_t \leftarrow$ COMPUTEREWARD(s_t)
11:        $(r_{best}, k) \leftarrow$ UPDATEBEST(r_t, r_{best}, k)
12:        **if** $t \bmod T = 0$ **then**
13:           $R = \frac{1}{T} \sum_{i=t-T}^{t} (\sum_{j=i}^{t} \gamma^{j-i}(r_i - r_{i-1}))$
14:           $\nabla_\theta J(\theta) \leftarrow -R \nabla_\theta \log p_\theta(a_t)$
15:           $\theta \leftarrow \theta +$ GRADIENTCLIP$(\alpha \nabla_\theta J(\theta))$
16:        **end if**
17:     **end while**
18: **end for**

---

yields the best results for the PRP. In fact, we confirm it in Fig. 4, where we show the comparison of the training convergence curves for the mentioned pairwise operators.

4) Training Algorithm: The implemented training algorithm is presented in Algorithm 1. The state s_t represents the instance and a candidate solution at time step t. At each epoch, a random state is generated with a random solution (line 3). Then, the model gives the probability vector (line 7), and an action is sampled and applied to the current solution (lines 8 and 9). The best found reward is saved (line 10) and the process is repeated until the model does not improve the best known solution for a specified number of steps (K_{max}) consecutively. For the sake of clarity, Algorithm 1 shows the training with one instance, even though a batch is used. To use a batch of instances, states $\mathbf{s}_t$, probabilities $\mathbf{p}_\theta$, actions $\mathbf{a}_t$, and rewards $\mathbf{r}_t$ will be vectors and the average of $\mathbf{r}_t$ will be used to control the improvement condition (line 11).

C. Applications of the NI Model

The HC heuristic is a procedure that continuously tries to improve a given solution performing local changes (for example, swapping two items) and looking for better candidate solutions in the neighborhood. Examples of conventional HC procedures include, among others, best first HC (BFHC), which selects the first found candidate (neighbor) that improves the present solution; steepest-ascent HC (SAHC), which exhaustively searches all the neighborhood and selects the best candidate solution from it; or stochastic HC (SHC), which randomly picks one solution from the neighborhood.

1) Neural Hill Climber: With the goal of reducing the large number of evaluations needed by the conventional HC, we propose the neural hill climber (NHC), which attempts to suggest the best neighbor based on a given solution and then uses this neighbor as the starting point for the next iteration. This cycle is repeated until the best possible solution is found. In general, HC heuristics do not allow the objective value to decrease. In our approach, to ensure that the selected action is an improving move, we sort the probability vector given by

the model and select the first action that improves the solution. Of course, other strategies could also be adopted.

2) Advanced Hill Climbers: Eventually, just as the conventional HC procedures, the NHC will get stuck in a local optimum where an improving move cannot be found. More advanced algorithms have been proposed in the literature that try to escape the local optima by performing a restart or a perturbation to the current solution [39]. One of the many examples is the multistart HC (msHC), which restarts the search departing from another random candidate solution whenever an improving move cannot be found.

An alternative to msHC is the iterated local search (ILS) [41]. Once the search gets stuck in a local optimum, instead of restarting the algorithm, ILS perturbs the best solution found so far and the search is resumed in this new solution. The perturbation level is dynamically changed based on the total budget left (number of evaluations or time).

The third example considered in this article is the TS [40], which enhances the performance of the HC method allowing worsening moves whenever a local optimum is reached. To avoid getting trapped in cycles, TS maintains a tabu memory of previously visited states to prevent visiting them again in the next m moves.

We will use the NI model to guide the local moves of a multistart neural HC (msNHC), a neural ILS (NILS), and a neural TS (NTS) and analyze their performance compared with the conventional versions in Section V.[2]

---

[2]For TS, a short-term memory of size 200 is used. For ILS, once a local optima is reached, the solution is perturbed with n random swaps, n being calculated based on the remaining evaluations (REs) in the execution: $n = \lfloor (N/2) * (\text{RE}/E) \rfloor$, where N is the problem size and E denotes the evaluation budget.

V. EXPERIMENTS

In this section, we present a thorough experimentation of the proposed NI model. First, we analyze the performance of the NI model in the short-term (*one-step*) and long-term (*multistep*) basis. Afterward, we test the efficiency of the NI model implemented as a building block of HC algorithms.

A. Setup

For the experiments, we deploy a three-layer GNN as the encoder, being the embedding dimension 128. As the decoder, we use an MLP of four layers with hidden dimensions (128, 64, 32, and 1) and ReLU activation. Regarding the training hyperparameters, a learning rate is set to 1e-4, the episode length to $T = 20$, the decay factor to $\gamma = 0.1$, and the maximum number of consecutive nonimproving moves to $K_{max} = 5$. Further details on the selection of the hyperparameters can be found in Appendix A. To train the NI model, we adopt a common practice of using randomly generated instances. Each epoch, a different batch of 64 instances is used, and gradients are averaged across this batch to update the model parameters. Due to limited computational resources, two different models have been trained using sizes $N = 20$ and $N = 40$, respectively. Both the models have been trained for 5000 epochs.[3]

Related to the size of the instances used for training DL-based models, a common drawback is the lack of scalability: most of the DL-based models need to be trained with instances of the same size to those used later for inference. However, the introduced model can be trained with a computationally affordable instance size, and then be used for solving larger size instances (discussed later in this section). If not mentioned differently, for large sizes, the model trained with instances of size ($N = 20$) will be used.

We have implemented the algorithms using *Python 3.8*. Neural models have been trained in an *Nvidia RTX A5000* GPU, while methods that do not need a GPU are run on computers with *Intel Xeon X5650* CPUs and 64 GB of memory.

B. NI Model Performance Analysis

During the optimization process, obtaining solutions that improve the current one becomes harder at each step. With the aim of testing the ability of the model, we present two scenarios: 1) one-step, that is, departing from a random solution, is the model able to find the best neighbor and 2) multistep, departing from a random solution, is the model able to propose a better solution repeatedly, improving the previously found solution in a consistent way throughout the optimization process.

1) One-Step: We focus on the capability of the NI model to provide a solution that outperforms the present one. In terms of neighborhood, we expect the NI model to be able to identify the best or at least one of the best neighboring moves.

---

[3]Even though we use instances of the same size to train each model, note that, due to the elementwise operations performed in the encoder, it is possible to combine instances of different sizes.
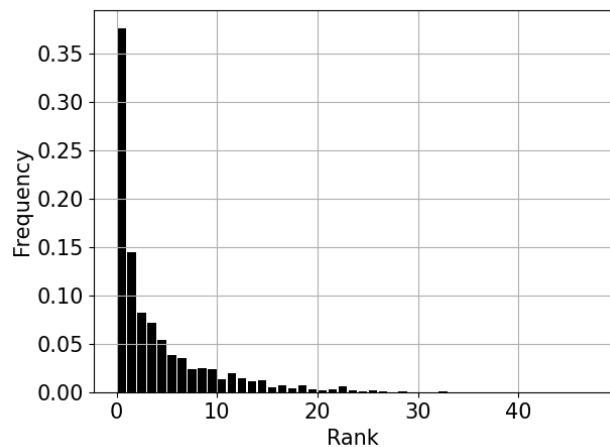

Fig. 5. Histogram that shows the ranking of the action selected by the model among all the possible actions on 2000 different instances of size $N = 20$. The first rank denotes the best possible action (based on the one-step improvement). Note that the x-axis is cut since none of the proposed actions has been ranked between the 50th and 361st positions.

We perform one-step predictions for 2000 instances of size $N = 20$. The results are depicted in Fig. 5 in the form of a histogram. Specifically, we calculated the ranking of the predicted move among all the available actions ($(N-1)^2$, 361). Note that the model selects the best neighbor in more than 35% of the times. Considering all the possible insert operations, on average, the action that the model takes is in the 99th percentile, ranked fourth out of 361.

In larger instance sizes, even if trained with $N = 20$, the NI model is not only able to maintain the good performance, but also improves the percentile rank of the selected action. It selects, on average, the 13rd best action out of 2401 for $N = 50$ and the 32nd out of 9801 for $N = 100$.

2) Multistep: Once tested that the NI model successfully learns to select one of the best possible actions (one-step), we still need to verify whether it obtains increasingly better solutions in a multistep scheme. For that purpose, we let the model make consecutive moves, being fed, at each iteration, with the solution obtained in the previous one. Fig. 6 illustrates the behavior of the model for consecutive steps. The figure shows that: 1) as expected the maximum obtainable reward decreases over the improvement steps, presumably increasing the difficulty, and 2) the action selected by the model is closer to the maximum reward than the minimum reward, which confirms the good performance of the model when several steps are completed.

C. Neural HC Performance Analysis

In this section, we compare the performance of NHC to two conventional approaches: steepest ascent (SAHC) and best first (BFHC). We let the algorithms run until they get trapped in a local optima and repeat the optimization for 2000 different instances. During the optimization run, we computed the gap (%) to the optimum objective value of the instance and the consumed evaluations.

Fig. 7 shows the results of the experiment. NHC, SAHC, and BFHC obtain an average gap of 0.28%, 0.29%, and 0.28%,
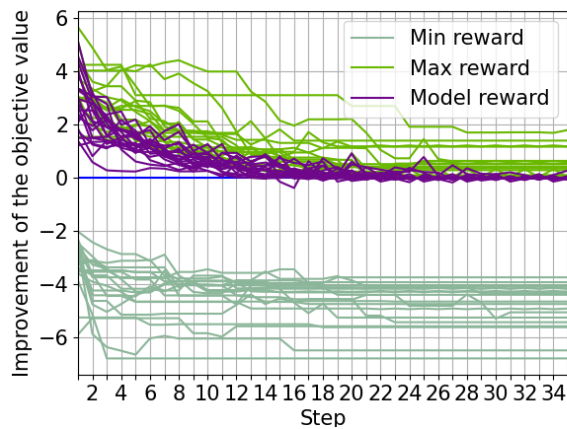
This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

8 IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS

Fig. 6. Line plot of 20 different executions of the NHC. Each purple line represents the reward, or the improvement of the objective value (y-axis), given by the selected action over consecutive steps (x-axis). The minimum and maximum rewards that could be obtained in each step are also shown. Instances of $N = 20$ are solved using models trained with randomly generated instances of the same size.
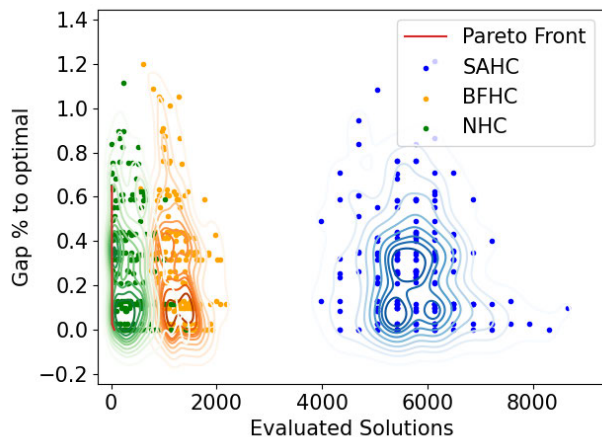

Fig. 7. Bicriteria analysis regarding the gap (%) to optimal value and the total number of objective value evaluations. The lower left corner is best. Instances of $N = 20$ are solved using models trained with randomly generated instances of the same size.

respectively. Regarding the number of solution evaluations, NHC is the cheapest among all the three procedures evaluating on average 355 solutions. In all, 948 solutions are explored by BFHC, and 6293 by SAHC, being the most expensive procedure since it needs to evaluate all the possible actions (entire neighborhood) before performing a move. All the solutions in the Pareto front belong to NHC (see the red line). In summary, NHC performs as well as SAHC and BFHC, but requires a significantly lower number of evaluations.

D. Advanced Hill Climbers' Performance Analysis

We expand here the analysis of the advanced algorithms explained in Section IV-C. The performance is measured for several instance sizes (20, 50, 100, and 500) setting three different maximum numbers of evaluations: 10, 100, and 1000 N, with N being the size of the instance. We incorporate the following algorithms for comparison: msHC with conventional search strategies, such as best first (msBFHC), steepest ascent (msSAHC), and stochastic (msSHC); the neural

version, guided with an NI model trained using instances of 20 and 40 (msNHC-20, msNHC-40); and TS algorithm with an underlying best-first strategy (BFTS), the NTS algorithm, an ILS algorithm with a best-first selection (BFILS), and its neural version (NILS). As a baseline method, we also add the Becker constructive method [42]. The performance is measured by means of the average gap percentage to the best known objective value, given by the state-of-the-art meta-heuristic [43]. For each size, we use 512 randomly generated instances.

The results are presented in Table II. As we can see, the neural variants (msNHC, NTS, and NILS) outperform the respective conventional ones in all the cases. While the conventional methods perform similarly, the neural multistart method shows the largest improvement compared with its conventional variant.

Training this kind of DL-based models for large size instances becomes computationally demanding. Thus, designing a model that can be trained on small sizes and later applied to larger instances becomes advisable or, depending on the context, even mandatory. As shown in Table II, models trained for sizes 20 and 40 are able to outperform their traditional counterpart on larger instances (up to 500). Particularly, even though msNHC-40 shows a better behavior compared with msNHC-20, the differences between them are quite small, demonstrating proper generalization of the model in larger sizes. We further evaluate the generalization of the model trained with randomly generated instances to other types of instances in Appendix B.

E. Computational Time

A common discussion in the optimization area is whether the number of evaluations or the computational time should be used when comparing different algorithms. However, it is important to note that different programming languages and hardware platforms may produce varying results. Therefore, this section has been added to supplement the previous one, by offering information on the required computational times for both training and inference of the presented model.

1) Inference Time: The performance of the HC algorithms has been measured using a maximum number of solution evaluations as a limit. However, one could argue that neural methods have, for an equal number of evaluations, a higher time cost compared with the conventional ones, i.e., getting the next movement from the NN is more costly than choosing a movement randomly or selecting the next movement of a greedy sequence. Nevertheless, the reduced amount of evaluations of NHC compensates this, obtaining a far better performance, especially for larger instances.

In fact, as can be seen in Fig. 8, NI-based heuristics outperform conventional heuristics under the time criterion. Neural variants obtain a major advantage in the beginning of the execution, where the decrease in the optimality gap is steeper. Note that to fairly compare with the conventional search strategies, we do not make use of a batch of instances for the model inference; instead, we only feed a unique instance simultaneously. Otherwise, NHC would greatly

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

IRAZUSTA GARMENDIA et al.: NI HEURISTICS FOR GRAPH COPs

9

TABLE II
COMPARISON OF IMPROVEMENT METHODS. AVERAGE GAP (%) TO THE BEST KNOWN VALUE FOR DIFFERENT MAXIMUM NUMBERS OF SOLUTION EVALUATIONS (E). BEST RESULTS AMONG EACH KIND OF ALGORITHMS ARE HIGHLIGHTED IN BOLD

| Method	N=20			N=50			N=100			N=500		
	E20	E200	E2000	E50	E500	E5000	E100	E1000	E10000	E500	E5000	E50000
Becker		3.38%			3.30%			2.87%			1.63%	
msBFHC	10.97%	2.58%	0.28%	8.60%	2.89%	0.68%	6.80%	2.67%	0.82%	4.03%	1.87%	0.73%
msSAHC	13.59%	7.72%	0.49%	10.92%	8.93%	2.35%	8.67%	7.91%	3.95%	4.51%	4.44%	3.89%
msSHC	13.30%	10.21%	8.07%	10.60%	9.32%	8.42%	8.50%	7.86%	7.36%	4.44%	4.31%	4.22%
msNHC-20	2.46%	**0.85%**	**0.14%**	2.93%	1.34%	0.65%	3.42%	1.66%	0.79%	3.07%	1.71%	0.78%
msNHC-40	**2.09%**	0.92%	0.16%	**2.25%**	**1.24%**	**0.63%**	**2.98%**	**1.66%**	0.77%	**2.92%**	**1.59%**	**0.69%**
BFTS	10.94%	2.56%	0.36%	8.57%	2.88%	0.71%	6.79%	2.64%	0.81%	3.52%	1.72%	0.75%
NTS	2.46%	**0.85%**	0.25%	2.93%	1.34%	0.64%	3.42%	1.67%	**0.75%**	3.07%	1.69%	0.72%
BFILS	10.98%	2.46%	0.28%	8.55%	2.91%	0.68%	6.79%	2.65%	0.82%	3.52%	1.74%	0.74%
NILS	2.46%	**0.85%**	0.15%	2.93%	1.34%	0.63%	3.42%	1.66%	0.78%	3.07%	1.61%	0.71%
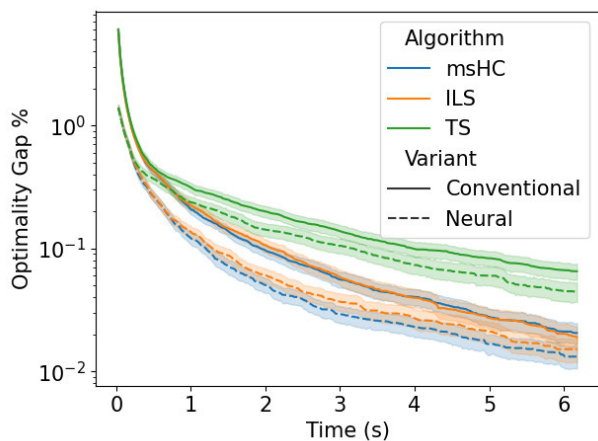
Fig. 8. Average optimality gap % of 512 executions solving instances of $N = 20$. The x-axis denotes the computation time.
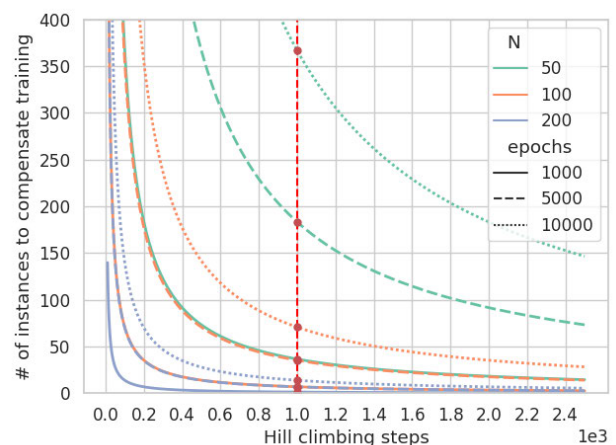

Fig. 9. Plot illustrates the minimum number of instances to be solved to compensate the training of the model, for each execution time. Trained model sizes are differentiated with different colors, while different numbers of epochs have different line styles (5000 epochs are used in this article).

benefit from using a batch of instances, as done in training, due to parallelization.

2) Training Time: When evaluating a learning framework, apart from the inference time, considering training time is also a crucial factor. While computation time for inference is well-researched in the literature, training time is often given less attention. This can pose a challenge when, for example, a model must be trained very frequently, and it is applied to solve a few instances each time. The RL frameworks typically require a large amount of computation time for training, which can be expensive in terms of resources and time. If a pretrained model is not available, conventional methods may be more suitable for solving a single instance since training time is much longer than the execution time required by conventional methods. However, if a pretrained model is accessible through open-source platforms or from prior practices, and generalizes well to the specific instance being solved, practitioners can still benefit from using the model.

To analyze the point where it becomes cost-effective to train and use a neural method, the following experiment is conducted. We assume that the practitioner does not possess any pretrained model. It requires the practitioner to weight the cost of training a neural model against the cost of using traditional techniques (inference time) while also considering

the expected performance improvement that the neural model can provide. For illustrative purposes, the total time required to complete an execution of the conventional HC and the neural HC is defined in (8) and (9), where y_{conv} and y_{ni} are the total time, an t_{train}, t_{infer}, and t_{neigh} denote the training and inference time of the NI model and the time required by the conventional SAHC to evaluate an entire neighborhood, respectively

$$y_{conv} = t_{neigh} * n_{steps} \tag{8}$$
$$y_{ni} = t_{train} + t_{infer} * n_{steps}. \tag{9}$$

Let T be the number of steps performed in an execution, (10) defines the minimum number of instances that need to be solved, to compensate for the training time of the NI model

$$n_{comp} \geq \left\lceil \frac{t_{train}}{T * (t_{neigh} - t_{infer})} \right\rceil. \tag{10}$$

Based on our implementation and hardware used (an RTX A5000 GPU and Intel Xeon X5650 CPUs), the conventional SAHC requires an average of 0.40, 2.06, and 10.41 s to explore an entire neighborhood of the PRP of size 50, 100, and 200, respectively. The inference time of the NI model is 4.6, 13, and 48 ms for PRP instances of sizes 50, 100, and 200, respectively. Considering these values, Fig. 9 depicts,

for these three problem sizes and three training budgets, the variation in the minimum number of instances that need to be solved to compensate for the training time. The red vertical line represents the conventional stopping criteria (1000 N^2 evaluations, $T = 1000$) used in [44], and the red dots depict the intersection with different curves. For the training duration of 5000 epochs used in this article, the minimum number of instances that need to be solved is 183, 35, and 7 for sizes 50, 100, and 200, respectively. It is noteworthy that as opposed to neural methods, the conventional methods require a search in the quadratic neighborhood, and consequently, the difference in execution times between these two approaches becomes more pronounced with larger instance sizes and the training time gets compensated with fewer instances. Therefore, the advantage provided by using the NI model increases dramatically as the instance size increases.

VI. Extension to Other Problems

In Section V, we have approached the PRP as an illustrative problem where the instance information is exclusively stored in the edges of graph representation of the problem. Nevertheless, the proposed model is not exclusive to the PRP and can be extended to any problem that falls in this category.

In fact, the proposed NI model can be used for other combinatorial problems performing only some minor changes. The main change the practitioner needs to perform resides in the edge feature selection. The NI model uses edges as the main message-passing elements. Extracting instance stationary information and dynamic solution information from edges may not be trivial for every problem. Apart from that, the practitioner needs to select the most efficient operator for the problem at hand.

To illustrate the process, we give examples on how to adapt the model to solve two different problems: the TSP [18] and the 2-partition balanced GPP [17]. The reader is headed to the corresponding papers for detailed information about these problems.

A. Traveling Salesman Problem

Given a set of n cities and their coordinates in a 2-D space $s = \{\mathbf{c_i}\}_{i=1}^{n}$ where each $\mathbf{c_i} \in \mathbb{R}^2$, the TSP consists of finding a permutation ω that orders the cities in a tour that visits each city once and has the minimum length. Formally, the TSP can be defined as a fully connected graph where the nodes represent the cities and weighted edges denote the pairwise distances or costs between cities.

In the TSP, as opposed to the PRP, certain information is contained in the nodes, namely, the city coordinates. Meanwhile, edge features ($\mathbf{x_{ij}} \in \mathbb{R}^2$) are obtained directly by the edge weights in a similar way as done for the PRP: the first dimension in $\mathbf{x_{ij}}$ denotes the distance if the edge is part of the current solution (cities i and j are consecutively visited) and zero otherwise. Similarly, the second dimension is set to nonzero (distance) for the edges that are not part of the solution.

Regarding the operator, as seen in recent works [13], [26], the 2-opt operator is better suited for routing problems than
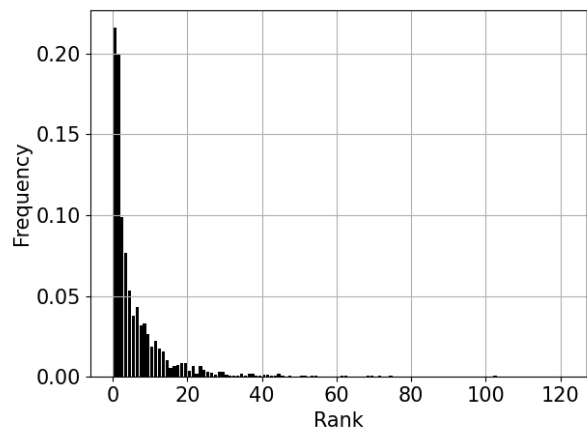

Fig. 10. TSP. Histogram showing the rankings (x-axis) of the action selected by the model among all the possible actions.

the insert or the swap operators. However, as done in Fig. 4, a set of candidate operators should be considered and evaluated for a few training epochs, to select the best one. In fact, in the experiments we opted to use the insert operation since it performs better than 2-opt.

B. 2-Partition Balanced Graph Partitioning Problem

Given a graph $G(N, E, b)$ where N denotes the set of nodes and E the set of edges with weights b, the 2-partition balanced GPP consists of finding a 2-partition of N, where the number of nodes is balanced among the sets, and which minimizes the sum of the weights of edges going from the nodes of one partition to the nodes in the other.

Regarding feature extraction, an identity vector is used as node feature, as in the previous cases. Edge features $x_{ij} \in \mathbb{R}^2$ are again obtained from the edge weights. $x_{ij} = (b_{ij}, 0)$ when edge (i, j) belongs to the cut, i.e., it is an edge between clusters, and $x_{ij} = (0, b_{ij})$ if it is not.

Regarding the operator, in this case we propose to use swaps between a pair of items in order to guarantee the balance between the two clusters. Even though the model may choose a pair of items in the same cluster, the solution would not change, and thus, the improvement will be null, forcing the model to pay attention to pairs of items that belong to different clusters.

C. Preliminary Results on TSP and GPP

We evaluated the application of the NI model in the one-step scenario for the described problems using 2000 instances. For the TSP, we used instances of 20 cities, placed uniformly at random in the unit square. For the GPP, we created random graphs with 20 nodes and 50% of connectivity with weights sampled from a uniform distribution between $(0, 1)$.

We trained an NI model for each problem following the setup described in Section V-A. In the TSP, the NI model selects, on average, the action ranked fifth out of 361 (see Fig. 10 for further details on the ranks of the selected actions). In the GPP, the NI model selects, on average, the action ranked tenth; however, among the 2000 instances, the model selected an invalid swap (swap between two items in the same cluster) in 89 cases (4%), and thus, some actions seem to be ranked
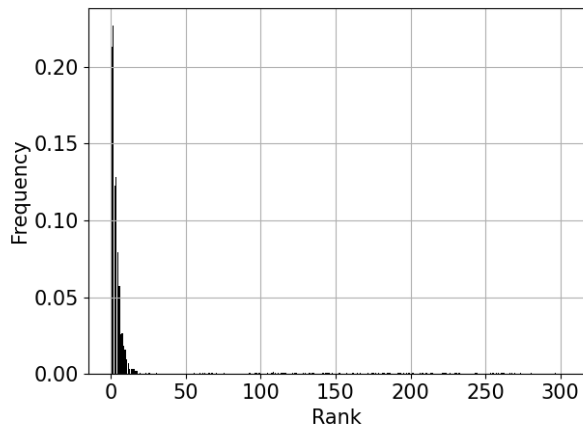
This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

IRAZUSTA GARMENDIA et al.: NI HEURISTICS FOR GRAPH COPs 11

Fig. 11. GPP. Histogram showing the rankings (*x*-axis) of the action selected by the model among all the possible actions.
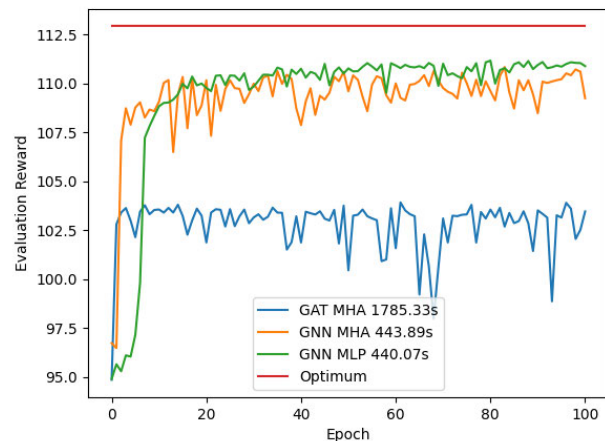

Fig. 12. Evaluation reward during training for different model architectures. Three different architectures are tested: (1) GAT encoder and MHA decoder, (2) GNN encoder and MHA decoder, and (3) GNN encoder and MLP decoder.

worse than usual (see Fig. 11). If we masked the invalid moves, the NI selects, on average, the action ranked third.

VII. CONCLUSION

This article presents an NI model for graph-based problems which, given a candidate solution, is able to propose a pairwise modification that (almost) always generates a new better solution. We have experimentally demonstrated that the NI module could replace traditional local-search strategies, since it requires less computational effort to obtain similar results, it is more flexible, and it can efficiently guide a variety of hill-climbing algorithms. This work has major implications for most of the state-of-the-art meta-heuristics used to solve CO problems, as they commonly include conventional local search procedures.

The research presented in this article represents a promising avenue for future investigation. In particular, we emphasize the importance of investigating different training strategies, given that the performance of the model is heavily influenced by the instances used in training. To this end, we suggest exploring advanced curriculum learning strategies that intelligently select training instances, as well as real-world instance generators that would allow to generate a broader range of instance closer to the target distribution.

Although the model has shown good performance, local search can be prone to becoming trapped in local optima, as is the case with the conventional approaches. We think that a more advanced model that can overcome this limitation should be developed. In particular, we suggest incorporating strategies such as curiosity-driven learning [45] and memory-based learning [46] to improve the exploration of the NI model and prevent the model from revisiting previously explored states. By taking these steps, we believe that the potential of the NI model can be fully realized, opening up exciting opportunities for future research in this area.

APPENDIX A
HYPERPARAMETER AND ARCHITECTURE SELECTION

To select the best model, we have experimented with several options. As an encoder, we have evaluated two of the most popular options, graph attention networks (GAT) [47] and

anisotropic GNN [48]. For the GAT encoder to be able to consider edges, we have used the edge-featured GAT [49]. As the decoder, we have tested a multihead attention mechanism (MHA) [50], which performs a self-attention pass over the edge embeddings before generating the output probabilities, and a simple MLP, which directly produces the output probabilities without any attention.

For evaluating different encoder and decoder architectures, we have trained three different models for 100 epochs and evaluated on a set of test instances every epoch. Fig. 12 shows the reward curves for three different architectures: 1) GAT encoder and MHA decoder; 2) GNN encoder and MHA decoder; and 3) GNN encoder and MLP decoder. As shown in the figure, the GNN encoder obtains better results compared with GAT, while there is not such difference between decoders. Moreover, the GAT encoder is time-consuming (four times slower than GNN). As a result, we have chosen GNN as the encoder. Among decoders, we have selected the MLP, since it is slightly faster and more stable than MHA.

Table III provides a description of the hyperparameters used in the proposed model. We have set the embedding dimension to 128, which is a commonly used power-of-two value. Through experimentation, we determined that this value provides the best tradeoff between performance and computational efficiency. In addition, we have set the number of layers to $L = 3$. Fewer layers are insufficient for correctly encoding the graph structure, while larger GNNs may encounter the issue of oversmoothing. Parameters (3–4) have been set with values reported with similar architectures [8], [22], [26]. The discount factor is also a key aspect in the reward engineering process. We found out that using a low discount factor ($\gamma = 0.1$) yields better results. The episode length, denoted as T, specifies how frequently the reward is updated. A model trained using shorter episodes, such as $T = 1$, tends to learn policies similar to SAHC. In contrast, longer episodes do not necessarily promote greedy behavior; instead, they allow the model to optimize for long-term rewards. The parameter $K_{\max}$ denotes the maximum number of steps the model can take

TABLE III
TRAINING AND MODEL HYPERPARAMETERS

	Hyperparameter	Symbol	Value
1	Embedding dimension	d	128
2	Number of encoding layers	L	3
3	Tanh clipping	C	10
4	Gradient norm clipping	-	1
5	Learning rate	α	1e-4
6	Batch size	B	64
7	Number of epochs	n_{epochs}	5000
8	Discount factor	γ	0.1
9	Episode length	T	20
10	Max num. of consecutive non-improving moves	K_{max}	5

TABLE IV

COMPARISON OF THE PERFORMANCE OF NHC AND THE BEST
CONVENTIONAL HC (BFHC) IN DIFFERENT INSTANCES OF
THE XLOLIB BENCHMARK. THE SHOWN VALUES ARE
AVERAGE GAPS TO THE OPTIMAL VALUE
TAKEN FROM 100 EXECUTIONS

Instance	NHC	BFHC	Instance	NHC	BFHC
N-t65b11xx_250	**3.14**%	3.66%	N-tiw56n62_250	**3.57**%	4.42%
N-stabu3_250	**3.34**%	3.97%	N-be75eec_250	**3.85**%	3.91%
N-t74d11xx_250	**3.08**%	3.61%	N-tiw56n66_250	**2.93**%	3.53%
N-be75oi_250	**3.12**%	3.32%	N-t70d11xn_250	**2.75**%	3.68%
N-t59n11xx_250	**2.89**%	4.89%	N-t75k11xx_250	**3.37**%	3.79%
N-tiw56r58_250	**3.34**%	3.78%	N-tiw56r54_250	**3.16**%	3.40%
N-t59f11xx_250	**3.36**%	3.77%	N-t70n11xx_250	**4.08**%	4.42%
N-tiw56n54_250	**2.78**%	3.65%	N-tiw56r72_250	**3.17**%	3.59%
N-t69r11xx_250	**3.70**%	3.86%	N-t65f11xx_250	**3.82**%	3.97%
N-t75d11xx_250	**3.06**%	3.68%	N-t70f11xx_250	**4.11**%	4.27%
N-be75np_250	**3.39**%	3.62%	N-tiw56n67_250	**3.45**%	3.79%
N-t75n11xx_250	**3.73**%	3.99%	N-tiw56r66_250	**3.97**%	4.20%
N-t75e11xx_250	**3.64**%	4.07%	N-t65d11xx_250	**3.43**%	4.12%
N-tiw56n58_250	**3.57**%	3.68%	N-stabu1_250	**3.00**%	3.72%
N-stabu2_250	**2.95**%	3.46%	N-tiw56n72_250	**2.89**%	3.79%
N-tiw56r67_250	**2.65**%	3.70%	N-t70b11xx_250	3.52%	**3.50**%
N-t70l11xx_250	**4.12**%	4.71%	N-t59d11xx_250	**3.37**%	3.80%
N-t65n11xx_250	**3.65**%	4.68%	N-t70d11xx_250	**3.51**%	4.38%
N-t65l11xx_250	**2.93**%	4.01%	N-be75tot_250	**3.53**%	3.96%
N-t59b11xx_250	**3.75**%	4.37%			

without improving the best discovered reward. Increasing K_{max} can provide the model with more opportunities to explore high-quality solutions. However, excessively large values are computationally expensive and unnecessary.

APPENDIX B
GENERALIZATION TO OTHER INSTANCE TYPES

The proposed model has been designed for the PRP; however, since there is no any widely used benchmark in the PRP, we have opted for the linear ordering problem (LOP) [51], a classical CO problem that can be seen as a mathematical formulation of the PRP. To further validate the proposal in real-world data, and demonstrate its ability to generalize to different instance distributions, the NHC method has been evaluated in the most popular library for the LOP, the LOLIB [52]. Particularly, we selected the most challenging instance type, the XLOLIB of size 250. Table IV summarizes the average performance gap to the best known value obtained by [43] in 100 executions limited by 1 min. NHC obtains better average gap for 38 out of 39 instances compared with the best performing conventional HC method (BFHC).

REFERENCES

[1] V. T. Paschos, *Applications of Combinatorial Optimization*, vol. 3. Hoboken, NJ, USA: Wiley, 2014.

[2] G. Naseri and M. A. G. Koffas, "Application of combinatorial optimization strategies in synthetic biology," *Nature Commun.*, vol. 11, no. 1, p. 2446, May 2020.

[3] M. R. Garey and D. S. Johnson, "Computers and intractability," in *A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Macmillan, 1979.

[4] J. Chen, C. Liu, and M. Tomizuka, "FOAD: Fast optimization-based autonomous driving motion planner," in *Proc. Annu. Amer. Control Conf. (ACC)*, Jun. 2018, pp. 4725–4732.

[5] E.-G. Talbi, "Machine learning into metaheuristics: A survey and taxonomy," *ACM Comput. Surv.*, vol. 54, no. 6, pp. 1–32, Jul. 2022.

[6] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Comput. Oper. Res.*, vol. 134, p. 105400, 2021.

[7] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, 2021.

[8] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016, *arXiv:1611.09940*.

[9] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" 2018, *arXiv:1803.08475*.

[10] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "POMO: Policy optimization with multiple optima for reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 21188–21198.

[11] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, p. 32.

[12] H. Lu, X. Zhang, and S. Yang, "A learning-based iterative method for solving vehicle routing problems," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[13] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 9, pp. 5057–5069, Sep. 2022.

[14] L. Accorsi, A. Lodi, and D. Vigo, "Guidelines for the computational testing of machine learning approaches to vehicle routing problems," *Oper. Res. Lett.*, vol. 50, no. 2, pp. 229–234, Mar. 2022.

[15] A. I. Garmendia, J. Ceberio, and A. Mendiburu, "Neural combinatorial optimization: A new player in the field," 2022, *arXiv:2205.01356*.

[16] R. Heckel, M. Simchowitz, K. Ramchandran, and M. Wainwright, "Approximate ranking from pairwise comparisons," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2018, pp. 1057–1066.

[17] K. Andreev and H. Räcke, "Balanced graph partitioning," in *Proc. 16th Annu. ACM Symp. Parallelism Algorithms Archit.*, Jun. 2004, pp. 120–124.

[18] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ, USA: Princeton Univ. Press, 2011.

[19] J. J. Hopfield and D. W. Tank, "'Neural' computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, no. 3, pp. 141–152, Jul. 1985.

[20] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 2692–2700.

[21] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković, "Combinatorial optimization and reasoning with graph neural networks," 2021, *arXiv:2102.09544*.

[22] C. K. Joshi, Q. Cappart, L.-M. Rousseau, and T. Laurent, "Learning the travelling salesperson problem requires rethinking generalization," 2020, *arXiv:2006.07054*.

[23] A. Hottung and K. Tierney, "Neural large neighborhood search for the capacitated vehicle routing problem," 2019, *arXiv:1911.09539*.

[24] P. da Costa, J. Rhuggenaath, Y. Zhang, A. Akcay, and U. Kaymak, "Learning 2-opt heuristics for routing problems via deep reinforcement learning," *Social Netw. Comput. Sci.*, vol. 2, no. 5, pp. 1–16, Sep. 2021.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

IRAZUSTA GARMENDIA et al.: NI HEURISTICS FOR GRAPH COPs
13

[25] J. K. Falkner, D. Thyssens, A. Bdeir, and L. Schmidt-Thieme, "Learning to control local search for combinatorial optimization," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2023, pp. 361–376.

[26] Y. Ma et al., "Learning to iteratively solve routing problems with dual-aspect collaborative transformer," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 11096–11107.

[27] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," 2019, *arXiv:1906.01227*.

[28] Z. H. Fu, K. B. Qiu, and H. Zha, "Generalize a small pre-trained model to arbitrarily large TSP instances," in *Proc. AAAI Conf. Artif. Intell.*, 2021, vol. 35, no. 8, pp. 7474–7482.

[29] R. Tromble and J. Eisner, "Learning linear ordering problems for better translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2009, pp. 1007–1016.

[30] W. Leontief, *Input-Output Economics*. London, U.K.: Oxford Univ. Press, 1986.

[31] H. Achatz, P. Kleinschmidt, and J. Lambsdorff, "Der corruption perceptions index und das linear ordering problem," in *Proc. ORNews*, vol. 26, 2006, pp. 10–12.

[32] P. Anderson, T. Chartier, and A. Langville, "The rankability of data," *SIAM J. Math. Data Sci.*, vol. 1, no. 1, pp. 121–143, Jan. 2019.

[33] T. R. Cameron, S. Charmot, and J. Pulaj, "On the linear ordering problem and the rankability of data," 2021, *arXiv:2104.05816*.

[34] N. B. Shah, J. K. Bradley, A. Parekh, N. Wainwright, and K. Ramchandran, "A case for ordinal peer-evaluation in MOOCs," in *Proc. NIPS Workshop Data Driven Educ.*, vol. 15, 2013, p. 67.

[35] Y. D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon, "Matrix encoding networks for neural combinatorial optimization," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 7, 2021, pp. 5138–5149.

[36] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3, pp. 229–256, 1992.

[37] Y. Bengio, J. Louradour, and R. Collobert, "Curriculum learning," in *Proc. Int. Conf. Mach. Learn.*, Aug. 2009, pp. 41–48.

[38] M. Laguna, R. Marti, and V. Campos, "Intensification and diversification with elite Tabu search solutions for the linear ordering problem," *Comput. Oper. Res.*, vol. 26, no. 12, pp. 1217–1230, Oct. 1999.

[39] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, Sep. 2003.

[40] F. Glover and E. Taillard, "A user's guide to Tabu search," *Ann. Oper. Res.*, vol. 41, no. 1, pp. 1–28, 1993.

[41] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search: Framework and applications," in *Handbook of Metaheuristics*. Berlin, Germany: Springer, 2019, pp. 129–168.

[42] O. Becker, "Das Helmstädtersche Reihenfolgeproblem—Die Effizienz verschiedener Näherungsverfahren," in *Proc. Comput. Uses Social Sci., Berichteiner Work. Conf.*, Vienna, Austria, 1967.

[43] L. Lugo, C. Segura, and G. Miranda, "A diversity-aware memetic algorithm for the linear ordering problem," *Memetic Comput.*, vol. 14, no. 4, pp. 395–409, 2022.

[44] V. Santucci and J. Ceberio, "Using pairwise precedences for solving the linear ordering problem," *Appl. Soft Comput.*, vol. 87, Feb. 2020, Art. no. 105998.

[45] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2778–2787.

[46] A. P. Badia et al., "Agent57: Outperforming the Atari human benchmark," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 507–517.

[47] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.

[48] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," 2020, *arXiv:2003.00982*.

[49] J. Chen and H. Chen, "Edge-featured graph attention network," 2021, *arXiv:2101.07671*.

[50] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 6000–6010.

[51] J. Ceberio, A. Mendiburu, and J. A. Lozano, "The linear ordering problem revisited," *Eur. J. Oper. Res.*, vol. 241, no. 3, pp. 686–696, Mar. 2015.

[52] G. Reinelt, "Linear ordering library (LOLIB)," Univ. Heidelberg, Heidelberg, Germany, Tech. Rep., 2002.
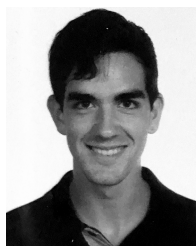
Andoni I. Garmendia received the B.Sc. and M.Sc. degrees in industrial engineering from the University of the Basque Country (UPV/EHU), Donostia-San Sebastian, Spain, in 2018 and 2020, respectively, and the M.Sc. degree in computational engineering from the Illinois Institute of Technology, Chicago, IL, USA, in 2020.

Since 2020, he has been the Ph.D. student with the Intelligent Systems Group, UPV/EHU. His current research is on the use of reinforcement learning for combinatorial optimization.

Josu Ceberio (Member, IEEE) received the Ph.D. degree from the University of the Basque Country (UPV/EHU), Donostia-San Sebastian, Spain, in 2014.

He is an Associate Professor with the Department of Computer Science and Artificial Intelligence, UPV/EHU. His main research areas are evolutionary computation, combinatorial optimization problems, and reinforcement learning.

Dr. Ceberio has been a member of the editorial board of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION since 2022.

Alexander Mendiburu (Member, IEEE) received the Ph.D. degree from the University of the Basque Country (UPV/EHU), Donostia-San Sebastian, Spain, in 2006.

He is a Full Professor with the Department of Computer Architecture and Technology, UPV/EHU. His main research areas are optimization, cybersecurity, and high-performance computing.