Applying neural combinatorial optimization for the generalized regenerator location problem

Lucas Martín-García[1 [0009−0009−7840−931X]], Isaac Lozano-Osorio[1⋆ [0000−0002−2608−8464]], Abraham Duarte[1 [0000−0002−4532−3124]], and J. Manuel Colmenar[1 [0000−0001−7490−9450]]

Universidad Rey Juan Carlos, Madrid, Spain
{lucas.martin,isaac.lozano,abraham.duarte,josemanuel.colmenar}@urjc.es

Abstract. Neural Combinatorial Optimization (NCO) has shown promising results in graph-based optimization problems, although its application to facility location problems remains limited. This work studies the use of NCO for the Generalized Regenerator Location Problem (GRLP), a network design problem where the objective is to minimize the number of active regenerators required to connect all demand nodes. A constructive algorithm guided by a Graph Neural Network (GNN) is proposed, where the model is trained through Reinforcement Learning (RL) on generated GRLP instances and later combined with a local search phase. The method is compared with exact, heuristic, and metaheuristic approaches from the literature. The results show that the NCO approach is competitive with the state of the art and, in the largest instances, obtains better constructive solutions than a Greedy Randomized Adaptive Search Procedure (GRASP) method using the same improvement phase. Despite the explainability and hardware limitations associated with these models, these results indicate that NCO algorithms can learn better construction strategies than problem-specific heuristics without the need for expert knowledge.

Keywords: Neural Combinatorial Optimization · Generalized Regenerator Location Problem · Reinforcement Learning · Metaheuristics.

1 Introduction

The continuous advancement of Deep Learning has enabled new ways to address optimization problems through data-driven methods. Within this context, Neural Combinatorial Optimization (NCO) offers a new paradigm where artificial neural networks are trained to autonomously discover and execute strategies to solve optimization problems [1]. The NCO process includes a learning phase, where the network adjusts its parameters via supervised or Reinforcement Learning (RL), and an inference phase, where the trained model solves new instances. This framework enables the model to autonomously extract the underlying problem structure and learn effective solving policies directly from data, eliminating the reliance on manually engineered heuristics.

⋆ Corresponding author.

Given that many combinatorial optimization problems can be naturally represented as graphs, Graph Neural Networks (GNN) [13] have become popular within the NCO framework. These architectures are well-suited for this domain because they can extract meaningful information from both the overall topology of the instance and the specific features embedded within its elements.

To date, most NCO research has focused on the Traveling Salesman Problem and similar routing problems [1]. While recent efforts have extended these methodologies to scheduling and ranking [8,9], the application of NCO to other combinatorial domains remains largely unexplored [4]. Motivated by this gap, this work aims to expand NCO research into Facility Location Problems, a family of problems that has received limited attention but presents a strong potential for NCO. These problems are particularly well-suited for GNN-based approaches, as their customer-facility network structure can be naturally represented as graphs, allowing the models to extract relevant information and learn solution policies.

A prominent and challenging example within this family is the Generalized Regenerator Location Problem (GRLP) [3], a relevant optimization task in the design of telecommunications and optical networks. The GRLP extends the classical Regenerator Location Problem (RLP) [2], which aims to minimize the number of located regenerators while ensuring that every pair of nodes in a network can communicate without exceeding the maximum allowed distance. Specifically, while the RLP assumes that regenerators can be installed at any node and that all nodes must communicate with each other, the GRLP restricts the regenerator allocations to a specific subset of candidate nodes, and communication requirements to a specific subset of terminal nodes. The remainder of this paper is organized as follows. Section 2 reviews the related literature on both NCO and the GRLP. Then, Section 3 details the proposed NCO methodology applied to solve the GRLP. Section 4 describes the experimental setup and analyzes the computational results. Finally, the conclusions are summarized in Section 5.

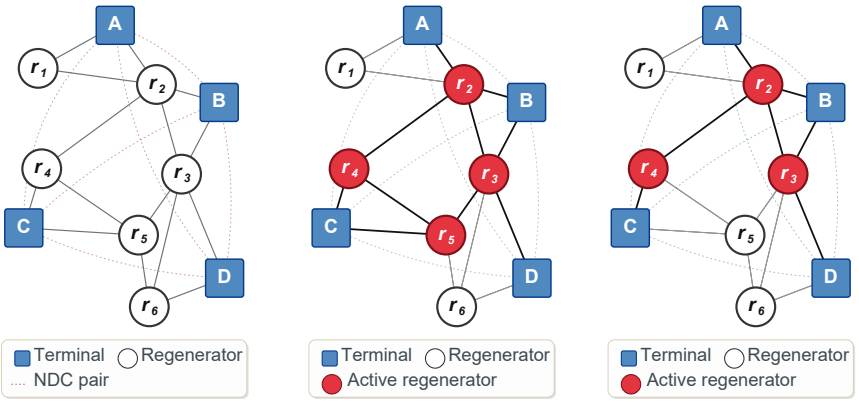
2 Related work

As a generalization of the Regenerator Location Problem (RLP) [2], the GRLP is also a $\mathcal{NP}$-hard problem [3]. Due to this computational complexity, exact mathematical models fail to scale on large instances, shifting the focus towards metaheuristics to find high-quality solutions within reasonable times. The problem was first introduced in [3] alongside a node-weighted directed Steiner forest formulation, a Branch-and-Cut algorithm, basic heuristics, and a generated public benchmark. To overcome the scalability limits of exact approaches, a Greedy Randomized Adaptive Search Procedure (GRASP) [12] was proposed, combining a randomized greedy constructive phase with a local search that removes redundant regenerators. After evaluating several greedy strategies, it was determined that the most effective constructive heuristic for this problem consists of iteratively selecting candidate locations that maximize the number of new connections between disconnected terminal nodes. More recently, an improved Basic Variable Neighborhood Search (iBVNS) [11] further advanced the state-of-

the-art using dynamic neighborhoods. While a weighted variant of the problem (WGRLP) exists, which minimizes the total installation cost rather than the number of regenerators [3,11], this work focuses only on the unweighted GRLP.

Following the literature, in this work, the GRLP is considered through its communication graph $G = (N, E)$, where N is the set of nodes and E contains the pairs of nodes that can communicate directly without requiring an intermediate regenerator. The node set is divided into candidate regenerator locations R and terminal nodes T, with $R \cap T = \emptyset$ and $R \cup T = N$. Terminal pairs that are not directly connected in G are denoted as Not Directly Connected (NDC) pairs and define the communications that must be supported by the selected regenerators. Therefore, a GRLP instance, denoted by I, contains the communication graph G, the candidate regenerator set R, the terminal set T, and the NDC set $\mathcal{D} \subseteq T \times T$. A solution is a subset $S \subseteq R$ of active regenerators, and it is feasible if every pair in $\mathcal{D}$ can be connected in G using active regenerators as intermediate support nodes. Finally, given a feasible solution, the objective function is the number of active regenerators, $\mathcal{F}(S) = |S|$, with the aim of obtaining the minimum $|S|$, making the GRLP a minimization problem.

Figure 1 illustrates a small example of the problem. The communication graph, depicted in Figure 1a, contains terminal nodes (depicted as squares), which define the required communications, and candidate regenerator nodes (depicted as circles), which can be activated to connect NDC pairs. Figure 1b shows one feasible solution obtained by activating a subset of candidate regenerators. Figure 1c shows an optimal solution with three active regenerators. Since the objective value in the GRLP is the number of selected regenerators, another optimal solution can be obtained by activating r_5 and deactivating r_4.



(a) Communication graph. (b) Feasible solution. (c) Optimal solution.

Fig. 1: Example instance and solutions for the GRLP.

In parallel to traditional metaheuristics, Neural Combinatorial Optimization (NCO) trains deep neural networks, typically via RL, to act as solving heuris-

tics [7]. Graph Neural Networks (GNNs) [13] are particularly suitable for this framework to process problem instances directly from their graph representations. Early NCO approaches focused on constructing solutions for classic graph and routing problems such as the Minimum Vertex Cover, Maximum Cut, and the Traveling Salesman Problem [5]. Recent research has evaluated the applicability of these models [7,8], expanding NCO to other domains like scheduling and ranking [8]. Furthermore, the field has evolved to include Neural Improvement models that guide local search operations [9], as well as strategies that take advantage of the fast inference of GNNs to optimize large-scale instances [10].

3 Methodology

The objective of this work is to evaluate the performance of the constructive NCO approach applied to a new problem family different from the routing and ordering problems already studied in the literature. In particular, the aim is to study whether a GNN-based constructive policy can learn useful decision rules for the GRLP, and whether these rules remain effective when the model is evaluated on larger instances not used in the training process. The proposed methodology uses the neural network as a constructive heuristic that sequentially selects candidate regenerator locations until a feasible solution is obtained. This constructive strategy is common in NCO [5].

3.1 Graph Neural Network

The neural model used in this work is based on the GNN paradigm, originally introduced to extend neural learning to data represented by graphs [13]. A GNN associates a state or embedding with each node and updates these embeddings by exchanging information through the edges of the graph. After several message-passing layers, each node representation contains information about its attributes, its neighborhood, and progressively larger parts of the graph. This mechanism is appropriate for the GRLP because the quality of selecting a candidate regenerator depends not only on the node itself, but also on its relation with terminal nodes, NDC pairs, and other candidate locations.

In our proposal, each node receives a 4-dimensional feature vector representing relevant information of the nodes for the neural model. These dimensions encode the following features: 1) whether the node belongs to R or T, 2) the normalized incidence of the node in NDC pairs, 3) the normalized communication degree, and 4) whether the node is currently active in the solution. Each edge is represented by one scalar feature indicating adjacency in the communication graph. The architecture used in the experiments is an edge-aware graph transformer with an attention decoder. The implementation is based on the public NCO Library (NCOLib)[1], which provides an NCO framework to solve optimization problems. The configuration considered in this work uses a node embedding

[1] `https://github.com/TheLeprechaun25/NCOLib`

dimension of 128 and includes encoder layers, attention heads, a feed-forward multiplier, a dropout regularization process, an auxiliary global node for the GNN, and logit clipping.

This representation has two relevant advantages for NCO. First, it is invariant to the input order. The same neural functions are applied to every node and edge, and the information received from neighbors is aggregated through permutation-invariant operations. Second, the GNN models are not restricted to a fixed number of nodes. Traditional neural network architectures require fixed-size inputs. In contrast, GNNs are invariant to input size thanks to the fact that the learnable parameters in this architecture are shared across nodes, edges, and message-passing layers, rather than being associated with a specific input position. This size invariance property allows a single model trained on instances of a specific size to process and solve instances of any size, effectively generalizing the learned policy to much larger networks. This capability is highly relevant in combinatorial optimization, where instances from the same problem vary in size while preserving the same relational structure.

3.2 Neural Combinatorial Optimization algorithm

The algorithm proposed in this work to solve the GRLP combines metaheuristic components with a GNN policy trained through RL. The GNN guides the constructive phase, while classical improvement mechanisms are used after construction to refine the solution. Let θ be the trained parameters of the GNN. The constructive algorithm, detailed in Algorithm 1, starts with an empty solution and iteratively adds one candidate regenerator from R until the solution becomes feasible (lines 2–6). At each iteration, the procedure $\texttt{NodeFeatures}(I, S)$ builds the input representation used by the GNN (line 3), described in Section 3.1, including which candidate regenerators are already active. The GNN, using the given parameters θ, returns a vector of positive values $p = (p_v)_{v \in R \setminus S}$, where each value p_v is the probability assigned by the GNN to the inactive node v as the next possible decision, based on the policy learned during training (line 4).

Algorithm 1 $\texttt{NCO}(I, \theta)$

1: $S \leftarrow \emptyset$
2: **while not** $\texttt{Feasible}(I, S)$ **do**
3: $\quad X \leftarrow \texttt{NodeFeatures}(I, S)$
4: $\quad p \leftarrow \texttt{GNN}_\theta(I, X, S)$
5: $\quad S \leftarrow S \cup \{\texttt{RandomWeighted}(p)\}$
6: **end while**
7: **return** S

Then, one regenerator $r \in R \setminus S$ is sampled according to these probabilities and added to the solution with the $\texttt{RandomWeighted}(p)$ method in line 5. This stochastic step allows the same trained model to generate different solutions for

the same instance. After each addition, the algorithm checks whether the current set S is feasible according to the definition of the GRLP instance (line 2), and the first feasible solution obtained is returned (line 7).

Once a feasible solution is constructed, an Improvement Phase (`IP`) is applied to refine it using two sequential methods from the literature [12]. First, a cleaning procedure identifies and removes redundant active regenerators. Subsequently, a first-improvement local search explores swap moves that replace two active regenerators with a single inactive candidate while maintaining feasibility.

In this work, multiple solutions are constructed for the same instance, applying the improvement phase to each one of them, and the best overall solution is returned, following a multi-start approach similar to the GRASP [6].

3.3 Reinforcement Learning

The GNN parameters are trained using RL, since the objective is not to predict a fixed label for each node, but to learn a sequence of constructive decisions that leads to good solutions. Each training episode consists of generating GRLP instances, constructing solutions with the current policy, computing the objective value of the obtained solutions, and updating the parameters of the model according to the quality of these solutions. In this setting, the learned policy is the probability distribution over the next regenerator to activate, defined by the logits returned by the GNN model during the constructive process.

The RL algorithm used to train the model is REINFORCE [14]. For each training instance I, the current policy constructs several stochastic solutions. Since the GRLP is a minimization problem, the reward is defined from the objective value with opposite sign, receiving higher rewards for solutions with fewer active regenerators. For each instance, the average reward of the generated solutions is used as a baseline $b(I)$. The effective learning signal for each sampled solution is the difference between its reward and this average baseline, $R(S) - b(I)$. To formalize this, the parameters θ of the neural model are optimized by minimizing the loss function defined in Equation 1,

$$\mathcal{L}(\theta|I) = \mathbb{E}_{S\sim p_\theta(\cdot|I)}\left[-(R(S) - b(I)) \ln p_\theta(S|I)\right] \tag{1}$$

where $p_\theta(S|I)$ represents the probability of generating the solution S under the current policy, and $\mathbb{E}$ denotes the expectation over the solution space [10]. Since evaluating all possible solutions is computationally infeasible, this expectation is approximated in practice by the empirical mean of the sampled constructions. If the reward is above the baseline, minimizing this loss increases the log-probabilities of the selected actions along that constructive trajectory. If the reward is below the baseline, the update decreases the probability of that sequence. This is done by applying backpropagation to update the GNN parameters using a gradient-based optimizer.

A generator of training instances is required due to the limited number of benchmark instances and the need of the neural model for a large number of different examples during training. The generator must also produce instances

with structural properties similar to the benchmark instances used in the final comparison. Otherwise, the model would learn a policy adapted to a different distribution of graphs and the results would not represent the target problem family. Although the GNN representation allows the model to generalize to any possible instance, the best performance is expected on instances with similar characteristics to those observed during training. For this reason, an analysis of the benchmark instances was carried out before defining the training generator with the objective of reproducing the main structural characteristics of the public GRLP instances. The resulting generator follows the same general principles described in the literature for creating GRLP benchmark instances [3].

4 Experimental results

The objective of our experimentation is to evaluate the performance of the proposed NCO approach. All the experiments were executed on servers equipped with NVIDIA A100-SXM4 GPUs with 80 GB of RAM and AMD EPYC 9655P limited to 32 CPU core processors and 32 GB of RAM. The training and inference phases of the proposal were implemented in `Python 3.10` and carried out on the GPU environment. Following the literature [12], the improvement phase was implemented in `Java 25` and executed on the CPU. The 300 public GRLP instances introduced in the literature [3] are used[2]. This dataset is divided into two groups based on their node sizes: Set 1 with sizes $|N| \in \{50, 75, 100, 125, 150\}$ and Set 3 with $|N| \in \{175, 200, 300, 400, 500\}$. Since Set 2 consists of WGRLP instances, it is not used in this work. Both groups use the same candidate regenerator percentages, $p\,(\%)$, with $p \in \{25, 50, 75\}$, providing 10 different instances for every combination of network size and regenerator percentage.

4.1 Training results

The training process is organized by the instance characteristics considered most relevant for the difficulty and average objective value: the number of nodes in the graph, $n = |N|$, and the percentage of candidate regenerators, $p\,(\%)$, with respect to the total number of nodes. Following previous NCO methodologies for constructive policies [7], the model is trained on instances with a fixed number of nodes, 50, and it is expected to generalize the learned strategies to larger instances. The first training stage uses instances with 25% candidate regenerators, the second stage uses 50%, and the final stage uses 75%. This order allows the model to learn from different ratios of candidate locations, ending with the setting that usually produces more difficult constructive decisions [3].

The configuration of the GNN model was conducted through a grid search focused on the most relevant hyperparameters: learning rate, dropout, batch size and number of stochastic constructions, which defines how many different solutions are sampled per training instance. The final hyperparameters were

[2] `https://drum.lib.umd.edu/items/aba073cf-991f-42f1-a009-d450d21fc289`

selected based on the best mean objective value achieved across a validation subset of 20 out of the 300 instances from the public benchmark. The best configuration explored consisted of a learning rate of 0.0005, a dropout rate of 0.0 (meaning that the model performs better without regularization), 10 stochastic constructions, and a batch size of 64. The final model was trained using this configuration for 100 epochs in each stage. Each epoch contains one training episode, where a batch of 64 generated instances is created and 10 stochastic constructive solutions are sampled for each instance. After every epoch, the model is evaluated in the validation subset using the deterministic policy, where the action with the highest probability is selected at each constructive step. The entire training process of 300 epochs was completed in under 5.5 hours.

Figure 2 reports the mean objective value obtained in the validation subset after each epoch. Since the training procedure is based on RL, policy updates can produce abrupt changes in the decisions made by the model. For this reason, the final model used is not the model obtained at the last epoch, but the model that achieved the best mean validation cost during the complete training process.

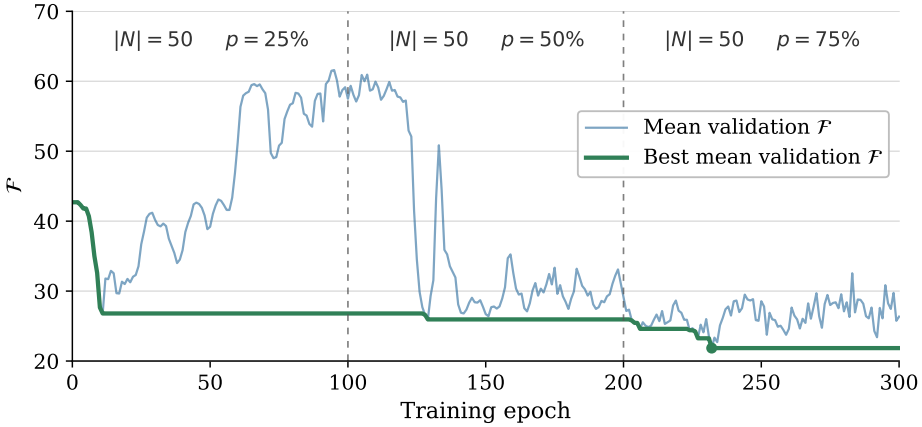


Fig. 2: Evolution of the mean validation objective value and best mean validation objective value during training.

As shown in Figure 2, the validation value improves during the initial epochs, indicating that the model quickly learns useful constructive decisions in the first stage. However, after this initial improvement, the validation cost increases. A possible explanation is that the policy learned on instances with a low percentage of candidate regenerators does not transfer equally well to validation instances with larger percentages. However, when the second stage starts, the validation cost decreases sharply as the model adapts its policy to instances with a larger percentage of candidate regenerators. After reaching the best validation value in the last stage, the model does not improve further and the training and validation cost oscillates around a similar range, showing that the model has converged.

4.2 Comparison with previous approaches

This section reports the performance of the constructive NCO algorithm implemented with the trained GNN model and the improvement phase, denoted as NCO+IP. This algorithm performs 10 iterations generating 10 different constructive solutions, applying the improvement phase to each of them, and reporting the best overall solution found. The comparison includes the mathematical model from the original GRLP study [3], denoted as MILP. In addition, the iBVNS [11], currently the most recent and competitive method proposed for the GRLP, is included. The comparison also considers a previous GRASP metaheuristic [12], since its constructive methodology is conceptually very similar to the proposed NCO approach. However, this last method only reports detailed results for the largest instances with 400 and 500 nodes, and therefore it is analyzed separately.

The algorithms from the literature could not be re-executed under the same hardware conditions, since their source codes are not publicly available. In addition, the corresponding publications do not provide the full disaggregated results for every individual instance. For this reason, the comparison uses the aggregated values reported in the original studies and compares them with the aggregated results obtained by the method proposed in this work. It is important to note that the computational times were obtained under different computational environments. Two metrics are used throughout the experiments. The first one is the objective function value, denoted by $\mathcal{F}$, which corresponds to the objective function of the best solution found by each algorithm for each instance. The second metric is the computational time, reported as T(s), which corresponds to the total execution time in seconds required by each algorithm for an instance.

Table 1 shows the results aggregated by instance size. The column NCO reports the NCO algorithm described in Algorithm 1 without the improvement phase (`IP`). Additionally, the best results are highlighted in bold. Note that the NCO method is excluded from this highlighting, as its execution times are lower due to the omission of the improvement phase.

The results show that the NCO+IP method is competitive with the best methods from the literature, although there is still room for improvement. The exact method obtains optimal or near-optimal solutions in the smaller instances. However, from $n = 300$ onward, the time limit is reached, and for $n = 400$ and $n = 500$ the reported objective values degrade as expected from an exact approach. The NCO+IP method, instead, keeps a stable behavior across the full benchmark and scales better in the larger instances. In addition, this NCO method obtains almost the same aggregated objective value as the exact method. The iBVNS obtains most of the best objective values, reaching the best aggregated quality with an average $\mathcal{F}$ value of 12.37. The proposed NCO+IP method remains close to this value, leaving a clear but limited margin for improvement with respect to the current state of the art.

Regarding computational time, the exact method shows the worst scalability, increasing from less than one second for the smallest instances to the 1-hour time limit in the largest ones. The iBVNS scales well, although its time increases noticeably for $n = 400$ and $n = 500$. The NCO approach shows a different

Table 1: Computational results aggregated by instance size.

Instance	MILP		IBVNS		NCO+IP		NCO	
n	$\mathcal{F}$	T(s)	$\mathcal{F}$	T(s)	$\mathcal{F}$	T(s)	$\mathcal{F}$	T(s)
50	**6.33**	0.17	**6.33**	**0.00**	**6.33**	0.11	8.80	0.11
75	**8.13**	1.40	**8.13**	**0.07**	8.17	0.24	11.83	0.22
100	**9.83**	10.47	**9.83**	**0.06**	**9.83**	0.62	13.93	0.36
125	**11.37**	94.33	**11.37**	**0.71**	11.63	0.77	16.47	0.50
150	**11.93**	500.60	**11.93**	**0.77**	12.17	1.16	17.77	0.69
175	**12.83**	1512.60	**12.83**	**1.49**	13.00	1.52	20.40	0.91
200	**13.40**	2375.67	**13.40**	**2.17**	13.67	2.22	22.00	1.20
300	**14.83**	3600.00	15.77	**6.49**	16.43	7.56	27.47	2.79
400	18.63	3600.00	**16.67**	16.33	17.47	**11.22**	30.57	4.94
500	19.63	3600.00	**17.40**	40.06	18.30	**21.97**	32.43	6.93
Aggregated	12.69	1529.52	**12.37**	6.81	12.70	**4.36**	20.17	1.86

behavior: once the model has been trained, inference is fast, and its time grows more moderately with the instance size. This is especially visible in the largest instances, where NCO+IP remains fast while still producing high-quality feasible solutions. However, it must be noted that this fast inference relies on GPU hardware to achieve maximum efficiency, unlike traditional algorithms that are typically executed on standard CPUs. The results show that the average time of the constructive NCO alone is 1.86 seconds, while the complete NCO+IP method requires 4.36 seconds. This indicates that most of the computational time of the algorithm comes from the improvement phase.

Table 2 focuses on the comparison with the GRASP method for the largest instances. This comparison is useful because both methods follow a similar resolution scheme: they generate constructive solutions following a random and greedy strategy, and then apply the exact same improvement procedure. The difference is that the GRASP constructive phase follows an expert-designed heuristic specific for this problem, whereas the proposed method uses a policy autonomously learned by the GNN from generated training instances.

The results show that the GNN has learned a constructive policy that is more effective than the constructive heuristic used in the GRASP method for this problem. For $n = 400$, NCO+IP obtains better average objective values than GRASP in the three regenerator percentage groups. For $n = 500$, NCO+IP is also better in two of the three groups. In the aggregated comparison, NCO+IP obtains $\mathcal{F} = 17.88$, compared to $\mathcal{F} = 18.08$ for GRASP, with a very similar reported time, 16.60 seconds and 16.38 seconds, respectively.

Since both methods use the exact same improvement phase after construction, the main difference is the strategy for constructing initial solutions. The GRASP construction is a problem-specific randomized heuristic designed using expert knowledge of the GRLP, while the GNN learns its constructive policy autonomously through RL. Without requiring expert information about the prob-

Table 2: Computational results for instance sizes 400 and 500.

Instance		GRASP		NCO+IP		NCO	
n	p (%)	$\mathcal{F}$	T(s)	$\mathcal{F}$	T(s)	$\mathcal{F}$	T(s)
400	25	20.90	**6.85**	**20.70**	11.88	31.50	5.15
	50	18.30	19.53	**18.00**	**14.25**	32.30	5.57
	75	13.90	**4.62**	**13.70**	7.54	27.90	4.08
500	25	23.40	**18.25**	**23.00**	25.89	37.20	8.13
	50	**19.60**	**19.19**	19.70	29.37	38.20	7.85
	75	12.40	29.82	**12.20**	**10.64**	21.90	4.82
Aggregated		18.08	**16.38**	**17.88**	16.60	31.50	5.93

lem, the model optimizes its decisions using self-generated rewards. This result shows a relevant advance of the NCO approach because the neural model is able to extract useful information and learn a better constructive strategy that outperforms a hand-designed metaheuristic from the literature. However, a notable disadvantage of NCO models is their lower explainability. Given that metaheuristics use problem-specific knowledge, it is straightforward to identify how they construct and improve solutions. In contrast, while there exist techniques to study the interpretability of neural networks, the decision-making process in deep learning models remains significantly more difficult to analyze.

Overall, these results are promising for an initial NCO approach to the GRLP. The proposed NCO algorithm does not yet outperform the strongest iBVNS method, and there is still room for improvement in both training and inference. However, the GNN is able to learn a constructive policy that obtains better constructed solutions than the problem-specific constructive heuristic of the GRASP from the literature.

5 Conclusions

This work explores the application of Neural Combinatorial Optimization (NCO) to the Generalized Regenerator Location Problem (GRLP). The proposed approach introduces a constructive algorithm guided by a Graph Neural Network (GNN) trained via Reinforcement Learning (RL). By evaluating stochastic constructions on generated instances, the model successfully learns an effective constructive policy for solving the problem.

Computational results demonstrate that the proposed constructive NCO method is competitive with state-of-the-art algorithms. Although it does not surpass the strongest iBVNS method, it obtains high-quality solutions and outperforms the analogous GRASP metaheuristic from the literature. This highlights the main advantage of the proposed approach: while traditional algorithms rely heavily on manual design and expert knowledge, the neural model learns a resolution policy autonomously without requiring problem-specific information. Outperforming the constructive scheme of a classical metaheuristic confirms that

GNN-based models can autonomously extract structural patterns to build better solutions, proving their value in the field of combinatorial optimization.

Despite these promising results, some limitations remain. The performance of the model relies on the quality of the training distribution and requires GPU hardware for efficient inference. Furthermore, the learned neural policy lacks the straightforward explainability of traditional heuristics, making it difficult to analyze the reasons behind its decisions. Future work will focus on exploring more diverse training strategies, improving inference efficiency, and incorporating interpretability tools to better understand the neural decision-making process.

Acknowledgments

This work was supported by the Ministerio de Economía y Competitividad (grant ref. PID2024-156045NB-I00), Ministerio para la Transformación Digital y de la Función Pública (Cátedra ENIA AI4DDS, grant ref. TSI-100930-2023-3), Ministerio de Ciencia, Innovación y Universidades (grant ref. FPU24/01134), and Comunidad Autónoma de Madrid (grant ref. TEC-2024/COM-404).

References

1. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural Combinatorial Optimization with Reinforcement Learning (Jan 2017).
2. Chen, S., Ljubić, I., Raghavan, S.: The regenerator location problem. Networks **55**(3), 205–220 (2010).
3. Chen, S., Ljubić, I., Raghavan, S.: The Generalized Regenerator Location Problem. INFORMS Journal on Computing **27**(2), 204–220 (Apr 2015).
4. Chung, K.T.: Neural combinatorial optimization with reinforcement learning in industrial engineering: A survey. Artificial Intelligence Review (2025).
5. Dai, H., Khalil, E.B., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. pp. 6351–6361. NIPS'17 (2017)
6. Feo, T.A., Resende, M.G.C.: A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters **8**(2), 67–71 (Apr 1989).
7. Garmendia, A.I., Ceberio, J., Mendiburu, A.: Neural Combinatorial Optimization: A New Player in the Field. http://arxiv.org/abs/2205.01356 (May 2022).
8. Garmendia, A.I., Ceberio, J., Mendiburu, A.: Applicability of Neural Combinatorial Optimization: A Critical View. ACM Trans. Evo. Learn. Optim. (2024).
9. Garmendia, A.I., Ceberio, J., Mendiburu, A.: Neural Improvement Heuristics for Graph Combinatorial Optimization Problems. IEEE TNNLS (2024).
10. Garmendia, A.I., Ceberio, J., Mendiburu, A.: Neural Combinatorial Optimization by Means of Partial Solution Strategies. In: (CEC) (2024).
11. Mrkela, L.: An Efficient Variable Neighborhood Search for Generalized Regenerator Location Problems. IJAIT **31**(05) (2022).
12. Quintana, J., Sánchez-Oro, J., Duarte, A.: Efficient Greedy Randomized Adaptive Search Procedure for the Generalized Regenerator Location Problem. International Journal of Computational Intelligence Systems **9**(6), 1016–1027 (Nov 2016).
13. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The Graph Neural Network Model. IEEE Transactions on Neural Networks **20**(1) (2009).
14. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8**(3-4), 229–256 (May 1992).