Supplementary Material for
-
MARCO: A Memory-Augmented Reinforcement framework for Combinatorial Optimization

Andoni I. Garmendia[1] , **Quentin Cappart**[2] , **Josu Ceberio**[1] and **Alexander Mendiburu**[1]
[1]University of the Basque Country (UPV/EHU), Donostia-San Sebastian, Spain
[2]Polytechnique Montréal, Montreal, Canada
{andoni.irazusta, josu.ceberio, alexander.mendiburu}@ehu.eus, quentin.cappart@polymtl.ca

A Further implementation details

In order to solve the Maximum Cut (MC), Maximum Independent Set (MIS) and Travelling Salesman Problem (TSP), MARCO receives information from three sources: the static instance information, the dynamic (partial) solution and the historical memory data.

A.1 MC and MIS

In MC and MIS, the instance information is given by binary edge features $\mathbf{y} \in \mathbb{R}^{|E|}$, which denote the existence of an edge in the set of edges E of the graph G. The solution is also encoded in binary node features $\mathbf{x} \in \mathbb{R}^{|V|}$, where V is the set of nodes. The memory data is a weighted average of the actions (node flips) performed in similar states and it is concatenated as an additional node feature.

A.2 TSP

In TSP, we follow prior works[Kool *et al.*, 2018; Kwon *et al.*, 2020] with some improvements:

- While prior works only use node features (city coordinates), we also consider the use of distances between cities as edge features.
- We implemented a Graph Transformer (GT) [Dwivedi and Bresson, 2020] encoder layer that also considers the edge features.
- We increased the model size (embedding dimension). We obtained better results scaling it. See the selected model hyperparameters in Table 1 (lines 1-5).
- ReLU activation function is replaced by SwiGLU [Shazeer, 2020].
- We implemented a function to compute an arbitrary number of data augmentations (coordinate rotations).

The memory data is aggregated as an edge feature, and gives information about the number of times each edge have been considered in previous solutions or routes. These edge features are used in the MHA mechanism of the decoder[Kool *et al.*, 2018; Kwon *et al.*, 2020]. Where a linear projection of the memory features is added to the attentions weights.

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \mathbf{E}\right)V \qquad (1)$$

A.3 Training in MARCO

The training hyperparameters can be found in Table 1 (lines 6-15). We trained a unique model for each problem. Each epoch of training comprised 1000 episodes. Within each episode, 128 new instances of an arbitrary size (50-200 for MC and MIS and 20-100 for TSP) were randomly generated. The optimization of model parameters was conducted using the AdamW optimizer [Loshchilov and Hutter, 2017], and we clipped the gradient norm at a value of 1.0.

The training of *improvement models* (for MC and MIS) requires additional hyperparameters, such as the discount factor and episode length. This necessity comes from the reward mechanism employed in these models, which is based on the concept of discounted future rewards, i.e., focusing on maximizing future gains. A discount factor of 0.95 and an episode length of 20 steps were selected.

We adopted the following two-phase training pipeline for *constructive models*. The initial phase comprises 200 epochs of training without the incorporation of memory, following the training of POMO [Kwon *et al.*, 2020].

Subsequently, we fix all model weights and introduce a trainable Feed-Forward network into the decoder, which processes the data retrieved from the memory. For the TSP, the processed memory content is incorporated into the edges, and it is used within the Multi-Head Attention mechanism of the decoder in the same fashion as in the GT encoder. This second training stage extends for an additional 50 epochs. Within each training episode during this stage, the process begins with a deterministic rollout performed with an empty memory. This is followed by executing 5 construction iterations (line 15) using identical problem instances. In these iterations, the model is penalized based on the similarity of its proposed solutions to those generated in previous iterations.

A.4 Inference in MARCO

To obtain the performance tables in the main paper, we have used the hyperparameters in Table 1 (16-21).

Lines 16-18 have common parameters for both improvement and constructive methods: the number (K) of neighbors to search in memory, the number of parallel execution threads, and the maximum number of solutions stored in memory (once this number is reached the oldest solutions are replaced).

Table 1: Hyperparameters used in MARCO.

	Hyperparameter	MC	MIS	TSP
	MODEL HP			
1	Embedding dim	64	64	512
2	Encoding layers	3	3	6
3	Attention heads	8	8	16
4	FF hidden dim	512	512	2048
5	Tanh clipping	10	10	10
	TRAINING HP			
6	Instance sizes	50-200	50-200	20-100
7	Learning rate	1e-4	1e-4	1e-4
8	Batch size	128	128	128
9	Number of epochs	100	100	250
10	Number of episodes	1000	1000	1000
11	Repetition penalty	1.0	0.01	0.1
12	Gradient clipping	1	1	1
13	Discount factor	0.95	0.95	-
14	Episode length	20	20	-
15	Constructions	-	-	5
	INFERENCE HP			
16	k in retrieval	20	20	3
17	Num of threads	50	50	100
18	Max memory size	10^5	10^5	10^5
19	Max steps	$2 \times N$	$2 \times N$	-
20	Constructions	-	-	10
21	Retrieval frequency	-	-	10

The *Max steps* parameter denotes the number of improvement steps, while *Constructions* denote the number of solutions built in constructive methods.

Lastly, for the TSP, we modulate the frequency of data retrieval from the memory. Instead of conducting this search at every construction step, we establish a predetermined retrieval frequency (*retrieval frequency*) for performing it, significantly accelerating the process, while having minimal impact on performance due to the marginal changes to the partial solution in consequent steps.

B Impact of k on MARCO's Performance

The number of neighbors, denoted as k, considered during the k nearest neighbor search, plays a crucial role in MARCO's performance. As depicted in Figure 1, small values of k result in poor performance due to the restricted information that can be retrieved at each step. On the other hand, when k is large, the algorithm takes into account numerous neighbors, including those with lower similarity. However, the influence of each retrieved neighbor is mitigated by its similarity, effectively addressing this issue.

C Impact of Repetition Penalty Coefficient

The penalty coefficient, as indicated in line 11 of Table 1, specifies the magnitude of the penalty applied to the reward mechanism when the algorithm suggests a repetitive action.

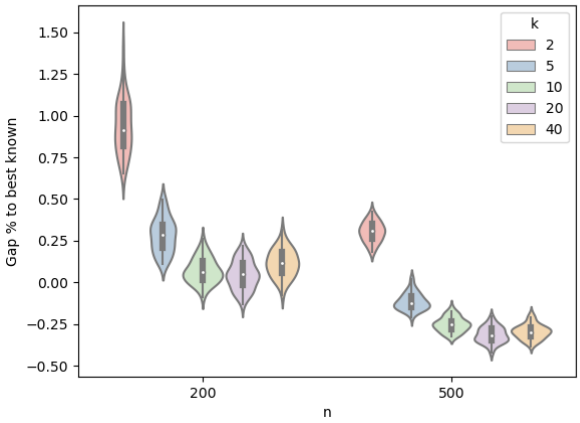

Figure 1: Performance of MARCO in MC with different k nearest neighbors retrieved from memory for graphs with 200 and 500 nodes.

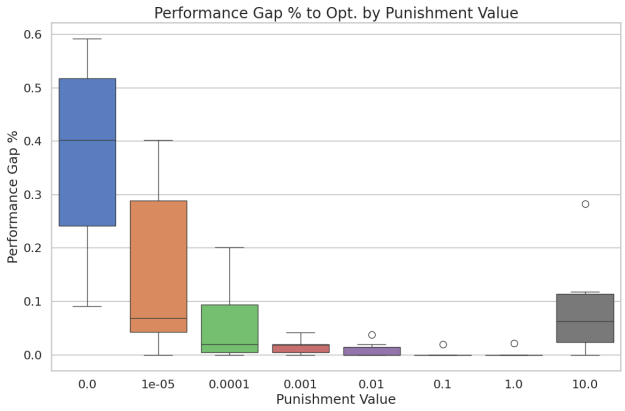

Figure 2: Performance of MARCO in MC based on the punishment weight in the reward used during training.

This parameter requires careful adjustment to ensure it harmonizes with the problem's objective value in the reward function. A penalty that is too small may lead the algorithm to frequently repeat actions, whereas an excessively high penalty can generate very large gradient values. Figure 2 illustrates the results of experiments conducted with various penalty coefficients for the MC problem. We trained five distinct models for each penalty coefficient, following the methodology of the main experiments, and conducting evaluations across 100 graph instances with 200 nodes. The findings reveal that penalty coefficients of 0.1 and 1.0 yield the most favorable performance outcomes.

D Analysis of the Memory Module

D.1 Theoretical Memory Complexity Analysis

The core elements of MARCO that contribute to memory consumption include the *Graph Transformer model* (GT) and the *memory module*.

Let us denote $N = |V|$ as the number of nodes in the instance graph. The memory complexity of *transformers* [Vaswani *et al.*, 2017], and by extension the GT, is known to scale quadratically with N ($\mathcal{O}(N^2)$).

The complexity associated with the *memory module* varies based on the method employed for storing solutions encountered during the optimization process:

- *Node-wise storage*. In MC and MIS, the solutions are encoded in nodes, represented by a vector of length N.
- *Edge-wise storage*. In TSP, each solution is represented by a vector of size proportional to the number of edges, which in a dense graph is of length N^2.

Additionally, with each optimization step t, a new solution is added to the memory. Consequently, if the optimization process runs for T steps, the memory requirement for storing these solutions scales linearly with T.

Therefore, the overall memory complexity of MARCO, depending on the storage method, can be summarized as:

- *Node-wise storage*: $\mathcal{O}(N^2 + T \cdot N)$
- *Edge-wise storage*: $\mathcal{O}(N^2 + T \cdot N^2)$

In scenarios where T is considerably large, the memory requirement associated with solution storage may become the predominant factor in the system's memory complexity.

D.2 Empirical Analysis of Memory Usage

In this section, we provide an empirical analysis of MARCO's memory usage to complement the theoretical analysis.

Figure 3 shows the progression of both the objective function and MARCO's memory usage across different sizes of MC instances throughout the optimization process. Consistent with our theoretical findings, the memory usage scales linearly with the number of optimization steps T, given a fixed instance size. Notably, the observed memory requirements for achieving (near) optimal solutions are relatively modest, especially when compared to the memory capacities of modern GPUs.

It is important to note that in our implementation for the TSP, where we store the solutions using graph edges (vectors of length N^2 rather than N), memory consumption is higher. To evaluate the feasibility of running MARCO with this edge-based representation in TSP, we analyzed its memory usage. Our findings reveal that for TSP instances with 100, 200, and 500 cities, our implementation of MARCO required an average memory of 0.04 MB, 0.15 MB, and 0.9 MB, respectively, for storing each construction.

Given that a modern GPU is typically equipped with more than 8GB of memory, the number of possible constructions for the largest TSP size evaluated (500 cities) stands at approximately $\frac{8 \times 1024}{0.9} \approx 9000$. However, in scenarios where MARCO might be required to run for extended periods or handle even larger problem instances, there are strategies, as discussed in the future work section of our main paper, for optimizing memory usage.

References

[Dwivedi and Bresson, 2020] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[Kool *et al.*, 2018] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[Kwon *et al.*, 2020] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

[Loshchilov and Hutter, 2017] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[Shazeer, 2020] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

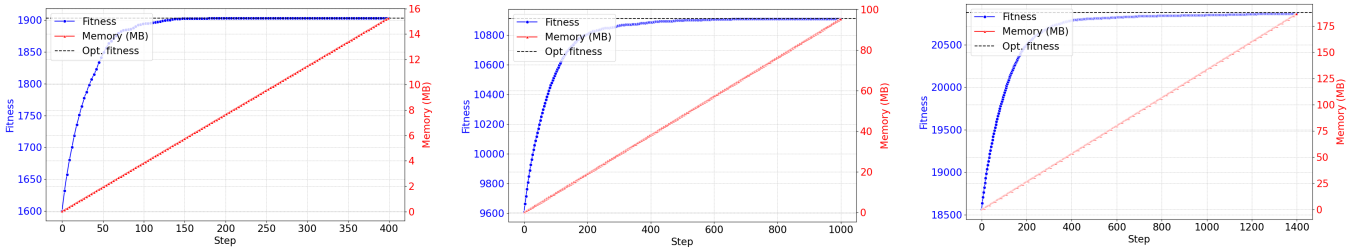

Figure 3: Performance and Memory Consumption Analysis of MARCO in MC. This graph illustrates the objective value (performance) and its GPU memory usage across successive improving iterations for instances of size (a) 200, (b) 500 and (c) 700.