Memory-Enhanced Neural Solvers for Routing Problems

Felix Chalumeau[*1] **Refiloe Shabe**[1] **Noah De Nicola**[**2]
Arnu Pretorius[1] **Thomas D. Barrett**[†1] **Nathan Grinsztajn**[†1]

[1]InstaDeep
[2]University of Cape Town

Abstract

Routing Problems are central to many real-world applications, yet remain challenging due to their (NP-)hard nature. Amongst existing approaches, heuristics often offer the best trade-off between quality and scalability, making them suitable for industrial use. While Reinforcement Learning (RL) offers a flexible framework for designing heuristics, its adoption over handcrafted heuristics remains incomplete. Existing learned methods still lack the ability to adapt to specific instances and fully leverage the available computational budget. Current best methods either rely on a collection of pre-trained policies, or on RL fine-tuning; hence failing to fully utilize newly available information within the constraints of the budget. In response, we present MEMENTO, an approach that leverages memory to improve the search of neural solvers at inference. MEMENTO leverages online data collected across repeated attempts to dynamically adjust the action distribution based on the outcome of previous decisions. We validate its effectiveness on the Traveling Salesman and Capacitated Vehicle Routing problems, demonstrating its superiority over tree-search and policy-gradient fine-tuning; and showing that it can be zero-shot combined with diversity-based solvers. We successfully train all RL auto-regressive solvers on large instances, and verify MEMENTO's scalability and data-efficiency: pushing the state-of-the-art on 11 out of 12 evaluated tasks.

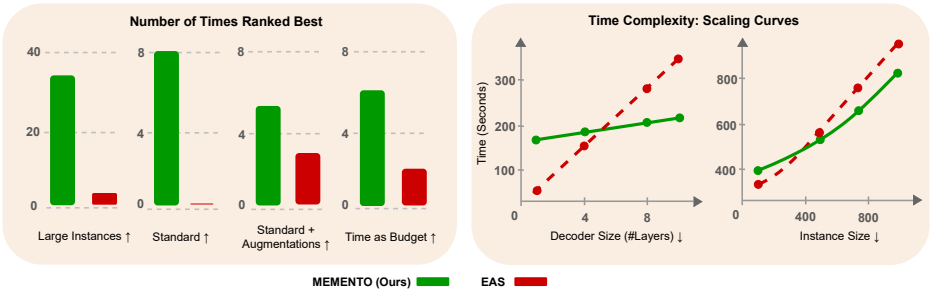

Figure 1: **MEMENTO outperforms Efficient Active Search** on a wide range of instance sizes and different evaluation budgets. Its time complexity makes it scalable to large instances and policies.

[*]Corresponding author: f.chalumeau@instadeep.com
[**]Work completed during an internship at InstaDeep
[†]Equal supervision

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

https://doi.org/10.52202/085713-1108

1 Introduction

Combinatorial Optimization (CO) encompasses a vast range of applications, ranging from transportation (J-F Audy and Rousseau, 2011) and logistics (Dincbas et al., 1992) to energy management (Froger et al., 2016). These problems involve finding optimal orderings or labels to optimize given objective functions. Real-world CO problems are typically NP-hard with a solution space growing exponentially with the problem size, making it intractable to find the optimal solution. Hence, industrial solvers rely on sophisticated heuristic approaches to solve them in practice. Reinforcement Learning (RL) provides a versatile framework for learning such heuristics and has demonstrated remarkable success in tackling CO tasks (Mazyavkina et al., 2021).

Traditionally, RL methods train policies to incrementally construct solutions. However, achieving optimality in a single attempt for NP-hard problems is impractical. Therefore, pre-trained policies are often combined with search procedures. The literature introduces a range of these procedures, from stochastic sampling (Kool et al., 2019; Kwon et al., 2020; Grinsztajn et al., 2023), beam search (Steinbiss et al., 1994; Choo et al., 2022), Monte Carlo Tree Search (MCTS) (Browne et al., 2012) to online fine-tuning (Bello et al., 2016; Hottung et al., 2022) and searching a space of diverse pre-trained policies (Chalumeau et al., 2023b). One popular online fine-tuning strategy, Efficient Active Search (EAS) (Hottung et al., 2022), uses transitions from generated solutions to derive policy gradient updates. However, it suffers from the inherent drawbacks of back-propagation, in particular having each data point only impacting the update as much as what is enabled by the learning rate. The leading strategy, COMPASS (Chalumeau et al., 2023b), relies on a space of diverse pre-trained policies for fast adaptation, but is limited to selecting the most appropriate one, and lacks an update mechanism using the collected data. Meanwhile, the use of memory has grown in modern deep learning, demonstrated by the success of retrieval augmented approaches in Natural Language Processing (Lewis et al., 2020), and to a smaller extent, its use in RL (Humphreys et al., 2022). These mechanisms create a closer link between collected experience and policy update, making them a promising candidate to improve adaptation.

In this vein, we introduce MEMENTO, a method to update the action distribution of neural solvers online, leveraging a memory of recently collected data. The approach is model agnostic and can be applied to existing neural solvers. MEMENTO learns expressive update rules, which experimentally prove to outperform standard policy-gradient updates. We evaluate our method on two popular routing tasks, the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP). We evaluate all methods both in and out-of-distribution and tackle instances up to size 500 to better understand their scaling properties. We provide an analysis of the update rules learned by MEMENTO, examining how it enables to adapt faster than policy gradient methods like EAS.

Our contributions come as follows: **(i)** We introduce MEMENTO, a memory and a processing module, enabling efficient adaptation of policies at inference time. **(ii)** We provide experimental evidence that MEMENTO can be combined with existing approaches to boost performance in and out-of-distribution, even for large instances and unseen solvers, reaching state-of-the-art (SOTA) on 11 out of 12 tasks. **(iii)** Whilst doing so, we train and evaluate leading construction methods on TSP and CVRP instances of size 500 solely with RL, outperforming all existing RL methods. **(iv)** We open-source our implementations in JAX (Bradbury et al., 2018), along with test sets and checkpoints.

2 Related work

Construction methods for CO These refer to methods that incrementally build a solution one action after the other. Hopfield and Tank (1985) was the first to use a neural network to solve TSP, followed by Bello et al. (2016) and Deudon et al. (2018) who respectively added an RL loss and an attention-based encoder. These works were further extended by Kool et al. (2019) and Kwon et al. (2020) to use a general transformer architecture (Vaswani et al., 2017), which has become the standard model choice that we also leverage in this paper. These works have given rise to several variants, improving the architecture (Xin et al., 2021; Luo et al., 2023) or the loss (Kim et al., 2021, 2022; Drakulic et al., 2023; Sun et al., 2024). These improvements are orthogonal to our work and could a priori be combined with our method. Construction approaches are not restricted to routing problems: numerous works have tackled various CO problems, especially on graphs, like Maximum Cut (Dai et al., 2017; Barrett et al., 2020), or Job Shop Scheduling Problem (JSSP) (Zhang et al., 2020; Park et al., 2021).

https://doi.org/10.52202/085713-1108

Construction methods make use of a predetermined compute budget to generate one valid solution for a problem instance, depending on the number of necessary action steps. In practice, it is common to have a greater, fixed compute budget to solve the problem at hand such that several trials can be attempted on the same instance. However, continuously rolling out the same learned policy is inefficient as (i) the generated solutions lack diversity (Grinsztajn et al., 2023) and (ii) the information gathered from previous rollouts is not utilized. How to efficiently leverage this extra budget has drawn some attention recently. We present the two main types of approaches that augment RL construction methods with no problem-specific knowledge (hence excluding *solution improvement methods*).

Diversity-based methods The first type of approach focuses on improving the diversity of the generated solutions (Kwon et al., 2020; Grinsztajn et al., 2023; Chalumeau et al., 2023b; Hottung et al., 2024). POMO (Kwon et al., 2020) uses different starting points to enable the same policy to generate diverse candidates. POPPY (Grinsztajn et al., 2023) leverages a population of agents with a loss targeted at specialization on sub-distribution of instances. It was extended in Chalumeau et al. (2023b) and Hottung et al. (2024), respectively replacing the population with a continuous latent space (COMPASS) or a discrete context vector (PolyNet).

Policy improvement at inference time The second category of methods, which includes MEMENTO, addresses the improvement aspect. These methods are theoretically orthogonal and can be combined with those mentioned earlier. EAS (Hottung et al., 2022) and Meta-SAGE (Son et al., 2023) employ a parameter-efficient fine-tuning approach on the test instances, whereas SGBS (Choo et al., 2022) enhances this strategy by incorporating tree search. While demonstrating good performance, they rely on rigid, handcrafted improvement and search procedures, which may not be optimal for diverse problems and computational budgets.

In contrast, several methods aim to learn these improvement mechanisms. Macfarlane et al. (2022) train a Graph Neural Network to perform a tree search akin to MCTS. MOCO, FER, and MARCO (Dernedde et al., 2024; Jingwen et al., 2023; I. Garmendia et al., 2024) learn the policy improvement update. MOCO (Dernedde et al., 2024) introduces a meta-optimizer that learns to calculate flexible parameter updates based on the reinforce gradient, remaining budget, and best solutions discovered so far. MARCO (I. Garmendia et al., 2024) aggregates graph edges' features and introduce problem-dependent similarity metrics to improve exploration of the solution space. Both approaches enable to learn adaptation strategies but are tied to heatmap-based policies, preventing their applicability to certain CO problems like CVRP. FER (Jingwen et al., 2023) learns a rule to update the instance nodes' embedding, whereas MEMENTO directly updates the action logits, hence being architecture-agnostic.

3 Methods

3.1 Preliminaries

Formulation A CO problem can be represented as a Markov Decision Process (MDP) denoted by $M = (S, A, R, T, H)$. This formulation encompasses the state space S with states $s_i \in S$, the action space A with actions $a_i \in A$, the reward function $R(r|s, a)$, the transition function $T(s_{i+1}|s_i, a_i)$, and the horizon H indicating the episode duration. Here, the state of a problem instance is portrayed as the sequence of actions taken within that instance, where the subsequent state s_{t+1} is determined by applying the chosen action a_t to the current state s_t. An agent is introduced in the MDP to engage with the problem and seek solutions by learning a policy $\pi(a|s)$. The standard objective considered in RL works is the single shot optimisation, i.e. finding a policy that generates a trajectory τ which maximises the collected reward: $\pi^* = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}[R(\tau)]$, where $R(\tau) = \sum_{t=0}^{H} R(s_t, a_t)$.

The specificity of most practical cases in RL for CO is that the policy is given a number of attempts allowed per instance (budget B), to find the best possible solution, rather than a single attempt. Consequently, the learning objective should rather be : $\pi^* = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}[\max_{i=1,\dots,B} R(\tau_i)]$.

3.2 MEMENTO

Adaptation to unseen instances is crucial for neural solvers. Even when evaluated in the distribution they were trained on, neural solvers are not expected to provide the optimal solution on the first shot,

https://doi.org/10.52202/085713-1108

due to the NP-hardness of the problems tackled. Making clever use of the available compute budget for efficient online adaptation is key to performance.

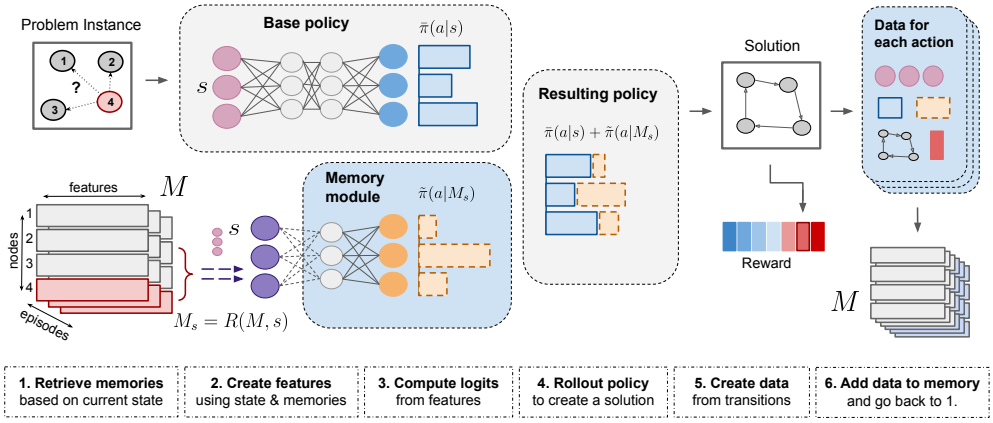

Figure 2: **MEMENTO uses a memory to adapt neural solvers at inference time.** When taking a decision, data from similar states is retrieved and prepared (1,2), then processed by a MLP to derive correction logits for each action (3). Summing the original and new logits enables to update the action distribution. The resulting policy is then rolled out (4), and transitions' data is stored in a memory (5,6), including node visited, action taken, log probability, and return obtained.

A compelling approach is to store all past attempts in a memory that can be leveraged for subsequent trajectories. This ensures that no information is lost and that promising trajectories can be used more than once. There are many ways of implementing such framework depending on how the information is stored, retrieved, and used in the policy. We would like the memory-based update mechanism to be (i) *learnable* (how to use past trajectories to craft better trajectories should be learnt instead of harcoded) (ii) *light-weight* to not compromise inference time unduly (iii) *agnostic* to the underlying model architecture (iv) able to leverage *pre-trained memory-less* policies.

Overview To this end, we introduce MEMENTO, a method that dynamically adapts the action distribution of the neural solver using online collected data. This approach, illustrated on Fig. 2, achieves the four desiderata: it stores light-weight data about past attempts, and uses an auxiliary model to update the action logits based on the previous outcomes. It can be used on top of any pre-trained policy (architecture-agnostic). In principle, MEMENTO can be combined with existing construction RL algorithms: Attention-Model (Kool et al., 2019), POMO (Kwon et al., 2020), POPPY (Grinsztajn et al., 2023), COMPASS (Chalumeau et al., 2023b), which we confirm experimentally. It learns an update rule during training, enabling data-efficient adaptation compared to policy gradient methods. The data retrieved from the memory and used by the auxiliary model to compute the new action logits contains more than the typical information used to derive a policy gradient update. For instance, the *budget remaining*, enabling to calibrate the exploration/exploitation trade-off and discover superior update rules. We show in Appendix F that MEMENTO has capacity to at least rediscover REINFORCE, and we show empirically in Section 4.1 that we outperform the leading policy-gradient updates adaptation method.

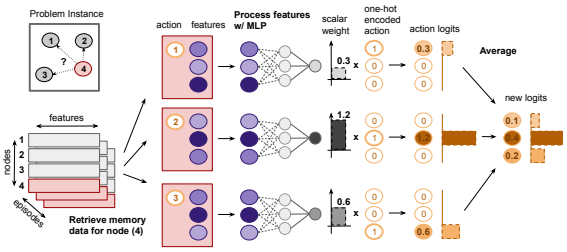

Figure 3: **Building a new action distribution using the memory.** Relevant data is retrieved and processed by a MLP to derive logits for each possible action.

Storing data in memory In the memory, we store data about past attempts (Fig. 2, step 6). Akin to a replay buffer, this data needs to reflect past decisions, their outcome, and must enable to take better future decisions. For each transition experienced while constructing a solution, the memory stores the following: (i) node visited (i.e., in the context of TSP or CVRP, this corresponds to the

current city or customer location), (ii) action taken, corresponding to the node that the policy decided to visit next (iii) log-probability given to that action by the model (iv) return of the entire trajectory (negative cost of the solution built) (v) budget at the time that solution was built (vi) log-probability that was suggested by the memory for that action (vii) log-probability associated with the entire trajectory and (viii) the log-probability associated with the remaining part of the trajectory. We can observe that elements (ii, iii and iv) are those needed to reproduce a REINFORCE update, and other features are additional context that will help for credit assignment and distribution shift estimation. The remaining budget is also added to the data when deriving the policy update.

Retrieving data from the memory Each time an action is taken in a given state, we retrieve data from the memory as shown in step 1 of Fig. 2. We retrieve data that inform the policy on past decisions that were taken in similar situations. To achieve a good balance between speed and relevance, we retrieve data collected in the same node as we are currently in; showing to be significantly faster than k-nearest neighbour retrieval, while extracting data of similar relevance. More details and motivation for this design choice provided in Appendix I.

Processing data to update actions Once data has been retrieved, it is used to derive a policy update. We separate the actions from their associated features (logp, return, etc...). We concatenate the remaining budget to the features. Each feature is normalised, and the resulting feature vector (Fig. 2, step 2) is processed by a Multilayer Perceptron (MLP) H_{θ_M} which outputs a *scalar weight*. We compute the logits from the MLP output (Fig. 2, step 3): each action is one-hot encoded, and a weighted average of the actions vectors is computed based on the scalar weight obtained. This aggregation outputs a new vector of action logits. We sum the new vector of logits with the vector of logits output by the base model. The following paragraph introduces the mathematical formalism, and Fig. 3 illustrates it schematically.

More formally, when visiting a node, with a given state s, we retrieve from the memory data experienced in the past when visiting that same node. This data, M_s, is a sequence of tuples (a_i, f_{a_i}), where a_i are the past actions tried and f_{a_i} are various features associated with the corresponding trajectories, as discussed above. The update is computed by $l_M = \sum_i \mathbf{a_i} H_{\theta_M}(f_{a_i})$, where $\mathbf{a_i}$ is the one-hot encoding of action a_i. Let l be the logits from the base policy, the final logits used to sample the next action are given by $l + l_M$. We show in Appendix F that MEMENTO has enough capacity to re-discover the REINFORCE update.

Training Existing construction methods are trained for one-shot optimization, with a few exceptions trained for few-shots (Grinsztajn et al., 2023; Chalumeau et al., 2023b; Hottung et al., 2024). We adapt the training process to fit the multi-shot setting in which methods are used in practice. We rollout the policy (Fig. 2, step 4-5) as many times as the budget allows on the same instance before taking an update. Our loss is inspired by ECO-DQN (Barrett et al., 2020) and ECORD (Barrett et al., 2022): for a new given trajectory, the reward is the ReLU of the difference between the return and the best return achieved so far. Hence, those summed rewards equal the best score found over the budget. In practice, to account for the fact that it is harder to get an improvement as we get closer to the end of the budget, we multiply this loss with a weight that increases logarithmically as the budget is consumed. See Appendix B for explicit formula and pseudo-code.

Inference The memory processing module can be used with any neural solver. The parameters of the base neural solver and those of the memory processing module are frozen (no more backpropagation), and the action updates are derived by filling the memory with the collected attempts and by processing the data retrieved from the memory, as described in the previous paragraphs, until the inference budget is consumed. All hyperparameters can be found in Appendix C.

4 Experiments

We evaluate our method across widely recognized routing problems TSP and CVRP. These problems serve as standard benchmarks for evaluating RL-based CO methods (Deudon et al., 2018; Kool et al., 2019; Kwon et al., 2020; Grinsztajn et al., 2023; Hottung et al., 2022; Choo et al., 2022; Chalumeau et al., 2023b). In Section 4.1, we validate our method by comparing it to leading approach for single-policy adaptation using online collected data, EAS (Hottung et al., 2022). We present their comparative performance in line with established standard benchmarking from literature, and

 https://doi.org/10.52202/085713-1108

elucidate the mechanisms employed to adapt the action distribution of the base policy during inference. We then demonstrate how MEMENTO can be combined with SOTA neural solver COMPASS with no additional retraining in Section 4.2. Finally, we scale those methods to larger instances in Section 4.3, outperforming heatmap-based methods.

Table 1: **MEMENTO against baselines on a standard benchmark comprising 8 datasets with 4 distinct instance sizes on (a) TSP and (b) CVRP.** Methods are evaluated on instances from the training distribution (100) and on larger instance sizes to test generalization. MEMENTO outperforms policy-gradient method EAS and tree-search SGBS, with significant improvement over the base policy POMO. SGBS * results are reported from Choo et al. (2022) (bias discussed in Appendix A.1).

(a) TSP

	Training distr.		**Generalization**					
	$n = 100$		$n = 125$		$n = 150$		$n = 200$	
Method	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
LKH3	7.765	0.000%	8.583	0.000%	9.346	0.000%	10.687	0.000%
POMO (greedy)	7.796	0.404%	8.635	0.607%	9.440	1.001%	10.933	2.300%
POMO (sampling)	7.779	0.185%	8.609	0.299%	9.401	0.585%	10.956	2.513%
SGBS *	*7.769**	0.058%	-	-	*9.367**	0.220%	*10.753**	*0.619%*
EAS-Emb	7.778	0.161%	8.604	0.238%	9.380	0.363%	10.759	0.672%
MEMENTO	**7.768**	**0.045%**	**8.592**	**0.109%**	**9.365**	**0.202%**	**10.758**	**0.664%**

(b) CVRP

	Training distr.		**Generalization**					
	$n = 100$		$n = 125$		$n = 150$		$n = 200$	
Method	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
LKH3	15.65	0.000%	17.50	0.000%	19.22	0.000%	22.00	0.000%
POMO (greedy)	15.874	1.430%	17.818	1.818%	19.750	2.757%	23.318	5.992%
POMO (sampling)	15.713	0.399%	17.612	0.642%	19.488	1.393%	23.378	6.264%
SGBS *	*15.659**	0.08%	-	-	*19.426**	1.08%	*22.567**	2.59%
EAS-Emb	15.663	0.081%	17.536	0.146%	19.321	0.528%	22.541	2.460%
MEMENTO	**15.657**	**0.066%**	**17.521**	**0.095%**	**19.317**	**0.478%**	**22.492**	**2.205%**

Code availability We provide access to the code[2] utilized for training our method and executing all baseline models. We release our checkpoints for all problem types and scales, accompanied by the necessary datasets to replicate our findings. We implement our method and experiments in JAX (Bradbury et al., 2018). The two problems are also JAX implementations from Jumanji (Bonnet et al., 2023). We use TPU v3-8 for our experiments.

4.1 Benchmarking MEMENTO against policy gradient fine-tuning

The most direct way to leverage online data for policy update is RL/policy-gradients. By contrast, MEMENTO learns an update based on the collected data. We therefore want to see how these two approaches compare. Since EAS (Hottung et al., 2022) is SOTA method using policy gradient, we benchmark MEMENTO against it on a standard set of instances used in the literature (Kool et al., 2019; Kwon et al., 2020; Hottung et al., 2022). Specifically, for TSP and CVRP, these datasets comprise 10 000 instances drawn from the training distribution. These instances feature the positions of 100 cities/customers uniformly sampled within the unit square. The benchmark also includes three datasets of distributions not encountered during training, each comprising 1000 problem instances with larger sizes: 125, 150, and 200, generated from a uniform distribution across the unit square. We employ the exact same datasets as those utilized in the literature.

Setup For routing problems such as TSP and CVRP, POMO (Kwon et al., 2020) is the base single-agent, one-shot architecture that underpins most RL construction solvers. In this set of experiments, MEMENTO and EAS both use POMO as a base policy, and adapt it to get the best possible performance within a given budget. We train MEMENTO until it converges, on the same instance distribution as that used for the initial checkpoint. When assessing active-search performance, each method operates within a fixed budget of 1600 attempts, a methodology akin to Hottung et al. (2022). In this setup, each attempt comprises of one trajectory per possible starting point. This standardized approach

[2]Code, checkpoints, and evaluation sets available at https://github.com/instadeepai/memento

facilitates direct comparison to POMO and EAS, which also utilize rollouts from all starting points at each step. We do not use *augmentations with symmetries* in our main results since they consume 87.5% of the budget to exploit network variance, which blurs the efficacy of the comparison and creates impractical settings for larger instances. Nevertheless, we report them in Appendix A.1 for continuity with previous literature.

Results The average performance of each method across all problem settings are presented in Table 1. The observations we draw are three-fold. First, MEMENTO outperforms its base model (POMO) on the entire benchmark by a significant margin: showing that its adaptive search is superior to stochastic sampling. Second, MEMENTO outperforms EAS on all 8 tasks (spanning both in- and out-of-distribution) for both environments, highlighting the efficacy of learned policy updates when compared to vanilla policy gradients. Finally, we note that this improvement is significant; for example, on TSP100, MEMENTO is doing 60% better than sampling, while EAS only 6%. We provide all time costs and performance comparison based on a time budget in Appendix A.1.

Analysing the update rule To understand how MEMENTO uses its memory to derive a policy update, we analyse the logit update of an action with respect to its associated data, inspired by Lu et al. (2022); Jackson et al. (2024), and compare it to the REINFORCE policy gradient. On Fig. 4, we plot the heatmap of the logit update with respect to the log probability (logp) and return associated with an action. We observe that the main rules learned by MEMENTO are similar to REINFORCE: an action with low logp and high return gets a positive update while an action with high logp and low return gets discouraged. It is interesting to see discrepancies. In particular, MEMENTO's rules are not symmetric with respect to $x = 0$, it only encourages action that are strictly above the mean return. A similar dissymetry would be expected with EAS since it combines REINFORCE with a term that increases the likelihood to generate the best solution found. Appendix J extends this analysis by visualising the evolution of the update rule for different remaining budget.

Additionally, MEMENTO uses *more inputs than REINFORCE*: current budget available, trajectory logp, age of the data, and previous MEMENTO logit are also used to derive the new action logit. Appendix G provides an ablation study of these additional inputs, validating their importance.

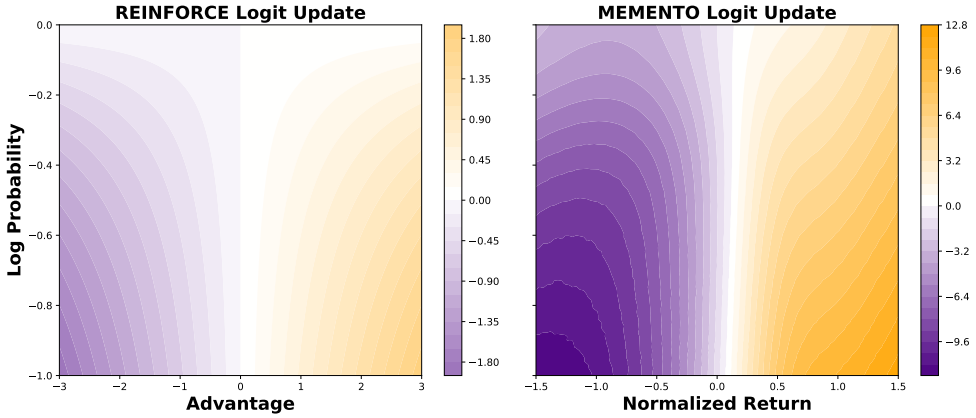

Figure 4: **Akin to REINFORCE (left), MEMENTO (right) encourages actions with high returns, particularly when they have low probability.** MEMENTO learns an asymmetric rule: requiring the normalised return to be strictly positive to reinforce an action, but encouraging it even more.

4.2 Zero-shot combination with unseen solvers

In this section, we demonstrate zero-shot combination (no retraining) with COMPASS. COMPASS adapts by searching amongst a collection of diverse pre-trained policies.

Although providing fast adaptation, it can hardly improve further once the appropriate pre-trained policy has been found. In particular, online collected data cannot be used to update that policy's action distribution. We use MEMENTO's memory module, trained with POMO as a base policy, apply it

https://doi.org/10.52202/085713-1108

on COMPASS without additional retraining, and show empirically that we can *switch on* MEMENTO's adaptation mechanism during the search and reach a new SOTA on 11 out of 12 tasks.

Methodology We combine our checkpoint of MEMENTO with a released model of COMPASS (Chalumeau et al., 2023b). For the first half of the budget, we do not use MEMENTO, to let COMPASS find the appropriate pre-trained policy. Once the search starts narrowing down, we activate MEMENTO to adapt the pre-trained policy, with no need to turn off COMPASS' search. Fig. 5 shows the typical trends observed at inference. We recognise the typical search reported in Chalumeau et al. (2023b). At 50% of budget consumption, MEMENTO is activated and one can observe a clear and significant drop in averaged tour length of the latest sampled solutions. The standard deviation also illustrates how the search is narrowed down.

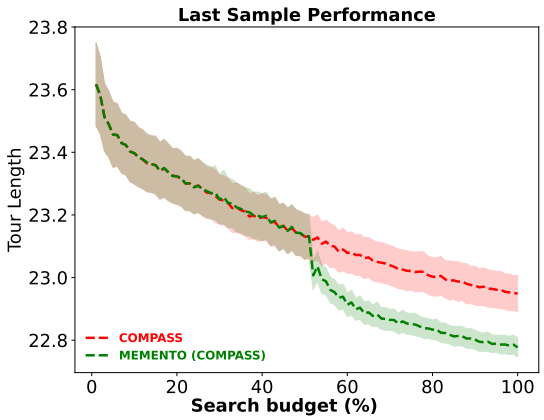

Figure 5: **Combining MEMENTO and COMPASS** during search on CVRP200, no re-training needed.

Results We evaluate the zero-shot combination of MEMENTO and COMPASS on the standard benchmark reported in Section 4.1. Results for CVRP are presented in Table 2, showing tour length and optimality gap in-distribution (CVRP100) and out-of-distribution. Although requiring no re-training, MEMENTO's rules combine efficiently with COMPASS and achieves new SOTA. Results on TSP are reported in Appendix A, and results on larger instances (TSP & CVRP) in Section 4.3.

Table 2: **SOTA performance on CVRP via zero-shot combination** of MEMENTO and COMPASS. The rules learned by MEMENTO transfer to the population-based method COMPASS.

	$n = 100$		$n = 125$		$n = 150$		$n = 200$	
Method	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
COMPASS	15.644	-0.019%	17.511	0.064%	19.313	0.485%	22.462	2.098%
MEMENTO (COMPASS)	**15.634**	**-0.082%**	**17.497**	**-0.041%**	**19.290**	**0.336%**	**22.405**	**1.808%**

4.3 Scaling RL construction methods to larger instances

Scaling neural solvers to larger instances is a crucial challenge for the field. Auto-regressive construction-based solvers are promising approaches, requiring limited expert knowledge whilst providing strong performance. But are still hardly ever trained on larger scales with RL. As a result, their performance reported in the literature is usually quite unfair, being evaluated on large scales instances (>500) although having been trained on 100 nodes (Qiu et al., 2022). In this section, we explain how we train construction solvers POMO and COMPASS on instances of size 500 with RL (on TSP and CVRP, checkpoints released); and use them as new base models to train and validate properties of MEMENTO at that scale.

RL training on larger instances Training POMO on instances of size 500 involves three main steps. First, inspired by curriculum-based methods, we start from a checkpoint pre-trained on TSP100. Second, we use Efficient Attention (Rabe and Staats, 2022) to reduce the memory cost of multi-head attention, enabling to rollout problems in parallel despite the $O(n^2)$ memory requirement. Third, we use gradient accumulation to keep good estimates despite the constrained smaller instance batch size. These combined tricks enable to train POMO till convergence within 4 days, and consequently train COMPASS and MEMENTO. We also build the zero-shot combined MEMENTO (COMPASS) checkpoint.

Inference-time constraints On larger instances, the cost of constructing a solution is significantly increased (for POMO's architecture: squared in computation and memory, and linear in sequential operations). It is also harder to get numerous parallel shots, which most methods depend on. It hence becomes crucial to develop data-efficient methods which stay robust to low budget regimes; and important to know which method to use for each budget constraint.

https://doi.org/10.52202/085713-1108

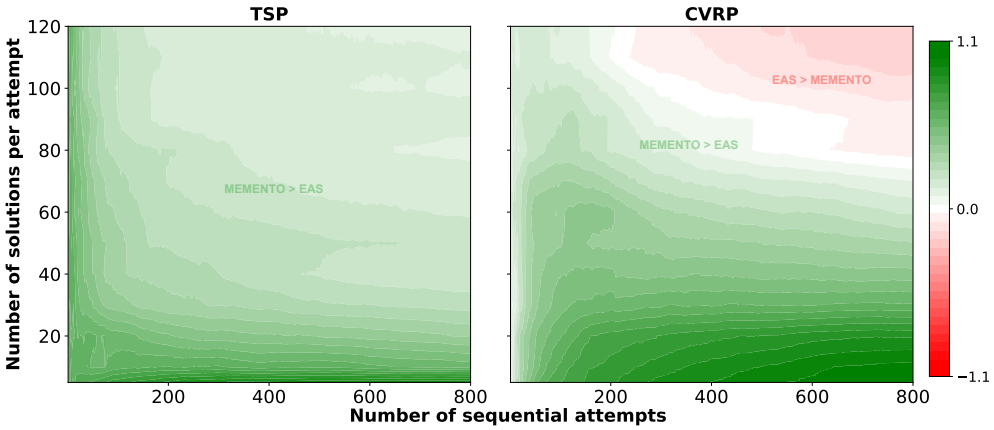

Figure 6: **MEMENTO outperforms EAS on instances of size 500 across batch sizes and sequential attempts.** Green areas indicate settings where MEMENTO adapts more efficiently. It consistently outperforms EAS on TSP and in most CVRP settings, with strong gains under low budgets.

MEMENTO on larger instances We compare MEMENTO and EAS on a set of 128 unseen instances (of size 500), for a range of budget expressed in number of sequential attempts and number of parallel attempts attempts. Figure 6 reports the percentage of improvement (in absolute cost) brought by MEMENTO over EAS. From this figure, we can observe three main tendencies. First, the data-efficiency of MEMENTO is illustrated by its consistent superiority on low budget (lower left of plots), reaching significant improvement compared to EAS. Then, we can see that, on CVRP, the gap increases with the sequential budget for low parallelism. It demonstrates that MEMENTO's update stays robust, whereas the gradient estimates of EAS get deteriorated and cannot bring further improvement despite additional sequential budget. To finish with, we can see that MEMENTO fully dominates the heatmap on TSP, even for higher budget. However, when budget is increased for CVRP, EAS is able to outperform MEMENTO, given large enough batches (> 80).

Results In Table 3 and Table 4, we report performance of our checkpoints, using datasets introduced in Fu et al. (2021) and commonly used in the literature. For each environment, we report the results on two different setting: (i) Low Budget, where the methods are given 25 000 attempts, (ii) High Budget, where 100 000 attempts are available. We also report results from concurrent RL methods (Qiu et al., 2022; Dernedde et al., 2024; Sun and Yang, 2023; Ye et al., 2023), without method-agnostic local search; and industrial solvers LKH3 (Helsgaun, 2017) and Concorde (Applegate et al., 2006).

Table 3: **MEMENTO achieves SOTA performance on TSP500.** It outperforms baselines in both budget regimes, attaining single-agent and overall SOTA when combined with POMO and COMPASS, respectively.

Method	Obj.	Gap	Time
Concorde	16.55	0.000%	38M
LKH-3	16.55	0.003%	46M
DIMES	17.80	7.55%	2H
Difusco	17.23	4.08%	11M
DeepACO	18.74	13.23%	-
MOCO	16.84	1.72%	-
Pointerformer	17.14	3.56%	1M
	Low Budget		
POMO (sampling)	16.999	2.736%	4M
EAS	16.878	2.006%	10M
COMPASS	16.811	1.603%	9M
MEMENTO (POMO)	16.838	1.766%	9M
MEMENTO (COMPASS)	**16.808**	**1.586%**	10M
	High Budget		
POMO (sampling)	16.986	2.659%	9M
EAS	16.844	1.804%	24M
COMPASS	16.800	1.539%	22M
MEMENTO (POMO)	16.823	1.673%	23M
MEMENTO (COMPASS)	**16.795**	**1.507%**	29M

First, we observe that stochastically sampling solutions with POMO for less than 10 minutes already provides competitive results, ranking second among the five leading neural solvers. Adding EAS on top of POMO enables to compete with leading method MOCO on TSP500. This shows that auto-regressive RL constructive methods are *competitive at this scale*, contradicting previous literature (Qiu et al., 2022; Dernedde et al., 2024). It is the first time that POMO is trained at this scale, making it the first RL-trained neural solver that

https://doi.org/10.52202/085713-1108

can be used for large instances on both TSP and CVRP: most previous large scale RL methods are graph-specific, and hence cannot be applied on CVRP.

These results confirm that MEMENTO scales to instance size, and can be zero-shot combined with COMPASS. On this benchmark, MEMENTO achieved SOTA on 3 of the 4 tasks when zero-shot combined with COMPASS, showing significant improvement compared to previous SOTA MOCO: the gap to optimality is reduced by more than 10%. It is the best single agent method on all TSP instances, and on the low budget CVRP task.

Time complexity Time performance at scale is key to the adoption of neural solvers, because the number of sequential batches of attempts that can be achieved within a time budget will mostly depend on it. Search methods must be able to tackle large instances in reasonable time, and must scale well with the size of the base policy used, since Luo et al. (2023) demonstrated the importance of using large and multi-layered decoders in neural CO.

Table 4: **MEMENTO outperforms baselines on CVRP500.** It achieves single-agent SOTA on low budget regime with POMO and overall with COMPASS.

Method	Obj.	Gap	Time
LKH-3	37.229	0.00%	6H
	Low Budget		
POMO (sampling)	37.501	0.731%	9M
EAS	37.425	0.525%	16M
COMPASS	37.336	0.287%	10M
MEMENTO (POMO)	37.367	0.369%	16M
MEMENTO (COMPASS)	**37.309**	**0.215%**	12M
	High Budget		
POMO (sampling)	37.456	0.608%	20M
EAS	**37.185**	**-0.120%**	36M
COMPASS	37.279	0.133%	25M
MEMENTO (POMO)	37.306	0.206%	65M
MEMENTO (COMPASS)	37.251	0.059%	36M

Hence, we evaluate MEMENTO and EAS on a set of increasing instance sizes and decoder sizes and report their time complexity on Fig. 1 (right). These plots show that, despite being slower for small settings, MEMENTO's adaptation mechanism scales better than EAS. For an instance size of 1000, MEMENTO becomes 20% faster. And even for small instances (size 100), using 10 layers in the decoder's architecture (1M parameters) makes EAS 40% slower than MEMENTO. These scaling laws illustrate another benefit of using an approach that learns the update rather than relying on back-propagation to adapt neural solvers at inference time.

5 Conclusion

We present MEMENTO, a method to improve adaptation of neural CO solvers to unseen instances by conditioning a policy directly on data collected online during search and search budget. In practice, it proves to outperform stochastic sampling, tree search and policy-gradient fine-tuning, and shows zero-shot combination with an unseen solver. Additionally, MEMENTO respects several key properties, like robustness to low-budget regimes, and favourable time performance scalability. We demonstrate the efficacy of MEMENTO by achieving SOTA single-policy adaptation on a standard benchmark on TSP and CVRP, both in and out-of-distribution. Moreover, we show that MEMENTO finds interpretable update rules to the underlying policy; trading off exploration and exploitation over the search budget and outperforming REINFORCE-style updates. We further demonstrate zero-shot combination of MEMENTO and COMPASS, achieving overall SOTA on a benchmark of 12 tasks.

Limitations and future work MEMENTO incurs additional compute and memory-usage compared to the memory-less base policies which it augments. Practically, we find that the performance gain significantly outweighs these overheads. A potential mitigation could be to derive the mathematical update learned by MEMENTO to avoid relying on the MLP computation. While we demonstrate successful zero-shot combination of MEMENTO with COMPASS, training MEMENTO with a diversity-based method or fine-tuning MEMENTO specifically on the COMPASS policy represents a promising direction for future work.

Acknowledgements

We thank the Python and JAX communities for developing tools that made this research possible. We thank the anonymous reviewers for their valuable comments and suggestions that helped improve the final version of this paper. Finally, we thank Google's TPU Research Cloud (TRC) for supporting our research with Cloud TPUs.

References

D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde TSP solver, 2006.

I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, I. Danihelka, A. Dedieu, C. Fantacci, J. Godwin, C. Jones, R. Hemsley, T. Hennigan, M. Hessel, S. Hou, S. Kapturowski, T. Keck, I. Kemaev, M. King, M. Kunesch, L. Martens, H. Merzic, V. Mikulik, T. Norman, G. Papamakarios, J. Quan, R. Ring, F. Ruiz, A. Sanchez, L. Sartran, R. Schneider, E. Sezener, S. Spencer, S. Srinivasan, M. Stanojević, W. Stokowiec, L. Wang, G. Zhou, and F. Viola. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

T. Barrett, W. Clements, J. Foerster, and A. Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3243–3250, 2020.

T. D. Barrett, C. W. Parsonson, and A. Laterre. Learning to solve combinatorial graph partitioning problems via efficient exploration. *arXiv preprint arXiv:2205.14105*, 2022.

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

C. Bonnet, D. Luo, D. Byrne, S. Abramowitz, V. Coyette, P. Duckworth, D. Furelos-Blanco, N. Grinsztajn, T. Kalloniatis, V. Le, O. Mahjoub, L. Midgley, S. Surana, C. Waters, and A. Laterre. Jumanji: a suite of diverse and challenging reinforcement learning environments in jax, 2023. URL https://github.com/instadeepai/jumanji.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

F. Chalumeau, B. Lim, R. Boige, M. Allard, L. Grillotti, M. Flageat, V. Macé, A. Flajolet, T. Pierrot, and A. Cully. Qdax: A library for quality-diversity and population-based algorithms with hardware acceleration, 2023a.

F. Chalumeau, S. Surana, C. Bonnet, N. Grinsztajn, A. Pretorius, A. Laterre, and T. D. Barrett. Combinatorial optimization with policy adaptation using latent space search. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b.

J. Choo, Y.-D. Kwon, J. Kim, J. Jae, A. Hottung, K. Tierney, and Y. Gwon. Simulation-guided beam search for neural combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2207.06190.

H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, 2017.

T. Dernedde, D. Thyssens, S. Dittrich, M. Stubbemann, and L. Schmidt-Thieme. Moco: A learnable meta optimizer for combinatorial optimization. *arXiv preprint arXiv:2402.04915*, 2024.

M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau. Learning heuristics for the tsp by policy gradient. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 170–181. Springer International Publishing, 2018.

M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving a cutting-stock problem with the constraint logic programming language chip. *Mathematical and Computer Modelling*, 16(1):95–105, 1992. ISSN 0895-7177. doi: https://doi.org/10.1016/0895-7177(92)90081-U. URL https://www.sciencedirect.com/science/article/pii/089571779290081U.

D. Drakulic, S. Michel, F. Mai, A. Sors, and J.-M. Andreoli. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

 https://doi.org/10.52202/085713-1108

A. Froger, M. Gendreau, J. E. Mendoza, Éric Pinson, and L.-M. Rousseau. Maintenance scheduling in the electricity industry: A literature review. *European Journal of Operational Research*, 251 (3):695–706, 2016. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2015.08.045. URL https://www.sciencedirect.com/science/article/pii/S0377221715008012.

Z.-H. Fu, K.-B. Qiu, and H. Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):7474–7482, May 2021. doi: 10.1609/aaai.v35i8.16916. URL https://ojs.aaai.org/index.php/AAAI/article/view/16916.

N. Grinsztajn, D. Furelos-Blanco, S. Surana, C. Bonnet, and T. D. Barrett. Winner takes it all: Training performant rl populations for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

K. Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde University, Tech. Rep.*, 2017.

J. J. Hopfield and W. D. Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

A. Hottung, Y.-D. Kwon, and K. Tierney. Efficient active search for combinatorial optimization problems. In *International Conference on Learning Representations*, 2022.

A. Hottung, M. Mahajan, and K. Tierney. Polynet: Learning diverse solution strategies for neural combinatorial optimization. *arXiv preprint arXiv:2402.14048*, Feb 2024.

P. C. Humphreys, A. Guez, O. Tieleman, L. Sifre, T. Weber, and T. P. Lillicrap. Large-scale retrieval for reinforcement learning. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=Ya9lATuQ3gg.

A. I. Garmendia, Q. Cappart, J. Ceberio, and A. Mendiburu. Marco: A memory-augmented reinforcement framework for combinatorial optimization. In K. Larson, editor, *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 6931–6939. International Joint Conferences on Artificial Intelligence Organization, 8 2024. doi: 10.24963/ijcai.2024/766. URL https://doi.org/10.24963/ijcai.2024/766. Main Track.

S. D. J-F Audy and L.-M. Rousseau. Cost allocation in the establishment of a collaborative transportation agreement—an application in the furniture industry. *Journal of the Operational Research Society*, 62(6):960–970, 2011. doi: 10.1057/jors.2010.53. URL https://doi.org/10.1057/jors.2010.53.

M. T. Jackson, C. Lu, L. Kirsch, R. T. Lange, S. Whiteson, and J. N. Foerster. Discovering temporally-aware reinforcement learning algorithms. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=MJJcs3zbmi.

L. Jingwen, Y. Ma, Z. Cao, Y. Wu, W. Song, J. Zhang, and Y. M. Chee. Learning feature embedding refiner for solving vehicle routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, PP, 05 2023. doi: 10.1109/TNNLS.2023.3285077.

M. Kim, J. Park, and j. kim. Learning collaborative policies to solve np-hard routing problems. In *Advances in Neural Information Processing Systems*, volume 34, page 10418–10430. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/564127c03caab942e503ee6f810f54fd-Abstract.html.

M. Kim, J. Park, and J. Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2022. doi: 10.48550/arXiv.2205.13209.

W. Kool, H. van Hoof, and M. Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Y.-D. Kwon, B. K. Jinho Choo, Y. G. Iljoo Yoon, and S. Min. Pomo: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

C. Lu, J. G. Kuba, A. Letcher, L. Metz, C. Schroeder de Witt, and J. Foerster. Discovered policy optimisation. In *Advances in Neural Information Processing Systems*. FLAIR, University of Oxford, 2022.

F. Luo, X. Lin, F. Liu, Q. Zhang, and Z. Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. In *Advances in Neural Information Processing Systems*, 2023.

M. Macfarlane, D. M. Roijers, and H. van Hoof. Training graph neural networks with policy gradients to perform tree search. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2021.105400. URL https://www.sciencedirect.com/science/article/pii/S0305054821001660.

J. Park, J. Chun, S. H. Kim, Y. Kim, J. Park, et al. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research*, 59(11):3360–3377, 2021.

R. Qiu, Z. Sun, and Y. Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=9u05zr0nhx.

M. N. Rabe and C. Staats. Self-attention does not need $o(n^2)$ memory, 2022.

J. Son, M. Kim, H. Kim, and J. Park. Meta-sage: scale meta-learning scheduled adaptation with guided exploration for mitigating scale shift on combinatorial optimization. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

V. Steinbiss, B.-H. Tran, and H. Ney. Improvements in beam search. In *Third international conference on spoken language processing*, 1994.

R. Sun, Z. Zheng, and Z. Wang. Learning encodings for constructive neural combinatorial optimization needs to regret. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence*, 2024.

Z. Sun and Y. Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=JV8Ff0lgVV.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60:611–624, 06 2012. doi: 10.1287/opre.1120.1048.

L. Xin, W. Song, Z. Cao, and J. Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *In Proceedings of the 35th National Conference on Artificial Intelligence, AAAI*, 2021.

H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li. DeepACO: Neural-enhanced ant systems for combinatorial optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=cd5D1DD923.

 https://doi.org/10.52202/085713-1108

C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and C. Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

https://doi.org/10.52202/085713-1108

Justification: No theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Code, model checkpoints and validation sets are provided.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification:

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [No]

   Justification: They are in the zip file containing the code, and will be made open source upon acceptance.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

https://doi.org/10.52202/085713-1108

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: Not relevant.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

https://doi.org/10.52202/085713-1108

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification:

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

https://doi.org/10.52202/085713-1108

Appendix

A Extended results

In Section 4, we compare MEMENTO to popular methods from the literature in numerous settings. In particular, Table 1 compares using stochastic sampling, MEMENTO and EAS to adapt POMO on the standard benchmark; and Figure 6 compares the relative performances of MEMENTO and EAS on larger instances of TSP and CVRP for several values of batch sizes and budget. In this section, we report additional values and results of other methods for those experiments.

A.1 Standard benchmark

We evaluate our method, MEMENTO, against leading RL methods and industrial solvers. For RL specific methods, we provide results for POMO with greedy action selection and stochastic sampling, and EAS; an active search RL method built on top of POMO, that fine-tunes a policy on each problem instance. We also report population-based method COMPASS and the zero-shot combination of COMPASS with MEMENTO.

We compare to the heuristic solver LKH3 (Helsgaun, 2017); the current leading industrial solver of both TSP and CVRP, as well as an exact solver Concorde (Applegate et al., 2006) which is a TSP-specific industrial solver, and the CVRP-specific solver (Vidal et al., 2012).

We use datasets of 10,000 instances with 100 cities/customer nodes drawn from the training distribution, and three generalization datasets of 1,000 instances of sizes 125, 150, and 200, all from benchmark sets frequently used in the literature (Kool et al., 2019; Kwon et al., 2020; Hottung et al., 2022; Grinsztajn et al., 2023; Chalumeau et al., 2023b). Table 5 displays results for TSP and CVRP on the standard benchmark. The columns provide the absolute tour length, the optimality gap, and the total inference time that each method takes to solve one instance within the attempts budget.

Standard benchmark with augmentation trick Augmentation with symmetries is a problem-specific trick that can only be used for a few CO problems. Most prior work assumes that this additional x8 batching can be achieved seamlessly, which is unlikely in practice, when simulating complex real-world scenarios. Using this trick means decreasing the room for search and adaptation, since 87.5% of the budget is consumed to squeeze performance through uncontrolled network variance, rather than letting methods use principled strategies. Nevertheless, we report the standard benchmark with the "augmentation trick" in Table 6. It can be observed that MEMENTO outperforms EAS in distribution (CVRP 100), and for the large instances task (CVRP 200). EAS leads on the two other tasks. MEMENTO leads the augmented benchmark overall: it consistently leads in distribution, and both methods are competing closely out-of-distribution. It is to be noted that we have not tuned MEMENTO's hyper-parameters for these runs.

Times reported When possible, we decide to report the time to solve one instance rather the entire dataset following four main observations: (i) First, the literature use datasets of varying sizes, e.g. 10k for TSP100, 1k for TSP200, 128 for TSP500, hence time reported can be confusing, and do not enable to clearly see how methods' solving time scale with instance size. (ii) Second, the default real-world application consist in solving one instance at a time, or solving several instances at the same time but on separated hardware. (iii) Additionally, measuring on the entire dataset mixes several aspects, i.e. the instance solving time is mixed with the batch scalability of the method. We do think that this property is very interesting to know, but should be considered on the side, rather than mixed with the instance solving time. (iv) Moreover, this property will express differently depending on the hardware available. For instance, on a small hardware POMO sampling with n attempts will be n times slower than POMO greedy; but given a large enough GPU, POMO sampling can be parallelised n times and hence take exactly the same amount of time as POMO greedy.

We were able to do so on the standard benchmark since we have implementations of most methods we were comparing in a single framework. Since we are reporting several external methods in Table 8, we could only report time taken for the full dataset in that case.

Performance with runtime as budget Expressing budget as time taken to solve one instance is subject to high biases but it is gives interesting perspectives on the time analysis and effects of

parallelism. In Table 7, we provide results of using runtime in stead of attempts as budget given to each methods to solve TSP and CVRP instances. We use EAS runtime as the reference time. These results may only be considered moderately since they are subject to a number of limitations: (i) different labs use different implementations and frameworks (ii) labs have access to different hardwares (iii) time is sensitive to parallelism, whereas number of attempts is not. Hence, comparing methods with time is subject to a higher number of biases, and would make it almost impossible to compare papers without a common codebase and hardware. We observe that MEMENTO still leads the benchmark, with 6 out of 8 top results; and both methods are very close on the two tasks where EAS leads.

Additional comments about the results On Table 5, we can see that in the single-agent setting, MEMENTO leads the entire benchmark, and in the population-based setting, MEMENTO (COMPASS) is leading the whole benchmark. MEMENTO is able to give significant improvement to COMPASS on CVRP, pushing significantly the state-of-the-art on this benchmark. On TSP, the improvement is not significant enough to be visible on the rounded results. To finish with, we can observe that the time cost associated with MEMENTO is reasonable and is worth the performance improvement (except maybe for TSP results of MEMENTO (COMPASS)).

Notes concerning SGBS All neural methods reported in Table 5 are our own runs with standardised checkpoints and rollout strategies, except for the results of SGBS, which were taken from Choo et al. (2022). This introduces three biases: (i) the base POMO checkpoint used by SGBS is not exactly the same as our re-trained POMO checkpoint (ii) SGBS uses the domain-specific augmentation trick that we do not use (iii) SGBS pre-selects starting points in CVRP, which we do not do.

Overall, the results reported in SGBS show that SGBS alone is always outperformed by EAS; hence, it should be expected that in all the settings where we outperform EAS, we would significantly outperform SGBS if both methods were evaluated exactly in the same way. We hence expect the gap between MEMENTO and SGBS to be larger than the one reported here. Additionally, SGBS has yet never been validated on larger instances. Nevertheless, we think that SGBS is a very efficient method from the NCO toolbox and appreciate that MEMENTO and SGBS are orthogonal, and could be combined for further improvement. We leave this for future work.

A.2 Performance over different number of parallel and sequential attempts

In Section 4.3, we show performance improvement of MEMENTO over EAS on instances of size 500 on TSP and CVRP for increasing number of sequential attempts and size of attempt batch. We report extended results with an additional competitor, POMO. We compare the online adaptation of the methods over four different sizes of batched solutions across increasing sequential attempts for each CO problem. Results from Table 8 show results of the three methods on TSP and CVRP. The columns show the best tour length performance for various values of sequential attempts expressed as budget, and solution batches of size N. Performance is averaged over a set of 128 instances.

A.3 Evaluation over larger instances

In Section 4.3, we evaluate MEMENTO and baselines on instances of size 500. For TSP, we use the dataset from Fu et al. (2021). For CVRP, we use the dataset from Luo et al. (2023). We do not include LEHD in our results since we focus on methods trained with Reinforcement Learning, and LEHD can only be successfully trained with supervised learning at the time of writing. Nevertheless, the good performance achieved by LEHD has motivated us to study the scaling law of MEMENTO as the number of layers in the decoder increases, reported on Appendix A.4.

In order to run COMPASS and MEMENTO (COMPASS) with the same batch sizes as other methods on those large instances, we reduced the number of starting points used by COMPASS to ensure that this number multiplied by the number of latent vector sampled at the same time is equal to the number of starting point used by other methods (POMO, EAS, MEMENTO). In practice, we used a latent vector batch of size 10, and hence divided the number of starting points by 10. This is slightly disadvantaging COMPASS and MEMENTO (COMPASS) but enables to respect the constraint of number of parallel batches that can be achieved at once. Note that this slightly impacts the time performance reported since JAX jitting process will not fuse the operations in the same way, additionally, when combined with MEMENTO, this impact the size of the memory (we keep one per starting point to

 https://doi.org/10.52202/085713-1108

Table 5: Results of MEMENTO against the baseline algorithms for (a) TSP and (b) CVRP. The methods are evaluated on instances from training distribution ($n = 100$) as well as on larger instance sizes to test generalization. We use the same dataset as the rest of the literature, those contain 10 000 instances for $n = 100$ and 1000 instances for $n = 125, 150, 200$. Gaps are computed relative to LKH3. We report time needed to solve one instance. SGBS * results are reported from Choo et al. (2022), and do not use the same POMO checkpoint as other reported results. Additionally, they rely on problem-specific tricks that were not used by other methods. Details in Appendix A.1.

(a) TSP

	Training distr. $n = 100$			**Generalization** $n = 125$			$n = 150$			$n = 200$		
Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
Concorde	7.765	0.000%	0.49S	8.583	0.000%	0.72S	9.346	0.000%	1S	10.687	0.000%	1.86S
LKH3	7.765	0.000%	2.9S	8.583	0.000%	4.4S	9.346	0.000%	6S	10.687	0.000%	11S
POMO (greedy)	7.796	0.404%	0.16S	8.635	0.607%	0.2S	9.440	1.001%	0.29S	10.933	2.300%	0.45S
POMO (sampling)	7.779	0.185%	16S	8.609	0.299%	20S	9.401	0.585%	29S	10.956	2.513%	45S
SGBS*	*7.769**	*0.058%**	-	-	-	-	*9.367**	*0.220%**	-	*10.753**	*0.619%**	-
EAS	7.778	0.161%	39S	8.604	0.238%	46S	9.380	0.363%	64S	10.759	0.672%	91S
MEMENTO (POMO)	7.768	0.045%	43S	8.592	0.109%	52S	9.365	0.202%	77S	10.758	0.664%	115S
COMPASS	7.765	0.008 %	20S	8.586	0.036 %	24S	9.354	0.078%	33S	10.724	0.349%	49S
MEMENTO (COMPASS)	**7.765**	**0.008%**	32S	**8.586**	**0.035%**	39S	**9.354**	**0.077%**	58S	**10.724**	**0.348%**	88S

(b) CVRP

	Training distr. $n = 100$			**Generalization** $n = 125$			$n = 150$			$n = 200$		
Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
HGS	15.563	−0.536%	19S	-	-	-	19.055	−0.884%	32S	21.766	−1.096%	61S
LKH3	15.646	0.000%	52S	17.50	0.000%	-	19.222	0.000%	72S	22.003	0.000%	90S
POMO (greedy)	15.874	1.430%	24S	17.818	1.818%	34S	19.750	2.757%	52S	23.318	5.992%	87S
POMO (sampling)	15.713	0.399%	24S	17.612	0.642%	34S	19.488	1.393%	52S	23.378	6.264%	87S
SGBS*	*15.659**	*0.08%**	-	-	-	-	*19.426**	*1.08%**	-	*22.567**	*2.59%**	-
EAS	15.663	0.081%	66S	17.536	0.146%	82S	19.321	0.528%	123S	22.541	2.460%	179S
MEMENTO (POMO)	15.657	0.066%	118S	17.521	0.095%	150S	19.317	0.478%	169S	22.492	2.205%	392S
COMPASS	15.644	-0.019%	29S	17.511	0.064%	39S	19.313	0.485%	56S	22.462	2.098%	85S
MEMENTO (COMPASS)	**15.634**	**-0.082%**	82S	**17.497**	**-0.041%**	100S	**19.290**	**0.336%**	118S	**22.405**	**1.808%**	272S

Table 6: Results of MEMENTO and the baseline algorithms with instance augmentation for (a) TSP and (b) CVRP.

(a) TSP

	Training distr. $n = 100$			**Generalization** $n = 125$			$n = 150$			$n = 200$		
Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
LKH3	7.765	0.000%	2.9S	8.583	0.000%	4.4S	9.346	0.000%	6S	10.687	0.000%	11S
SGBS	7.769	0.058%	-	-	-	-	9.367	0.220%	-	10.753	0.619%	-
POMO (sampling)	7.767	0.026%	16S	8.594	0.128%	20S	9.376	0.321%	29S	10.916	2.14%	45S
EAS	7.768	0.038%	39S	8.590	0.080%	46S	9.361	0.159%	64S	**10.730**	**0.403%**	91S
MEMENTO (POMO)	**7.765**	**0.008%**	43S	**8.586**	**0.035%**	52S	**9.355**	**0.091%**	77S	10.743	0.526%	115S

(b) CVRP

	Training distr. $n = 100$			**Generalization** $n = 125$			$n = 150$			$n = 200$		
Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
HGS	15.563	−0.536%	-	-	-	-	19.055	−0.884%	-	21.766	−1.096%	-
LKH3	15.646	0.000%	-	17.50	0.000%	-	19.222	0.000%	-	22.003	0.000%	-
SGBS	15.659	0.08%	-	-	-	-	19.426	1.08%	-	22.567	2.59%	-
POMO (sampling)	15.67	0.18%	24S	17.56	0.33%	34S	19.43	1.08%	52S	23.24	5.64%	87S
EAS	15.623	-0.175%	66S	**17.473**	**-0.153%**	82S	**19.261**	**0.213%**	123S	22.556	2.49%	179S
MEMENTO (POMO)	**15.616**	**-0.196%**	118S	17.511	0.040%	150S	19.316	0.477%	169S	**22.515**	**2.308%**	392S

adapt to POMO, although this is completely agnostic to MEMENTO's method in itself). Since those factors impacts TSP and CVRP performance in different ways, this explains why their relative speed differ, i.e. why MEMENTO (COMPASS) is slower on TSP but faster on CVRP.

A.4 Time and memory complexity analysis

Time complexity To get the time curves reported in Fig. 7, we used CVRP, since it is the environment were MEMENTO was the slowest compared to EAS. We hence expect curves on TSP to be

Table 7: Results of MEMENTO and the baseline algorithms with budget expressed as runtime for (a) TSP and (b) CVRP.

(a) TSP

	Training distr.			**Generalization**								
	$n = 100$			$n = 125$			$n = 150$			$n = 200$		
Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
LKH3	7.765	0.000%	-	8.583	0.000%	-	9.346	0.000%	-	10.687	0.000%	-
POMO (sampling)	7.779	0.216%	39S	8.609	0.299%	46S	9.401	0.585%	64S	10.956	2.513%	91S
EAS	7.778	0.161%	39S	8.604	0.238%	46S	9.380	0.363%	64S	**10.759**	**0.672%**	91S
MEMENTO	**7.768**	**0.046%**	39S	**8.592**	**0.110%**	46S	**9.365**	**0.203%**	64S	10.760	0.681%	91S

(b) CVRP

	Training distr.			**Generalization**								
	$n = 100$			$n = 125$			$n = 150$			$n = 200$		
Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
LKH3	15.647	0.000%	-	17.504	0.000%	-	19.225	0.000%	-	22.007	0.000%	-
POMO (sampling)	15.713	0.399%	66S	17.612	0.642%	82S	19.488	1.393%	123S	23.378	6.264%	179S
EAS	15.663	0.081%	66S	17.536	0.146%	82S	19.321	0.528%	123S	**22.541**	**2.460%**	179S
MEMENTO	**15.660**	**0.086%**	66S	**17.526**	**0.127%**	82S	**19.321**	**0.502%**	123S	22.546	2.450%	179S

Table 8: Results of MEMENTO, POMO and EAS on instances of size 500 of (a) TSP and (b) CVRP for increasing number of sequential attempts and sizes of attempt batch.

(a) TSP

	$N = 20$					$N = 40$				
Method	200	400	600	800	1000	200	400	600	800	1000
POMO (sampling)	16.9810	16.9707	16.9638	16.9619	16.9574	16.9717	16.96	16.9549	16.9523	16.9493
EAS	16.9223	16.8923	16.8754	16.864	16.8556	16.8777	16.8468	16.83	16.8208	16.8143
MEMENTO	**16.8312**	**16.81**	**16.799**	**16.7939**	**16.7902**	**16.8187**	**16.8018**	**16.7926**	**16.7855**	**16.7815**
	$N = 60$					$N = 80$				
Method	200	400	600	800	1000	200	400	600	800	1000
POMO (sampling)	16.9678	16.9587	16.9539	16.952	16.9503	16.9634	16.9547	16.9513	16.9495	16.9474
EAS	16.8616	16.8353	16.8211	16.8134	16.8084	16.8521	16.8234	16.8106	16.8014	16.7941
MEMENTO	**16.8129**	**16.7966**	**16.7867**	**16.7816**	**16.7780**	**16.8087**	**16.7935**	**16.7854**	**16.7811**	**16.7770**

(b) CVRP

	$N = 20$					$N = 40$				
Method	200	400	600	800	1000	200	400	600	800	1000
POMO (sampling)	37.5256	37.4917	37.4763	37.4639	37.4570	37.4858	37.4599	37.4475	37.4372	37.4307
EAS	37.6107	37.5988	37.5914	37.582	37.5769	37.5022	37.4546	37.422	37.4017	37.3849
MEMENTO	**37.3398**	**37.2992**	**37.2731**	**37.2589**	**37.2527**	**37.3172**	**37.2776**	**37.2515**	**37.2357**	**37.2260**
	$N = 60$					$N = 80$				
Method	200	400	600	800	1000	200	400	600	800	1000
POMO (sampling)	37.4598	37.436	37.4228	37.4191	37.4135	37.4588	37.4335	37.4198	37.4113	37.4050
EAS	37.4569	37.3679	37.3248	37.2892	37.2624	37.3588	37.2804	**37.2346**	**37.2021**	**37.1747**
MEMENTO	**37.292**	**37.2607**	**37.2305**	**37.2171**	**37.2092**	**37.2887**	**37.2556**	37.2382	37.2268	37.2142

even better for MEMENTO. For the instance size scaling, we use 50 starting points and 100 sequential attempts, and 1 layer in the decoder, and evaluate several instance sizes going from 100 to 1000. For the decoder layer depth scaling, we use CVRP100, 100 starting points and 160 sequential attempts. We then evaluate methods for a depth going from 1 to 10.

Computational and memory complexity analysis The use of a memory to compute corrected action logits (i.e., MEMENTO's policy update rule) introduces a computational overhead, whose scaling properties depend on specific design choices and problem parameters.

Consider the TSP100 example with the memory design presented in the paper. The memory contains 100 slots (one per node), each storing 40 entries with 5 float32 values, repeated for each starting point (100). This results in roughly 8 MB of memory data. The memory footprint therefore scales linearly with the instance size (since there is one slot per node), but remains independent of the base policy size. In addition, the MLP module used to process memory entries is small, consisting of two layers with eight hidden units.

https://doi.org/10.52202/085713-1108

Regarding computational cost, the main overhead comes from processing the retrieved entries through the MLP. At each step, 40 entries are retrieved and processed in parallel, so the effective cost is that of 40 independent MLP forward passes. This operation does not depend on the instance size or the policy's parameter count, but only on the number of retrieved entries. An additional cost arises from indexing the appropriate slice of the memory, which can grow slightly with the number of nodes.

In practice, these overheads make MEMENTO slower than EAS for small instance sizes or small neural networks. However, since its operations scale more favorably with instance size and parameter count (compared to backpropagation), MEMENTO becomes faster for larger problems.

Overall, the computational and memory overhead of MEMENTO is negligible when solving a few instances at a time, typical in practical settings where one receives a new problem instance and a limited compute budget, but it can become a bottleneck when solving large batches of problems in parallel, as often done in research benchmarks.

To provide a more concrete view of the memory-processing cost across instance sizes, we also report the runtime of POMO in Table 9. The difference between the two indicates the additional cost introduced by MEMENTO. Note that these values may depend on the backend: for example, JAX's `jit` compilation can impact relative timings, and alternative implementations (e.g., PyTorch) could yield different ratios.

Table 9: Average inference time (in seconds) for each algorithm across instance sizes.

Instance Size	100	250	500	750	875	950
POMO	57	114	225	360	443	504
EAS	84	194	446	774	973	1122
MEMENTO	148	212	425	684	833	940

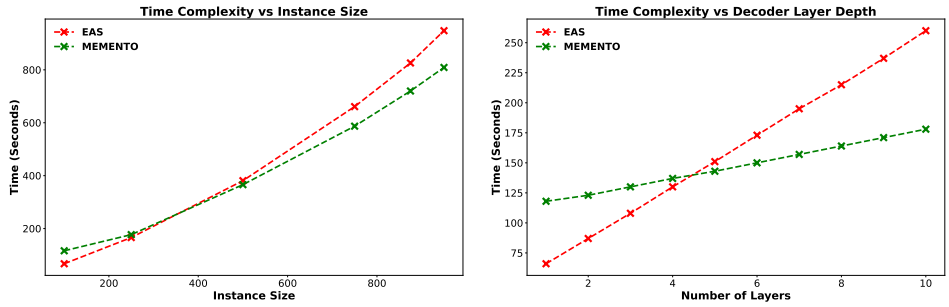

Figure 7: Time complexity of MEMENTO and EAS for increasing values of decoder size, and instance size. MEMENTO is shown to scale better in time than EAS.

B Training procedure

This section give a detailed description of how an existing pre-trained model can be augmented with MEMENTO, and how we train MEMENTO to acquire adaptation capacities.

Firstly, we take an existing single-agent model that was trained using the REINFORCE algorithm and reuse it as a base model. In our case, the base model is POMO. We augment POMO with a memory module and begin the training procedure which aims to create a policy that is able to use past experiences to make decisions in a multi-shot setting. We initialize the memory module weights with small values such that they barely affect the initial output. Hence, the initial MEMENTO checkpoint maintains the same performance as the pre-trained POMO checkpoint.

In the training procedure, the policy is trained to use data stored in the memory to take decisions and solve a problem instance. This is achieved by training the policy in a budgeted multi-shot setting on a problem instance where past experiences are collected and stored in a memory. The memory is organised by nodes such that only information about a specific node is found in a data row that

corresponds to that node. The policy retrieves data from the memory at each budget attempt and learns how to uses the data to decide on its next action.

The details of the MEMENTO training procedure are presented in Algorithm 1 and can be understood as follows. At each iteration, we sample a batch B of instances from a problem distribution $\mathcal{D}$. Then, for each instance ρ_i where $i \in 1, \ldots, B$, for K budget attempts, we retrieve data from the memory. This is done as follows; given that the current selected node is a_j, we only retrieve data $M(a_j)$ associated with node a_j and its starting point. However, the method is agnostic to starting point sampling and would work without it. The features associated with the retrieved action are then normalised. We add remaining budget as an additional feature and process the data by a Multilayer Perceptron (MLP) which outputs a scalar weight for each action. We compute correction logits by averaging the actions based on their respective weights. The correction logits are added to the logits of the base policy. We then rollout the resulting policy $\pi_{\tilde{\theta}}$ on the problem instance (i.e., generate a trajectory which represents a solution to the instance). After every policy rollout attempt, the memory is updated with transitions data such as the action taken, the obtained return, the log-probability of the action and the log-probability of the trajectory. These data are stored in the row that corresponds to the current selected node.

Details about the loss For each problem instance, we want to optimise for the best return. At each attempt, we apply the rectified linear unit (ReLU) function to the difference between the last return and the best return ever obtained. We use the rectified difference to compute the `REINFORCE` loss at each attempt to avoid having a reward that is too sparse and perform back-propagation through the network parameters of our model (including the encoder, the decoder and the memory networks). The sum of the rectified differences is equal to the best return ever obtained over the budget. As the budget is used, it becomes harder to improve over the previous best, the loss terms hence getting smaller. We found that adding a weight to the terms, with logarithmic increase, helped ensuring that the last terms would not vanish, and thus improved performance. We provide the mathematical formulation below.

Given a problem instance, we unroll B trajectories that we store iteratively in the memory. Each trajectory τ_i generates a return $R(\tau_i)$. The advantage for each trajectory is defined as $\tilde{R}(\tau_i) = \max(R(\tau_i) - R_{best}, 0)$, where R_{best} is the highest return found so far. The total loss for updating the policy is calculated using the REINFORCE algorithm: $\mathcal{L} = -\sum_{i=1}^{B} \log(1 + \epsilon + i)\tilde{R}(\tau_i) \sum_t \log \pi_M(a_t|s_t, M_t)$, where π_M is the policy enriched with the memory M_t, using the logits l_M defined in eq.1 in the paper. ϵ is a small number ensuring that the first term is not zero.

To keep the computations tractable, we still compute a loss at each step, estimate the gradient, and average them sequentially until the budget is reached, at which point we take a gradient update step and consider a new batch of instances.

In practice, we also observe that we can improve performance further by adding an optional refining phase where the base model is frozen, and only the memory module is trained, with a reduced learning rate (multiplied by 0.1), for a few hours.

C Hyper-parameters

We report all the hyper-parameters used during train and inference time. For our method MEMENTO, there is no training hyper-parameters to report for instance sizes 125, 150, and 200 as the model used was trained on instances of size 100. The hyper-parameters used for MEMENTO are reported in Table 10. Since we also trained POMO and COMPASS on larger instances, we report hyper-parameters used for POMO and COMPASS in Table 11 and Table 12, respectively.

D Model checkpoints

Our experiments focus on two CO routing problems, TSP and CVRP, with methods being trained on two distinct instance sizes: 100 and 500. Whenever possible, we re-use existing checkpoints from the literature; in the remaining cases, we release all our newly trained checkpoints in the repository *anonymised for the review process*.

https://doi.org/10.52202/085713-1108

Algorithm 1 MEMENTO Training

Input: problem distribution $\mathcal{D}$, problem size N, memory M, batch size B, budget K, number of training steps T, policy π_θ with pre-trained parameters θ.
initialize memory network parameters ϕ
combine pre-trained policy parameters and memory network parameters $\tilde{\theta} = (\theta, \phi)$
for step 1 to T **do**
  $\rho_i \leftarrow \text{Sample}(\mathcal{D}) \; \forall i \in 1, \dots, B$
  for attempt 1 to K **do**
    for node 1 to N **do**
      $m_j \leftarrow \text{Retrieve } (M) \; \forall j \in 1, \dots, B$ {Retrieve data from the memory}
      $\tau_i^j \leftarrow \text{Rollout } (\rho_i, \pi_{\tilde{\theta}}(\cdot|m_j)) \; \forall i, j \in 1, \dots, B$
    $m_j \leftarrow f(m_j, \tau_i^j)$ {Update the memory with transition data}
    $R_i^* \leftarrow \max(R_i^*, \mathcal{R}(\tau_i^j)) \; \forall i \in 1, \dots B$ {Update best solution found so far}
    $\nabla L(\tilde{\theta}) \leftarrow \frac{1}{B} \sum_{i \leq B} \texttt{REINFORCE}(\texttt{ReLU}(\tau_i^j - R_i^*))$ {Estimate gradient}
  $\nabla L(\tilde{\theta}) \leftarrow \frac{1}{K} \sum_{i=1}^{K} \nabla L(\tilde{\theta})$ {Accumulate gradients}
  $\tilde{\theta} \leftarrow \tilde{\theta} - \alpha \nabla L(\tilde{\theta})$ {Update parameters}

Table 10: The hyper-parameters used in MEMENTO

(a) TSP

Phase	Hyper-parameters	TSP100	TSP(125, 150)	TSP200	TSP500
Train time	budget	200	-	-	200
	instances batch size	64	-	-	32
	starting points	100	-	-	30
	gradient accumulation steps	200	-	-	400
	memory size	40	-	-	80
	number of layers	2	-	-	2
	hidden layers	8	-	-	8
	activation	GELU	-	-	GELU
	learning rate (memory)	0.004	-	-	0.004
	learning rate (encoder)	0.0001	-	-	0.0001
	learning rate (decoder)	0.0001	-	-	0.0001
Inference time	policy noise	1	0.2	0.1	0.8
	memory size	40	40	40	40

(b) CVRP

Phase	Hyper-parameters	CVRP100	CVRP(125, 150)	CVRP200	CVRP500
Train time	budget	200	-	-	200
	instances batch size	64	-	-	8
	starting points	100	-	-	100
	gradient accumulation steps	200	-	-	800
	memory size	40	-	-	40
	number of layers	2	-	-	2
	hidden units	8	-	-	8
	activation	GELU	-	-	GELU
	learning rate (memory)	0.004	-	-	0.004
	learning rate (encoder)	0.0001	-	-	0.0001
	learning rate (decoder)	0.0001	-	-	0.0001
Inference time	policy noise	0.1	0.1	0.1	0.3
	memory size	40	40	40	40

We evaluate MEMENTO on two CO problems, TSP and CVRP, and compare the performance to that of two main baselines: POMO (Kwon et al., 2020) and EAS (Hottung et al., 2022). The checkpoints used to evaluate POMO on TSP and CVRP are the same as the one used in Grinsztajn et al. (2023) and Chalumeau et al. (2023b), and the EAS baseline is executed using the same POMO checkpoint. These checkpoints were taken in the publicly available repository https://github.com/instadeepai/poppy. The POMO checkpoint is used in the initialisation step of MEMENTO (as described in Appendix B). To combine MEMENTO and COMPASS on TSP100, we re-use the COMPASS checkpoint available at https://github.com/instadeepai/compass and add the memory processing layers from the MEMENTO checkpoint trained on TSP100. This checkpoint is also released at *anonymised for the review process*.

Table 11: The hyper-parameters used in POMO

Phase	Hyper-parameters	TSP500	CVRP500
Train time	starting points	200	200
	instances batch size	64	32
	gradient accumulation steps	1	2
Inference time	policy noise	1	1
	sampling batch size	8	8

Table 12: The hyper-parameters used in COMPASS

Phase	Hyper-parameters	TSP500	CVRP500
Train time	latent space dimension	16	16
	training sample size	64	32
	instances batch size	8	8
	gradient accumulation steps	8	16
Inference time	policy noise	0.5	0.3
	num. CMA-ES components	1	1
	CMA-ES init. sigma	100	100
	sampling batch size	8	8

For larger instances, we compare our MEMENTO method to three baselines: POMO, EAS and COMPASS. Since no checkpoint of POMO and COMPASS existed, we trained them with the tricks explained in Section 4. The process to generate the MEMENTO checkpoint and the MEMENTO (COMPASS) checkpoint is then exactly the same. All those checkpoints are available, for both TSP and CVRP.

E Implementation details

The code-base in written in JAX (Bradbury et al., 2018), and is mostly compatible with recent repositories of neural solvers written in JAX, i.e. Poppy and COMPASS. The problems' implementation are also written in JAX and fully jittable. Those come from the package Jumanji (Bonnet et al., 2023). CMA-ES implementation to mix MEMENTO and COMPASS is taken from the research package QDax (Chalumeau et al., 2023a). Neural networks, optimizers, and many utilities are implemented using the DeepMind JAX ecosystem (Babuschkin et al., 2020).

F Can MEMENTO discover the REINFORCE update?

In Section 3.2, we presented the architecture used by the auxiliary model that processes the memory data to derive the new action logits. The intuition behind this architecture choice is that it should be able to learn the REINFORCE update rule. Indeed the REINFORCE loss associated with a new transition is $R\log(\pi_\theta(a))$, such that $\frac{\partial R\log(\pi_\theta(a))}{\partial l_a} = R(1-\pi_\theta(a))$. Therefore, simply having $H_{\theta_M}(f_a)$ match $R(1-\pi_\theta(a))$ would recover a REINFORCE-like update. This is feasible, as R and $\log(\pi_\theta(a))$ are included in the features f_a.

We compare the rule learned by MEMENTO to REINFORCE in Fig. 4 and provide an analysis of how the rule learned by MEMENTO evolve over different budgets in Appendix J. In Appendix G, we present an ablation study of the features used in the update rule of MEMENTO, showing how much performance can be gained from the use of more information to derive the update rule.

G Ablation Study: MEMENTO input features

In Section 3.2, we present MEMENTO, in particular, we present all the inputs that are used by the neural module to derive the new action logits from the memory data. These inputs, or features, are information associated to each past action taken, and that help decide whether those actions should be taken again or not. In Section 4.1, we compare the rule learned by MEMENTO compared to the policy-gradient update REINFORCE. This comparison is made on the features that both REINFORCE and MEMENTO use: i.e. the action log probability, and the return (or advantage if a baseline is used). Although REINFORCE only uses those two features, MEMENTO uses more, which enables to refine even more the update, and also to adapt it to the budget remaining in the search process. As a recall, in addition to the log probability and return, MEMENTO uses: the log probability

https://doi.org/10.52202/085713-1108

of the full trajectory, the log probability of the rest of the trajectory after the action was taken, the budget at the time the action was taken, the action logit suggested by MEMENTO when the action was taken, and the budget currently remaining.

To validate the impact of all the features used in the memory, we provide an ablation study of those features. To highlight the interest of using all the additional features, we report the performance of MEMENTO using only the return and the log probability, against the performance of MEMENTO with all features, on Fig. 8. We also report on Fig. 9 a bigger ablation where components are added one after the others.

We can extract two main observations: (i) first, adding the remaining budget completely changes the strategy. We can see on the right panel of Fig. 8 that having access to this additional feature enables MEMENTO to explore much more, and then to focus on high-performing solutions when it gets closer to the end of the budget; (ii) then, Fig. 9 confirms empirically that all features contribute to improving the overall performance.

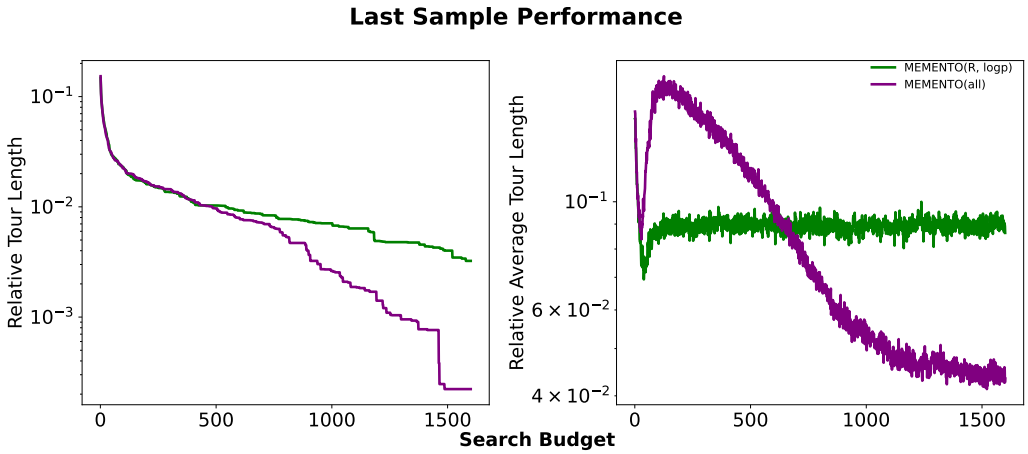

Figure 8: Ablation study of MEMENTO: comparing the impact of memory features that are not available in usual policy gradient estimations methods. The left plot reports the best solution found so far. The right plot shows the performance of the latest solution sampled. The plot illustrates how the additional features enable to achieve a complex exploration strategy, reaching a significantly more efficient adaptation mechanism.

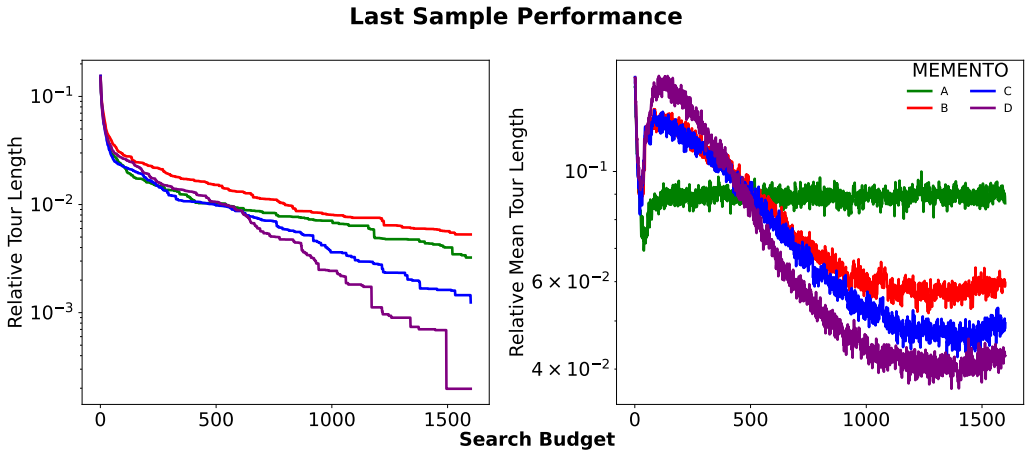

Figure 9: Full ablation study of MEMENTO: comparing the impact of memory features that are not available in usual policy gradient estimations methods. The left plot reports the best solution found so far. The right plot shows the performance of the latest solution sampled. A: return + logp; B: A + remaining budget; C: B + budget when action was taken; D: C + memory logp + trajectory logp + end trajectory logp.

H Evaluation metrics during training phase

In this section, we provide two plots of MEMENTO's training phase. They show the evolution of performance over time during MEMENTO's training on CVRP100. The left plot reports the evolution of the best tour length obtained during validation over time. The right plot reports the evolution of the improvement delta over time, i.e. the difference between the quality of the best solution generated minus the quality of the first solution generated. This metric shows well how the training phase results in MEMENTO learning an update rule that is able to significantly improve the base policy.

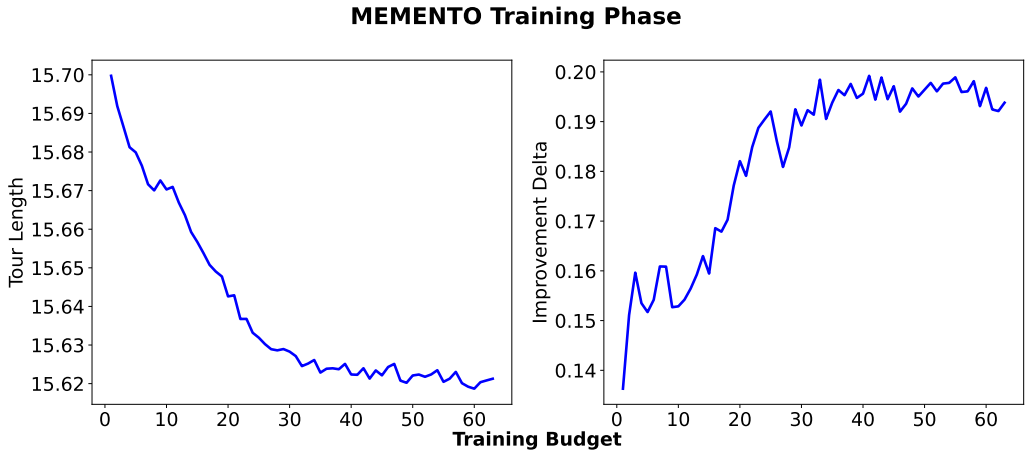

Figure 10: Evaluation metrics during the training phase of MEMENTO.

I Memory retrieval mechanism

In this section, we provide additional motivation for the choice of the node retrieval mechanism used in MEMENTO.

Ideally, the retrieval mechanism should retrieve data based on similarity to the current partial solution. But this comes with a computation cost since it requires getting a similarity measure and extracting the k most similar points in the entire memory. We observe that retrieving from the same node is an excellent proxy for similarity, and that the most similar points are very likely to come from the same node. This retrieval strategy hence provides a better trade-off between quality and computation cost, which is why it was selected as the final strategy for MEMENTO.

Nevertheless, MEMENTO comes as a framework, and each practitioner is free to change part of the method to fit each specific problem. One can hence update the retrieval mechanism if desired. Below is an example of an alternative strategy for retrieval that does consider the partial solution constructed. Since the output of the multi-head attention of POMO 's decoder builds a low-dimensional representation of the partial solution, one can store this vector in the memory, and when building a new solution, retrieve only the k-nearest neighbors of the current partial solution's representation. One could even apply further dimensionality reduction to reduce the cost of the nearest neighbor search. This approach brings a significant cost increase, even when using state-of-the-art approximated nearest neighbor JAX implementation. This effect gets worse when batching the attempts, or the problem instances. In the problems and settings considered in this paper, our simplified retrieval approach maintains similar results, while significantly improving scaling.

J Evolution of MEMENTO learned update over budget

We further analyse how the update rule learned by the MLP evolves over the course of the budget. Specifically, we compute the MLP's logit correction values across a grid of inputs by varying the normalised return and the log probability of the action, while fixing all the other inputs. These allows us to isolate the learned update pattern across the return and log probability space. As shown in Fig. 11, MEMENTO strongly upweights low-probability, high-return actions, consistent

https://doi.org/10.52202/085713-1108

with advantage-weighted updates, but exhibits a sharper concentration around high-value transitions. Importantly, we observe a temporal shift: early in the budget, the MLP assigns nonzero corrections even to uncertain or suboptimal transitions promoting broad exploration. As the remaining budget decreases, the corrections become increasingly selective, focusing on high-return, high-confidence actions.

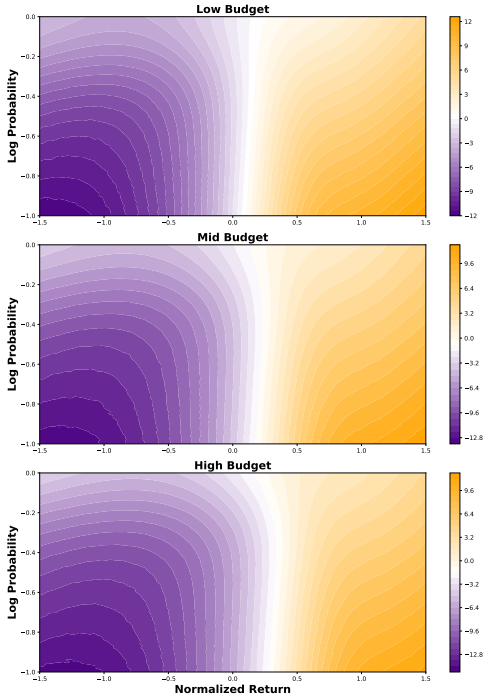

Figure 11: The update rule discovered by MEMENTO across different budget stages (low, mid, high). The learned update rule shifts from broad exploration to sharper exploitation as the budget decreases.

K Impact of train-time budget

The main experiments report results for MEMENTO trained with a budget of 200 sequential attempts. In this analysis, we evaluate variants of MEMENTO trained under alternative training budgets, and evaluate their performance for increasing inference budgets. We report the results in Table 13.

Despite being trained with different budgets, all variants of MEMENTO exhibit stable performance across a wide range of inference budgets. This confirms that conditioning on the remaining budget enables flexible behaviour even when the actual inference budget differs from that used in training. Overall, we observe that MEMENTO performs robustly across budget mismatches and that all variants generalise reasonably well beyond their training budget. Notably, the model trained with 200 attempts consistently performs the best which we attribute to a tuning bias: the hyperparameters were selected based on validation performance at budget 200, and were not re-tuned for other settings.

Table 13: Performance of MEMENTO on TSP100 across different inference budgets, trained with three budget settings.

	Inference Budgets																	
	$B = 50$		$B = 100$		$B = 200$		$B = 300$		$B = 400$		$B = 500$		$B = 600$		$B = 1200$		$B = 1600$	
Budget	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap
100	7.769	0.0077%	7.767	0.0063%	7.767	0.0058%	7.767	0.0058%	7.767	0.0054%	7.767	0.0056%	7.767	0.0055%	7.767	0.005%	7.766	0.005%
200	7.768	0.0074%	7.767	0.0058%	7.766	0.005%	7.766	0.0042%	7.766	0.0043%	7.766	0.0042%	7.766	0.004%	7.765	0.0035%	7.765	0.0035%
300	7.768	0.0075%	7.768	0.0066%	7.767	0.0056%	7.767	0.0057%	7.767	0.0057%	7.767	0.005%	7.766	0.0043%	7.766	0.0045%	7.766	0.0045%

https://doi.org/10.52202/085713-1108