BQ-NCO: BISIMULATION QUOTIENTING FOR GENERALIZABLE NEURAL COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite the success of Neural Combinatorial Optimization methods for end-to-end heuristic learning, out-of-distribution generalization remains a challenge. In this paper, we present a novel formulation of combinatorial optimization (CO) problems as Markov Decision Processes (MDPs) that effectively leverages symmetries of the CO problems to improve out-of-distribution robustness. Starting from the standard MDP formulation of constructive heuristics, we introduce a generic transformation based on bisimulation quotienting (BQ) in MDPs. This transformation allows to reduce the state space by accounting for the intrinsic symmetries of the CO problem and facilitates the MDP solving. We illustrate our approach on the Traveling Salesman and Capacitated Vehicle Routing Problems. We present a BQ reformulation of these problems and introduce a simple attention-based policy network that we train by imitation of (near) optimal solutions for small instances from a single distribution. We obtain new state-of-the-art generalization results for instances with up to 1000 nodes from synthetic and realistic benchmarks that vary both in size and node distributions.

1 INTRODUCTION

Combinatorial Optimization problems are crucial in many application domains such as transportation, energy, logistics, etc. Because they are generally NP-hard (Cook et al., 1997), their resolution at real-life scales is mainly done by heuristics, which are efficient algorithms that generally produce good quality solutions (Boussaïd et al., 2013). However, strong heuristics are generally problem-specific and designed by domain experts. Neural Combinatorial Optimization (NCO) is a relatively recent line of research that focuses on using deep neural networks to learn such heuristics from data, possibly exploiting information on the specific distribution of problem instances of interest (Bengio et al., 2021; Cappart et al., 2021). Despite the impressive progress in this field over the last few years, their out-of-distribution generalization, especially to larger instances, remains a major hurdle (Joshi et al., 2022; Manchanda et al., 2022).

In this paper, we are interested in constructive NCO methods, which build a solution incrementally, by applying a sequence of elementary steps. These methods are often quite generic, see e.g. the seminal papers by Khalil et al. (2017); Kool et al. (2019). Most CO problems can indeed be represented in this way, although the representation is not unique as the nature of the steps is, to a large extent, a matter of choice. Given a choice of step space, solving the CO problem amounts to computing an optimal policy for sequentially selecting the steps in the construction. This task can typically be performed in the framework of Markov Decision Processes (MDP), through imitation or reinforcement learning. The exponential size of the state space, inherent to the NP-hardness of combinatorial problems, usually precludes other methods such as (tabular) dynamic programming. Whatever the learning method used to solve the MDP, its efficiency, and in particular its out-of-distribution generalization capabilities, depend greatly on the state representation. The state space is often characterized by deep symmetries, which, if they are not adequately identified and leveraged, hinders the training process by forcing it to independently learn the policy at states which in fact are essentially the same (modulo some symmetry).

In this work, we investigate a type of symmetries which often occurs in MDP formulations of constructive CO heuristics. We first introduce a generic framework to systematically derive a naive CO problem-specific MDP. We formally demonstrate the equivalence between solving the MDP

and solving the CO problem and highlight the flexibility of the MDP formulation, by defining a minimal set of conditions for the equivalence to hold. Our framework is general and easy to specialize to encompass previously proposed learning-based construction heuristics. We then show that the state space of this naive MDP is inefficient because it fails to capture deep symmetries of the CO problem, even though such symmetries are easy to identify. Therefore, we propose a method to transform the naive MDP, based on the concept of *bisimulation quotienting* (BQ), in order to get a reduced state space, which is easier to process by the usual (approximate) MDP solvers. We illustrate our approach on two well-known CO problems, the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP). Furthermore, we propose a simple transformer-based architecture for these problems, that we train by imitation of expert trajectories derived from (near) optimal solutions. In particular, we show that our model is well-suited for our BQ formulation: it spends a monotonically increasing amount of computation as a function of the subproblem size (and therefore complexity), in contrast to most previous models. Finally, extensive experiments confirm the validity of our approach, and in particular its state-of-the-art out-of-distribution generalization capacity. In summary, our contributions are as follows:

- We present a generic and flexible framework to define a construction heuristic MDP for arbitrary CO problems;
- We propose a method to simplify commonly used naive MDPs for constructive NCO via symmetry-focused bisimulation quotienting.
- We design an adequate transformer-based architecture for the new MDP, for both the TSP and CVRP;
- We achieve state-of-the-art generalization performance on these two problems.

2 Combinatorial Optimization as a Markov Decision Problem

In this section, we present a generic formalization of constructive heuristics which underlies their MDP formulation. A deterministic CO problem is denoted by a pair $(\mathcal{F}, \mathcal{X})$, where $\mathcal{F}$ is its objective function space and $\mathcal{X}$ its (discrete) solution space. A problem instance $f{\in}\mathcal{F}$ is a mapping $f{:}\mathcal{X}{\to}\mathbb{R}{\cup}\{\infty\}$, with the convention that $f(x){=}\infty$ if x is infeasible for instance f. A solver for problem $(\mathcal{F}, \mathcal{X})$ is a functional:

$$\textsc{solve} : \mathcal{F} \to \mathcal{X} \qquad \text{satisfying} \qquad \textsc{solve}(f) = \arg\min_{x \in \mathcal{X}} f(x). \tag{1}$$

Incremental solution construction Constructive heuristics for CO problems build a solution sequentially, starting at an empty partial solution and expanding it at each step until a finalized solution is reached. Many NCO approaches are based on a formalization of that process as an MDP, e.g. Khalil et al. (2017); Kool et al. (2019); Zhang et al. (2020). Such an MDP can be obtained, for an arbitrary CO problem $(\mathcal{F}, \mathcal{X})$, using the following ingredients:

• **Steps**: $\mathcal{T}$ is a set of available steps to construct solutions. A *partial solution* is a pair $(f, t_{1:n})$ of a problem instance $f{\in}\mathcal{F}$ and a sequence of steps $t_{1:n}{\in}\mathcal{T}^*$. Observe that a partial solution (in $\mathcal{F}{\times}\mathcal{T}^*$) is *not* a solution (in $\mathcal{X}$), but may *represent* one.

• **Representation**: $\textsc{sol}{:}\mathcal{F}{\times}\mathcal{T}^*{\to}\mathcal{X}{\cup}\{\perp\}$ is a mapping that takes a partial solution and returns either the feasible solution which it represents if it is *finalized*, or $\perp$ otherwise.

• **Evaluation**: $\textsc{val}{:}\mathcal{F}{\times}\mathcal{T}^*{\to}\mathbb{R}{\cup}\{\infty\}$ is a mapping that takes a partial solution and returns an estimate of the minimum value of its expansions into finalized solutions. If the returned value is finite, the partial solution is said to be *admissible*.

In order to define an MDP using these ingredients, we assume they satisfy the following axioms:

$$\forall f{\in}\mathcal{F}, x{\in}\mathcal{X}, \quad f(x) < \infty \;\Leftrightarrow\; \exists t_{1:n} \in \mathcal{T}^* \text{ such that } \textsc{sol}(f, t_{1:n}) = x, \tag{2a}$$

$$\forall f{\in}\mathcal{F}, t_{1:n}{\in}\mathcal{T}^*, \quad \textsc{sol}(f, t_{1:n}) \neq \perp \;\Rightarrow\; \forall m{\in}\{1{:}n{-}1\},\ \textsc{sol}(f, t_{1:m}) = \perp, \tag{2b}$$

$$\forall f{\in}\mathcal{F}, t_{1:n}{\in}\mathcal{T}^*, x{\in}\mathcal{X}, \quad \textsc{sol}(f, t_{1:n}) = x \;\Rightarrow\; \begin{cases} \textsc{val}(f, t_{1:n}) = f(x), \\ \forall m \in \{1{:}n{-}1\},\ \textsc{val}(f, t_{1:m}) < \infty. \end{cases} \tag{2c}$$

Equation 2a states that the feasible solutions are exactly those represented by a finalized partial solution; Equation 2b states that if a partial solution is finalized then none of its preceding partial solutions in the construction can also be finalized; Equation 2c states that the evaluation of a finalized

partial solution is the value of the solution it represents, and all its preceding partial solutions are admissible.

We call a triplet $\langle \mathcal{T}, \text{SOL}, \text{VAL} \rangle$ satisfying the above axioms a *specification* of problem $(\mathcal{F}, \mathcal{X})$. Note that a specification is not intrinsic to the problem. The step space $\mathcal{T}$ results from a choice of how to construct a solution sequentially. Once $\mathcal{T}$ is chosen, SOL is determined, and must satisfy Equation 2a and 2b. Then VAL is only loosely constrained by Equation 2c, and can be chosen among a wide range of alternatives, including the following straightforward, uninformed one and the ideal, but intractable one (and, more likely, somewhere in between these two extremes):

$$\text{VAL}^{\text{uninformed}}(f, t_{1:n}) =_{\text{def}} f(x) \text{ if } [\text{SOL}(f, t_{1:n}) = x \neq \bot] \text{ else } 0,$$
$$\text{VAL}^{\text{ideal}}(f, t_{1:n}) =_{\text{def}} \min\{f(x) | x \in \mathcal{X}, \exists u_{1:m} \in \mathcal{T}^* \text{ s.t. } \text{SOL}(f, t_{1:n}u_{1:m}) = x\},$$

with the convention $\min \emptyset = \infty$. Value 0 in the uninformed case can be replaced by any constant.

Solution construction as an MDP Using a specification $\langle \mathcal{T}, \text{SOL}, \text{VAL} \rangle$ of problem $(\mathcal{F}, \mathcal{X})$, one can derive a "naive" MDP as follows. **States** are partial solutions (in $\mathcal{F} \times \mathcal{T}^*$); **actions** are steps (in $\mathcal{T}$); a state is **terminal** if it is a finalized partial solution; **transitions**: action $u \in \mathcal{T}$ applied to a non-terminal state $(f, t_{1:n})$ leads to state $(f, t_{1:n}u)$ where u is appended to the sequence so far, with **reward** $\text{VAL}(f, t_{1:n}) - \text{VAL}(f, t_{1:n}u)$, conditioned on $\text{VAL}(f, t_{1:n}u)$ being finite. Note that VAL has the double role of providing a reward and specifying the set of allowed actions. The number of these is expected to be linear, or at worst polynomial, in the size of the instance, since picking a step should not be as complex as solving the whole problem.

Now, assume we have access to a generic solver $\text{SOLVE}^{\text{MDP}}$, which, given an MDP $\mathcal{M}$ and one of its states s_o, returns an optimal trajectory starting at that state, i.e. $\arg\max_\tau R(\tau)$ where τ ranges over the $\mathcal{M}$-trajectories starting at s_o and ending in a terminal state, and $R(\tau)$ denotes its cumulated reward. Note that because we are dealing with deterministic MDPs, looking for an optimal policy is the same as looking for an optimal trajectory for a given set of initial states. That is why $\text{SOLVE}^{\text{MDP}}$ is defined here directly in terms of trajectories rather than policies. $\text{SOLVE}^{\text{MDP}}$ can then be specialized into a solver for the specific CO problem $(\mathcal{F}, \mathcal{X})$:

Proposition 1 *Let $\mathcal{M}_o$ be the naive MDP obtained from specification $\langle \mathcal{T}, \text{SOL}, \text{VAL} \rangle$. The procedure defined as follows (where ϵ denotes the empty sequence) satisfies the requirement of equation 1:*

$$\text{SOLVE}(f \in \mathcal{F}) =_{def} \{\text{SOL}(s) | s \textit{ is the last state of the trajectory } \text{SOLVE}^{MDP}(\mathcal{M}_o, (f, \epsilon))\}.$$

In other words, solving the naive MDP is equivalent to solving the CO problem. The detailed proof of Proposition 1 is in Appendix E. Of course, procedure $\text{SOLVE}^{\text{MDP}}$ may be approximate, in which case so is procedure SOLVE. Moreover, its performance depends on that of $\text{SOLVE}^{\text{MDP}}$, esp. its out-of-distribution generalization capacity, but also on the choice of specification, esp. of action space. It is a distinguishing feature of CO from an MDP perspective that the action space is *not* prescribed by the problem.

The impact of the choice of the VAL mapping depends on the type of learning used by $\text{SOLVE}^{\text{MDP}}$. When $\text{SOLVE}^{\text{MDP}}$ learns by *reinforcement*, VAL is essential, as it provides the rewards which guide the resolution. For example, $\text{VAL}^{\text{uninformed}}$ leads to the notoriously hard case of sparse rewards, while $\text{VAL}^{\text{ideal}}$ (were it tractable) would lead to the trivial case where a myopic policy (greedy in its immediate reward) is optimal. Although we do not provide a generic method to design VAL, we argue that there are natural candidates, typically based on extending the objective function to partial solutions (not just finalized ones). When $\text{SOLVE}^{\text{MDP}}$ learns by *imitation* instead, the choice of VAL has a much more limited impact: it only serves to define the allowed actions. The critical factor in that case is the construction of the training dataset of expert trajectories to imitate.

Example on TSP Consider the widespread CO problem known as the Traveling Salesman Problem (TSP) in a Euclidian space V. A TSP solution (in $\mathcal{X}$) is a path, i.e. a finite sequence of pairwise distinct nodes taken from a given infinite set of abstract nodes. A TSP instance (in $\mathcal{F}$) is given by a labeling of a finite set D of nodes by points in V, inducing a metric on the nodes of D, and maps any solution (path) to the length of that path (closed at its ends) if it visits exactly all the nodes of D, and ∞ otherwise (infeasible solutions). A simple specification $\langle \mathcal{T}, \text{SOL}, \text{VAL} \rangle$ for the TSP is given by: the step space $\mathcal{T}$ is the set of nodes; for a given instance f and sequence $t_{1:n}$ of steps,

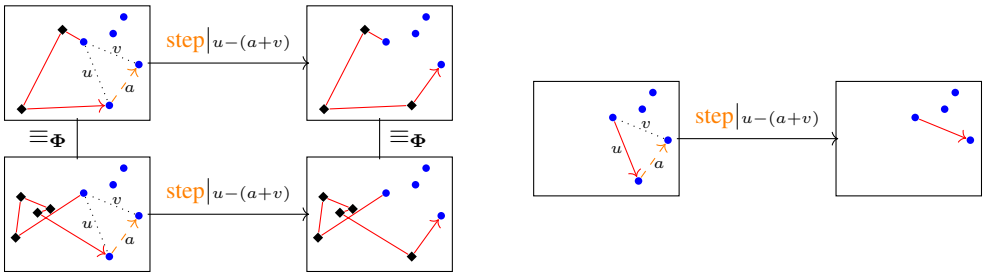


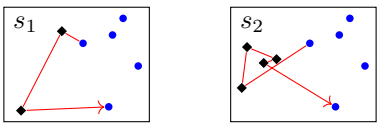
Figure 1: An example of bisimulation commutation in TSP-MDP, and the corresponding path-TSP-MDP transition. The step is the same in all three transitions: it is the end node of the dashed arrow. And the reward is also the same: it depends only on the distances a, u, v, and not on any of the previously visited nodes.

$\text{SOL}(f, t_{1:n})$ is either the sequence $t_{1:n}$ if it forms a path which visits exactly all the nodes of f, or $\perp$ otherwise; and $\text{VAL}(f, t_{1:n})$ is either the length of path $t_{1:n}$ (closed at its ends) if it forms a path which visits only nodes of f (maybe not all), or ∞ otherwise. It is easy to show that we thus obtain a specification (as defined by the axioms above) of the TSP. In TSP-MDP, the naive MDP obtained from it, the reward of taking action u (a node) at state $(f, t_{1:n})$ is $\delta_f(t_n, t_1) - (\delta_f(t_n, u) + \delta_f(u, t_1))$ where δ_f is the node distance measured on the corresponding points of V in f, conditioned on $t_{1:n}u$ being pairwise distinct nodes of f. Observe that when allowed, the reward depends only on the end nodes t_1, t_n of the step sequence.

3 Bisimulation Quotienting for Combinatorial Optimization

In our context of deterministic CO problems and therefore deterministic MDPs, the general notion of *bisimilarity* is simplified (Givan et al., 2003): two states are said to be bisimilar if they spawn exactly the same action-reward sequences. Likewise, the notion of a binary relation $\mathcal{R}$ on states being a *bisimulation* reduces to a commutation between the (deterministic) transitions of the MDP and that relation: if $s_1 \mathcal{R} s_2$ and action a applied to state s_1 leads to state s_1' with reward r, then action a applied to state s_2 leads to a state s_2' with the same reward r, and $s_1' \mathcal{R} s_2'$. An illustration is given in Fig 1. Bisimilarity is equivalently defined as the largest bisimulation (the union of all bisimulations).

Bisimilarity-induced symmetries In the naive MDP obtained from a specification of a given CO problem, a state is a partial solution and carries the whole information about the "past" decisions (steps) leading to it, which may not all be useful for the "future" decisions, i.e. the completion of that partial solution. Consider for example the following two states in TSP-MDP, in which the sequence of steps of the partial solution is represented as a directed path in red among some of the problem instance nodes:



Observe that s_1, s_2 differ only in the inner nodes of the red path (black diamond-shaped nodes). Now, it is easy to see that the successful completions of these two partial solutions are identical: they each consist of a path visiting the (same) unvisited (blue) nodes, starting at the end node of the red path and ending at its start node, with the same rewards defined by VAL. Consequently, in the MDP, the two states s_1, s_2 spawn exactly the same action-reward sequences and form a *bisimilar* pair. This is the kind of deep symmetries of the problem which we want the MDP to leverage. Of course, there exist other kinds of symmetries, e.g. rotational symmetries: if s_2 is obtained from s_1 by applying an isometric transformation to all the points in the problem instance, then s_1, s_2 also form a bisimilar pair. However, the latter symmetry is specific to the Euclidian version of the TSP. We focus here on the former kind of symmetry as it is more general. Although it has previously been

noted for routing problems (Peng et al., 2020; Xin et al., 2021b), we show here that it is an inherent characteristic of constructive CO approaches.

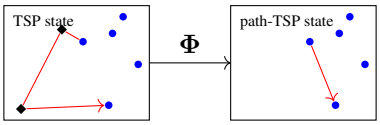
Bisimulation quotienting A classical result on MDPs (Givan et al., 2003) states that *all* such symmetries in *any* MDP can be leveraged by *quotienting* it by its bisimilarity relation, i.e. the set of all bisimilar pairs. Of course, there is no free lunch: constructing the bisimilarity of an MDP is in general intractable. Still, the result remains valuable because it holds for any bisimulation, not just the bisimilarity. Therefore one can control the amount of symmetries captured in the quotienting by carefully choosing the bisimulation, trading off its closeness to full bisimilarity for tractability.

We now assume that, for a given CO problem $(\mathcal{F}, \mathcal{X})$ we have access not only to a specification $\langle \mathcal{T}, \text{SOL}, \text{VAL} \rangle$ with its associated naive MDP, but also to a mapping $\mathbf{\Phi}{:}\mathcal{F}\times\mathcal{T}^*\to\hat{\mathcal{S}}$ from partial solutions to some new space $\hat{\mathcal{S}}$. Typically, $\mathbf{\Phi}(f, t_{1:n})$ should capture, within the partial solution $(f, t_{1:n})$, a piece of information as small as possible but sufficient to determine the set of action-reward sequences it spawns in the MDP – in other words, a summary of its "past" which is sufficient to determine its "future". We can then define an equivalence relation where two partial solutions are equivalent if they have the same image by $\mathbf{\Phi}$. For it to be a bisimulation, $\mathbf{\Phi}$ must satisfy:

$$\forall s_1, s_2 \in \mathcal{F}\times\mathcal{T}^*, \quad \mathbf{\Phi}(s_1)=\mathbf{\Phi}(s_2) \Rightarrow \begin{cases} \forall u \in \mathcal{T}, \ \mathbf{\Phi}(s_1 u)=\mathbf{\Phi}(s_2 u) \\ \text{and } \text{VAL}(s_1)-\text{VAL}(s_1 u)=\text{VAL}(s_2)-\text{VAL}(s_2 u). \end{cases} \tag{3}$$

Under that assumption, we can construct a new MDP (the quotient of the original one by the bisimulation) which is equivalent, as far as policy optimization is concerned, to the original one, but captures more symmetries of the problem. This allows to reduce the state space and should lead to a better performance, whatever the generic MDP solver used afterwards. Furthermore, by construction, the equivalence classes are in one-to-one correspondence with the states in $\hat{\mathcal{S}}$, so that the new MDP can be formulated on that space directly.

Application to the TSP and CVRP Let $\mathbf{\Phi}$ be the mappings from TSP-MDP states (TSP states for short) into new objects called "path-TSP" states, informally described by the following diagram:



The inner nodes (black diamonds) on the red path of visited nodes in the TSP state are removed, leaving only the two ends of the red path which constitute two distinguished nodes in the path-TSP state. Mapping $\mathbf{\Phi}$ has been designed to satisfy equation 3, so it induces a bisimulation on TSP-MDP (see Figure 1), and TSP-MDP can be turned into an equivalent "path-TSP-MDP" on path-TSP states. This path-TSP-MDP can be viewed as solving a variant of the TSP known as path-TSP, hence its name. However it is *not* the naive MDP for that variant since it forgets as it progresses, while naive MDPs always accumulate.

With CVRP, we define a step as the pair of a node and a binary flag specifying whether that node is reached via the depot or directly. We can define a mapping $\mathbf{\Phi}$ similarly to the TSP case, except it is not sufficient to summarize the "past" (the visited nodes) by just the two ends of their path: to guarantee equation 3 and the bisimulation property, an additional piece of information must be preserved from the past, namely the remaining capacity at the end of the current path. More details are available in Appendix A.

4 NEURAL ARCHITECTURE FOR PATH-TSP

Model We now describe our proposed policy network for the path-TSP-MDP above. The model for path-CVRP differs only slightly and is presented in Appendix A. A visual animation of its use in test is also available as Supp. Mat. Most neural models for TSP utilize an encoder-decoder architecture, in which the encoder computes a representation of the entire graph, and the decoder constructs a solution by taking into consideration the representation of the whole graph and the partial solution, e.g. Attention Model Kool et al. (2019), or PointerNetworks (Vinyals et al., 2015).

In our case, the path-TSP formulation allows us to forget the nodes in the graph that have already been visited, except the distinguished origin and destination nodes. As a corollary, it also requires re-encoding the remaining nodes at each decoding step – hence removing the need for a separate decoder. To encode a path-TSP state, we use a Transformer model (Vaswani et al., 2017). Each node is represented by its (x, y) coordinates, so that the input feature matrix for an N-node state is an $N \times 2$ matrix. We embed these features via a linear layer. The remainder of the encoder is based on Vaswani et al. (2017) with the following differences. First, we do not use positional encoding since the input nodes have no order. Instead, we learn two special embedding vectors for the two distinguished nodes (origin and destination), that are added to the feature embeddings at the relevant positions. Second, we use ReZero (Bachlechner et al., 2021) normalization, which leads to more stable training and better performance in our experiments (see ablation study in Appendix C). Finally, a last linear layer projects the encoder's output into a vector of size N, from which unfeasible actions, corresponding to the origin and destination nodes, are masked out, before applying a softmax operator so as to interpret the scalar node values for all allowed nodes as action probabilities.

Training We train our model by imitation of expert trajectories, using a plain cross-entropy loss (behaviour cloning). Such trajectories are extracted from pre-computed optimal (or near optimal) solutions for instances of a (relatively small) fixed size. Note that (optimal) solutions are not directly in the form of trajectories. Equation 2a guarantees that a trajectory exists for any solution, but it is usually far from unique. Besides, optimal solutions are costly, so we seek to make the most out of each of them. In the TSP case, we observe that given an optimal solution (tour) to an instance, any sub-path of that tour is also an optimal solution to the associated path-TSP sub-problem, hence amenable to our path-TSP model. We therefore form minibatches by first sampling a number n between 4 and N (path-TSP problems of size lower than 4 are trivial), then sampling sub-paths of length n – the same n for all the minibatch entries so as to simplify batching – from the initial solution set. In the CVRP case, the procedure is similar, except that, first, the extracted sub-paths must end at the depot, and, second, they can follow the sub-tours of the full solution in any order. We observed experimentally that the way that order is sampled has an impact on the performance (see Appendix D).

Complexity Note that our model has $\mathcal{O}(N^3)$ complexity where N is the problem instance size, which is the same as with encoder-decoder architectures (Bresson & Laurent, 2021). Note, however, that the way computation is allocated to different time steps is drastically different. Standard encoder-decoder models spend $\mathcal{O}(t^2)$ operations (in the decoder) when the size of the remaining graph is $N-t$. In other words, they spend *more* computation time as the remaining problem gets easier. In contrast, our model spends $\mathcal{O}(t^2)$ operations (in the encoder) when the remaining graph size is t. In other words, it spends *less* computation time as the remaining problem gets easier. We believe that this is a useful inductive bias, which enables better generalization in particular for larger problem sizes. This hypothesis is supported by the fact that replacing the self-attention component with a linear time alternative (i.e., spending $\mathcal{O}(t)$ operations per step) drastically degrades the generalization ability to larger instances, as we show in Appendix C.

Summary By reformulating TSP-MDP into path-TSP-MDP, the state is made to contain only a very concise summary of the "past" of a partial solution (how it was formed) as two distinguished nodes, but sufficient to determine its "future" (how it can be completed). Furthermore, at train time, we sample optimal solutions and associated path-TSP states amongst all the possible infixes of solutions of full problems. These proposed modifications go hand-in-hand. Thanks to the transformation to path-TSP-MDP, our model enables better generalization in two important ways: 1) Due to step-wise re-encoding at each step, the encoder produces a graph representation that is specific to the current path-TSP-MDP state. Graphs in these states vary in size and distribution, implicitly encouraging the model to work well across sizes and node distributions, and generalize better than if such variations were not seen during the training. In this regard, our model is similar to the SW-AM model (Xin et al., 2021b), except that our model does not have a decoder, making it simpler. 2) By sampling subsequences from our training instances, we automatically get an augmented dataset, which some previous models had to explicitly design their model for (Kwon et al., 2021).

5 RELATED WORK

NCO approaches Many NCO approaches construct solutions sequentially, via auto-regressive models. Starting with the seminal work by Vinyals et al. (2015), which proposed the Pointer network that was based on RNNs and trained in a supervised way, Bello et al. (2017) trained the same model by RL for the TSP and Nazari et al. (2018) adapted it for the CVRP. Kool et al. (2019) introduced an attention-based encoder-decoder architecture (AM) trained with RL to solve several variants of routing problems – which is reused by Kwon et al. (2021) along with a few extensions (POMO). Bresson & Laurent (2021) use a similar architecture with a different decoder on TSP problems. Another line of works is concerned with directly producing a heat-map of solution segments: Nowak et al. (2018) trained a Graph Neural Network in a supervised manner to output an adjacency matrix, which is converted into a feasible solution using beam search. Joshi et al. (2019) followed a similar framework and trained a deep Graph Convolutional Network instead, that was used by (Fu et al., 2020) as well.

Step-wise methods Peng et al. (2020) first pointed out the limitation of the original AM (Kool et al., 2019) approach in representing the dynamic nature of routing problems. They proposed to update the encoding after each subtour completion for the CVRP. Xin et al. (2021b) proposed a similar step-wise strategy but the encodings recomputed after each decision. In practice, their architecture is the most similar to ours for the TSP. However, thanks to our principled MDP transformations based on bisimulation quotienting, we obtain a superior representation for CVRP: In contrast to our approach, their CVRP architecture only provides censored information by omitting the remaining vehicle capacity and simply restricting the state to the nodes whose demand is below the remaining capacity. Xin et al. (2020) extended on this idea by proposing the Multi-Decoder Attention Model (MDAM) that in particular contains a special layer to efficiently approximate the re-embedding process. As MDAM constitutes the most advanced version, we employ it as a baseline in our experiments.

Generalizable NCO Generalization to different instances distributions, and esp. larger instances, is regarded as one of the major limitations of current NCO approaches (Joshi et al., 2022; Mazyavkina et al., 2020). Fu et al. (2020) trained a Graph Convolution model in a supervised manner on small graphs and used it to solve large TSP instances, by applying the model on sampled subgraphs and using an expensive MCTS search to improve the final solution (Att-GCN+MCTS). While this method achieves excellent generalization on TSP instances, MCTS requires a lot of computing resources and is essentially a post-learning search strategy. Geisler et al. (2022) investigate the robustness of NCO solvers through adversarial attacks and find that existing neural solvers are highly non-robust to out-of-distribution examples. They conclude that one way to address this issue is through adversarial training. In particular, Xin et al. (2021a) trains a GAN to generate instances that are difficult to solve for the current model. Manchanda et al. (2022) take a different approach and leverage meta-learning to learn a model in such a way that it is easily adaptable to new distributions. Accounting for symmetries in a given CO problem is a powerful idea to boost the generalization performance of neural solvers. Both Kwon et al. (2021) and Kim et al. (2022) make use of solution symmetricity as part of their loss function during training. Problem instance symmetry can be used at training time to augment the dataset (Kwon et al., 2021) or enforce robust representations (Kim et al., 2022), or it can be used at inference time to augment the set of solutions (Kwon et al., 2021).

Please note that all of the above are orthogonal to our approach: rather than augmenting data or changing the training paradigm, our approach simplifies the state space by transforming the MDP, which has beneficial effects irrespective of the method of training.

6 EXPERIMENTS

To verify the effectiveness of our method, we train our model and all baselines on synthetic TSP and CVRP instances of size 100, generated using the procedures described in Joshi et al. (2019) and Nazari et al. (2018). We choose graphs of size 100 because it is the largest size for which (near) optimal solutions are still reasonably fast to obtain, and such training datasets are commonly used in the literature. Then, we evaluate trained models on synthetic instances of size 100, 200, 500 and 1K generated from the same distribution, as well as the standard TSPLib and CVRPLib datasets.

Hyperparameters and training procedure We use the same hyperparameters for both problems. The model has 9 layers, each built with 8 attention heads with embedding size of 128 and dimension of feed-forward layer 512. Our model is trained on 1 million instances with 100 nodes split into batches of size 1024, for 1000 epochs. Solutions of these problems are obtained by using the Concorde solver (Applegate et al., 2015) for TSP and LKH heuristic (Helsgaun, 2017) for CVRP. We use Adam (Kingma & Ba, 2017) as optimizer with an initial learning rate of $7.5\mathrm{e}{-4}$ and decay of 0.98 every 20 epochs.

Evaluation We compare our model with existing state-of-the-art methods: OR-Tools (Perron & Furnon, 2022), LKH (Helsgaun, 2017), and Hybrid Genetic Search for the CVRP (HGS) (Vidal, 2022) as traditional non-neural methods; Att-GCN+MCTS and NeuralRewriter (Chen & Tian, 2019) as hybrid methods for TSP and CVRP respectively; and the following deep learning-based constructive methods: AM, MDAM and POMO, which were discussed in Section 5. For all deep learning baselines we use the model *trained on graphs of size 100* and the best decoding strategy. Following the same procedure as in Fu et al. (2020), we generate four test datasets with graphs of sizes 100, 200, 500 and 1000. For CVRP, we use capacities of 50, 80, 100 and 250, respectively. In addition, we report the results on TSPLib instances with up to 4461 nodes and all CVRPLib instances with node coordinates in the Euclidian space. For all models, we report the optimality gap and the inference time. The optimality gap for TSP is based on the optimal solutions obtained with Concorde. For CVRP, although HGS gives better results than LKH, we use the LKH solution as a reference to compute the "optimality" gap, in order to be consistent (and easily comparable) with previous works. While the optimality gap is easy to compute and compare, measurements of running times are much harder: they may vary due to the implementation platforms (Python, C++), hardware (GPU, CPU), parallelization, batch size, etc. Therefore, we also report the number of solutions generated by each of the constructive deep learning models. In our experiments, we run all deep learning models on a single Nvidia Tesla V100-S GPU with 24GB of memory, and other solvers on Intel(R) Xeon(R) CPU E5-2670 with 256GB memory, in one single thread.

Results Tables 1a and 1b summarize our results on TSP and CVRP, respectively. For both problems, our model shows superior generalization on larger graphs, even with the greedy decoding strategy, which generates only a single solution while all others generate several hundreds (and select the best among them). In terms of running time with greedy decoding, our model is competitive with the POMO baseline, and significantly faster than other models. Beam search decoding with beam size 16 further improves the quality of solutions, but as expected, it takes approximately 16 times longer. Figure 2 shows optimality gap versus running time for our model and other baseline models. Our model clearly outperforms other models in terms of generalization to larger instances. The only model that is competitive with ours is Att-GCN+MCTS, but it is 2-15 times slower and is designed for TSP only. In addition to synthetic datasets, we test our model on TSPLib and VRPLib instances, which are of varying graph sizes, node distributions, demand distributions and vehicle capacities. Table 1c shows our model's strength over MDAM and POMO, even with the greedy decoding strategy. The effectiveness of our MDP transformation method and the resulting neural architecture is confirmed by the results. Thanks to our more principled approach that leads to better state representations and a simpler architecture without a decoder, by generating a single solution, it is able to outperform MDAM (with 250 solutions), which is closest to our model conceptually. Moreover, an ablation study in Appendix C suggests that spending appropriate amounts of compute for each subproblem is a crucial factor in our model.

7 CONCLUSION

We have presented a flexible framework to derive MDPs that sequentially construct solutions to CO problems. Starting from a naive MDP, we introduced a generic transformation using bisimulation quotienting, which reduces the state space by leveraging its symmetries. We applied this transformation on the TSP and CVRP, for which we also designed a simple attention-based model, well-suited to the transformed state representation. We show experimentally that this combination of state representation, simple model, and training procedure yields state-of-the-art generalization results on diverse benchmarks. While training on relatively small instances allowed us to use imitation learning, our approach and model could be similarly used with reinforcement learning. Finally, we have focused on deterministic CO problems, leaving the adaptation of our framework to stochastic problems as future work.

Table 1: Summary of the experimental results. The bold values represent the best optimality gap (lower is better) and fastest inference time. The underlined cells represent the best ratio between the quality of the solution and the inference time.

	#s	TSP100		TSP200		TSP500		TSP1000	
Concorde	-	0.000%	38m	0.000%	2m	0.000%	40m	0.000%	2.5h
OR-Tools	-	3.765%	1.1h	4.516%	4m	4.891%	31m	5.021%	2.4h
Att-GCN+MCTS*	-	0.037%	15m	0.884%	2m	2.536%	6m	3.223%	13m
AM bs1024	1024	2.510%	20m	6.176%	1m	17.978%	8m	29.750%	31m
MDAM bs50	250	0.395%	45m	2.044%	3m	9.878%	13m	19.965%	1.1h
POMO augx8	$8N$	0.134%	**1m**	1.572%	**5s**	20.182%	**1m**	40.603%	10m
BQ (ours) greedy	1	0.540%	**1m**	0.793%	**5s**	1.425%	**1m**	2.335%	**7m**
BQ (ours) bs16	16	**0.032%**	18m	**0.166%**	1m	**0.682%**	15m	**1.311%**	1.8h

(a) Experimental results on TSP. #s refers to number of solutions generated. *We could not run Att-GCN+MCTS code on our architecture so we report results from the original paper.

	#s	CVRP100		CVRP200		CVRP500		CVRP1000	
LKH	-	0.000%	15.3h	0.000%	30m	0.000%	1.3h	0.000%	2.8h
HGS	-	-0.510%	15.3h	-1.024%	30m	-1.252%	1.3h	-1.104%	2.8h
OR-Tools	-	9.617%	15.3h	10.700%	30m	11.403%	1.3h	13.559%	2.8h
NeuRewriter *	-	3.456%	1.1h	29.460%	9m	25.051%	32m	29.542%	1.8h
AM bs1024	1024	4.180%	24m	7.786%	1m	16.964%	8m	86.410%	31m
MDAM bs50	250	2.206%	56m	4.332%	3m	9.994%	14m	28.015%	1.4h
POMO augx8	$8N$	**0.689%**	**1m**	4.767%	**5s**	20.575%	**1m**	141.058%	10m
BQ (ours) greedy	1	4.832%	**1m**	3.723%	**5s**	3.429%	**1m**	6.809%	**7m**
BQ (ours) bs16	16	1.798%	18m	**1.375%**	1m	**0.817%**	15m	**2.048%**	1.8h

(b) Experimental results on CVRP. *We could not reproduce the reported results for NeuRewriter, so for CVRP100 we report results from the original paper and for other sizes we report the best result we got.

Size	MDAM bs50	POMO x8	BQ (ours) greedy	BQ (ours) bs16
<100	3.06%	0.42%	0.38%	**0.06%**
100-200	5.14%	2.31%	2.82%	**1.61%**
200-500	11.32%	13.32%	3.31%	**2.07%**
500-1K	20.40%	31.58%	10.08%	**3.04%**
>1K	40.81%	62.61%	11.87%	**8.61%**
All	19.01%	26.30%	6.22%	**3.94%**

Set (size)	MDAM bs50	POMO augx8	BQ (ours) greedy	BQ (ours) bs16
A (32-80)	6.17%	4.86%	5.85%	**1.96%**
B (30-77)	8.77%	5.13%	7.04%	**3.50%**
F (44-134)	16.96%	15.49%	7.20%	**3.04%**
M (100-200)	5.92%	4.99%	6.69%	**1.85%**
P (15-100)	8.44%	14.69%	4.71%	**1.32%**
X (100-1K)	34.17%	21.62%	10.74%	**8.35%**
All (15-1K)	22.36%	15.58%	8.58%	**5.60%**

(c) Experimental results on TSPLib (left) and CVRPLib (right).

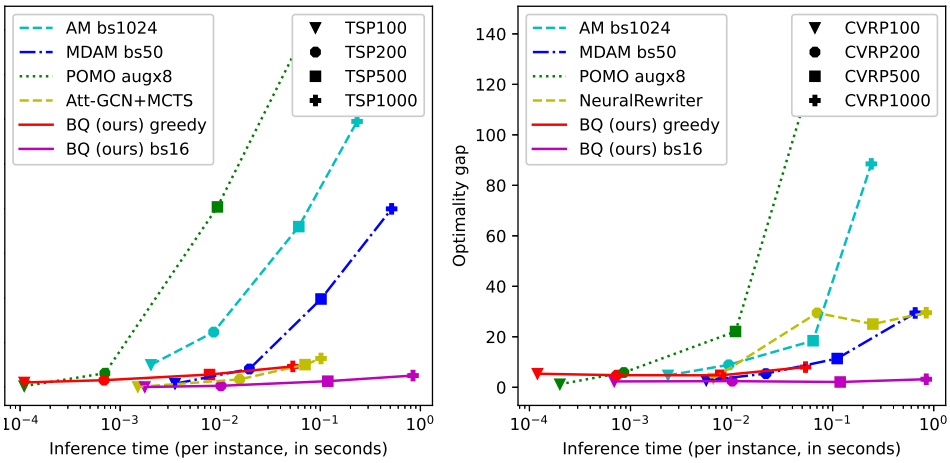


Figure 2: Generalization results on different graph sizes for TSP (left) and CVRP (right). Lower and further left is better.

REPRODUCIBILITY STATEMENT

In order to ensure the reproducibility of our approach, we have:

- described precisely our generic theoretical framework (Section 2) and provided a detailed proof of Proposition 1 in Appendix E. This should in particular serve to adapt the framework to other CO problems;
- explained in detail our proposed model (Section 4 for TSP and Appendix A for CVRP), described precisely the training procedure and listed the hyperparameters (Section 6);
- used public datasets referenced in Section 6.

Furthermore, we plan to make our code public upon acceptance.

REFERENCES

David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver. University of Waterloo, 2015.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization, July 2016.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. ReZero is all you need: Fast convergence at large depth. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 1352–1361. PMLR, December 2021.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. *arXiv:1611.09940 [cs, stat]*, January 2017.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021.

Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, July 2013. ISSN 0020-0255. doi: 10.1016/j.ins.2013.02.041.

Xavier Bresson and Thomas Laurent. The Transformer Network for the Traveling Salesman Problem. *arXiv:2103.03012 [cs]*, March 2021.

Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv:2102.09544 [cs, math, stat]*, February 2021.

Xinyun Chen and Yuandong Tian. Learning to Perform Local Rewriting for Combinatorial Optimization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. Wiley-Interscience, New York, 1st edition edition, November 1997. ISBN 978-0-471-55894-1.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a Small Pre-trained Model to Arbitrarily Large TSP Instances. *arXiv:2012.10658 [cs]*, December 2020.

Simon Geisler, Johanna Sommer, Jan Schuchardt, Aleksandar Bojchevski, and Stephan Günnemann. Generalization of Neural Combinatorial Solvers Through the Lens of Adversarial Robustness, March 2022.

Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1):163–223, July 2003. ISSN 0004-3702. doi: 10.1016/S0004-3702(02)00376-4.

Keld Helsgaun. An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems. Technical report, Roskilde University, Roskilde, Denmark, 2017.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 448–456. PMLR, June 2015.

Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem. *arXiv:1906.01227 [cs, stat]*, June 2019.

Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27(1-2):70–98, April 2022. ISSN 1383-7133. doi: 10.17863/CAM.85173.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning Combinatorial Optimization Algorithms over Graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6348–6358. Curran Associates, Inc., 2017.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-NCO: Leveraging Symmetricity for Neural Combinatorial Optimization, May 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! In *International Conference on Learning Representations*, 2019.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. *arXiv:2010.16011 [cs]*, July 2021.

Florian Mai, Arnaud Pannatier, Fabio Fehr, Haolin Chen, Francois Marelli, Francois Fleuret, and James Henderson. HyperMixer: An MLP-based Green AI Alternative to Transformers, March 2022.

Sahil Manchanda, Sofia Michel, Darko Drakulic, and Jean-Marc Andreoli. On the Generalization of Neural Combinatorial Optimization Heuristics. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, Grenoble, France, September 2022.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement Learning for Combinatorial Optimization: A Survey. *arXiv:2003.03600 [cs, math, stat]*, March 2020.

MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. Reinforcement Learning for Solving the Vehicle Routing Problem. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Alex Nowak, David Folqué, and Joan Bruna. Divide and Conquer Networks. In *6-Th International Conference on Learning Representations*, Vancouver, Canada, April 2018.

Bo Peng, Jiahai Wang, and Zizhen Zhang. A Deep Reinforcement Learning Algorithm Using Dynamic Attention Model for Vehicle Routing Problems. *arXiv:2002.03282 [cs, stat]*, February 2020.

Laurent Perron and Vincent Furnon. OR-tools. Google, 2022.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient Transformers: A Survey. *ACM Computing Surveys*, April 2022. ISSN 0360-0300. doi: 10.1145/3530811.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, June 2017.

Thibaut Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140:105643, April 2022. ISSN 0305-0548. doi: 10.1016/j.cor.2021.105643.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 2692–2700. Curran Associates, Inc., 2015.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-Decoder Attention Model with Embedding Glimpse for Solving Vehicle Routing Problems, December 2020.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Generative Adversarial Training for Neural Combinatorial Optimization Models, September 2021a.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Step-Wise Deep Learning Models for Solving Routing Problems. *IEEE Transactions on Industrial Informatics*, 17(7):4861–4871, July 2021b. ISSN 1941-0050. doi: 10.1109/TII.2020.3031409.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. *arXiv:2010.12367 [cs, stat]*, October 2020.

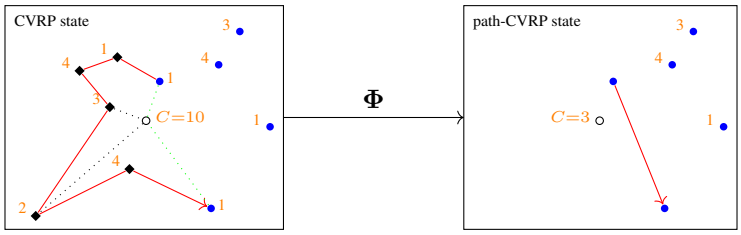
A DETAILS ON CVRP

Problem definition and specification The Capacitated Vehicle Routing Problem (CVRP) is a vehicle routing problem in which a vehicle (here, a single one) with limited capacity must deliver items from a depot location to various customer locations. Each customer has an associated *demand*, which represents an amount of items, and the problem is for the vehicle to provide all the customers in the least travel distance, returning as many times as needed to the depot to refill, but without ever exceeding the vehicle capacity.

Formally, we assume given an infinite set of abstract nodes, each with a demand (positive scalar), plus a distinguished depot node. A CVRP solution (in $\mathcal{X}$) is a demand compatible path ("d-path" for short), i.e. a finite sequence of nodes starting at the depot, which are pairwise distinct except for the depot, and respecting the capacity constraint: the total demand of any contiguous sub-sequence of non depot nodes is below the vehicle capacity. A CVRP instance (in $\mathcal{F}$) is given by a labelling of a finite set D of nodes, including the depot, by points of the Euclidian space V, and maps any solution (d-path) to the length of that path, closed at its end and measured by taking the distances on the corresponding points in V, if the path visits exactly all the nodes of D, or ∞ otherwise (unfeasible solutions).

A possible specification $\langle \mathcal{T}, \text{SOL}, \text{VAL} \rangle$ for the CVRP is defined as follows. The step space $\mathcal{T}$ is the set of pairs of a non depot node and a binary flag indicating whether that node is to be reached via the depot or directly. The extension $\bar{t}$ of a step t is either the singleton of its node component if its flag is 0 or the pair of the depot node and its node component if its flag is 1. For a given problem instance f and sequence $t_{1:n}$ of steps, $\text{SOL}(f, t_{1:n})$ is either the sequence $\bar{t}_{1:n}$ if it forms a d-path which visits exactly all the nodes of f , or $\perp$ otherwise. And $\text{VAL}(f, t_{1:n})$ is either the total length of $\bar{t}_{1:n}$ (closed at its end) if it forms a d-path which visits only nodes of f (maybe not all), or ∞ otherwise. It is easy to show that $\langle \mathcal{T}, \text{SOL}, \text{VAL} \rangle$ forms a specification for the CVRP (i.e. satisfies the axioms of specifications introduced in Section 2). The naive MDP obtained from it is denoted CVRP-MDP.

Bisimulation quotienting Just as with TSP, we can define a mapping $\mathbf{\Phi}$ from CVRP-MDP states to a new "path-CVRP" state space, informally described by the following diagram.



Here, the capacity of the vehicle is $C{=}10$, shown next to the (colourless) depot node, and the demand of each node is shown next to it, in orange. The black dotted line indicates that the action which introduced the node with demand 2 was via the depot: its flag was set to 1 (all the other actions had their flag set to 0 in this simple example). The green dotted line indicates how the path is closed to measure its length. After the node with demand 2, the path of visited nodes (in red) continues with nodes with demand 4 and 1, respectively, so that the remaining capacity at the end of the path is $C{-}(2{+}4{+}1){=}3$. Compared to TSP, this is the additional piece of information in the summary of the "past" (path of visited nodes) which is preserved in the path-CVRP state, together with the origin and destination of the path. Mapping $\mathbf{\Phi}$ thus defined satisfies Equation 3, hence induces a bisimulation on CVRP-MDP states, and by quotienting, one obtains an MDP which can be defined directly on path-CVRP states.

Model architecture for CVRP The model architecture for CVRP is almost the same as for TSP, with a slight difference in the input sequence and in the output layer. In the TSP model, the input to the node embedding layer for a N-node state is a $2{\times}N$ matrix of coordinates. For CVRP, we use two additional channels: one for node demands, and one for the current vehicle capacity, repeated across all nodes. The demand is set to zero for the origin and destination nodes. We obtain a $4{\times}N$ matrix of features, which is passed through a learned embedding layer. As for TSP, a learned origin-signalling

		Greedy		Beam size 16		Beam size 64	
TSP200	Full graph	**0.79**%	5s	**0.17**%	1m	**0.08**%	5m
	100KNNs	1.31%	3s	0.23%	33s	0.10%	3m
TSP500	Full graph	1.71%	1m	0.68%	15m	0.54%	1h
	100KNNs	2.58%	18s	0.92%	3m	0.69%	12m
	250KNNs	**1.56**%	32s	**0.67**%	9m	**0.53**%	30m
TSP1000	Full graph	**2.34**%	7m	**1.31**%	1.8h	**1.19**%	7.3h
	100KNNs	3.34%	25s	1.69%	6m	1.45%	24m
	250KNNs	2.53%	1m	1.43%	23m	**1.19**%	1.4h
CVRP200	Full graph	4.80%	5s	2.42%	1m	1.82%	5m
	100KNNs	5.18%	3s	2.12%	33s	1.68%	3m
CVRP500	Full graph	4.74%	1m	2.10%	15m	1.59%	1h
	100KNNs	5.14%	18s	2.02%	3m	1.74%	12m
	250KNNs	**4.58**%	32s	**1.86**%	9m	**1.14**%	30m
CVRP1000	Full graph	8.00%	7m	3.19%	1.8h	2.39%	7.3h
	100KNNs	8.25%	25s	4.76%	6m	3.58%	24m
	250KNNs	**7.51**%	1m	**3.08**%	23m	**2.28**%	1.4h

Table 2: Improving the model performance using a k-nearest-neighbor heuristic.

(resp. destination-signalling) vector is added to the corresponding embeddings. The rest of the architecture, in the form of attention layers, is identical to TSP, until after the action scores projection layer. In the case of TSP, the projection layer returns a vector of N scores, where each score, after a softmax, represents the probability of choosing the node as the next step in the construction. In the case of CVRP, the model returns a matrix of scores of dimension $N \times 2$, corresponding to each possible actions (node-flag pair) and the softmax scopes over this whole matrix. As usual, a mask is always applied to unfeasible actions before the softmax operator: those which have higher demand than the remaining vehicle capacity, as well as the origin and destination nodes.

B IMPROVING THE MODEL PERFORMANCE WITH A K-NEAREST-NEIGHBOR HEURISTIC

Our decoding strategy could be further improved by using a k-nearest-neighbor heuristic to restrict the search space and reduce the inference time. For both greedy and beam search strategies, at every step, it is possible to reduce the remaining graph by considering only a certain number of neighbouring nodes. Table 2 presents the experiments where we apply the model just on a certain number on nearest neighbours of the origin. This approach clearly reduces the execution time, but also in some cases even improves the performance in terms of optimality gap.

C ABLATION STUDY

C.1 TRANSFORMER VS HYPERMIXER AS MODEL

In Section 6 we have shown that our model has an excellent generalization ability to graphs of larger size. In Section 4, we hypothesize that this has to do with the fact that a subproblem of size t spends $\mathcal{O}(t^2)$ computation operations due to the quadratic complexity of the Transformer encoder's self-attention component, which is responsible for mixing node representations. To test this hypothesis, we experiment with replacing self-attention with an efficient mixing component (see Tay et al. (2022) for an overview), namely the recent linear-time HyperMixer (Mai et al., 2022). We chose this model because it does not assume that the input is ordered, unlike e.g. sparse attention alternatives.

Experimental Details For comparability, we set the model and training parameters to the same as for Transformers, so the experiments only differ in token mixing component that is used. The only other difference is that we used Layer Normalization Ba et al. (2016) instead of ReZero Bachlechner

Seed	TSP100	TSP200	TSP500	TSP1000
1	2.10%	8.38%	34.91%	71.30%
2	1.38%	3.54%	98.59%	628.71%
3	1.93%	4.14%	120.18%	216.77%
4	1.37%	4.54%	46.23%	104.85%
5	1.25%	3.66%	61.99%	524.43%

Table 3: Experimental results on TSP with HyperMixer for five different seeds.

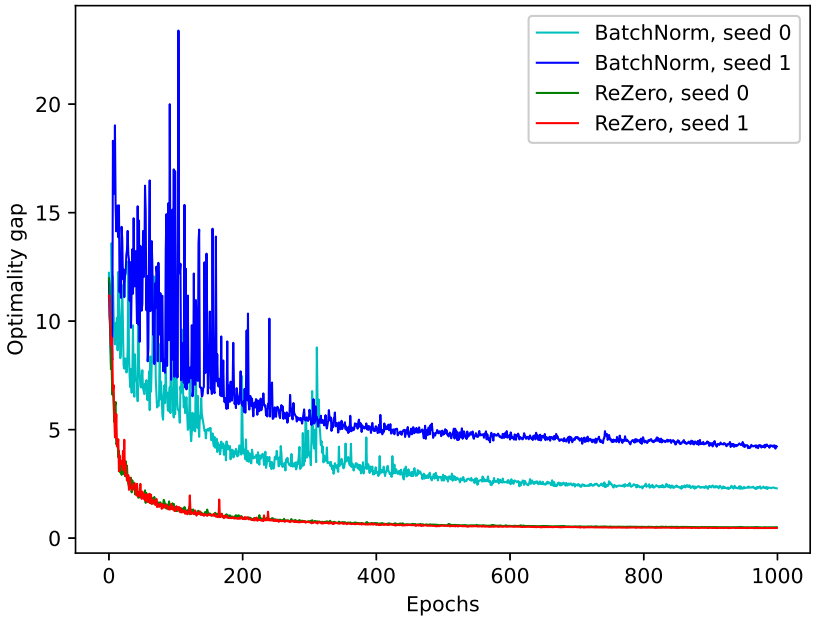


Figure 3: Training curves showing the effect of the choice of normalization layer on validation performance

et al. (2021), which leads to more stable training for HyperMixer. Since we observed relatively large sensitivity to model initialization, we are reporting the results for 5 different seeds.

Results Table 3 shows the result for HyperMixer with greedy decoding. While the model reaches lower but acceptable performance than Transformers on TSP100, it generalizes poorly to instances of larger size. Moreover, performance is very sensitive to the seed. These results suggest that the computation spent by self-attention is indeed necessary to reach the generalization ability of our model, which increases the compute with the size of the (sub)problem.

C.2 ReZero vs BatchNorm as normalization

Most NCO works that use transformer networks (Kool et al., 2019)(Kwon et al., 2021)(Xin et al., 2020) use batch normalization(Ioffe & Szegedy, 2015) rather than layer normalization (Ba et al., 2016) in attention layers. We find ReZero normalization (Bachlechner et al., 2021) to work even better. Figure 3 shows the effect of using ReZero compared to batch normalization in our Tranformer network. Using it leads to more stable training, better performance, and drastically lower variance between seeds.

D ON THE IMPACT OF EXPERT SOLUTIONS

Our datasets consist of pairs of a problem instance and a solution (tour). On the other hand, in this paper, we use imitation learning, which requires instead pairs of a problem instance and (expert) trajectory in the MDP. Now, a solution may be obtained from multiple trajectories in the MDP. For example, with TSP, a solution is a loop in a graph, and one has to decide at which node its construction started and in which direction it proceeded. With CVRP, the order in which the subtours are constructed needs also to be decided. Hence, all our datasets are pre-processed to transform solutions into corresponding construction trajectories (a choice for each or even all possible ones). We experimentally observed that this transformation has an impact on the performance. For example, with CVRP, choosing, for each solution, the construction in the order in which LKH3 *displays* it, which does not seem arbitrary, yields to 1.3 point better opt-gap performance compared to following a random ordering of the sub-tours. We hypothesize that if there is any bias in the display of the optimal solution - for example, shorter tour first, or closest node first - it requires slightly less model capacity to learn action imitation *for this display* rather than *for all possible displays*.

E PROOF OF PROPOSITION 1 (SOUNDNESS OF THE NAIVE MDP)

We show here that procedure SOLVE satisfies $\text{SOLVE}(f)=\arg\min_{x\in\mathcal{X}} f(x)$. We first show the following general lemma:

Let $\mathcal{Y}\xrightarrow{\psi}\mathcal{X}\xrightarrow{f}\mathbb{R}\cup\{\infty\}$ be arbitrary mappings, if ψ is surjective then

$$\arg\min_{x\in\mathcal{X}} f(x) = \psi(\arg\min_{y\in\mathcal{Y}} f(\psi(y)))$$

Simple application of the definition of $\arg\min$ (as a set). The subscript $_*$ denotes the steps where the assumption that ψ is a surjection is used:

$$\begin{aligned}
x' \in \psi(\arg\min_{y} f(\psi(y))) \quad &\text{iff} \quad \exists y' \in \arg\min_{y} f(\psi(y))\ x' = \psi(y') \\
&\text{iff} \quad \exists y'\ x' = \psi(y')\ \forall y\ f(\psi(y')) \leq f(\psi(y)) \\
&\text{iff} \quad \exists y'\ x' = \psi(y')\ \forall y\ f(x') \leq f(\psi(y)) \\
&\text{iff}_* \quad \forall y\ f(x') \leq f(\psi(y)) \quad \text{iff}_* \quad \forall x\ f(x') \leq f(x) \\
&\text{iff} \quad x' \in \arg\min_{x} f(x)
\end{aligned}$$

Let $(\mathcal{F},\mathcal{X})$ be a CO problem with specification $\langle\mathcal{T},\text{SOL},\text{VAL}\rangle$ and $\mathcal{M}$ the naive MDP obtained from it. For each $f\in\mathcal{F}$, let $v_f=\text{VAL}(f,\epsilon)$, $\mathcal{X}_f=\{x\in\mathcal{X}|f(x)<\infty\}$ and let $\mathcal{Y}_f$ be the set of $\mathcal{M}$-trajectories which start at (f,ϵ) and end at a stop state.

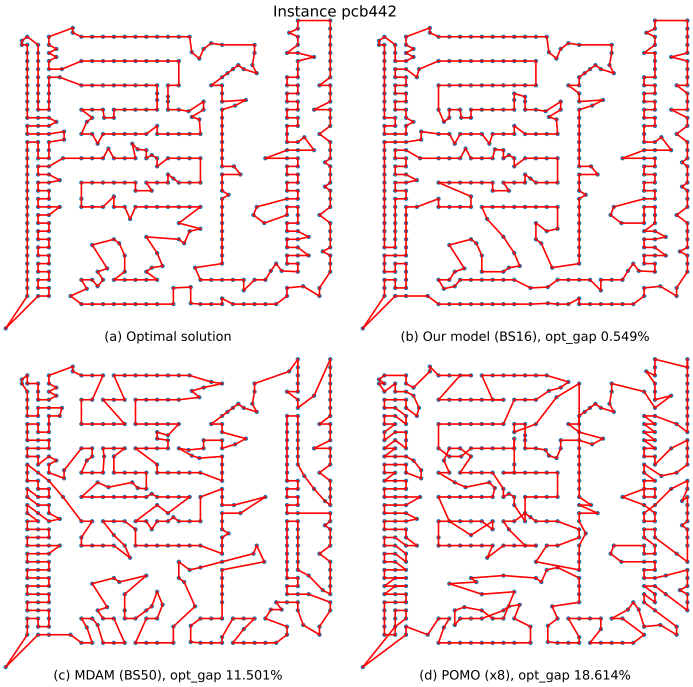
- For any $\mathcal{M}$-trajectory $\tau=s_0t_1r_1s_1\cdots t_nr_ns_n$ in $\mathcal{Y}_f$, define $\psi(\tau)\ =_{\text{def}}\ \text{SOL}(s_n)$. Since $\tau\in\mathcal{Y}_f$, we have $s_0=(f,\epsilon)$ and s_n is a stop state, i.e. $\text{SOL}(s_n)=\psi(\tau)\in\mathcal{X}$, and by Equation 2a, $f(\psi(\tau))<\infty$. Hence $\psi{:}\mathcal{Y}_f\mapsto\mathcal{X}_f$.
- By construction, $s_m=(f,t_{1:m})$ for all $m\in 1{:}n$ and each transition in τ has a finite reward $\text{VAL}(s_{m-1})-\text{VAL}(s_m)$ (condition for it to be valid). Hence the cumulated reward is given by $R(\tau)=\text{VAL}(s_0)-\text{VAL}(s_n)$. Now, $\text{VAL}(s_0)=v_f$ which is independent of τ and by Equation 2c, $\text{VAL}(s_n)=f(\psi(\tau))$. Hence $f(\psi(\tau))=v_f-R(\tau)$.
- Let's show that ψ is surjective. Let $x\in\mathcal{X}_f$. Equation 2a ensures that $x=\text{SOL}(f,t_{1:n})$ for some $t_{1:n}\in\mathcal{T}^*$. For each $m\in\{0{:}n\}$, let $s_m=(f,t_{1:m})$ and consider the sequence $\tau=s_0t_1r_1s_1\cdots t_nr_ns_n$. Now, $\text{SOL}(s_n)=x\neq\perp$ hence τ ends in a stop state and starts at (f,ϵ). By Equation 2c we have $\text{VAL}(s_n)=f(x)$, hence $\text{VAL}(s_n)<\infty$, and $\text{VAL}(s_m)<\infty$ for all $m\in\{0{:}n-1\}$. And by Equation 2b $\text{SOL}(s_m)=\perp$, hence all the transitions in τ are valid in $\mathcal{M}$. Hence $\tau\in\mathcal{Y}_f$ and by definition, $\psi(\tau)=x$.

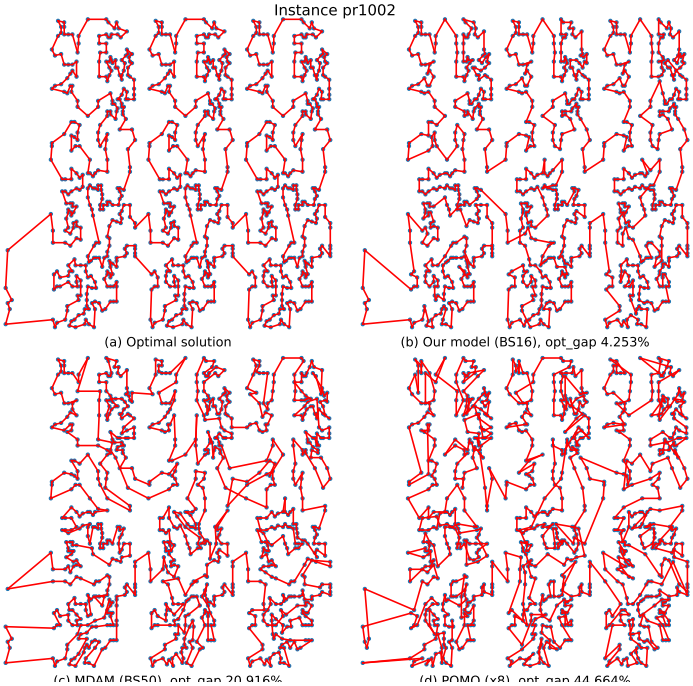
Therefore we can apply the lemma proved above:

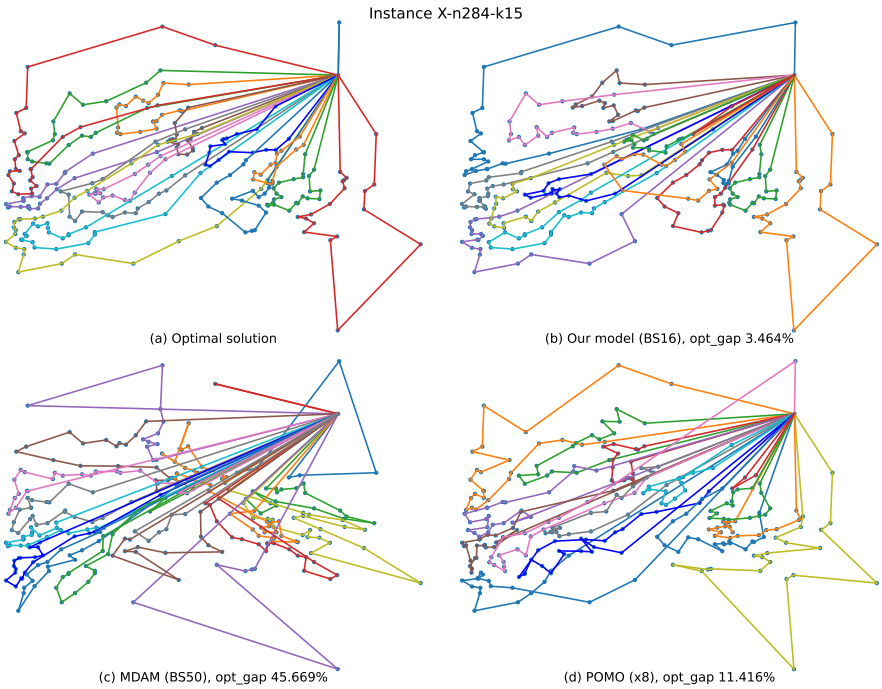
$$\begin{aligned}
\arg\min_{x\in\mathcal{X}_f} f(x) &= \psi(\arg\min_{\tau\in\mathcal{Y}_f} f(\psi(\tau))) = \psi(\arg\min_{\tau\in\mathcal{Y}_f} v_f-R(\tau)) \\
&= \psi(\arg\max_{\tau\in\mathcal{Y}_f} R(\tau)) = \psi(\text{SOLVE}^{\text{MDP}}_{\mathcal{M}}(f,\epsilon)) = \text{SOLVE}(f)
\end{aligned}$$

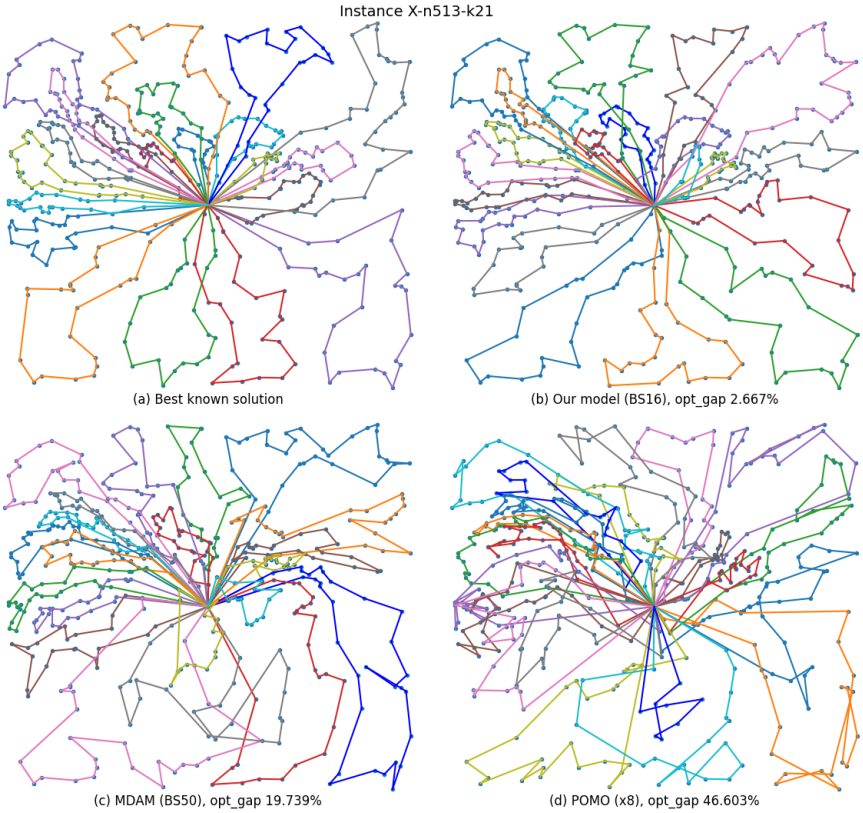
Now, obviously, $\arg\min_{x\in\mathcal{X}} f(x) = \arg\min_{x\in\mathcal{X}_f} f(x)$, since by definition f is infinite on $\mathcal{X}\backslash\mathcal{X}_f$.

F PLOTS OF SOME TSPLIB AND CVRPLIB SOLUTIONS






Instance X-n284-k15
(a) Optimal solution
(b) Our model (BS16), opt_gap 3.464%
(c) MDAM (BS50), opt_gap 45.669%
(d) POMO (x8), opt_gap 11.416%


Instance X-n513-k21
(a) Best known solution
(b) Our model (BS16), opt_gap 2.667%
(c) MDAM (BS50), opt_gap 19.739%
(d) POMO (x8), opt_gap 46.603%