POPULATION-BASED REINFORCEMENT LEARNING FOR COMBINATORIAL OPTIMIZATION PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Applying reinforcement learning (RL) to combinatorial optimization problems is attractive as it removes the need for expert knowledge or pre-solved instances. However, it is unrealistic to expect an agent to solve these (often NP-)hard problems in a single shot at inference due to their inherent complexity. Thus, leading approaches often implement additional search strategies, from stochastic sampling and beam-search to explicit fine-tuning. In this paper, we argue for the benefits of learning a population of complementary policies, which can be simultaneously rolled out at inference. To this end, we introduce Poppy, a simple theoretically grounded training procedure for populations. Instead of relying on a predefined or hand-crafted notion of diversity, Poppy induces an unsupervised specialization targeted solely at maximizing the performance of the population. We show that Poppy produces a set of complementary policies, and obtains state-of-the-art RL results on three popular NP-hard problems: the traveling salesman (TSP), the capacitated vehicle routing (CVRP), and 0-1 knapsack (KP) problems. On TSP specifically, Poppy outperforms the previous state-of-the-art, dividing the optimality gap by 5 while reducing the inference time by more than an order of magnitude.

1 INTRODUCTION

In recent years, machine learning (ML) approaches have overtaken algorithms that use handcrafted features and strategies across a variety of challenging tasks (Mnih et al., 2015; van den Oord et al., 2016; Silver et al., 2017; Brown et al., 2020). In particular, solving combinatorial optimization (CO) problems – where the maxima or minima of an objective function acting on a finite set of discrete variables is sought – has attracted significant interest (Bengio et al., 2021) due to both their (often NP) hard nature and numerous practical applications across domains varying from logistics (Sbihi & Eglese, 2007) to fundamental science (Wagner, 2020).

As the search space of feasible solutions typically grows exponentially with the problem size, exact solvers can be challenging to scale, hence CO problems are often also tackled with handcrafted heuristics using expert knowledge. Whilst a diversity of ML-based heuristics have been proposed, reinforcement learning (RL; Sutton & Barto, 2018) is a promising paradigm as it does not require pre-solved examples of these hard problems. Indeed, algorithmic improvements to RL-based CO solvers, coupled with low inference cost, and the fact that they are by design targeted at specific problem distributions, have progressively narrowed the gap with traditional solvers.

To improve the quality of proposed solutions, RL methods typically generate multiple candidates with additional search procedures, which can be divided into two families (Mazyavkina et al., 2021). First, *improvement methods* start from a feasible solution and iteratively improve it through small modifications (actions). However, such incremental search cannot quickly access very different solutions, and requires handcrafted procedures to define a sensible action space. Second, *construction methods* incrementally build a solution by selecting one element at a time. Multiple solutions can be built using sampling strategies, such as stochastic sampling policies or beam search. However, just as improvement methods are biased by the initial starting solution, construction methods are biased by the single underlying policy. Thus, a balance must be struck between the exploitation of the learned policy (which may be ill-suited for a given problem instance) and the exploration of different solutions (where the extreme case of a purely random policy will likely be highly inefficient).

In this work, we propose Poppy, a construction method that uses a *population* of agents with suitably diverse policies to improve the exploration of the solution space of hard CO problems. Whereas a single agent aims to perform well across the entire problem distribution, and thus has to make compromises, a population can learn a set of heuristics such that only one of these has to be performant on any given problem instance. However, realizing this intuition presents several challenges: (i) naïvely training a population of agents is expensive and challenging to scale, (ii) the trained population should have complementary policies that propose different solutions, and (iii) the training approach should not impose any handcrafted notion of diversity within the set of policies given the absence of clear behavioral markers aligned with performance for typical CO problems.

Poppy addresses (i) by sharing a large fraction of the computations across the population, specializing only lightweight policy heads to realize the diversity of agents. Challenges (ii) and (iii) are jointly achieved by introducing an RL objective aimed at specializing agents on distinct subsets of the problem distribution. Concretely, we derive a lower bound of the true population-level objective, which corresponds to training only the agent which performs best on each problem. This is intuitively justified as the performance of the population on a given problem is not improved by training an agent on an instance where another agent already has better performance. Strikingly, we find that judicious application of this conceptually simple objective gives rise to a population where the diversity of policies is obtained without explicit supervision (and hence is applicable across a range of problems without modification) and essential for strong performance.

Our contributions are summarized as follows.

1. We introduce and motivate the use of populations for combinatorial problems.
2. We show that an encoder-decoder architecture can be used to efficiently train a population.
3. We derive a new training objective and present a practical training procedure that encourages performance-driven diversity (i.e. effective diversity without the use of explicit behavioral markers or other external supervision).
4. We evaluate Poppy on three CO problems: TSP, CVRP, and 0-1 knapsack (KP). On TSP and KP, Poppy significantly outperforms other RL-based approaches. On CVRP, it consistently outperforms other inference-only methods and approaches the performance of actively fine-tuning problem-specific policies.

2 Related Work

ML for Combinatorial Optimization The first attempt to solve TSP with neural networks is due to Hopfield & Tank (1985), which only scaled up to 30 cities. Recent developments of bespoke neural architectures (Vinyals et al., 2015; Vaswani et al., 2017) and performant hardware have made ML approaches increasingly efficient. Indeed, several architectures have been used to address CO problems, such as graph neural networks (Dai et al., 2017), recurrent neural networks (Nazari et al., 2018), and attention mechanisms (Deudon et al., 2018). In this paper, we use an encoder-decoder architecture that draws from that proposed by Kool et al. (2019). The costly encoder is run once per problem instance, and the resulting embeddings are fed to a small decoder iteratively rolled out to get the whole trajectory, which enables efficient inference. This approach was furthered by Kwon et al. (2020), who leveraged the underlying symmetries of typical CO problems (e.g. of starting positions and rotations) to realize improved training and inference performance using instance augmentations.

Additionally, ML approaches usually rely on mechanisms to generate multiple candidate solutions (Mazyavkina et al., 2021). One such mechanism consists in using improvement methods on an initial solution: de O. da Costa et al. (2020) uses policy gradients to learn a policy that selects local operators (2-opt) given a current solution in TSP, while Lu et al. (2020) and Wu et al. (2021) extend this method to CVRP. This idea has been extended to enable searching a learned latent space of solutions (Hottung et al., 2021). However, these approaches have two limitations: they are environment-specific, and the search procedure is inherently biased by the initial solution.

An alternative exploration mechanism is to generate a diverse set of trajectories by stochastically sampling a learned policy, potentially with additional beam search (Joshi et al., 2019), Monte Carlo tree search (Fu et al., 2021), dynamic programming (Kool et al., 2021) or active search (Hottung

et al., 2022). However, intuitively, the generated solutions tend to remain close to the underlying deterministic policy, implying that the benefits of additional sampled candidates diminish quickly.

Population-Based RL Populations have already been used in RL to learn diverse behaviors. In a different context, Gupta et al. (2018), Eysenbach et al. (2019), Hartikainen et al. (2020) and Pong et al. (2020) use a single policy conditioned on a set of goals as an implicit population for unsupervised skill discovery. Closer to our approach, another line of work revolves around explicitly storing a set of distinct policy parameters. Doan et al. (2020), Hong et al. (2018), Jung et al. (2020) and Parker-Holder et al. (2020) use a population to achieve a better coverage of the policy space. This is especially useful in a sparse reward context or to avoid deceptive local optima. However, they enforce explicit attraction-repulsion mechanisms based on policy distances, which are added to the loss function. They thus rely on a performance-diversity tradeoff, which is a major difference with respect to our approach where diversity is a pure byproduct of performance optimization.

Our method is also related to approaches combining RL with evolutionary algorithms (EA; Khadka & Tumer, 2018; Khadka et al., 2019; Pourchot & Sigaud, 2019), which benefit from the sample-efficient RL policy updates while enjoying evolutionary population-level exploration. However, the population is a means to learn a unique strong policy, whereas Poppy learns a set of complementary strategies. More closely related, Quality-Diversity (QD; Pugh et al., 2016; Cully & Demiris, 2018) is a popular EA framework which maintains a portfolio of diverse policies. Pierrot et al. (2022) has recently combined RL with a QD algorithm, Map-Elites (Mouret & Clune, 2015); unlike Poppy, these methods rely on expert knowledge to define behavioral markers, which is not easily amenable to the CO context.

One of the drawbacks of population-based RL is its expensive cost. However, recent approaches have shown that modern hardware as well as targeted frameworks enable efficient vectorized population training (Flajolet et al., 2022), opening the door to a wider range of applications.

3 METHODS

3.1 BACKGROUND AND MOTIVATION

RL Formulation A CO problem instance ρ sampled from some distribution $\mathcal{D}$ consists of a discrete set of N variables (e.g. city locations in TSP). We model a CO problem as a Markov decision process (MDP) defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition function T, a reward function R and a discount factor γ. A state is a trajectory through the problem instance $\tau_t = (x_1, , \ldots, x_t) \in \mathcal{S}$ where $x_i \in \rho$, and thus consists of an ordered list of variables (not necessarily of length N). An action, $a \in \mathcal{A} \subseteq \rho$, consists of choosing the next variable to add; thus, given state $\tau_t = (x_1, \ldots, x_t)$ and action a, the next state is $\tau_{t+1} = T(\tau_t, a) = (x_1, \ldots, x_t, a)$. Let $\mathcal{S}^* \subseteq \mathcal{S}$ be the set of *solutions*; that is, states that comply with the problem's constraints (e.g., a sequence of cities such that each city is visited once and ends with the starting city in TSP). The reward function $R : \mathcal{S}^* \to \mathbb{R}$ maps solutions into scalars. We assume the reward is maximized by the optimal solution (e.g. R returns the negative tour length in TSP).

A *policy* π_θ parameterized by θ can be used to generate solutions for any instance $\rho \sim \mathcal{D}$ by iteratively sampling the next action $a \in \mathcal{A}$ according to the probability distribution $\pi_\theta(\cdot \mid \rho, \tau_t)$. We learn π_θ using REINFORCE (Williams, 1992). This method aims at maximizing the RL objective $J(\theta) \doteq \mathbb{E}_{\rho\sim\mathcal{D}}\mathbb{E}_{\tau\sim\pi_\theta,\rho}R(\tau)$ by adjusting θ such that good trajectories are more likely to be sampled in the future. Formally, the policy parameters θ are updated by gradient ascent using $\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\mathcal{D}}\mathbb{E}_{\tau\sim\pi_\theta,\rho}(R(\tau) - b_\rho)\nabla_\theta \log(p_\theta(\tau))$, where $p_\theta(\tau) = \prod_t \pi_\theta(a_{t+1} \mid \rho, \tau_t)$ and b_ρ is a baseline. The gradient of the objective, $\nabla_\theta J$, can be estimated empirically using Monte Carlo simulations.

Motivating Example We argue for the benefits of training a population using the example in Figure 1. In this environment, there are three actions: **Left**, **Right**, and **Up**. **Up** leads to a medium reward, while **Left/Right** lead to low/high or high/low rewards (the configuration is determined with equal probability at the start of each episode). Crucially, the left and right paths are intricate, so the agent cannot easily infer from its observation which one leads to a higher reward. Then, the best strategy for a single agent is to always go **Up**, as the guaranteed medium reward (2 scoops) is higher than the expected reward of guessing left or right (1.5 scoops). In contrast, two agents

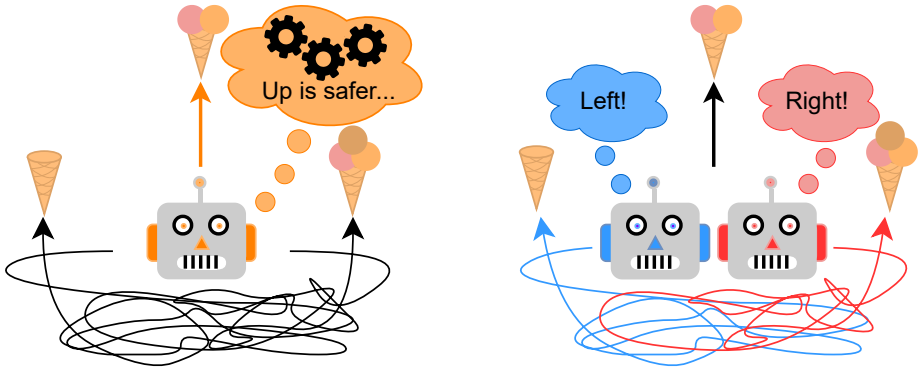


Figure 1: In this environment, the upward path always leads to a medium reward, while the left and right paths are intricate such that either one may lead to a low reward or high reward with equal probability. **Left**: An agent trained to maximize a sum of rewards will converge to taking the safe upward road since it does not have enough information to act optimally. **Right**: A two-agent population can always try the left and right paths and thus get the largest reward in any instance.

in a population can go in opposite directions and always find the maximum reward. There are two striking observations: (i) the agents do not need to perform optimally for the population performance to be optimal (one agent gets the maximum reward), and (ii) the average performance is worse than in the single-agent case.

Specifically, the discussed phenomenon can occur when (i) some optimal actions are too difficult to infer from observations and (ii) choices are irreversible (i.e. it is not possible to recover from a suboptimal decision). These conditions usually hold when solving hard CO problems. In these situations, as shown above, maximizing the performance of a population will require agents to specialize and likely yield better results than in the single agent case.

3.2 POPPY

We present the three distinct components of Poppy: an architecture enabling efficient population rollouts, an RL objective encouraging agent specialization, and the overall training procedure.

Architecture We use the attention model introduced by Kool et al. (2019), which decomposes the policy model into two parts. First, a large encoder h_ψ, takes an instance ρ as input and outputs embeddings ω for each of the variables in ρ. Second, a smaller decoder q_ϕ takes the embeddings ω and a trajectory τ_t as input, and outputs the probabilities of each of the next possible actions. Crucially, the expensive computation of embeddings can be done once for every instance at the beginning of an episode since it is independent of any trajectories; hence, only lightweight decoding operations are performed at each timestep.

We exploit this framework to build a population of K agents. The encoder h_ψ is shared as a common backbone for the whole population, whereas the decoders $q_{\phi_1}, q_{\phi_2}, \ldots, q_{\phi_K}$ are unique to each agent. This is motivated by (i) the encoder learning general representation useful for all agents, and (ii) reducing the overhead of training a population and keeping the total number of parameters low. A discussion on the model sizes is provided in Appendix A.1.

Population-Based Training Objective The usual RL objective $J(\theta) = \mathbb{E}_{\rho\sim\mathcal{D}}\mathbb{E}_{\tau\sim\pi_\theta,\rho}R(\tau)$ was previously presented in Section 3.1. Intuitively, the *population objective* corresponds to rolling out every agent in the population for each problem and using the best solution. For the parameters $\bar{\theta} = \{\theta_1, \theta_2, \ldots, \theta_K\}$ of a population of K agents, the population objective can be thus defined as:

$$J_{\text{pop}}(\bar{\theta}) \doteq \mathbb{E}_{\rho\sim\mathcal{D}}\mathbb{E}_{\tau_1\sim\pi_{\theta_1},\tau_2\sim\pi_{\theta_2}\ldots,\tau_K\sim\pi_{\theta_K}} \max\left[R(\tau_1), R(\tau_2), \ldots, R(\tau_K)\right].$$

In order to update $\bar{\theta}$, we derive a simple lower bound designed at inducing agent specialization:

$$J_{\text{pop}}(\bar{\theta}) = \mathbb{E}_{\rho\sim\mathcal{D}}\mathbb{E}_{\tau_1\sim\pi_{\theta_1},\tau_2\sim\pi_{\theta_2},\ldots,\tau_K\sim\pi_{\theta_K}} \max\left[R(\tau_1), R(\tau_2), \ldots, R(\tau_K)\right],$$
$$J_{\text{pop}}(\bar{\theta}) \geq \mathbb{E}_{\rho\sim\mathcal{D}} \max\left[\mathbb{E}_{\tau_1\sim\pi_{\theta_1}} R(\tau_1), \mathbb{E}_{\tau_2\sim\pi_{\theta_2}} R(\tau_2), \ldots, \mathbb{E}_{\tau_K\sim\pi_{\theta_K}} R(\tau_K)\right],$$
$$J_{\text{pop}}(\bar{\theta}) \geq \mathbb{E}_{\rho\sim\mathcal{D}}\left[\mathbb{E}_{\tau\sim\pi_{\theta^*_\rho}} R(\tau)\right] \text{ with } \theta^*_\rho = \arg\max_{\theta_i\in\bar{\theta}} \mathbb{E}_{\tau\sim\pi_{\theta_i},\rho} R(\tau).$$

This lower bound, which we denote J_{poppy} and treat as our RL objective, corresponds to evaluating only the best performing agent on each instance. The parameters θ^*_ρ can be estimated as the agent with the highest reward on ρ (breaking ties arbitrarily). Importantly, J_{poppy} can be optimized using standard policy gradients (Sutton et al., 1999), with the difference that for each problem only the best agent is trained. This formulation is applicable across a variety of problems and directly optimizes for population-level performance without explicit supervision or handcrafted behavioral markers.

Algorithm 1: Poppy training

input: problem distribution $\mathcal{D}$, number of agents K, batch size B, number of training steps H, a pretrained encoder h_ψ and a set of K decoders $q_{\phi_1}, q_{\phi_2}, \ldots, q_{\phi_K}$;

for *step 1 to H* **do**

$\rho_i \leftarrow \texttt{Sample}(\mathcal{D})\ \forall i \in 1, \ldots, B$;

${}^k\tau_i \leftarrow \texttt{Rollout}(\rho_i, h_\psi, q_{\phi_k})\ \forall i \in 1, \ldots, B, \forall k \in 1, \ldots, K$;

`/* Select the best trajectory for each problem` ρ_i. `*/`

$\mathcal{T}^* \leftarrow \texttt{SelectBest}\left(\{{}^k\tau_i \mid i \in 1, \ldots, B, k \in 1, \ldots, K\}\right)$;

`/* Propagate the gradients through these only. */`

$\nabla L(h_\psi, q_{\phi_1}, q_{\phi_2}, \ldots, q_{\phi_K}) \leftarrow \frac{1}{|\mathcal{T}^*|}\sum_{\tau\in\mathcal{T}^*} \texttt{Reinforce}(\tau)$;

$(h_\psi, q_{\phi_1}, q_{\phi_2}, \ldots, q_{\phi_K}) \leftarrow (h_\psi, q_{\phi_1}, q_{\phi_2}, \ldots, q_{\phi_K}) - \alpha\nabla L(h_\psi, q_{\phi_1}, q_{\phi_2}, \ldots, q_{\phi_K})$;

Training Procedure The training procedure consists of three phases:

1. A single-decoder architecture (i.e. single agent) is trained from scratch. This step is identical to the training process of Kwon et al. (2020).
2. The trained decoder is discarded and a population of decoder heads is randomly initialized. With the parameters of the shared encoder frozen, the decoders are trained in parallel using the same training instances, as shown in Figure 2 (left).
3. The encoder is unfrozen and trained jointly with the population decoders using the J_{poppy} objective, as described in Algorithm 1 and illustrated in Figure 2 (right). Agents implicitly specialize on different types of problem instances during this phase.

Phase 1 enables training the large encoder without the computational overhead of a population. Whilst this is not strictly necessary from a performance perspective, we observed that jointly training a shared encoder alongside a large population from scratch (i.e. combining the first two phases) is less efficient and not more performant. Phase 2, where the population is pre-trained before specializing, is also critical. Randomly initialized decoders are often ill-distributed, such that a single (or few) agent(s) dominate the performance across all instances; therefore, without previously pre-training the decoders, only initially dominating agents would receive a training signal, further widening the performance gap. We found that pre-training the entire population to have reasonable policies before specialization was a simple and reliable method to avoid this issue. Finally, Phase 3 is also applicable with the encoder frozen; however, we choose to unfreeze it to maximize performance.

Starting points Following Kwon et al. (2020), we generate multiple solutions for each instance ρ by considering a set of $P \in [1, N]$ *starting points*, where N is the number of instance variables. For example, a starting point in a TSP instance could be any of its cities. Therefore, across the different training phases, agents generate trajectories for (instance, starting point) pairs. The average reward is used as the REINFORCE baseline. A key consideration is at what level to apply the Poppy objective (i.e. how do agents specialize during Phase 3). As agents are trained across several starting

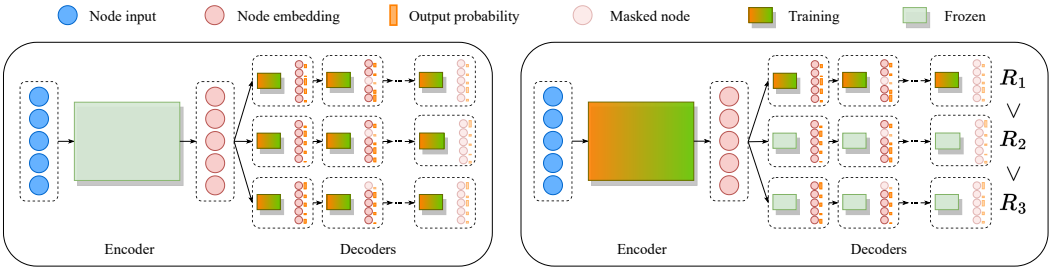


Figure 2: The last two phases of the training process. **Left (Phase 2)**: the population is trained in parallel on top of a frozen encoder. **Right (Phase 3)**: training according to the Poppy objective (i.e., the gradient is only propagated through the decoder which yielded the highest reward).

points for each instance, we distinguish between two kinds of specialization. First, when the optimal solution depends on the starting point, we consider each (instance, starting point) pair as a separate problem; thus, we train the best agent for each starting point. Second, when the optimal solution does not depend on the starting point, we push the specialization further onto the starting points: for each instance, only the single best (agent, starting point) pair is used for training. The former case is more general, whereas the latter case holds for TSP since any starting position can lead to the same solution. An extended discussion can be found in Appendix A.2.

4 Experiments

We evaluate Poppy on three CO problems: TSP, CVRP, and KP. To emphasize its generality, we use the same hyperparameters for each problem, taken from Kwon et al. (2020). We run Poppy for populations of 4, 8, 16, or 32 agents, which exhibit various time-performance tradeoffs.

Training One training step corresponds to computing policy gradients over the same batch of 64 instances for each agent in the population. Training time varies with problem complexity and training phase. For instance, in TSP with 100 cities, Phases 1 and 2 take 4.5M (5 days) and 25k (10H) steps, respectively. Phase 3 takes 800k training steps and lasts 1-4 days depending on the population size. Our JAX-based implementation using environments from the Jumanji suite (Bonnet et al., 2022), along with problem instances to reproduce this work, are available at `[MASKED-FOR-BLIND-REVIEW]`. All experiments were run on a v3-8 TPU.

Inference We greedily roll out each agent in the population, and use the augmentations proposed by Kwon et al. (2020) for TSP and CVRP. Additionally, to give a sense of the performance of Poppy with a larger time budget, we implement a simple sampling strategy. Given a population of K agents, we first greedily rollout each of them on every starting point, and evenly distribute any remaining sampling budget across the most promising K (agent, starting point) pairs for each instance.

Baselines We compare Poppy against exact solvers, heuristics, and state-of-the-art ML methods. Some baseline performances taken from Hottung et al. (2022) were obtained with different hardware and framework (Nvidia Tesla V100 GPU with Pytorch); thus, for fairness, we mark these times with $*$ in our tables. As a comparison guideline, we informally note that these GPU inference times should be approximately divided by 2 to get the converted TPU time.

4.1 Traveling Salesman Problem (TSP)

Given a set of n cities, the goal in TSP is to visit every city and come back to the starting city while minimizing the total traveled distance.

Setup We use the architecture used by Kool et al. (2019) and Kwon et al. (2020) with slight modifications (see Appendix B). The testing instances are taken from Kool et al. (2019) for $n = 100$, and from Hottung et al. (2022) for $n \in \{125, 150\}$. The training is done on $n = 100$ instances. We

compare Poppy to (i) the specialized supervised learning (SL) methods GCN-BS (Joshi et al., 2019), CVAE-Opt (Hottung et al., 2021), DPDP (Kool et al., 2021); (ii) the fast RL methods with limited sampling budgets POMO with greedy rollout, and POMO with 16 stochastic rollouts (to match Poppy 16 runtime); and (iii) the slow RL methods (characterized by the inclusion of extensive search strategies) 2-Opt-DL (de O. da Costa et al., 2020), LIH (Wu et al., 2021), POMO with 200 rollouts, and the fine-tuning approach EAS (Hottung et al., 2022). In addition to these ML-based approaches, we also compare to the exact solver Concorde (Applegate et al., 2006) and the heuristic solver LKH3 (Helsgaun, 2017). Poppy 1 is not equivalent to POMO here due to the starting point specialization.

Results Table 1 displays the average tour length, the optimality gap, and the total runtime for each test set. The best algorithm remains Concorde as it is a highly specialized TSP solver. Remarkably, Poppy 16 with greedy rollouts reaches the best performance across every category in just a few minutes, except for one case where DPDP performs better; however, DPDP tackles specifically routing problems and makes use of expert knowledge. With extra sampling, Poppy reaches a performance gap of 0.001%, which establishes a state-of-the-art for general ML-based approaches. Compared to DPDP, Poppy improves 0-shot performance, suggesting that it is more robust to distribution shifts.

Table 1: TSP results.

		Inference (10k instances)			**0-shot** (1k instances)					
		$n = 100$			$n = 125$			$n = 150$		
	Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
	Concorde	7.765	0.000%	82M	8.583	0.000%	12M	9.346	0.000%	17M
	LKH3	7.765	0.000%	8H	8.583	0.000%	73M	9.346	0.000%	99M
SL	GCN-BS	7.87	1.39%	40M*	-	-	-	-	-	-
SL	CVAE-Opt	-	0.343%	6D*	8.646	0.736%	21H*	9.482	1.45%	30H*
SL	DPDP	7.765	0.004%	2H*	8.589	0.070%	31M*	9.434	0.94%	44M*
RL (fast)	POMO	7.774	0.13%	37S	8.605	0.26%	6S	9.393	0.50%	10S
RL (fast)	POMO (16 samples)	7.770	0.073%	9M	8.597	0.16%	1M	9.385	0.41%	2M
RL (fast)	**Poppy 1**	7.768	0.044%	37S	8.593	0.12%	6S	9.372	0.27%	10S
RL (fast)	**Poppy 4**	7.766	0.016%	2M	8.588	0.057%	23S	9.363	0.17%	38S
RL (fast)	**Poppy 8**	7.765	0.010%	5M	8.587	0.043%	45S	9.360	0.15%	1M
RL (fast)	**Poppy 16**	**7.765**	**0.008%**	9M	**8.586**	**0.038%**	1M	**9.359**	**0.14%**	2M
RL (slow)	2-Opt-DL	7.83	0.87%	41M*	-	-	-	-	-	-
RL (slow)	LIH	7.87	1.42%	2H*	-	-	-	-	-	-
RL (slow)	POMO (200 samples)	7.769	0.056%	2H	8.594	0.13%	20M	9.376	0.31%	32M
RL (slow)	EAS	7.768	0.048%	5H*	8.591	0.091%	49M*	9.365	0.20%	1H*
RL (slow)	**Poppy 16 (200 samples)**	**7.765**	**0.001%**	2H	**8.584**	**0.008%**	20M	**9.352**	**0.06%**	32M

Analysis To better understand how Poppy affects the training process, we display Phases 2 and 3 for a 16-agent population on the top plots of Figure 3. When switching to J_{poppy}, the population-level performance improves sharply, while the average agent performance deteriorates (top left), and the number of agents reaching the same best score suddenly diminishes (top right). This corroborates the intuition that agents specialize. Moreover, the bottom right plot shows that the average agent performance decreases with the population size, whereas the population-level performance increases. This demonstrates a stronger specialization for larger populations, which J_{poppy} appears to balance. Since Poppy produces an unsupervised specialization favoring the best agents, it is not straightforward to predict how the performance will distribute across the population. In the bottom left figure, we can see that the performance is evenly distributed across the population of Poppy 16; hence, showing that the population has not collapsed to a few high-performing agents, and that Poppy fully benefits from the population size. Additional analyses are made in Appendix B.1.4.

4.2 Capacitated Vehicle Routing Problem (CVRP)

Given a vehicle with limited capacity departing from a depot node and a set of n nodes with different demands, the goal is to find an optimal set of routes such that each node (except for the depot) is visited exactly once and has its demand covered. The vehicle's capacity diminishes by the demand of the visited node (which must be fully covered) and it is restored when the depot is visited.

Setup We use the test instances employed by Kwon et al. (2020) for $n = 100$, and the sets from Hottung et al. (2022) to evaluate generalization to larger problems. We evaluate Poppy with popu-

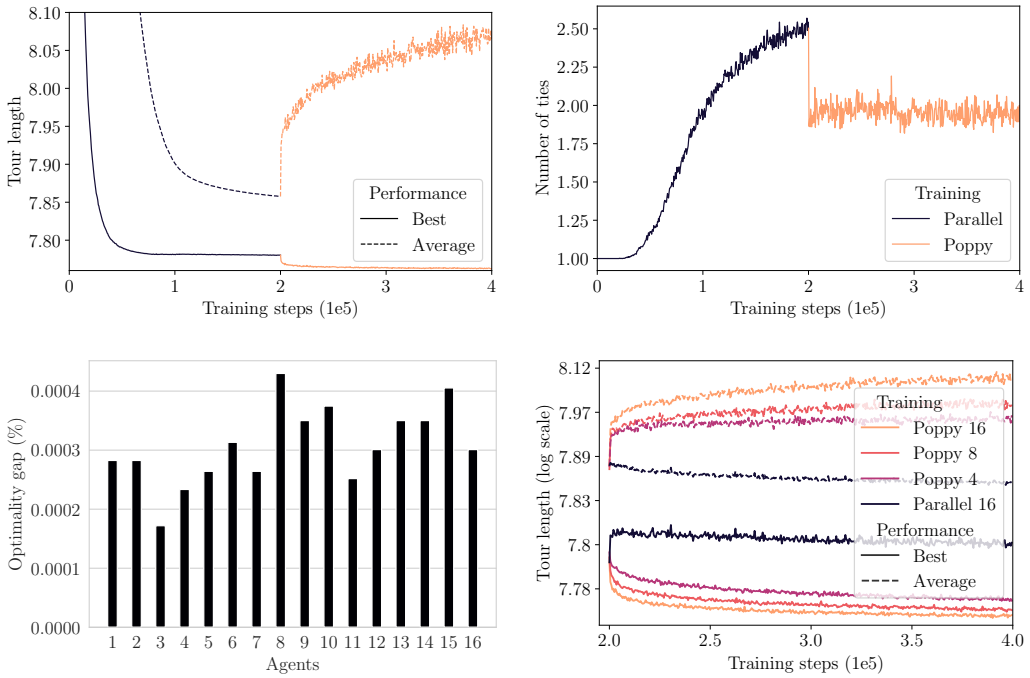


Figure 3: The first three plots are an analysis of Poppy 16 on TSP100. **Top left**: Both average and best tour lengths decrease during the second training phase (dashed and solid blue lines). In Phase 3, the average performance drops while the population performance rises abruptly. **Top right**: The number of agents reaching the same best performance for a given instance increases during Phase 2 (blue), and suddenly drops and stays constant when switching to Phase 3 (red). **Bottom left**: Optimality gap loss suffered when removing any agent from the population. Although some contribute more (e.g. 8, 15) and some less (e.g. 3, 4), the distribution is strikingly even, although no explicit mechanism enforces this behavior. **Bottom right**: With J_{poppy}, the average agent gets worse as the population size increases, although the population-level performance improves. Training 16 agents in parallel leads to a better average but worse population performance.

lations of 4 and 8 agents, and include a population of 32 since the performances were improving. We compare Poppy to the heuristic solver LKH3 (Helsgaun, 2017), taken as a reference to compute the gaps although its performances are not optimal. We also report results of the supervised ML methods CVAE-Opt (Hottung et al., 2021), DPDP (Kool et al., 2021), and RL methods NeuRewriter (Chen & Tian, 2019), NLNS (Hottung & Tierney, 2020), POMO (Kwon et al., 2020), LIH (Wu et al., 2021), and EAS (Hottung et al., 2022). As for TSP, we evaluate POMO with greedy rollouts, as well as 32 stochastic samples to match the runtime of our largest population, and 200 samples.

Results Table 2 shows Poppy has the best time-performance tradeoff among the fast approaches, e.g. Poppy 32 has the same runtime as POMO with 32 stochastic rollouts while dividing by 1.5 the optimal gap. A fine-grained comparison between POMO with stochastic sampling and Poppy is in Appendix 7. Among the slow RL approaches, adding sampling to Poppy makes it on par with DPDP depending on the problem size, and it is only outperformed by the active search approach EAS, which gives large improvements on CVRP. However, active search (i) prevents parallelization (as it scales linearly with the number of samples) and (ii) could be added on top of Poppy instead of stochastic sampling to further boost performance, which we leave for future work.

4.3 0-1 Knapsack (KP)

We evaluate Poppy on KP to demonstrate its applicability beyond routing problems. Given a set of n items with specific weights and values and a bag of limited capacity, the goal is to determine which items to add such that the total weight does not exceed the capacity and the value is maximal.

Table 2: CVRP results.

	Method	**Inference** (10k instances) $n = 100$ Obj.	Gap	Time	**0-shot** (1k instances) $n = 125$ Obj.	Gap	Time	**0-shot** (1k instances) $n = 150$ Obj.	Gap	Time
	LKH3	15.65	0.000%	6D	17.50	0.000%	19H	19.22	0.000%	20H
SL	CVAE-Opt	-	1.36%	11D*	17.87	2.08%	36H*	19.84	3.24%	46H*
SL	DPDP	15.63	−0.13%	23H*	17.51	0.07%	3H*	19.31	0.48%	5H*
RL (fast)	NeuRewriter	16.10	-	66M*	-	-	-	-	-	-
RL (fast)	NLNS	15.99	2.23%	62M*	18.07	3.23%	9M*	19.96	3.86%	12M*
RL (fast)	POMO	15.76	0.76%	2M	17.68	1.02%	<1M	19.58	1.85%	1M
RL (fast)	POMO (32 samples)	15.70	0.32%	43M	17.59	0.50%	8M	19.48	1.35%	12M
RL (fast)	**Poppy 4**	15.72	0.45%	5M	17.62	0.69%	2M	19.49	1.38%	2M
RL (fast)	**Poppy 8**	15.70	0.32%	11M	17.60	0.54%	3M	19.46	1.22%	5M
RL (fast)	**Poppy 32**	**15.68**	**0.20%**	43M	**17.57**	**0.40%**	8M	**19.41**	**1.00%**	12M
RL (slow)	LIH	16.03	2.47%	5H*	-	-	-	-	-	-
RL (slow)	POMO (200 samples)	15.67	0.18%	4H	17.56	0.33%	43M	19.43	1.08%	1H
RL (slow)	EAS	**15.62**	**-0.14%**	8H*	**17.50**	**0.00%**	80M*	19.36	0.72%	2H*
RL (slow)	**Poppy 32 (200 samples)**	15.63	−0.10%	4H	17.51	0.02%	42M	**19.34**	**0.61%**	1H

Setup We use the setting employed by Kwon et al. (2020): an action corresponds to putting an item in the bag, which is filled iteratively until no remaining item can be added. We evaluate Poppy on 3 population sizes against the optimal solution based on dynamic programming, a greedy heuristic, and POMO with and without sampling.

Results Table 3 shows that Poppy leads to improved performance with a population of 16 agents, dividing the optimality gap with respect to POMO by 45 and 12 on KP100 and KP200 respectively, and by 12 and 2 with respect to POMO with 16 stochastic samples for the exact same runtime.

Table 3: KP results.

Method	**Testing** (10k instances) $n = 100$ Obj.	Gap	Time	**Testing** (1k instances) $n = 200$ Obj.	Gap	Time
Optimal	40.437	-		57.729	-	
Greedy	40.387	0.1250%		57.672	0.0986%	
POMO	40.428	0.0224%	8S	57.718	0.0191%	4S
POMO (16 samples)	40.435	0.0060%	2M	57.727	0.0032%	1M
Poppy 4	40.434	0.0081%	33S	57.723	0.0099%	16S
Poppy 8	40.436	0.0032%	1M	57.726	0.0058%	33S
Poppy 16	**40.437**	**0.0005%**	2M	**57.728**	**0.0015%**	1M

5 CONCLUSIONS

Poppy is a population-based RL method for CO problems. It uses an RL objective that incurs agent specialization with the purpose of maximizing population-level performance. Crucially, Poppy does not rely on handcrafted notions of diversity to enforce specialization. We show that Poppy achieves state-of-the-art performance on three popular NP-hard problems: TSP, CVRP, and KP.

This work raises several questions. First, we have experimented on populations of at most 32 agents; therefore, it is unclear what the consequences of training larger populations are. We hypothesize that the population performance would eventually collapse, leaving agents with null contributions behind. Exploring this direction, including possible strategies to prevent such collapses, is an interesting direction for future work. Second, we recall that the motivation behind Poppy was dealing with problems where predicting optimal actions from observations is too difficult to be solved reliably by a single agent. We believe that such settings may not be strictly limited to canonical CO problems, and that population-based approaches offer a promising direction for many challenging RL applications. In this direction, we hope that approaches (such as Poppy) that alleviate the need for handcrafted behavioral markers, whilst still realizing performant diversity, could broaden the range of applications of population-based RL.

REPRODUCIBILITY STATEMENT

We provide in Appendix A.2 the detailed pseudo-code of our method. Model architectures as well as the training hyperparameters are given in Appendix B.1.2, Appendix B.2.2 and Appendix B.4 for TSP, CVRP and KP respectively. Moreover, all the code used in our experiments as well as the test instances are available at [MASKED-FOR-BLIND-REVIEW].

REFERENCES

David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver, 2006.

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.

Clément Bonnet, Donal Byrne, Victor Le, Laurence Midgley, Daniel Luo, Cemlyn Waters, Sasha Abramowitz, Edan Toledo, Cyprien Courtot, Matthew Morris, Daniel Furelos-Blanco, Nathan Grinsztajn, Thomas D. Barrett, and Alexandre Laterre. Jumanji: Industry-driven hardware-accelerated RL environments, 2022. URL https://github.com/instadeepai/jumanji.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and Chris Hesse. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2019.

Antoine Cully and Yiannis Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2018. doi: 10.1109/TEVC.2017.2704781.

Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, 2017.

Paulo R. de O. da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian Conference on Machine Learning*, 2020.

Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 170–181. Springer International Publishing, 2018.

Thang Doan, Bogdan Mazoure, Moloud Abdar, Audrey Durand, Joelle Pineau, and R. Devon Hjelm. Attraction-repulsion actor-critic for continuous control reinforcement learning. *arXiv preprint arXiv:1909.07543*, 2020. URL http://arxiv.org/abs/1909.07543.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

Arthur Flajolet, Claire Bizon Monroc, Karim Beguir, and Thomas Pierrot. Fast population-based reinforcement learning on a single machine. In *International Conference on Machine Learning*, 2022.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large TSP instances. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021,*

The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pp. 7474–7482. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/16916.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *International Conference on Neural Information Processing Systems*, 2018.

Kristian Hartikainen, Xinyang Geng, Tuomas Haarnoja, and Sergey Levine. Dynamical distance learning for semi-supervised and unsupervised skill discovery. In *International Conference on Learning Representations*, 2020.

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

Zhang-Wei Hong, Tzu-Yun Shann, Shih-Yang Su, Yi-Hsiang Chang, and Chun-Yi Lee. Diversity-driven exploration strategy for deep reinforcement learning. In *International Conference on Neural Information Processing Systems*, 2018.

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

André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *24th European Conference on Artificial Intelligence (ECAI 2020)*, 2020.

André Hottung, Bhanu Bhandari, and Kevin Tierney. Learning a latent search space for routing problems using variational autoencoders. In *International Conference on Learning Representations*, 2021.

André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. In *International Conference on Learning Representations*, 2022.

Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Whiyoung Jung, Giseung Park, and Youngchul Sung. Population-guided parallel policy search for reinforcement learning. In *International Conference on Learning Representations*, 2020.

Shauharda Khadka and Kagan Tumer. Evolution-guided policy gradient in reinforcement learning. In *Conference on Neural Information Processing Systems*, 2018.

Shauharda Khadka, Somdeb Majumdar, Tarek Nassar, Zach Dwiel, Evren Tumer, Santiago Miret, Yinyin Liu, and Kagan Tumer. Collaborative evolutionary reinforcement learning. In *International Conference on Machine Learning*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. Deep policy dynamic programming for vehicle routing problems. *arXiv preprint arXiv:2102.11756*, 2021.

Yeong-Dae Kwon, Byoungjip Kim Jinho Choo, Youngjune Gwon Iljoo Yoon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*, 2020.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, Apr 2015.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takáč. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, 2018.

Jack Parker-Holder, Aldo Pacchiano, Krzysztof Choromanski, and Stephen Roberts. Effective diversity in population based reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Thomas Pierrot, Valentin Macé, Félix Chalumeau, Arthur Flajolet, Geoffrey Cideron, Karim Beguir, Antoine Cully, Olivier Sigaud, and Nicolas Perrin-Gilbert. Diversity policy gradient for sample efficient quality-diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2022.

Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *International Conference on Machine Learning*, 2020.

Aloïs Pourchot and Olivier Sigaud. Cem-rl: Combining evolutionary and gradient-based methods for policy search. In *International Conference on Learning Representations*, 2019.

Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3, 2016. ISSN 2296-9144. doi: 10.3389/frobt.2016.00040.

Abdelkade Sbihi and Richard W. Eglese. Combinatorial optimization and green logistics. *4OR*, 5: 99–116, 2007.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Richard S. Sutton, David A. McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pp. 1057–1063, 1999.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *ArXiv*, abs/1609.03499, 2016.

Ashish Vaswani, Google Brain, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, 2015.

Adam Zsolt Wagner. Refuting conjectures in extremal combinatorics via linear programming. *Journal of Combinatorial Theory, Series A*, 169:105130, Jan 2020. doi: 10.1016/j.jcta.2019.105130. URL `https://www.sciencedirect.com/science/article/abs/pii/S0097316519301116`.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2021. doi: 10.1109/TNNLS.2021.3068828.

A Additional Details on Poppy

A.1 Number of Parameters

Table 4 shows the total number of parameters of our models as a function of the population size. Since the decoder represents less than 10% of the parameters, scaling the population size can be done efficiently. For instance, a population of 16 agents roughly doubles the model size.

Table 4: Number of model parameters for different population sizes.

	Encoder	Decoder	Population size 1	4	8	16	32
Parameters	1,190,016	98,816	1,288,832	1,585,280	1,980,544	2,771,072	4,352,128
Extra parameters	-	-	0%	23%	54%	115%	238%

A.2 Possible Flavors of Poppy

We present the detailed algorithms for the two Poppy variations used in our experiments. Algorithm 2 is the most general version, which consists in training only the best agent for each (instance, starting point) pair. Using K agents and P different starting points means that for any given instance $P \times K$ trajectories are rolled out, among which only P are effectively used for training. We apply this version in CVRP and KP.

Algorithm 3 adds a layer of specialization: for each instance, the best agent is trained from its best starting point only. As before, $P \times K$ trajectories are sampled, but the loss is computed from a single trajectory.

Algorithm 2: Poppy training with starting points (general)

input: problem distribution $\mathcal{D}$, number of starting points per instance P, number of agents K, batch size B, number of training steps H, a pretrained encoder h_ψ and a set of K decoders $q_{\phi_1}, q_{\phi_2}, \dots, q_{\phi_K}$;

for *step 1 to H* **do**

$\rho_i \leftarrow \texttt{Sample}(\mathcal{D})\ \forall i \in 1, \dots, B$;

$\alpha_i^1, \dots, \alpha_i^P \leftarrow \texttt{SelectStartPoints}(\rho_i, P)\ \forall i \in 1, \dots, B$;

${}^k\tau_i^p \leftarrow \texttt{Rollout}(\rho_i, \alpha_i^p, h_\psi, q_{\phi_k})\ \forall i \in 1, \dots, B, \forall p \in 1, \dots, P, \forall k \in 1, \dots, K$;

${}^kb_i \leftarrow \frac{1}{P}\sum_p R({}^k\tau_i^p)$;

/* Select the best agent per (instance, start point). */

$k_{i,p}^* \leftarrow \arg\max_{k \leq K} R({}^k\tau_i^p)\ \forall i \in 1, \dots, B, \forall p \in 1, \dots, P$;

/* Propagate the gradients through these only. */

$\nabla L(h_\psi, q_{\phi_1}, q_{\phi_2}, \dots, q_{\phi_K}) \leftarrow -\frac{1}{BP}\sum_{i,p}(R({}^{k_{i,p}^*}\tau_i^p) - {}^{k_{i,p}^*}b_i)\nabla \log p_{\psi,\phi_{k_{i,p}^*}}({}^{k_{i,p}^*}\tau_i^p)$;

$(h_\psi, q_{\phi_1}, q_{\phi_2}, \dots, q_{\phi_K}) \leftarrow (h_\psi, q_{\phi_1}, q_{\phi_2}, \dots, q_{\phi_K}) - \alpha\nabla L(h_\psi, q_{\phi_1}, q_{\phi_2}, \dots, q_{\phi_K})$;

B Problems

We here describe the details of the CO problems we have used to evaluate Poppy, including instance generation and training details (e.g. architecture, hyperparameters). In the case of TSP and CVRP, we show some example solutions obtained by a population of agents. Besides, we thoroughly analyze the performance of the populations in TSP.

B.1 Traveling Salesman Problem (TSP)

B.1.1 Instance Generation

The n cities that constitute each problem's instance are uniformly sampled from $[0, 1]^2$.

Algorithm 3: Poppy training with starting points and starting point specialization (TSP)

input: problem distribution $\mathcal{D}$, number of starting points per instance P, number of agents K, batch size B, number of training steps H, a pretrained encoder h_ψ and a set of K decoders $q_{\phi_1}, q_{\phi_2}, \dots, q_{\phi_K}$;

for *step 1 to H* **do**

$\rho_i \leftarrow \texttt{Sample}(\mathcal{D})\ \forall i \in 1, \dots, B$;

$\alpha_i^1, \dots, \alpha_i^P \leftarrow \texttt{SelectStartPoints}(\rho_i, P)\ \forall i \in 1, \dots, B$;

${}^k\tau_i^p \leftarrow \texttt{Rollout}(\rho_i, \alpha_i^p, h_\psi, q_{\phi_k})\ \forall i \in 1, \dots, B, \forall p \in 1, \dots, P, \forall k \in 1, \dots, K$;

${}^kb_i \leftarrow \frac{1}{P}\sum_p R({}^k\tau_i^p)$;

/* Select the best agent and starting point per problem. */

$k_i^*, p_i^* \leftarrow \arg\max_{p \leq P, k \leq K} R({}^k\tau_i^p)\ \forall i \in 1, \dots, B$;

/* Propagate the gradients through these only. */

$\nabla L(h_\psi, q_{\phi_1}, q_{\phi_2}, \dots, q_{\phi_K}) \leftarrow -\frac{1}{B}\sum_i (R({}^{k_i^*}\tau_i^{p_i^*}) - {}^{k_i^*}b_i)\nabla \log p_{\psi, \phi_{k_i^*}}({}^{k_i^*}\tau_i^{p_i^*})$;

$(h_\psi, q_{\phi_1}, q_{\phi_2}, \dots, q_{\phi_K}) \leftarrow (h_\psi, q_{\phi_1}, q_{\phi_2}, \dots, q_{\phi_K}) - \alpha\nabla L(h_\psi, q_{\phi_1}, q_{\phi_2}, \dots, q_{\phi_K})$;

B.1.2 Training Details

Architecture We use the same model as Kool et al. (2019) and Kwon et al. (2020) except for the batch-normalization layers, which are replaced by layer-normalization to ease parallel batch processing. We invert the mask used in the decoder computations (i.e., masking the available cities instead of the unavailable ones) after having experimentally observed faster convergence rates.

Hyperparameters To match the setting used by Kwon et al. (2020), we use the Adam optimizer (Kingma & Ba, 2015) with a learning rate $\mu = 10^{-4}$, and a L_2 penalization of 10^{-6}. The encoder is composed of 6 multi-head attention layers with 8 heads each. The dimension of the keys, queries and values is 16. Each attention layer is composed of a feed-forward layer of size 512, and the final node embeddings have 128 dimensions. The decoders are composed of 1 multi-head attention layer with 8 heads and 16-dimensional key, query and value.

The number of starting points P is 20 for each instance. We determined this value after performing a grid-search based on the first training steps with $P \in \{20, 50, 100\}$.

B.1.3 Example Solutions

Figure 5 shows some trajectories obtained from a 16-agent population on TSP100. Even though they look similar, small decisions differ between agents, thus frequently leading to different solutions. Interestingly, some agents (especially 1 and 5) give very poor trajectories. We hypothesize that it is a consequence of specializing since agents have no incentive to provide a good solution if either another agent is already better on this instance, or if other starting points yield better performance.

B.1.4 Population Analysis

Figure 4 shows some additional information about individual agent performances. In the left figure, we observe that each agent gives on average the best solution for 25% of the instances, and that for 2.5% it gives the unique best solution across the population. These numbers are evenly distributed, which shows that every agent contributes to the whole population performance. Furthermore, the best performance is reached by a single agent in almost 40% of the cases, as shown in the bottom figure. On the right is displayed the performance of several sub-populations of agents for Poppy 4, 8 and 16. Unsurprisingly, any fixed size sub-population is better when sampled from smaller populations: Poppy 16 needs 6 agents to recover the performances of Poppy 4, and 11 agents to recover the performance of Poppy 8 for example. This highlights the fact that agents have learned complementary behaviors which might be sub-optimal if part of the total population is missing.

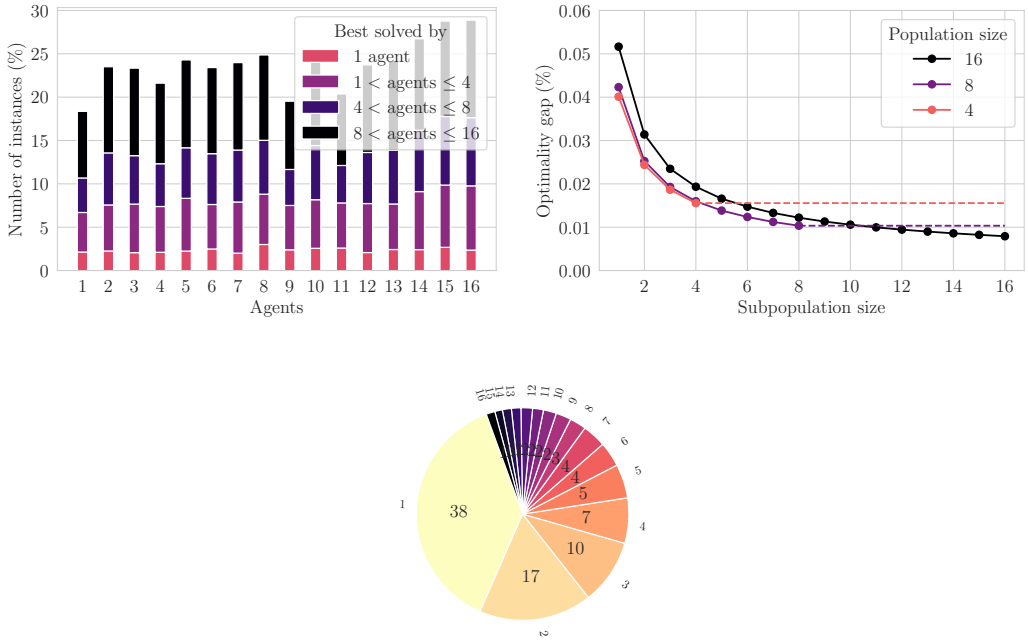


Figure 4: **Left**: Proportion of instances that each agent solves best among the population for Poppy 16 on TSP100. Colors indicate the number of agents in the population giving the same solution for these sets of instances. **Right**: The mean performance of 1,000 randomly drawn sub-populations for Poppy 1, 4, 8 and 16. **Bottom**: Proportion of test instances where any number of agents reach the exact same best solution. A 38% of the instances are best solved by a single agent, and more than a 50% by less than two agents.

B.2 Capacitated Vehicle Routing Problem (CVRP)

B.2.1 Instance Generation

The locations of the n customer nodes and the depot are uniformly sampled in $[0, 1]^2$. The demands are uniformly sampled from the discrete set $\{\frac{1}{D}, \frac{2}{D}, \ldots, \frac{9}{D}\}$ where $D = 50$ for CVRP100, $D = 55$ for CVRP125, and $D = 60$ for CVRP150. The maximum vehicle capacity is 1. The deliveries cannot be split: each customer node is visited once, and its whole demand is taken off the vehicle's remaining capacity.

B.2.2 Training Details

Architecture We use the same model as in TSP. However, unlike TSP, the mask is not inverted; besides, it does not only prevent the agent from revisiting previous customer nodes, but also from visiting the depot if it is the current location, and any customer node whose demand is higher than the current capacity.

Hyperparameters We use the same hyperparameters as in TSP except for the number of starting points P per instance used during training, which we set to 100 after performing a grid-search with $P \in \{20, 50, 100\}$.

B.2.3 Example Solutions

Figure 6 shows some trajectories obtained by 16 agents from a 32-agent population on CVRP100. Unlike TSP, the agent/vehicle performs several tours starting and finishing in the depot.

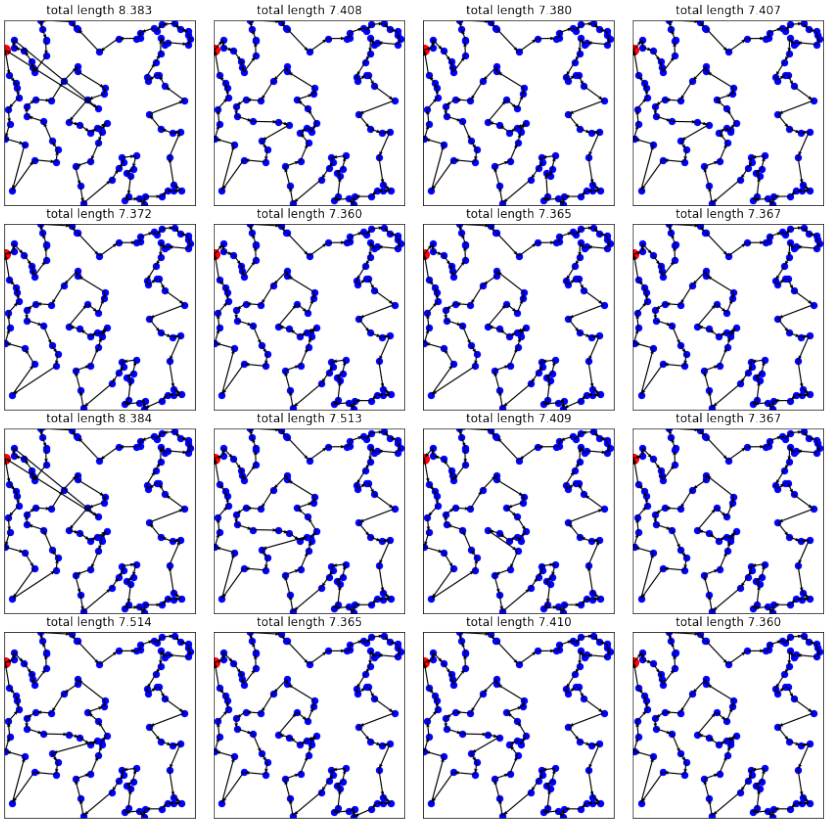


Figure 5: Example TSP trajectories given by Poppy for a 16-agent population from one starting point (red).

B.3 0-1 KNAPSACK (KP)

B.3.1 INSTANCE GENERATION

Item values and weights are uniformly sampled in $[0, 1]$. The bag capacity is fixed to 25.

B.4 TRAINING DETAILS

Architecture We use the same model as in TSP. However, the mask used when decoding is not inverted, and the items that do not fit in the bag are masked together with the items taken so far.

Hyperparameters We use the same hyperparameters as in TSP except for the number of starting points P used during training, which we set to 100 after performing a grid-search with $P \in \{20, 50, 100\}$.

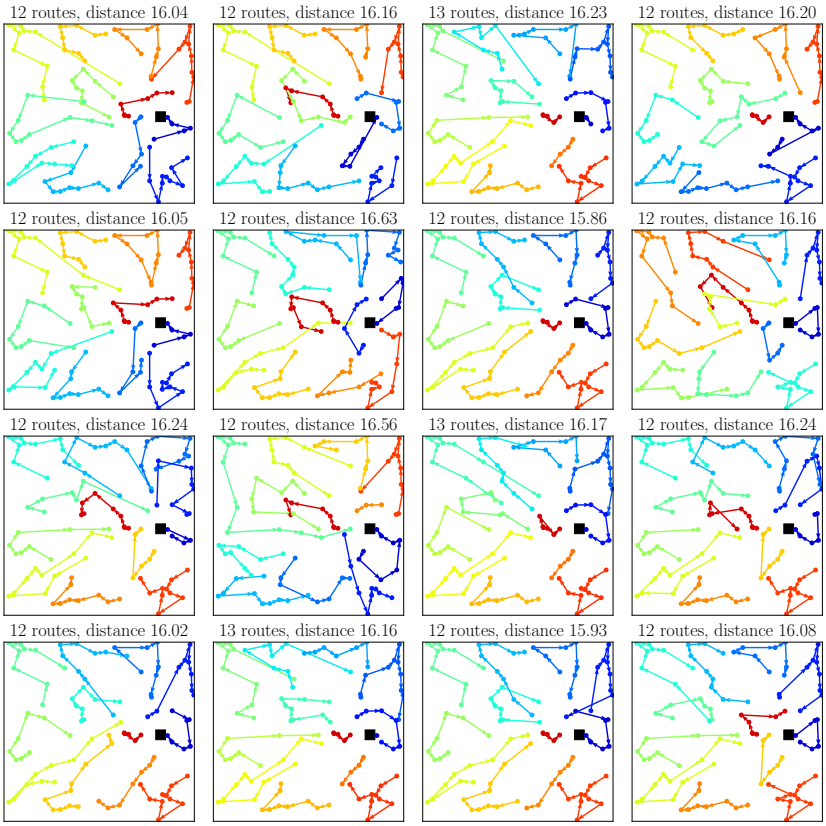
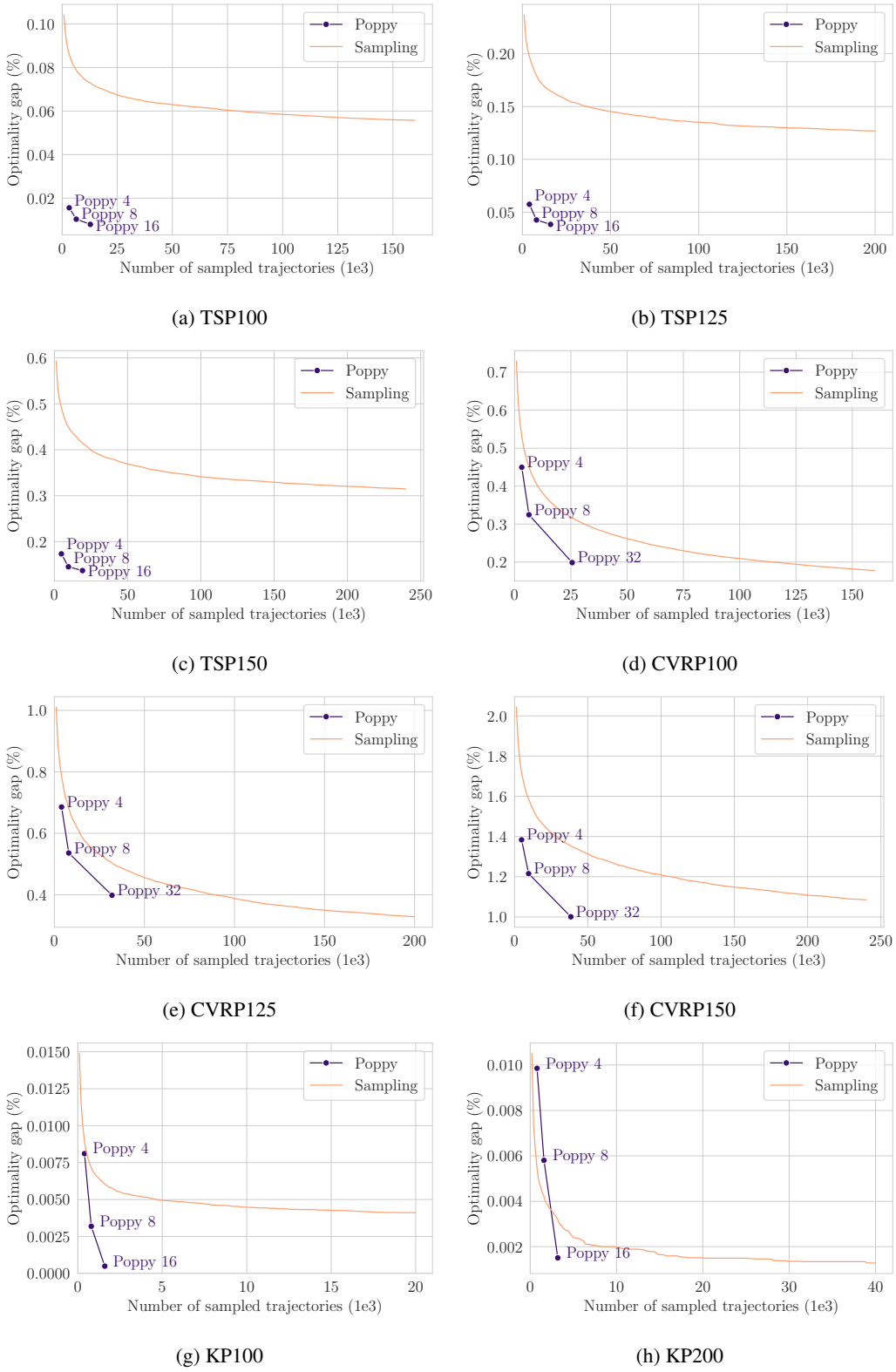


Figure 6: Example CVRP trajectories given by Poppy for 16 agents from a 32-agent population. The depot is displayed as a black square. The edges from/to the depot are omitted for clarity.

C TIME-PERFORMANCE TRADEOFF

We present on Figure 7 a comparison of the time-performance Pareto front between Poppy and POMO as we vary respectively the population size and the amount of stochastic sampling. Poppy consistently provides better performance for a fixed number of trajectories. Strikingly, in some environments like TSP100, TSP125, TSP150, CVRP150 and KP100, matching Poppy's performance by increasing the number of stochastic samples does not appear tractable.



(a) TSP100

(b) TSP125

(c) TSP150

(d) CVRP100

(e) CVRP125

(f) CVRP150

(g) KP100

(h) KP200

Figure 7: Comparison of the time-performance Pareto front of Poppy and POMO, for each problem used in the paper. The x-axis is the number of trajectories sampled per test instance, while the y-axis is the gap with the optimal solution for TSP and KP, and the gap with LKH3 for CVRP.