Ant Colony Sampling with GFlowNets for Combinatorial Optimization

Minsu Kim[*]
Mila, Université de Montréal, KAIST

Sanghyeok Choi[*]
KAIST

Hyeonah Kim[†]
Mila, Université de Montréal

Jiwoo Son
Omelet

Jinkyoo Park
KAIST, Omelet

Yoshua Bengio
Mila, Université de Montréal

Abstract

We present the Generative Flow Ant Colony Sampler (GFACS), a novel meta-heuristic method that hierarchically combines amortized inference and parallel stochastic search. Our method first leverages Generative Flow Networks (GFlowNets) to amortize a *multi-modal* prior distribution over combinatorial solution space that encompasses both high-reward and diversified solutions. This prior is iteratively updated via parallel stochastic search in the spirit of Ant Colony Optimization (ACO), leading to the posterior distribution that generates near-optimal solutions. Extensive experiments across seven combinatorial optimization problems demonstrate GFACS's promising performances.

1 Introduction

Combinatorial optimization (CO) aims to optimize an objective function over discrete, finite combinatorial structures under given constraints. It has broad applications in operations research (Zhang et al., 2019), scientific discovery (Naseri and Koffas, 2020) and machine learning (Chickering et al., 1995, 2004). Traditionally, integer programming approaches have been extensively studied to solve CO problems (Miller et al., 1960). While these methods offer guaranteed optimality, they often become impractical for large-scale instances due to their exponential complexity. An alternative is to design problem-specific heuristics, but it requires domain knowledge and labor-intensive design, limiting flexibility to variant problems.

Recent advancements in deep learning (LeCun et al., 2015) brought alternative directions to CO. For a more detailed review, refer to our related works in Appendix F and the survey by Bengio et al. (2021b). Among deep learning for CO methods, supervised learning (SL) approaches distill mathematical programming methods or heuristics into powerful deep neural networks, enabling fast and scalable solution generation (Vinyals et al., 2015; Joshi et al., 2021; Sun and Yang, 2023; Luo et al., 2023). However, SL methods rely on expert-labeled data, limiting their applicability as general-purpose CO solvers.

Reinforcement learning (RL) is another popular approach to CO (Khalil et al., 2017; Kool et al., 2019; Kwon et al., 2020; Ye et al., 2023). Unlike supervised learning, RL methods train deep neural networks directly using the objective function of a CO problem as a reward, without requiring expert-labeled data. This makes RL methods particularly useful because they can adapt flexibly to a wide range of CO problems, even where specific heuristics are unavailable or unexplored. However, RL requires extensive exploration of vast solution space and often involves non-trivial credit assignments across multiple decisions, which introduces significant complexity to the learning process compared to SL (Minsky, 1961).

Hierarchical methods that leverage RL in a two-stage problem-solving procedure tackle such exploration and credit assignment problems through hierarchical decomposition. Previous works by Bello et al. (2016), Hottung et al. (2022), and Son et al. (2023) pretrain

---

* Equal contribution. † Work done while Hyeonah Kim was at KAIST. Code available at: https://github.com/ai4co/gfacs. Correspondence to: Minsu Kim (*minsu.kim@mila.quebec*) and Sanghyeok Choi (*sanghyeok.choi@kaist.ac.kr*).
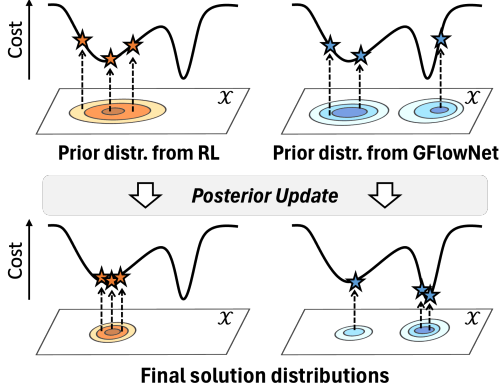
Figure 1: Synergy of multi-modal prior distribution of solutions and iterative posterior update with parallel stochastic search in combinatorial optimization.

large models on massive problem instances using RL and fine-tune these models on specific problem instances at test time using a method called active search (AS). Ye et al. (2023) pretrain relatively small models with significantly reduced training times. They pretrain a neural network with RL to produce a prior distribution over solution space and iteratively update the distribution using ant colony optimization (ACO), an effective parallel stochastic search algorithm.

Contribution. In this work, we propose a novel probabilistic meta-heuristic called Generative Flow Ant Colony Sampler (GFACS), which leverages the synergy between a multi-modal amortized prior and iterative update of it using parallel stochastic search. Unlike previous approaches that use RL pretraining aimed at maximizing expected rewards, we employ Generative Flow Networks (GFlowNets; Bengio et al., 2021a, 2023), designed to learn a distribution where the probability of a sample is proportional to its reward, *i.e.*, $p(x) \propto R(x)$. This leads to a multi-modal prior distribution of solutions, allowing it to benefit more from subsequent iterative posterior updates compared to the unimodal priors pretrained with reward maximizing RL, as illustrated in Figure 1. Moreover, to enhance and stabilize GFlowNet training, we introduce several new techniques that have been verified to be significantly more effective than existing GFlowNet training methods.

Our posterior update rules follow the most progressive methods of ant colony optimization (ACO) (Ye et al., 2023), creating synergies between our multi-modal amortized prior and their iterative refinement via parallel stochastic search. Additionally, we also validate our method with another posterior search method, active search (AS; Hottung et al., 2022), demonstrating the flexibility of our approach.

Through extensive experiments, we demonstrate the

superiority of our training methods over general-purpose reinforcement learning algorithms for providing effective prior distribution on ACO search across seven combinatorial optimization benchmarks:

- **Routing problems**:
  - Traveling Salesman Problem (TSP)
  - Capacitated Vehicle Routing Problem (CVRP)
  - CVRP with Time Windows (CVRPTW)
  - Prize-Collecting TSP (PCTSP)
  - Orienteering Problem (OP)
- **Scheduling problem**: Single Machine Total Weighted Tardiness Problem (SMTWTP)
- **Grouping problem**: Bin Packing Problem (BPP)

For problems such as the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP), we compare our GFACS method against problem-specific reinforcement learning algorithms that utilize significantly larger and specialized deep networks.

2 Preliminary

2.1 Traveling salesman problem

The traveling salesman problem (TSP) is a fundamental combinatorial optimization problem, which serves as a building block for more complex CO problems like the capacitated vehicle routing problem and orienteering problem. In light of this and for simplicity, this paper presents formulations using the two-dimensional Euclidean TSP as an illustrative example.

A TSP instance is defined on a (fully connected) graph $\mathcal{G} = (\mathcal{V}, \mathcal{D})$, where $\mathcal{V}$ and $\mathcal{D}$ are a set of nodes and edges, respectively. Each node $i \in \mathcal{V}$ has two-dimensional coordinates $v_i \in [0, 1]^2$ as a feature, and each edge $(i, j) \in \mathcal{D}$ has a feature $d_{i,j}$ which is the distance between node i and j, i.e., $d_{i,j} = \|v_i - v_j\|_2$. A solution to TSP, x corresponds to a Hamiltonian cycle for a given problem instance $\mathcal{G}$, which can be represented as a permutation of nodes $\pi = (\pi_1, \ldots, \pi_N)$. The tour length is computed as $\mathcal{E}(x; \mathcal{G}) = d_{\pi_N, \pi_1} + \sum_{t=1}^{N-1} d_{\pi_t, \pi_{t+1}}$. The optimization objective for TSP is to find x that minimizes $\mathcal{E}(x; \mathcal{G})$.

2.2 Generative flow networks

Generative Flow Networks, or GFlowNets (Bengio et al., 2021a, 2023), provide an effective training framework for learning a policy that samples compositional objects $x \in \mathcal{X}$. The objective is to generate x with probability proportional to a given non-negative reward function $R(x)$. GFlowNets follow a constructive generative process, using discrete *actions* to iteratively modify a *state* which represents a partially
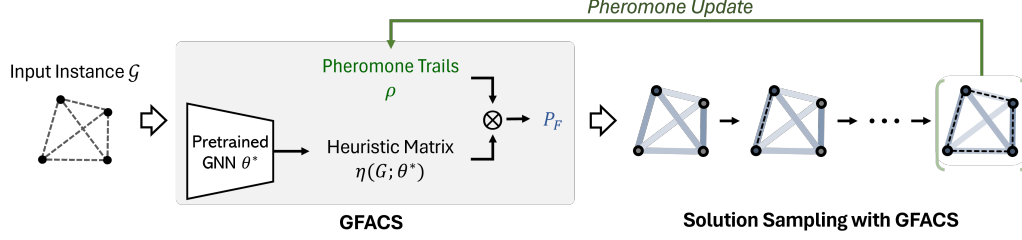
Minsu Kim[*], Sanghyeok Choi[*], Hyeonah Kim, Jiwoo Son, Jinkyoo Park, Yoshua Bengio

Figure 2: Solution sampling process of GFACS. The GNN, pretraind with a GFlowNet loss, serves as an expert heuristic to pick the next action to construct a solution, such as a tour in the TSP.

constructed object. All the ways to construct these objects can be described by a directed acyclic graph (DAG), $\mathcal{H} = (\mathcal{S}, \mathcal{A})$, where $\mathcal{S}$ is a finite set of all possible states, and $\mathcal{A}$ is a subset of $\mathcal{S} \times \mathcal{S}$, representing directed edges. Within this framework, we define the *children* of state $s \in \mathcal{S}$ as the set of states connected by edges whose head is s, and the *parents* of state s as the set of states connected by edges whose tail is s.

We define a (complete) *trajectory* $\tau = (s_0 \to \ldots \to s_N) \in \mathcal{T}$ from the initial state s_0 to terminal state s_n. We define the *forward policy* to model the forward transition probability $P_F(s'|s)$ from s to its child s'. Similarly, we also consider the *backward policy* $P_B(s|s')$ for the backward transition $s' \dashrightarrow s$, where s is a parent of s'. We can use the *forward policy* to compute in a forward way the probability of a trajectory: $P_F(\tau; \theta) = \prod_{t=1}^{n} P_F(s_t|s_{t-1}; \theta)$. Similarly, we can use the *backward policy* to compute in a backward way the probability of trajectory ending at a given object $x = s_n$: $P_B(\tau|x; \theta) = \prod_{t=1}^{n} P_B(s_{t-1}|s_t; \theta)$.

Trajectory balance (Malkin et al., 2022). Trajectory balance (TB) is the most widely used training objective for GFlowNets. It works with three key components: a learnable scalar Z_θ for the partition function, a learnable forward policy $P_F(\tau; \theta)$, and a backward policy $P_B(\tau|x)$ that can either be learnable or fixed with a predefined configuration (Zhang et al., 2022a). TB loss to minimize for any trajectory τ is defined as:

$$\mathcal{L}_{\text{TB}}(\tau; \theta) = \left(\log \frac{Z_\theta P_F(\tau; \theta)}{R(x) P_B(\tau|x)} \right)^2. \qquad (1)$$

When the trajectory balance loss is minimized to zero for all complete trajectories, the probability of sampling x with the forward policy is $P_F^\top(x) \propto R(x)$, as desired, where $P_F^\top(x) = \sum_{(\tau \to x) \in \mathcal{T}} P_F(\tau)$ with $(\tau \to x)$ indicating that the terminal state of τ is x.

2.3 Ant colony optimization

Ant colony optimization (ACO; Dorigo et al., 2006) is a meta-heuristic to refine the distribution over solution

space by an iterative procedure of solution construction and pheromone update. Given a problem instance $\mathcal{G}$, a corresponding heuristic matrix $\eta(\mathcal{G}) \in \mathbb{R}^{N \times N}$ and a pheromone trails $\rho \in \mathbb{R}^{N \times N}$, a solution is constructed by auto-regressively sampling edges from the probability distribution:

$$P(\pi_{t+1} = j | \pi_t = i; \rho, \eta(\mathcal{G})) \propto \begin{cases} \rho_{i,j} \eta_{i,j}(\mathcal{G}) & \text{if } j \notin \pi_{1:t} \\ 0 & \text{otherwise,} \end{cases}$$
$$(2)$$

where $\rho_{i,j}$ and $\eta_{i,j}(\mathcal{G})$ are the pheromone weight and heuristic weight for edge $d_{i,j}$, respectively.

At each iteration, K artificial ants independently sample solutions $\{\pi^k\}_{k=1}^{K}$ by following Eq.(2). The pheromone trails ρ are then adjusted using the sampled solutions and their corresponding costs, thereby updating the solution distribution. The exact pheromone update rules depend on the specific ACO algorithm being used. As a representative example, in Ant System (Dorigo et al., 1996), all K solutions contribute to the pheromone update as follows:

$$\rho_{i,j} \leftarrow (1 - \gamma)\rho_{i,j} + \sum_{k=1}^{K} \Delta_{i,j}^k, \qquad (3)$$

where γ is the decaying coefficient (or evaporation rate) and $\Delta_{i,j}^k$ is the amount of pheromone laid by the k-th ant on the edge (i, j). The value of $\Delta_{i,j}^k$ is proportional to the inverse of tour length if edge (i, j) is part of π^k, and zero otherwise:

$$\Delta_{i,j}^k = \begin{cases} C/f(\pi^k) & \text{if } (i, j) \in \pi^k \\ 0 & \text{otherwise,} \end{cases} \qquad (4)$$

where C is a constant. The smaller the cost $f(\pi^k)$, the more pheromone is laid on the edges of π^k. This mechanism encourages future ants to follow promising paths that lead to high-quality solutions.

Note that the heuristic matrix $\eta(\mathcal{G})$ is not updated during the search process; it only serves as an initial *prior*. This prior is augmented with the pheromone ρ that is typically initialized as a constant matrix and then iteratively updated, resulting in the posterior solution distribution that converges to the Dirac-delta distribution

of optimal solution under certain conditions (Dorigo and Gambardella, 1997; Dorigo and Blum, 2005).

In DeepACO (Ye et al., 2023), the learned graph neural network (GNN), parameterized by θ, is utilized to construct the heuristic matrix $\eta(\mathcal{G}, \theta)$ for a given problem instance $\mathcal{G}$, providing an informative prior for guiding probabilistic solution search process of ACO. The limitation of the approach is that they use REINFORCE (Williams, 1992) for GNN training, which suffers from mode-collapse, restricting further improvement from the subsequent iterative pheromone update. Our work seeks to overcome this limitation by incorporating a diversified, multi-modal solution distribution, which shares similar motivation with approximate global optimization methods such as simulated annealing (Kirkpatrick et al., 1983).

3 Generative flow ant colony sampler

In this section, we provide the inference procedure for the Generative Flow Ant Colony Sampler (GFACS). GFACS enhances Ant Colony Optimization (ACO) by integrating it with multi-modal prior from GFlowNet-based training.

3.1 MDP formulation

We illustrate the solution construction procedure as a Markov decision process (MDP), using the Traveling Salesman Problem (TSP) as an example. This formulation can be easily adapted to general combinatorial optimization problems with minor modifications.

State s: The initial state s_0 is an empty set. The state s_t at $0 < t < n$ is a partial combinatorial solution presented with previous actions: $\{a_0, \ldots, a_{t-1}\}$ and s_n is a completed solution.

Action a: The action a_t is a selection of the next node to visit at time t among un-visited nodes: $a_t \in \{1, \ldots, N\} \setminus \{a_0, \ldots, a_{t-1}\}$. If all nodes are visited, a deterministic terminating action is applied, which maps a final state s_N into a solution, x.

Reward R: The reward is given by $R(x; \mathcal{G}, \beta) = e^{-\beta \mathcal{E}(x;\mathcal{G})}$. Here, $\mathcal{E}(x; \mathcal{G})$ represents the energy of the solution x, which corresponds to metrics such as the tour length in the TSP. The inverse temperature β is a hyperparameter. For simplicity, we will omit $\mathcal{G}$ from R and $\mathcal{E}$ except when its inclusion is necessary for clarity.

3.2 Policy configuration for prior modeling

We parameterize the forward policy P_F using a graph neural network (GNN) with parameter θ that maps an input graph instance $\mathcal{G}$ to the output edge matrix

$\eta(\mathcal{G}; \theta)$, following Ye et al. (2023). For simplicity, we use notation $\eta(\mathcal{G})$ without θ. With a given $\eta(\mathcal{G})$, the forward policy when $t < N$ is expressed as follows:

$$P_F(s_{t+1}|s_t; \eta(\mathcal{G}), \rho) \propto \begin{cases} \rho_{a_{t-1}, a_t} \eta_{a_{t-1}, a_t}(\mathcal{G}) & \text{if } a_t \notin s_t \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Note that ρ and $\eta(\mathcal{G})$ denote the pheromone trails and learned heuristic matrix in ACO. Subsequently, we can define a trajectory from the initial state to a terminal solution, with $\tau = (s_0, \ldots, s_N, x)$, and the forward probability of that trajectory as

$$P_F(\tau; \eta(\mathcal{G}), \rho) = \prod_{t=0}^{N-1} P_F(s_{t+1}|s_t; \eta(\mathcal{G}), \rho). \quad (6)$$

The forward probability for the transition from s_N to x is omitted since it is deterministic. During training, we fix ρ to $\mathbf{1}$, and thus we omit it from subsequent sections unless its presence is required for clarity.

For the backward policy, we first define $h(x)$ as the number of possible trajectories that yield the solution x, and we set $P_B(\tau|x) = \frac{1}{h(x)}$. This setting emulates the maximum entropy GFlowNets (Zhang et al., 2022a), treating all symmetric trajectories that terminate in the same solution equally. In the case of the TSP, where $h(x) = 2N$, we have $P_B = \frac{1}{2N}$, thereby imposing a uniform backward distribution across all $2N$ feasible trajectories leading to the same Hamiltonian cycle x. It is important to note that the number of symmetric trajectories varies depending on the specific CO problem; for a more detailed discussion on solution symmetry across various CO problems, please refer to Appendix E

3.3 ACO update on learned prior

In the sampling process, we assume the existence of the learned GNN parameterized with θ^*; see Section 4 for details of the training procedure. This learned GNN provides the prior parameter $\eta^*(\mathcal{G})$ for the ACO algorithm. The next step is to update the pheromone parameter ρ, using K agents (ants) to search over samples drawn from the amortized prior $P_F(\tau; \eta^*(\mathcal{G}))$ as described in Section 2.3.

Since the trained $\eta^*(\mathcal{G})$ samples diverse, high-reward (*i.e.*, low-energy) candidates, the distribution P_F provides strong prior candidates for the ants in the ACO algorithm. This learned prior is expected to highly improve the likelihood of discovering the global optimum after the iterations of the pheromone update.

The overall process is illustrated in Figure 2.

Minsu Kim*, Sanghyeok Choi*, Hyeonah Kim, Jiwoo Son, Jinkyoo Park, Yoshua Bengio
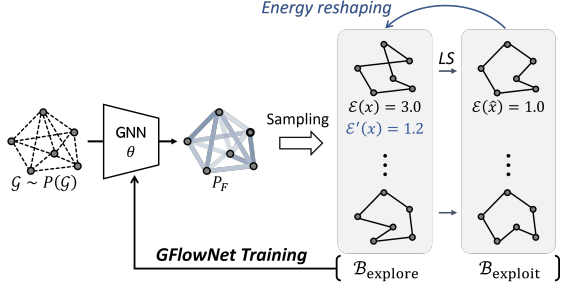
Figure 3: Experience collection procedure. Energy reshaping compensates for the energy of underrated samples that have potential to become low-energy samples after the local search.

4 Off-policy training of GFACS

This section introduces how to train GFACS, especially the GNN component parameterized with θ, in the GFlowNet framework. GFACS is trained via iterations of the off-policy experience collection (Section 4.1) and training (Section 4.2).

4.1 Off-policy experience collection

In this step, we collect experiences on given graph instances. The collected experiences will be used to train GFACS, as described in Section 4.2. At each training round, we first sample M problem instances from a random instance generator $P(\mathcal{G})$, i.e., $\mathcal{G}_1, \ldots, \mathcal{G}_M \sim P(\mathcal{G})$. Details of instance generation are described in Appendix B. For each problem instance $\mathcal{G}_m$, we collect K solutions by following Eq.(6), resulting in a batch of "exploration experiences,"

$$\mathcal{B}_{\text{explore}} = \{(\mathcal{G}_m, \tau_{m,k}, \mathcal{E}(x_{m,k})) \mid m \in [M], k \in [K]\}.$$

To exploit high-reward regions in the large-scale combinatorial space, we collect additional K experiences for each $\mathcal{G}_m$ using a local search (LS) operator, such as 2-opt for TSP, as a powerful behavior policy. The LS operator directly refines a given solution into a locally optimized one, i.e., $\hat{x} \leftarrow \text{LS}(x)$. We then use the backward policy P_B to sample a trajectory $\hat{\tau}$ for each $\hat{x}$, obtaining a batch of "exploitation experiences,"

$$\mathcal{B}_{\text{exploit}} = \{(\mathcal{G}_m, \hat{\tau}_{m,k}, \mathcal{E}(\hat{x}_{m,k})) \mid m \in [M], k \in [K]\}.$$

For problems that lack domain-specific LS operators, we also suggest a general-purpose destroy-and-repair method that partially destroys a solution and then repairs; see Appendix C for more details.

While the concept of using local search for exploitation has been explored in recent GFlowNet works (Kim

et al., 2024b), our distinct approach involves utilizing the local search operation to facilitate *hindsight observation*—a forward-looking insight derived from extending the search beyond the current state. Methodologically, we compensate for the solutions that were initially overlooked due to high energy (*i.e.*, low reward) but that can be readily transformed into low energy (*i.e.*, high reward) solutions through local search.

To achieve this, we modify the energy (in a way we call *energy reshaping*) of x in the *exploration* experiences, $\mathcal{B}_{\text{explore}}$. Specifically, for a given solution $x \in \mathcal{B}_{\text{explore}}$ and its corresponding refined solution $\hat{x} \leftarrow \text{LS}(x)$, we adjust the energy of x as follows:

$$\mathcal{E}'(x) = \alpha \mathcal{E}(\hat{x}) + (1 - \alpha)\mathcal{E}(x). \quad (7)$$

This reshaping captures the potential for samples to become low-energy solutions after local search.

The overall procedure of experience collection via sampling and local search, along with the energy reshaping, is illustrated in Figure 3.

4.2 Training with collected experiences

Given a batch of experiences $\mathcal{B}$, the trajectory balance (TB) loss in Eq.(1) can be rewritten as

$$L(\mathcal{B}) = \frac{1}{MK} \sum_{m=1}^{M} \sum_{k=1}^{K} \left(\log \frac{P_F(\tau_{m,k}; \eta(\mathcal{G}_m))Z(\mathcal{G}_m)}{(1/h(x_{m,k}))R(x_{m,k}; \mathcal{G}_m, \beta)} \right)^2, \quad (8)$$

where $1/h(x_{m,k})$ is to the uniform backward policy as described in Section 3.2. Since the loss is computed over multiple instances $\mathcal{G}_m$, the objective corresponds to the conditional trajectory balance.

As discussed in Section 2.2, our target sampling distribution is proportional to reward, *i.e.*, $p(x; \mathcal{G}) \propto R(x; \mathcal{G}, \beta)$. However, in most CO problems, including TSP, the scale of $R(x; \mathcal{G}, \beta)$ can vary significantly across instances, possibly leading to unstable training due to the high variance of logarithms of rewards. Similar to previous works that stabilize REINFORCE training (Kwon et al., 2020; Kim et al., 2022), we normalize the energy $\mathcal{E}$ using the sample mean over K samples for the same $\mathcal{G}_m$ as follows:

$$\tilde{\mathcal{E}}(x_{m,k}; \mathcal{G}_m) = \mathcal{E}(x_{m,k}; \mathcal{G}_m) - \frac{1}{K} \sum_{k=1}^{K} \mathcal{E}(x_{m,k}; \mathcal{G}_m). \quad (9)$$

Accordingly, we have $\tilde{R}(x_{m,k}; \mathcal{G}_m, \beta) = e^{-\beta \tilde{\mathcal{E}}(x_{m,k}; \mathcal{G}_m)}$. We call this technique a *shared energy normalization*. Note that this technique is applied independently to the two batches, $\mathcal{B}_{\text{explore}}$ and $\mathcal{B}_{\text{exploit}}$, as the energy distribution between the two batches is often significantly different from each other, especially when a heuristic local search is applied. By applying shared energy
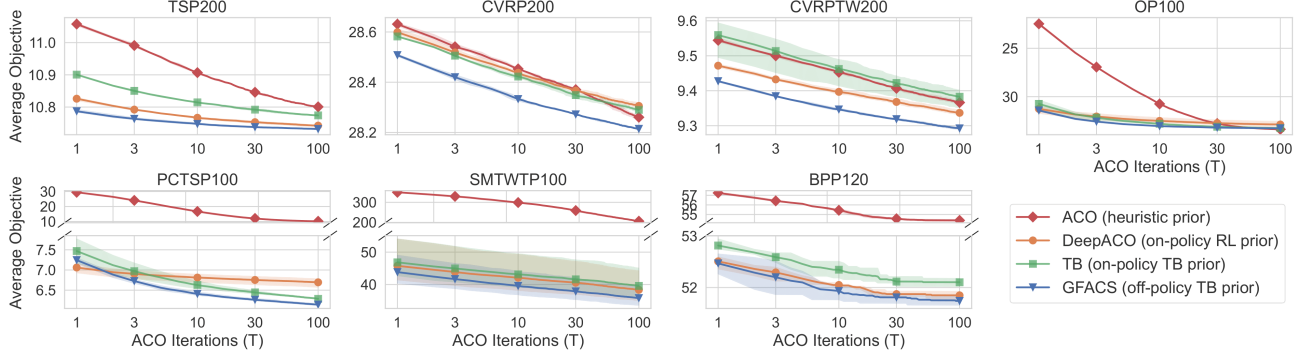
Figure 4: Results of ACO algorithms with different priors on various CO tasks. Our GFACS outperforms every ACO baseline. The results are averaged over 3 independent models evaluated on the held-out test sets, and the shade indicates the min-max range of the 3 models.

normalization to both batches, we obtain $\tilde{\mathcal{B}}_{\text{explore}}$ and $\tilde{\mathcal{B}}_{\text{exploit}}$, in which every $\mathcal{E}$ is replaced by $\tilde{\mathcal{E}}$.

The final loss to minimize is then defined as

$$\mathcal{L}(\theta) := \frac{1}{2}L(\mathcal{B}_{\text{explore}}) + \frac{1}{2}L(\mathcal{B}_{\text{exploit}}), \qquad (10)$$

where L is as defined in Eq.(8). Note the parameters θ govern the functions $Z(\mathcal{G}_m)$ and $P_F(\cdot; \eta(\mathcal{G}_m))$, although this dependence has been omitted for simplicity. The gradient of the loss with respect to θ is given by $\nabla_\theta \mathcal{L}(\theta)$, and the parameters are optimized using stochastic gradient descent methods.

5 Experiments

This section presents experimental results to validate our algorithm. We compare GFACS against other ACO algorithms with different priors and also against competitive reinforcement learning algorithms for vehicle routing problems. Moreover, we conducted an ablation study and an empirical comparison with previous GFlowNets training methods. Last but not least, we validate the efficacy of multi-modal prior with another posterior update method beyond ACO, the active search (AS). Below, we provide a brief overview of the experimental setup.

Implementation. GFACS is built upon the implementation established by DeepACO (Ye et al., 2023). As our methodology mainly focuses on improving training algorithms rather than network architecture, we simply adopt the GNN architecture proposed in DeepACO, except the additional parameters for partition function $Z(\cdot; \theta)$ in Eq.(8). For detailed network architecture, hyperparameters, and hardware settings, please refer to Appendix A. It is worth noting that one important hyperparameter that GFACS newly introduces is the inverse energy temperature β, which enables us to control the diversity-optimality trade-

Table 1: Experimental results on real-world datasets, TSPLib and CVRPLib. 'N' denotes the range of instance size and '#' denotes the number of instances in each set. We report the average optimality gap using the best-known costs.

	N	#	ACO	DeepACO	GFACS (ours)
TSPLib	100-299	30	1.71%	1.25%	**1.21**%
	300-699	10	4.26%	2.70%	**2.60**%
	700-1499	12	7.01%	4.08%	**3.88**%
CVRPLib	100-299	43	**2.50**%	3.27%	2.69%
	300-699	40	3.75%	4.22%	**3.71**%
	700-1001	17	4.64%	5.07%	**4.32**%

offs (see Appendix D.2 for detailed analysis). In addition, we employ Ant System (Section 2.3) for the pheromone updates throughout the experiments. We provide experimental results with other ACO variants in Appendix D.3.

Test settings. Each algorithm is tested on the held-out datasets, each with 100 instances generated in advance except TSP, where we obtain the TSP test dataset with 128 instances from Ye et al. (2023). Unless otherwise stated, the results presented throughout this section are the averaged values from three independent models, each trained with a distinct seed.

5.1 Comprison with different priors

Baselines. We compare GFACS against the vanilla ACO (heuristic prior) and the DeepACO (Ye et al., 2023) (on-policy RL prior) and ACO with TB prior as baseline methods. We aim to show our multi-modal prior from our off-policy GFlowNets training methods is beneficial when this is integrated with the iterative pheromone update of ACO. We use $K = 100$ ants and $T = 100$ ACO iterations for all algorithms.

Problems. We evaluate our algorithm in a range

Minsu Kim*, **Sanghyeok Choi***, **Hyeonah Kim, Jiwoo Son, Jinkyoo Park, Yoshua Bengio**

Table 2: Experimental results on TSP and CVRP. Results for methods with * are drawn from Ye et al. (2023), Cheng et al. (2023), and Jin et al. (2023). The reported objective value is averaged over 128 instances of TSP and 100 instances of CVRP. We report the average computation time per instance.

	TSP500			TSP1000				CVRP500			CVRP1000		
Method	Obj.	Gap(%)	Time	Obj.	Gap(%)	Time	Method	Obj.	Gap(%)	Time	Obj.	Gap(%)	Time
Concorde	16.55	-	10.7s	23.12	-	108s	PyVRP	62.96	-	21.0m	119.29	-	1.4h
LKH-3	16.55	0.02	31.7s	23.14	0.10	102s	LKH-3	63.07	0.18	18.6m	119.52	0.20	2.1h
AM*	21.46	29.07	0.6s	33.55	45.10	1.5s							
POMO*	20.57	24.40	0.6s	32.90	42.30	4.1s	POMO	68.69	9.08	2s	145.74	22.14	3s
+EAS	18.25	10.29	300s	30.77	33.06	600s	+EAS	64.76	2.85	5.0m	126.25	5.83	10.0m
+SGBS	18.86	13.98	318s	28.58	23.61	605s	+SGBS	65.44	3.93	5.2m	127.90	7.21	10.1m
DIMES*	17.01	2.78	11s	24.45	5.75	32s	Sym-NCO	68.81	9.28	2s	141.82	18.86	3s
SO*	16.94	2.40	15s	23.77	2.80	26s	+EAS	64.63	2.65	5.0m	125.58	5.27	10.0m
Pointerformer*	17.14	3.56	14s	24.80	7.90	40s	+SGBS	65.40	3.87	5.2m	127.53	6.90	10.1m
ACO	17.38	5.04	14s	24.76	7.09	57s	ACO	64.62	2.63	36s	122.82	2.96	1.3m
DeepACO	16.84	1.77	15s	23.78	2.87	66s	DeepACO	64.49	2.43	36s	122.15	2.40	1.3m
GFACS (ours)	**16.80**	**1.56**	15s	**23.72**	**2.63**	66s	GFACS (ours)	**64.19**	**1.95**	36s	**121.80**	**2.11**	1.3m

of CO problems, including five routing problems – TSP, Capacitated Vehicle Routing Problem (CVRP), CVRP with Time Window (CVRPTW), Prize-Collecting TSP (PCTSP), and Orienteering Problem (OP) –, a scheduling problem, the Single Machine Total Weighted Tardiness Problem (SMTWTP), and a grouping problem, the Bin Packing Problem (BPP). Detailed descriptions and configurations for each problem can be found in Appendix B. For TSP, CVRP and CVRPTW, we use local search heuristics (2-opt, Swap* and `LocalSearch` operator in PyVRP (Wouda et al., 2024), respectively) with perturbation scheme similar to Ye et al. (2023) for training GFACS and testing all the ACO algorithms. For others, we use the destroy-and-repair method. In Appendix C, further details are provided.

Results. Figure 4 shows GFACS consistently outperforms both the classic ACO algorithm and DeepACO across all benchmark problems. Notably, for routing problems, GFACS surpasses DeepACO by a significant margin. This is particularly remarkable given that both algorithms use identical neural networks and local search operators, underscoring the importance of the multi-modal prior. Additionally, the proposed off-policy training method enhances GFACS performance, demonstrating its effectiveness for GFlowNets.

Evaluation on real-world datasets. We compare the performance of our model against baseline models on real-world TSP and CVRP instances from the TSPLib (Reinelt, 1991) and CVRPLib X-instances (Uchoa et al., 2017). The models were trained on random uniform instances of TSP/CVRP with sizes 200 for dataset sizes 100-299, 500 for sizes 300-699, and 1000 for sizes 700-1499, with all other settings remaining the same. Table 1 presents the results. GFACS outperforms the baselines in TSPLib and large CVR-PLib instances but slightly underperforms compared

to vanilla ACO on small CVRPLib instances, where ACO with Swap* local search suffices for small-scale CVRP. As the problem scale increases, ACO's performance declines, underscoring the importance of effective priors using GFlowNets for efficient pheromone posterior updates in large combinatorial spaces.

5.2 Comparison with RL solvers

In this section, we present experimental results comparing our method with problem-specific RL approaches that leveraged specialized neural architectures or learning heuristics for the TSP and CVRP.

Baselines. For TSP baselines, we select the AM (Kool et al., 2019), POMO (Kwon et al., 2020), Sym-NCO (Kim et al., 2022) and Pointerformer (Jin et al., 2023) as RL-based constructive methods, DIMES (Qiu et al., 2022) as RL-based heatmap-based methods, the Select and Optimize (SO; Cheng et al., 2023) as RL-based improvement methods. We also applied two search algorithms, the Efficient Active Search (EAS; Hottung et al., 2022) and Simulation Guided Beam Search (SGBS; Choo et al., 2022), to POMO for a more thorough comparison. For CVRP,[1] we employ POMO and Sym-NCO as baselines with EAS and SGBS. Note that we restrict the duration of these searches to no more than 5 and 10 minutes for problems with 500 and 1000 nodes, respectively. We use Concorde (Applegate et al., 2006) for TSP and PyVRP (Wouda et al., 2024) for CVRP as oracle heuristic solvers to obtain near-optimal solutions that serve as baselines with 0.00% gap. For ACO algorithms, we use $K = 100$ ants with $T = 10$ with local search algorithms similar to Ye et al. (2023).

---

[1]Due to the inaccessibility of source codes, we compare ours with TAM using the reported result in the paper (Hou et al., 2023). See Appendix D.4.

Table 3: Results of the ablation study.

Method	TSP500	CVRP500	BPP120
GFACS	**16.80**	**64.19**	**51.84**
− *Off-policy*	17.09	64.42	52.32
− *Energy Reshaping*	16.87	64.41	52.07
− *Shared Energy Norm.*	17.52	65.51	90.73
+ *VarGrad*	16.86	64.28	51.95

Results. GFACS consistently shows superior or highly competitive performance compared to evaluated baseline methods (Table 2). Ours outperforms fast zero-shot methods like AM and POMO—even when they employ search techniques such as EAS or SGBS—and achieves better results than similarly paced algorithms like DIMES and Pointformer. Although it performs slightly worse than the heuristic solver LKH-3 (Helsgaun, 2017), GFACS offers faster inference times, especially in CVRP. As a general-purpose algorithm, it holds significant appeal in the competitive benchmarks of both TSP and CVRP. Notably, despite its small neural architecture and requiring less than three hours of training, GFACS outperforms larger models like POMO and Sym-NCO, which need over a week of training (Kim et al., 2022).

5.3 Ablation study

This section reports the ablation study results, validating our proposed training techniques for stabilizing and enhancing GFlowNets in large-scale combinatorial optimization tasks. As detailed in Table 3, our off-policy training method, incorporating local search, energy reshaping to value potential solutions, and shared ϵ-norm to stabilize conditional GFlowNets training, significantly improves performance across three tasks: TSP, CVRP, and BPP. For comprehensive results, including standard deviations, refer to Appendix D.5.

5.4 Comparison with other GFlowNet-based training methods

While this paper proposes a novel training method for GFlowNets designed for large-scale combinatorial optimization problems that are not locally decomposable due to their cyclic nature, such as in TSP and CVRP, we also compare existing GFlowNets training methods within this domain.

Forward-looking detailed balance (FL-DB; Pan et al., 2023; Zhang et al., 2023). FL-DB leverages local credits when the application benefits from useful local rewards for partial solutions. While graph combinatorial optimization tasks are generally locally decomposable (Ahn et al., 2020), yielding useful local rewards, tasks like TSP do not offer substantial local rewards due to their cyclic nature, which imposes a global con-
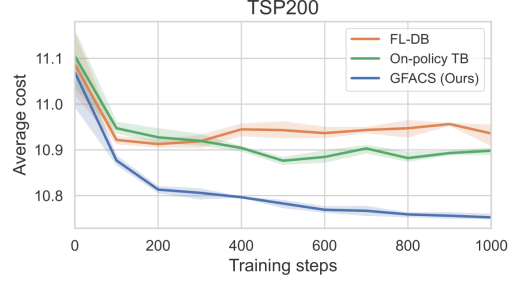

Figure 5: Validation cost on TSP with 200 nodes during training, compared to forward-looking detailed balance (FL-DB).

straint, thus complicating local credit assignment. As shown in Figure 5, various GFlowNet methods were used to train the ACO's heuristic matrix $\eta(\mathcal{G})$. While FL-DB did not surpass on-policy TB for TSP, our off-policy method demonstrated significant improvement. This trend holds across different deep learning models and inference tasks beyond ACO; see Appendix D.1.

Gradient of log-partition variance (VarGrad). Originally proposed by Richter et al. (2020) for variational inference, the VarGrad loss was applied to conditional GFlowNets for scheduling problems by Zhang et al. (2022b). This approach is vital for combinatorial optimization tasks requiring conditional inference on the problem instance $\mathcal{G}$. It computes the log-partition function via the consistency property over trajectories conditioned on identical instances, thus bypassing explicit training of $\log Z(\mathcal{G})$. Explicit training is challenging due to varying energy landscapes across different instances. Our method, utilizing explicit training of the log-partition $\log Z(\mathcal{G})$ as in Eq.(8), outperforms VarGrad in training conditional GFlowNets (Table 3). This improvement stems from our *Shared energy normalization*, which stabilizes training by normalizing energy values to sample means, resulting in an energy distribution with a zero mean across instances. Note that accurately estimating the explicit log-partition value is known to stabilize training of GFlowNets (Malkin et al., 2023).

5.5 Integrating GFACS with active search

Background. In this section, we explore the integration of active search (AS) as a posterior update on a parameterized prior distribution $p_\theta(\cdot|\mathcal{G})$. In AS, the prior parameter θ, pretrained on a large set of random problem instances $\mathcal{G}$, is adaptively updated for specific test-time instances $\mathcal{G}_{\text{test}}$ using the same objective function as in the prior training. We trained this prior p_θ using two methods: on-policy RL method (Kwon et al., 2020) that employ REINFORCE with shared baselines, and our off-policy GFlowNet training method as explained in Section 4.

Minsu Kim[∗]**, Sanghyeok Choi**[∗]**, Hyeonah Kim, Jiwoo Son, Jinkyoo Park, Yoshua Bengio**

Table 4: Evaluation results combine with active search (AS) as a posterior search method.

Method	TSP200		CVRP200	
	Obj.	Gap (%)	Obj.	Gap (%)
Concorde / PyVRP	10.72	0.00	27.93	0.00
RL prior + AS	10.79 ± 0.00	0.65	28.43 ± 0.02	1.79
GFlowNet prior + AS	$\mathbf{10.75} \pm 0.00$	**0.29**	$\mathbf{28.34} \pm 0.01$	**1.47**

Setting. For the active search procedure, we sample 20 solutions per test-time instance and perform 5 gradient updates at each iteration, updating θ over a total of 50 iterations.

Results. As shown in Table 4, the off-policy GFlowNet prior combined with AS posterior inference outperforms the RL counterpart. This result provides further evidence supporting our hypothesis that the multi-modal prior from GFlowNet training synergizes effectively with the subsequent iterative posterior update algorithm. Additionally, the results demonstrate the flexibility of our method beyond the ACO, suggesting a promising direction for future research into the benefits of using a multi-modal informative prior.

6 Conclusion

In this paper, we introduced a novel probabilistic meta-heuristic that hierarchically combines GFlowNets-based amortized multi-modal priors with posterior search using ant colony optimization (ACO) for combinatorial optimization problems. Our strategy achieved remarkable success across seven benchmark combinatorial optimization tasks. Furthermore, by applying our meta-heuristic to other posterior search methods like active search (AS), we demonstrated the expandability and modularity of our approach. Limitation of our work is that, while our method empirically improves solution quality, we do not provide formal theoretical guarantees in how our method gives better optimality than the reinforcement learning counterpart.

Acknowledgment

References

Ahn, S., Seo, Y., and Shin, J. (2020). Learning what to defer for maximum independent sets. In *International Conference on Machine Learning (ICML)*.

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

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

Bengio, E., Jain, M., Korablyov, M., Precup, D., and Bengio, Y. (2021a). Flow network based generative models for non-iterative diverse candidate generation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Bengio, Y., Lahlou, S., Deleu, T., Hu, E. J., Tiwari, M., and Bengio, E. (2023). Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55.

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

Berto, F., Hua, C., Park, J., Kim, M., Kim, H., Son, J., Kim, H., Kim, J., and Park, J. (2023). RL4CO: a unified reinforcement learning for combinatorial optimization library. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*.

Chalumeau, F., Surana, S., Bonnet, C., Grinsztajn, N., Pretorius, A., Laterre, A., and Barrett, T. (2023). Combinatorial optimization with policy adaptation using latent space search. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Cheng, H., Zheng, H., Cong, Y., Jiang, W., and Pu, S. (2023). Select and optimize: Learning to aolve large-scale tsp instances. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Chickering, D. M., Geiger, D., and Heckerman, D. (1995). Learning bayesian networks: Search methods and experimental results. In *Artificial Intelligence and Statistics (AISTATS)*.

Chickering, M., Heckerman, D., and Meek, C. (2004). Large-sample learning of bayesian networks is np-hard. *Journal of Machine Learning Research*, 5:1287–1330.

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

Croes, A. (1958). A method for solving traveling salesman problems. *Operations Research*, 5:791–812.

d O Costa, P. R., Rhuggenaath, J., Zhang, Y., and Akcay, A. (2020). Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian Conference on Machine Learning (ACML)*.

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical computer science*, 344(2-3):243–278.

Dorigo, M. and Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41.

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

Elfwing, S., Uchibe, E., and Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11.

Falkenauer, E. and Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1186–1192.

Fischetti, M., Gonzalez, J. J. S., and Toth, P. (1998). Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148.

Fu, Z.-H., Qiu, K.-B., and Zha, H. (2021). Generalize a small pre-trained model to arbitrarily large tsp instances. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Grinsztajn, N., Furelos-Blanco, D., and Barrett, T. D. (2023). Winner takes it all: Training performant rl populations for combinatorial optimization. *Advances in Neural Information Processing Systems (NeurIPS)*.

Helsgaun, K. (2017). An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, pages 966–980.

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

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

Hou, Q., Yang, J., Su, Y., Wang, X., and Deng, Y. (2023). Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *International Conference on Learning Representations (ICLR)*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*.

JIANG, Y., Cao, Z., Wu, Y., Song, W., and Zhang, J. (2023). Ensemble-based deep reinforcement learning for vehicle routing problems under distribution shift. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jin, Y., Ding, Y., Pan, X., He, K., Zhao, L., Qin, T., Song, L., and Bian, J. (2023). Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Joshi, C. K., Cappart, Q., Rousseau, L.-M., and Laurent, T. (2021). Learning TSP requires rethinking generalization. In *International Conference on Principles and Practice of Constraint Programming (CP)*.

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

Kim, H., Kim, M., Ahn, S., and Park, J. (2024a). Symmetric Replay Training: Enhancing sample efficiency in deep reinforcement learning for combinatorial optimization. In *International Conference on Machine Learning (ICML)*.

Kim, M., Park, J., and Kim, j. (2021). Learning collaborative policies to solve np-hard routing problems. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Kim, M., Park, J., and Park, J. (2022). Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Kim, M., Park, J., and Park, J. (2023). Learning to CROSS exchange to solve min-max vehicle routing problems. In *International Conference on Learning Representations (ICLR)*.

Kim, M., Yun, T., Bengio, E., Zhang, D., Bengio, Y., Ahn, S., and Park, J. (2024b). Local search GFlowNets. In *International Conference on Learning Representations (ICLR)*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Kool, W., van Hoof, H., Gromicho, J., and Welling, M. (2022). Deep policy dynamic programming for vehicle routing problems. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*.

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

Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., and Min, S. (2020). POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Li, Y., Guo, J., Wang, R., and Yan, J. (2023). From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Loshchilov, I. and Hutter, F. (2017a). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Loshchilov, I. and Hutter, F. (2017b). SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*.

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

Ma, Y., Cao, Z., and Chee, Y. M. (2023). Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ma, Y., Li, J., Cao, Z., Song, W., Guo, H., Gong, Y., and Chee, M. C. (2022). Efficient neural neighborhood search for pickup and delivery problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Ma, Y., Li, J., Cao, Z., Song, W., Zhang, L., Chen, Z., and Tang, J. (2021). Learning to iteratively solve routing problems with dual-aspect collaborative Transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Malkin, N., Jain, M., Bengio, E., Sun, C., and Bengio, Y. (2022). Trajectory balance: Improved credit assignment in GFlowNets. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Malkin, N., Lahlou, S., Deleu, T., Ji, X., Hu, E., Everett, K., Zhang, D., and Bengio, Y. (2023). GFlowNets and variational inference. In *International Conference on Learning Representations (ICLR)*.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.

Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Naseri, G. and Koffas, M. A. (2020). Application of combinatorial optimization strategies in synthetic biology. *Nature communications*, 11(1):2446.

Nazari, M., Oroojlooy, A., Snyder, L., and Takac, M. (2018). Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Pan, L., Malkin, N., Zhang, D., and Bengio, Y. (2023). Better training of GFlowNets with local credit and incomplete trajectories. In *International Conference on Machine Learning (ICML)*.

Pisinger, D. and Ropke, S. (2019). Large neighborhood search. *Handbook of metaheuristics*, pages 99–127.

Qiu, R., Sun, Z., and Yang, Y. (2022). DIMES: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384.

Richter, L., Boustati, A., Nüsken, N., Ruiz, F., and Akyildiz, O. D. (2020). Vargrad: A low-variance gradient estimator for variational inference. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Son, J., Kim, M., Choi, S., Kim, H., and Park, J. (2024). Equity-Transformer: Solving np-hard min-max routing problems as sequential generation with equity context. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Son, J., Kim, M., Kim, H., and Park, J. (2023). Meta-SAGE: scale meta-learning scheduled adaptation with guided exploration for mitigating scale shift on combinatorial optimization. In *International Conference on Machine Learning (ICML)*.

Stützle, T. and Hoos, H. H. (2000). MAX–MIN ant system. *Future Generation Computer Systems*, 16(8):889–914.

Sun, H., Goshvadi, K., Nova, A., Schuurmans, D., and Dai, H. (2023). Revisiting sampling for combinatorial optimization. In *International Conference on Machine Learning (ICML)*.

Sun, Z. and Yang, Y. (2023). DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., and Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations (ICLR)*.

Vidal, T. (2022). Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems (NIPS)*.

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

Wouda, N. A., Lan, L., and Kool, W. (2024). PyVRP: a high-performance VRP solver package. *INFORMS Journal on Computing*.

Xia, Y., Yang, X., Liu, Z., Liu, Z., Song, L., and Bian, J. (2024). Position: Rethinking post-hoc search-based neural approaches for solving large-scale traveling salesman problems. In *International Conference on Machine Learning (ICML)*.

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

Ye, H., Wang, J., Cao, Z., Liang, H., and Li, Y. (2023). DeepACO: Neural-enhanced ant systems for combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ye, H., Wang, J., Liang, H., Cao, Z., Li, Y., and Li, F. (2024). GLOP: Learning global partition and local construction for solving large-scale routing problems in real-time. In *AAAI Conference on Artificial Intelligence (AAAI)*.

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

Zhang, D., Dai, H., Malkin, N., Courville, A. C., Bengio, Y., and Pan, L. (2023). Let the flows tell: Solving graph combinatorial problems with gflownets. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhang, D., Malkin, N., Liu, Z., Volokhova, A., Courville, A., and Bengio, Y. (2022a). Generative flow networks for discrete probabilistic modeling. In *International Conference on Machine Learning (ICML)*.

Zhang, D. W., Rainone, C., Peschl, M., and Bondesan, R. (2022b). Robust scheduling with GFlowNets. In *International Conference on Learning Representations (ICLR)*.

Zhang, J., Ding, G., Zou, Y., Qin, S., and Fu, J. (2019). Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of intelligent manufacturing*, 30:1809–1830.

Zheng, Z., Zhou, C., Xialiang, T., Yuan, M., and Wang, Z. (2024). UDC: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhou, J., Cao, Z., Wu, Y., Song, W., Ma, Y., Zhang, J., and Xu, C. (2024). MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In *International Conference on Machine Learning (ICML)*.

Zhu, Y., Wu, J., Hu, C., Yan, J., Hou, T., Wu, J., et al. (2023). Sample-efficient multi-objective molecular optimization with gflownets. *Advances in Neural Information Processing Systems (NeurIPS)*.

Minsu Kim*, Sanghyeok Choi*, Hyeonah Kim, Jiwoo Son, Jinkyoo Park, Yoshua Bengio

A   Implementation

A.1   Learning architecture and optimizer

We utilize the anisotropic graph neural network with edge gating mechanisms, which is proposed by Joshi et al. (2021) and adopted by Ye et al. (2023), as the foundational GNN architecture for the construction of a heuristic matrix. This network offers node and edge embeddings that are more beneficial for edge prediction than other popular GNN variants, such as Graph Attention Networks (Veličković et al., 2018) or Graph Convolutional Networks (Kipf and Welling, 2017).

We represent the node feature of node v_i at the ℓ-th layer as h_i^ℓ, and the feature of the edge between nodes i and j at the same layer as e_{ij}^ℓ. These features undergo propagation to the next layer $\ell + 1$ through an anisotropic message passing scheme, detailed as follows:

$$h_i^{\ell+1} = h_i^\ell + \text{ACT}(\text{BN}(U^\ell h_i^\ell + \mathcal{A}_{j \in \mathcal{N}_i}(\sigma(e_{ij}^\ell) \odot V^\ell h_j^\ell))) \tag{11}$$

$$m^\ell = P^\ell e_{ij}^\ell + Q^\ell h_i^\ell + R^\ell h_j^\ell \tag{12}$$

$$e_{ij}^{\ell+1} = e_{ij}^\ell + \text{ACT}(\text{BN}(m^\ell)) \tag{13}$$

where $U^\ell, V^\ell, P^\ell, Q^\ell, R^\ell \in R^{d \times d}$ are the learnable parameters of layer ℓ, ACT denotes the activation function, BN (Ioffe and Szegedy, 2015) denotes batch normalization, $\mathcal{A}$ denote the aggregation function, σ is the sigmoid function, $\odot$ is Hadamard product, $\mathcal{N}_i$ denotes the neighborhoods of node v_i. In this paper we use ACT is SiLU (Elfwing et al., 2018) and $\mathcal{A}$ is mean pooling. We transform the extracted edge features into real-valued heuristic metrics by employing a 3-layer Multilayer Perceptron (MLP) that includes skipping connections to the embedded node feature h. The SiLU is the activation function for all layers except the output layer, in which the sigmoid function is applied to generate normalized outputs. The training process is optimized using the AdamW optimizer (Loshchilov and Hutter, 2017a) in conjunction with a Cosine Annealing Scheduler (Loshchilov and Hutter, 2017b)

A.2   Hyperparameters

Table 5: Training hyperparameter settings for each task. These hyperparameter settings are used both for GFACS and DeepACO training. The values with * indicate that the values are used for fine-tuning a pre-trained model.

	TSP200	TSP500	TSP1000	CVRP200	CVRP500	CVRP1000	OP100	PCTSP100	SMTWTP100	BPP120
N_{epoch}	50	50	20*	50	50	20*	20	20	20	20
N_{inst}	400	400	200*	200	200	100*	100	100	100	100
M	20	20	10	10	10	5	5	5	5	5
K	30	30	20	20	20	15	30	30	30	30

For a task of COP (e.g., TSP200), we train GFACS for N_{epoch} epochs. In each epoch, we use N_{inst} number of instances using mini-batch with size M; thus N_{inst}/M gradient steps are taken per epoch. For each problem instance, we sample K solutions to train with, as discussed in Section 4.1. Our main baseline algorithm, DeepACO, is also trained using the same hyperparameters to ensure a fair comparison. We summarize used value for N_{epoch}, N_{inst}, M and K in Table 5. Note that for TSP1000 and CVRP1000, we use model checkpoints pre-trained in TSP500 and CVRP500, respectively, to reduce the processing time for training.

There are other hyperparameters to be considered specifically for GFACS. β is one of the most important hyperparameters for training GFACS, which is analyzed in Appendix D.2. We use annealing for β using the following log-shaped scheduling:

$$\beta = \beta_{\min} + (\beta_{\max} - \beta_{\min}) \min \left(\frac{\log(i_{\text{epoch}})}{\log(N_{\text{epoch}} - N_{\text{flat}})}, 1 \right)$$

where i_{epoch} is the index of epoch and N_{flat} is the number of last epochs to have $\beta = \beta_{\max}$.

Table 6: Settings of β for each task.

	TSP	CVRP	CVRPTW	OP	PCTSP	SMTWTP	BPP
$\beta_{\min}$	200	500	500	5	20	10	1000
$\beta_{\max}$	1000	2000	2000	10	50	20	2000

We report the β value used for each task in Table 6. Note that β remains consistent across different scales of the same problem class.

In Equation (7), we also need a hyperparameter α that controls how much we augment the energy of a solution with the energy of the refined solution. If $\alpha = 0$, we do not augment at all, and if $\alpha = 1$, the energy of a solution is defined to be equal to the energy of the refined solution. We empirically found that linearly increasing α from 0.5 to 1.0 along the epochs works well, and we use that setting for all experiments.

A.3 Computing resource

Throughout the experiments, we utilize a single CPU, specifically an AMD EPYC 7542 32-Core Processor, and a single GPU, the NVIDIA RTX A6000, for training neural networks. After thet, the inference was conducted on a separate, standalone PC with a single AMD Ryzen 9 5900X 12-Core Processor and NVIDIA RTX 4070 Ti to assess the wall time of each algorithm fairly.

B Problem settings

In this section, we provide an overview of the combinatorial optimization problems we address. These encompass vehicle routing problems, bin packing problems, and scheduling problems. For a comprehensive understanding of each problem's specific settings, we direct readers to the work by Ye et al. (2023). Furthermore, for access to the exact source code pertaining to each problem, please refer to the repository available at [2] from Ye et al. (2023).

Traveling salesman problem In the traveling salesman problem (TSP), we aim to find the shortest route that visits a set of nodes once and returns to the origin. We consider Euclidean TSP, where the distance between two nodes is determined by Euclidean distance. A TSP instance can be determined as a set of nodes with 2D coordinates. We generate a random TSP instance by sampling the coordinate from a unit square, $[0, 1]^2$.

Capacitated vehicle routing problem Similar to TSP, capacitated vehicle routing problem (CVRP) also seeks the shortest path that visits all the nodes once. In CVRP, however, a vehicle has a limited capacity, and each node has a demand to be satisfied. The sum of the demand of each subroute should not exceed the capacity, which limits the number of nodes that can be visited by one vehicle. In our setting, CVRP is also defined in Euclidean space.

A CVRP instance can be represented by a set of customer nodes, a single depot node, and a capacity C. Each customer node has a position (2D coordinates) and a demand, while the depot only has a position. A random CVRP instance is generated by sampling the coordinates of both customer and depot from a unit square $[0, 1]^2$ and sampling demands for each customer from a predefined uniform distribution, $U[a, b]$. We use $a = 1$ and $b = 9$, and a fixed $C = 50$ for all problem scales, 200, 500 and 1000.

Capacitated vehicle routing problem with time window A uniform-capacity fleet of delivery vehicles is tasked with servicing customers who have specific demand and operating hours (earliest-latest arrival) for a single commodity. The vehicles should deliver to customer only the time available for service. The goals are to minimize the total travel distance.

A CVRPTW instance consists of a set of customer nodes, a single depot node, the demands of each customer node, the vehicle capacity (as in CVRP), and additionally, the earliest and latest arrival times for each node, as

---

[2]https://github.com/henry-yeh/DeepACO

well as the service time at each node. We generate the CVRPTW instance following the method provided by Berto et al. (2023), which can be accessed here.[3]

Orienteering problem In the orienteering problem (OP), the objective is to maximize the total prize of visited nodes within a given tour length limitation. This problem is similar to CVRP but focuses on maximizing gains rather than minimizing costs.

An OP instance comprises a set of nodes to visit, each with its position and prize, a depot, and a maximum length constraint. We sample a random OP instance by first sampling the position of nodes, including the starting node, from a unit square $[0, 1]^2$. And we set the prize for each node, following Fischetti et al. (1998); Ye et al. (2023), to be $p_i = (1 + \lfloor 99 \cdot \frac{d_{0,i}}{\max_{j=1}^n d_{0,j}} \rfloor)/100$, where $d_{0,i}$ is the Euclidean distance from the depot to node i. We set the maximum length to 4 in the OP100 task, following Kool et al. (2019).

Prize-collecting traveling salesman problem The prize-collecting traveling salesman problem (PCTSP) extends the TSP by associating a prize for visiting a node and a penalty for not visiting it. The goal is to find a tour that minimizes the total travel cost plus penalties for the unvisited nodes and also satisfies a predefined prize threshold. In PCTSP, balancing the need to collect prizes against the cost of visiting more nodes matters.

A PCTSP instance comprises a set of nodes to visit, a depot, and a minimum total prize threshold. The coordinates of nodes and depot are sampled in the unit square. The prize for each node is sampled from $U[0, 1]$, while the penalty for each node is sampled from $U[0, C_N]$, where C_N is set to 0.12 for $N = 100$. Lastly, the minimum total prize threshold is $N/4$ for problems with N nodes, i.e., 25 for PCTSP100.

Single machine total weighted tardiness problem This problem involves scheduling jobs on a single machine where each job has a due date, processing time, and weight. The objective is to minimize the total weighted tardiness, which is the sum of the weights of the jobs multiplied by their tardiness (the amount of time a job is completed after its due date).

To generate an SMTWTP instance, we sample a due date from $U[0, N]$, the weight from $U[0, 1]$, and the processing time from $U[0, 2]$, where N is the number of jobs to process.

Bin packing problem The bin packing problem (BPP) aims to minimize the number of bins when objects with different volumes need to be packed into a finite number of bins. The cost functions are often modified to maximize the utility of used bins (Falkenauer and Delchambre, 1992; Ye et al., 2023). However, the direct use of the objective function employed in DeepACO can lead to an excessively large β since the objective values tend to be saturated near 1.0; note that the objective values can not exceed 1.0 in that setting. Therefore, we slightly modify them to minimize a weighted sum of the inverse of the utilization of the used bins throughout all our BPP experiments.

To generate a BPP instance, we sample the N items with a random size from $U[20, 100]$. And we use 150 for the bin capacity.

C   Implementation of destroy-and-repair local search

As demonstrated in Section 4, our algorithm utilizes the local search for off-policy training. The refined solutions from local search serve as exploitation batches, and they are also used for energy-reshaping (Equation (7)). For some problems like traveling salesman problem (TSP), capacitated vehicle routing problem (CVRP), and CVRP with time window (CVRPTW), we used the well-established local search operators such as 2-opt, Swap* (Vidal, 2022) and granular neighborhood search from PyVRP (Wouda et al., 2024).

While domain-specific local search techniques are well developed for many combinatorial optimization problems, there are still tasks that lack effective local search solvers like Swap*. A potential candidate for a general-purpose local search solver in combinatorial optimization is the destroy-and-repair local search operator. This method is inspired by the large neighborhood search approach described by Pisinger and Ropke (2019). Our implementation utilizes an evolutionary-style algorithm that repeatedly performs destruction, reconstruction, and top-K selection over a specified number of rounds.

---

[3]https://github.com/ai4co/rl4co/blob/main/rl4co/envs/routing/cvrptw/generator.py

Symmetricity-aware destruction. The most naive way to destroy a solution x is simply dropping $N_{destroy}$ of the last nodes from the sampled trajectory $\tau_{\rightarrow x}$. However, it only performs the destruction with one trajectory, while there can be multiple trajectories that correspond to the same solution x. Considering this symmetric nature, our destruction phase consists of 1) selecting $\tau'_{\rightarrow x}$ randomly from the set of symmetric trajectories and then 2) destroying τ'_x by dropping the last $N_{destroy}$ nodes, resulting in a partial solution. Note that the number of symmetric trajectories for a given solution differs from problem types, as discussed in Appendix E.

Repair with temperature annealing. Given a partial solution, we reconstruct a new solution x' using the policy similar to the decoding steps explained in Section 3.2. Considering the purpose of the local search (refining a given solution), we further promote the exploitation by introducing an inverse temperature for the policy. Formally, we modify Equation (5) with the inverse temperature δ_r as follows:

$$P_F(s_{t+1}|s_t; \eta(\mathcal{G};\theta), \rho, \delta_r) \propto \begin{cases} \left(\rho_{a_{t-1},a_t} \eta_{a_{t-1},a_t}(\mathcal{G};\theta)\right)^{\delta_r} & \text{if } a_t \notin s_t \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

We increase δ_r linearly from 1.0 to a specified maximum value, $\delta_{\max}$, as the local search round progresses.

Top-K selection. For a given solution x, we perform the aforementioned destruction and repair multiple times (in batch), obtaining a set of refined solutions. Then we evaluate them and leave only top-K solutions among them in terms of solution quality. The remaining solutions are used as an initial solution for the next round of the search.

D   Additional experiments

D.1   Comparison between trajectory balance and forward-looking detailed balance

Table 7: Comparison between forward-looking detailed balance (FL-DB) and trajectory balance (TB) loss in TSP50, using Attention Model (AM) and heatmap-based method from Joshi et al. (2021) as backbone policies.

Method	TSP50	
	Obj.	Gap (%)
Concorde	5.69	0.00
AM + FL-DB (s.w. = 1280)	6.52	14.64
AM + TB (s.w. = 1280)	6.00	5.50
Joshi et al. + FL-DB (s.w. = 1280)	6.54	14.96
Joshi et al. + TB (s.w. = 1280)	6.09	7.05

We trained Attention Model (AM; Kool et al., 2019) and heatmap-based policy from Joshi et al. (2021) in an unsupervised way using FL-DB and TB loss. Table 7 shows the test results for 128 random TSP50 instances. The result is aligned with the previous observation from GFACS, showing the limitation of FL-DB loss for solving complex CO tasks.

D.2   Analysis of β and diversity-optimality trade-off

In this subsection, we explored the diversity-optimality trade-off by varying the inverse energy temperature β from low to high values. For analysis, we generate K solution samples from policies trained with GFACS with different values of β and DeepACO (with $\rho = 1$ so that the pheromone does not affect the result).

We use average pairwise Jaccard distance as a diversity metric. Formally, the diversity of a set of K solutions is calculated as $\frac{1}{K(K-1)} \sum_i \sum_j d_J(x_i, x_j)$, where d_J is a Jaccard distance between two solutions, defined as follows:

$$d_J(x_i, x_j) = 1 - J(x_i, x_j) = 1 - \frac{|E(x_i) \cap E(x_j)|}{|E(x_i) \cup E(x_j)|} \tag{15}$$

$E(x)$ is the set of edges composing the solution x. In TSP or CVRP, the (undirected) edge is a set of adjoining nodes in the solution path. For some CO problems, such as SMTWTP, the edge should be directed due to its asymmetric nature.

Figure 6 show that lower β values lead to more diverse solutions with higher costs, while higher β values result in less diverse solutions with lower costs. GFACS confirms the anticipated trade-off between diversity and optimality, establishing a Pareto frontier over DeepACO by offering more diverse and high-quality solutions. This advantage is particularly beneficial when an effective local search mechanism is available, increasing the likelihood of achieving superior local optima. However, excessively high β (e.g., 2000 in our result) degrades both diversity and solution quality, supposedly because too much high β hinders exploration during training. This empirical evidence supports our design choice of β annealing Appendix A.2, which encourages exploration in the early stage of training.
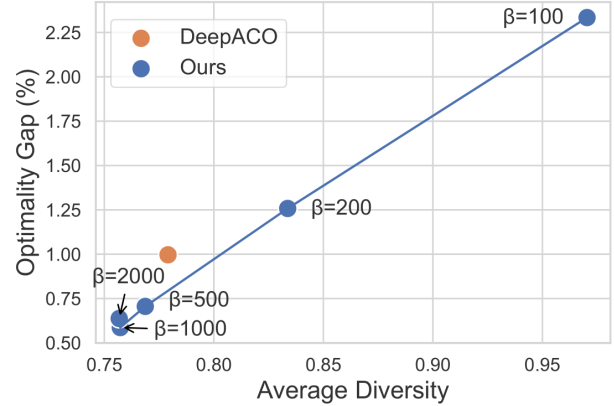

Figure 6: The trade-off between diversity and optimality as the inverse energy temperature β changes.

D.3 Evaluation with other ACO variants

Table 8: Evaluation results with different pheromone update rules, Elitist Ant System (Elitist) and MAX-MIN Ant System (MAX-MIN). We run $T = 10$ ACO rounds with 100 ants in all settings.

	TSP200		CVRP200	
Method	Obj.	Gap (%)	Obj.	Gap (%)
Concorde / PyVRP	10.72	0.00	27.93	0.00
Ant System	10.91	1.77	28.45	1.86
DeepACO + Ant System	10.77	0.45	28.44	1.80
GFACS + Ant System	**10.75**	**0.26**	**28.32**	**1.40**
Elitist	10.97	2.31	28.46	1.90
DeepACO + Elitist	10.77	0.49	28.45	1.84
GFACS + Elitist	**10.75**	**0.28**	**28.34**	**1.47**
MAX-MIN	10.97	2.34	28.46	1.88
DeepACO + MAX-MIN	10.78	0.57	28.44	1.82
GFACS + MAX-MIN	**10.75**	**0.32**	**28.33**	**1.43**

Our method is focused on enhancing the design of the heuristic matrix within the ACO framework. Thus, the advancement of the ACO algorithm itself (usually in pheromone update and action selection rule) is orthogonal to our work. To validate this claim, we evaluate our method with different ACO variants other than Ant System, namely Elitist Ant System (Dorigo et al., 1996), MAX-MIN Ant System (Stützle and Hoos, 2000). The results in Table 8 demonstrate that GFACS consistently outperforms both ACO with human-designed heuristic and DeepACO under all pheromone update rules tested. This underscores the superiority of GFACS in generating a high-quality heuristic matrix.

D.4 Comparison against TAM

We compare GFACS and DeepACO against CVRP instances used in Two-stage Divide Method (TAM; Hou et al., 2023). These instances have a different capacity constraint from those we used for the main experiment (Table 2). Please refer to Hou et al. (2023) for more details regarding the instance generation. Moreover, to match the processing time to be comparative to DeepACO, we set K (the number of ants for ACO) to 16.

D.5 Full results of the ablation study

In this section, we present the comprehensive results of our ablation study, including standard deviations from multiple independent runs (three runs) to ensure statistical significance. Table 10 demonstrates that each component of our technical proposals contributes to improvement, supported by empirical and statistically rigorous evidence.

Table 9: Results for CVRP instances used in TAM. The results for AM and TAM is drawn from Hou et al. (2023), and the result for DeepACO is drawn from Ye et al. (2023).

	CVRP100		CVRP400	
Method	Obj.	Time (s)	Obj.	Time (s)
AM*	16.42	0.06	29.33	0.20
TAM*	16.08	0.09	25.93	1.35
DeepACO ($T = 4$)	16.08	2.97	25.31	3.65
DeepACO ($T = 10$)	15.77	3.87	25.27	5.89
GFACS ($T = 4$)	15.77	2.91	24.95	3.23
GFACS ($T = 10$)	15.72	3.88	24.87	5.21

Table 10: Results of the ablation study. The mean and standard deviation are reported based on three independent runs. The reported gap is calculated against Concorde (Applegate et al., 2006) for TSP and PyVRP (Wouda et al., 2024) for CVRP.

	TSP500		CVRP500		BPP120
Method	Obj.	Gap (%)	Obj.	Gap (%)	Obj.
GFACS	**16.8042** $\pm$ 0.0043	**1.56**	**64.1858** $\pm$ 0.0071	**1.95**	**51.8428** $\pm$ 0.0706
– *Off-policy Training*	17.0861 $\pm$ 0.0028	3.26	64.4248 $\pm$ 0.0155	2.33	52.3158 $\pm$ 0.1171
– *Energy Reshaping*	16.8718 $\pm$ 0.0049	1.97	64.4121 $\pm$ 0.0069	2.31	52.0654 $\pm$ 0.0998
– *Shared ε-norm*	17.5209 $\pm$ 0.0062	5.87	65.5069 $\pm$ 0.0171	4.05	90.7332 $\pm$ 9.1649
+ *VarGrad* Zhang et al. (2022b)	16.8570 $\pm$ 0.0086	1.88	64.2771 $\pm$ 0.0293	2.09	51.9504 $\pm$ 0.1222

D.6 Results without local search on TSP and CVRP

Table 11: Results obtained with and without local search (LS) at test time on TSP and CVRP are reported. The values represent the objective value averaged over 128 test instances and further averaged across three independent runs.

Method	TSP200	TSP500	TSP500	CVRP500
ACO w/o LS	14.19	24.05	42.95	95.24
DeepACO w/o LS	11.59	18.83	32.12	73.20
GFACS w/o LS	12.63	21.72	34.04	81.03
ACO w/ LS	10.91	17.38	28.45	64.62
DeepACO w/ LS	10.77	16.84	28.44	64.49
GFACS w/ LS	10.75	16.80	28.32	64.19

We conduct an additional ablation study to assess the impact of local search at test time. Note that we use the same policies as in Section 5.1 or Section 5.2, which were trained *with* local search. The results are presented in Table 11. As expected, DeepACO (RL-based) performs better than ours when local search is not applied. This outcome can be attributed to two factors: (1) The RL-based method tends to generate non-diverse low-energy samples, whereas the GFlowNet-based method (GFACS) produces a more diverse set of candidates; and (2) Due to the energy reshaping (Eq. (7) in Section 4.1), the policy is optimized with respect to a modified energy map that assumes the application of local search, resulting in suboptimal performance when local search is omitted. This second factor also explains why GFACS outperforms other methods when local search is applied.

D.7 Runtime analysis

For completeness, in Table 12, we report the runtime for each algorithm on the real-world instances (Table 1 in Section 5.1). Although all algorithms have similar inference times, DeepACO and GFACS occasionally run faster than ACO. This is attributed to the increased complexity of real-world instances: both DeepACO and GFACS benefit from an NN-based amortized prior that enables them to start closer to a local optimum, resulting in earlier termination of the local search. This also explains why DeepACO is slightly faster than GFACS, as it produces a more locally optimized initial solution.

Table 12: Average running time (in seconds) to solve a single instance for Table 1.

	N	ACO	DeepACO	GFACS
TSPLib	100-299	1.3	1.3	1.3
	300-699	11.2	10.5	10.8
	700-1499	89.3	72.5	77.0
CVRPLib	100-299	15.5	15.5	15.5
	300-699	34.2	35.1	35.5
	700-1001	60.0	59.8	61.8

E Symmetric solutions of combinatorial optimization

This section delves into the symmetric characteristics of combinatorial solutions, wherein multiple trajectories, denoted as τ, can map to an identical combinatorial solution. We begin with a formal analysis focusing on two quintessential tasks: the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP). This analysis lays the foundation for understanding the connections to other related combinatorial optimization problems.

Proposition E.1 (Symmetry Solution in TSP (Hamiltonian Cycle))**.** *Let* $\mathcal{G} = (\mathcal{V}, \mathcal{D})$ *be a finite, undirected graph where* $\mathcal{V}$ *is the set of vertices and* $\mathcal{D}$ *is the set of edges. Assume* $\mathcal{G}$ *contains a Hamiltonian cycle. Denote* $N = |\mathcal{V}|$ *as the number of vertices. If* $N > 1$*, there exist exactly* $2N$ *symmetric trajectories (including the cycle and its reverse) starting from each vertex in the cycle. For* $N = 1$*, there is exactly one identical trajectory.*

Proof. A Hamiltonian cycle in $\mathcal{G}$ is a cycle that visits every vertex exactly once. For $N > 1$, each cycle can be traversed in two distinct directions (clockwise and counterclockwise) from each vertex, yielding 2 trajectories per vertex. Since there are N vertices, this results in $2N$ symmetric trajectories. For $N = 1$, the cycle is trivial and only has one trajectory. □

Proposition E.2 (Symmetric Solutions in CVRP)**.** *Consider a Capacitated Vehicle Routing Problem (CVRP) defined on a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{D})$ *with a designated depot vertex. Let there be* K *distinct Hamiltonian cycles (or routes) originating and terminating at the depot, and denote* S *as the number of single vertex cycles (excluding the depot) among these* K *cycles. The total number of symmetric solutions, considering the directionality of each cycle, is given by* $K! \times 2^{K-S}$*.*

Proof. Each of the K cycles can be ordered in $K!$ ways. Excluding the S single vertex cycles, each of the remaining $K - S$ cycles can be traversed in two directions (forward and reverse). Therefore, for these $K - S$ cycles, there are 2^{K-S} ways to choose a direction. The product $K! \times 2^{K-S}$ gives the total number of symmetric solutions for the CVRP under these conditions. □

The propositions discussed previously can be extended to various variants of vehicle routing problems. Specifically, in the context of the orienteering problem (OP) and prize collecting traveling salesman problem (PCTSP), which represent special cases where $K = 1$, the number of symmetric trajectories is determined by the number of vertices, N, in the solution. When $N > 1$, there are exactly 2 symmetric trajectories for each route. In contrast, for a single-vertex solution ($N = 1$), the trajectory is identical, reflecting the singularity of the route.

F Related works

F.1 Deep constructive policy for neural combinatorial optimization

Constructive policy methods systematically build solutions by incrementally adding elements, beginning with an empty set and culminating in a complete solution. This approach ensures compliance with constraints by progressively narrowing the scope of actions, making it versatile for application in domains like vehicle routing (Nazari et al., 2018; Kool et al., 2019), graph combinatorial optimization (Khalil et al., 2017; Ahn et al., 2020; Zhang et al., 2023), and scheduling (Zhang et al., 2020).

In deep learning, various techniques adopt this constructive policy framework. Notably, the attention model (AM; Kool et al., 2019) has been pivotal in vehicle routing problems (VRPs), establishing a new standard for stabilizing REINFORCE training. Subsequent advancements in AM have included enhanced decoder architecture and decoding procedure (Son et al., 2024; Xin et al., 2021; Luo et al., 2023; Drakulic et al., 2023), improved REINFORCE (Williams, 1992) baseline assessments (Kwon et al., 2020; Kim et al., 2022), leveraging symmetries for sample efficiency (Kim et al., 2024a), the application of population-based reinforcement learning (RL) (Grinsztajn et al., 2023), ensemble-based learning (JIANG et al., 2023; Zhou et al., 2024). Recent advancements in hierarchical algorithms exploit AM-based policies for solution optimization: one policy broadly generates solutions while another refines them locally (Kim et al., 2021; Hou et al., 2023; Ye et al., 2024; Zheng et al., 2024).

Graph combinatorial optimization, particularly in problems like maximum independent sets, has diversified from contrive VRPs policies due to their local decomposability and the relative ease of training value functions for partial solutions where VRPs are not locally decomposable due to their global contains. Starting with the influential works by Khalil et al. (2017), which employed deep Q networks (Mnih et al., 2013) for graph policy training, creating solutions constructively, Ahn et al. (2020) introduced an innovative Markov Decision Process (MDP) to generate multiple solution components per iteration by exploiting the local decomposability of graph combinatorial problems. Zhang et al. (2023) further developed this concept, training their MDP with forward-looking GFlowNets (Pan et al., 2023).

Our research aligns with the deep constructive policy framework, where we build a solution from an initial empty solution (s_0) based on the policy trained with GFlowNets. Additionally, our method is included in the heatmap-based approaches, detailed in Appendix F.3.

F.2   Deep improvement policy for neural combinatorial optimization

The improvement policy focuses on refining complete solutions, similar to local search methods like 2opt (Croes, 1958). Its advantage lies in achieving comparable performances to constructive policies, especially when allotted sufficient time for extensive improvement iterations. Researchers typically adapt existing heuristic local search techniques, creating learning policies that emulate these methods through maximizing expected returns via reinforcement learning (d O Costa et al., 2020; Hottung and Tierney, 2020; Ma et al., 2021, 2022, 2023; Kim et al., 2023). Also, recent works have explored active search to refine solutions via policy adjustments (Hottung et al., 2022; Son et al., 2023) or sampling the conditions from a continuous latent space (Chalumeau et al., 2023). Moreover, alternative strategies using discrete Langevin sampling instead of neural networks have shown promising outcomes (Sun et al., 2023). This approach serves as a complementary extension to deep constructive policies, as it often enhances solutions initially formulated by constructive methods. Additionally, our method, which integrates local search into the training loop, could see further enhancements if advancements in this area are incorporated into our training procedures.

F.3   Heatmap-based neural combinatorial optimization

The heatmap-based method (Joshi et al., 2021) entails the acquisition of a graph representation, referred to as the "Heatmap," from input graph instances. This Heatmap serves as a prior distribution, influencing the generation of a solution. However, this approach is not self-contained and requires an additional solution construction step, which may involve techniques like beam search (Joshi et al., 2021), dynamic programming (Kool et al., 2022), Monte Carlo tree search (Fu et al., 2021; Qiu et al., 2022; Sun and Yang, 2023; Xia et al., 2024) or greedy search with a guide using an objective function (Li et al., 2023). The key advantage of the heatmap-based method lies in its exceptional generalizability when compared to constructive policies. It operates at an abstract level, free from the constraints inherent in generating actual combinatorial solutions.

Our approach falls under heatmap-based methodologies, where we learn a heuristic metric (i.e., heatmap) and employ ant colony optimization for solution construction. A key advantage of our method over many other supervised learning-based heatmap approaches is its independence from optimally labeled data.

F.4   GFlowNets for neural combinatorial optimization

Originally proposed as a framework for learning policies over discrete compositional variables (Bengio et al., 2021a), GFlowNets have since been applied to combinatorial optimization problems (Zhang et al., 2023, 2022b). They offer unique advantages over reinforcement learning methods, particularly in generating diverse candidate solutions and enabling off-policy training. Although prior studies have explored the diversity aspect (Zhang et al., 2023, 2022b), our work is, to our knowledge, the first to empirically validate the benefits of off-policy training in combinatorial optimization settings.

To underscore our contribution, we conceptually compare our method against two recent approaches that incorporate off-policy training for GFlowNets: local search GFlowNets (LS-GFN) (LS-GFN; Kim et al., 2024b) and the hindsight-like off-policy strategy proposed by Zhu et al. (2023). Note that, in Section 5.4, we provide an empirical comparison of our approach with FL-DB (Zhang et al., 2023) and VarGrad (Zhang et al., 2022b).

The key difference between LS-GFN and our approach is that our method addresses conditional inference problems. This requires training conditional GFlowNets that operate under distinct reward landscapes for different conditions, whereas LS-GFN is designed for unconditional problems. Moreover, the local search operators themselves differ. LS-GFN uses a back-and-forth mechanism that partially backtracks trajectories and reconstructs them using its own policy, thereby limiting exploration to sequences sharing identical root segments. In contrast, our local search methods, such as the 2-opt operator for the TSP, explore the solution space within a Hamming ball, allowing for more diverse exploration without the constraint of shared initial sequences. Furthermore, our destroy-and-repair local search method (Appendix C) can be viewed as an extension of LS-GFN, incorporating symmetricity-aware destruction to mitigate the limitations of the back-and-forth operator.

In Zhu et al. (2023), they tries to model a family of policies using hypernetworks to tackle multi-objective optimization problems. They also propose a hindsight-like off-policy strategy, which enables sharing of off-policy samples among policies configured with varying objective mixing coefficients, thereby enhancing sample efficiency during training. In contrast, our method leverages hindsight observations from a (deterministic) local search (Section 4.1). Specifically, we adjust the energy of an original sample by incorporating the energy after local search, thereby quantifying how readily the sample can be converted into a low-energy solution.