Leveraging Structural Constraints for Diffusion-based Neural TSP Solvers

Mickaël Basson[1,2,3,4]✉ and Philippe Preux[1,2,3,4]

[1] Université de Lille, France
[2] CNRS, France
[3] Inria, France
[4] UMR 9189-CRIStAL, Lille, France
{mickael.basson, philippe.preux}@inria.fr

Abstract. Neural combinatorial optimization has recently achieved strong results on the Euclidean Traveling Salesman Problem (TSP) using generative models such as diffusion and consistency models. State-of-the-art approaches like FT2T combine fast consistency-based prediction with gradient-based inference time refinement. However, gradient search often incurs significant computational overhead and may not align with the discrete structure of feasible solutions. We introduce Projected Consistency Inference (PCI), a plug-and-play, retraining-free alternative that replaces gradient refinement with structure-aware projections: PCI decodes valid Hamiltonian tours from the consistency model output and applies a lightweight local search (e.g., 2-opt). PCI achieves an average optimality gap (OG) of 0.17% on TSP with 500 cities, and 0.31% on TSP with 1000 cities, outperforming FT2T best settings (OG 0.22% and 0.36%, respectively) while reducing the inference time up to 30 to 40%. PCI also exhibits lower variance and memory usage, and can surpass classical heuristics such as LKH3 in rapid solution generation. Our results demonstrate that structure-aware inference time operations provide a practical and principled path for neural TSP solvers, complementing training time objectives.

Keywords: Neural combinatorial optimization · Consistency models · Traveling Salesman Problem

1 Introduction

Combinatorial Optimization (CO) problems, such as the Traveling Salesman Problem (TSP), remain central to logistics, scheduling, network design, chip layout and many other areas. Yet these problems are NP-hard, making it challenging to solve large instances with exact methods [11]. Classical heuristics, e.g., LKH3 for the TSP, are remarkably strong but require expert hand-crafting and careful parameter tuning [11].

Solving a CO problem instance may be modeled as a supervised learning task. For a particular CO problem, we may assume we have a training set made of instances of this problem, each labeled with its optimal solution. Once trained, the

model would be able to generalize to yet unseen instances. "Neural CO", where "neural" refers to the fact that the model is a neural network, leverage the generalization ability of properly trained neural networks to this aim. Neural CO has emerged to reduce manual design while leveraging data. Early approaches based on pointer networks and then on attention-based transformers used sequence models to *construct* tours [28, 15]. However, their performances were not matching those of dedicated heuristics well-known in the CO community. More recently, diffusion and consistency-based generative paradigms have become competitive for CO by modeling instance-conditioned solution distributions and enabling powerful inference time refinement [26, 18, 3, 19]. In particular, DIFUSCO cast CO as structured denoising on graph adjacency matrices and delivered strong results on the TSP and the MIS [26]. T2TCO introduced a gradient-based inference time search [18] and FT2T replaced multi-step diffusion sampling with a consistency predictor for one step or few steps generation, again paired with inference time gradient search steps leading to massive speed-up and improved solution quality [19].

While inference time optimization is a powerful lever, two issues persist. First, iterative denoising/search can be time-consuming, particularly at scale and when augmented with gradient search steps [26, 18]. Second, gradient search steps operate in a relaxed latent continuous space which geometry may not align with the discrete and combinatorial nature of CO problems. IDEQ recently showed that injecting structural bias — most notably, biasing toward 2-opt basins and leveraging constraints of the TSP solution manifold — significantly improves the quality of solutions, and the generalization capability of the trained model [3].

In this paper, we introduce *Projected Consistency Inference (PCI)* that adapts IDEQ structural advantage at inference time to the consistency setting. We introduce a plug-and-play replacement for FT2T gradient search. For a given instance, PCI generates a solution in 2 steps: (i) a *feasibility projection* from the continuous latent space to the set of Hamiltonian tours, (ii) a *local search projection* to obtain a local optimum for a small computation cost (e.g., a 2-opt or 3-opt local optimum for TSP). PCI requires no retraining. Instead it leverages the previously trained models. Interestingly, PCI aligns with a broader trend in generative modeling that improves the quality of solution generation by refining the inference steps [21, 10]. PCI is the new state-of-the-art diffusion-based constructive neural method to solve TSP problems in terms of both speed and performance.

Contributions. This paper presents two main contributions. (1) We formalize *Projection-Enhanced Consistency Inference* (PCI) and instantiate it for the TSP as an alternative to FT2T gradient search, requiring no additional training [19]. (2) Empirically we show that PCI achieves significant quality and time improvement over FT2T gradient search on the TSP and is even able to outperform LKH3 on new instances drawn from the training distribution solved with a small computation time constraint.

We argue that structure-aware inference time operations are a principled and practical path for neural CO solvers, complementing training time objectives.

2 Background and Related Work

In this section, after a very brief recap on the TSP, we present the main components PCI rely on, that is diffusion models and their descendants, consistency models. Aside a brief presentation of these models, we present their use in the case of the resolution of the TSP. This section is completed by a brief overview of other neural CO methods though they do not compete with diffusion-based neural CO methods.

2.1 About the TSP

We consider the basic and usual definition of the Traveling Salesman Problem (see e.g. [13]). An instance of the TSP is defined in the Euclidean plane by a set of N cities. N is also known as the "size" of the instance. We will denote TSP-N a TSP instance of size N. The goal is to find a tour of minimal length that goes through all the cities. The TSP is a NP-hard problem. Today, the state-of-the-art exact solver Concorde [1] solves an instance of the TSP of size $N = 10^3$ in about 10 minutes on a laptop. For larger instances, one has to use heuristics that have no formal guarantee to find an optimal tour. The quality of a tour may be measured by its "optimality gap", defined by $\frac{\text{length of the tour}-\text{optimal tour length}}{\text{optimal tour length}}$: it is a non-negative real number equal to 0 only for an optimal tour[5].

An optimal tour goes once and only once per city: it is a Hamiltonian tour. Consequently, the search space may be reduced to the set of Hamiltonian tours. Given a Hamiltonian tour, a very simple heuristic known as "2-opt" may decrease its length. Given a Hamiltonian tour, a "2-change" consists in removing a pair of edges of the tour and reconstructing another tour (a single tour may be reconstructed). In 2D, when a pair of edges cross each other, applying a 2-change uncrosses the edges: this decreases the length of the tour. 2-opt considers all pairs of crossing edges and disentangles the pair that reduces the most the length of the resulting tour. 2-opt is very simple and rather effective in optimizing a tour. In practice, it is customary to pipeline a first heuristic that produces a Hamiltonian tour, and post-process it by iterating 2-opt to obtain a disentangled tour. There exists other such local optimizations, like the Lin-Kernighan operator (lk-opt) [20] and its more recent version LKH3 [11] that are more complex in terms of algorithm, though very efficient and effective in practice.

2.2 Diffusion for Combinatorial Optimization

Diffusion Models constitute a class of generative algorithm that sample from complex unknown distributions such as distributions of images or graphs. They

[5] In the case of large instances we use the best solution provided by heuristics as a proxy for the optimal solution. In this case, it is possible that the optimality gap is negative. Indeed, if the heuristics gives a non-optimal solution any solution between optimality and the heuristics one has a negative optimality gap. This has a very limited impact in practice.

do so by mimicking the inverse of a noising process. Indeed, by progressively adding carefully selected random perturbations to clean samples drawn from the target distribution we obtain pure noise after a sequence of steps. The careful choice of the noisy perturbations ensures that the resulting noise is an easy to sample from distribution such as a Gaussian distribution or a Bernoulli distribution. This is called the forward process. The goal of a diffusion model is to learn the reverse process, hence called the backward process: starting from a noise sample, it is denoised through a sequence of (backward) steps to obtain a clean image or graph. Diffusion models use the generalization capabilities of neural networks to learn this process in a supervised manner with the training samples generated from the forward process.

This diffusion process may be expressed in discrete time or in continuous time. In discrete-time, the Denoising Diffusion Probabilistic Model (DDPM) [12] starts from clean data $\mathbf{x}_0$ and sequentially adds Gaussian noise through the forward process (denoted as $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ where $\mathbf{x}_t$ is the generated sample at timestep t):

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\big(\sqrt{1-\beta_t}\,\mathbf{x}_{t-1},\, \beta_t \mathbf{I}\big), \quad t = 1, \dots, T, \tag{1}$$

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{2}$$

where β_t is the noise schedule, $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, and $\mathcal{N}(\mu, \mathbf{C})$ is the multivariate normal distribution of mean μ and covariance $\mathbf{C}$. The backward process is parameterized as $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I})$ and trained by predicting the noise $\boldsymbol{\epsilon}_\theta$ with the simplified objective[6]:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}}\big[\, \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\big(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon},\, t\big)\|_2^2\big], \tag{3}$$

where θ denotes the set of parameters of the neural network and is used as a subscript to indicate a predictor function parameterized by such a network.

Continuous-time score-based formulations cast diffusion as a stochastic differential equation (SDE) $d\mathbf{x} = f(\mathbf{x}, t)\, dt + g(t)\, d\mathbf{w}$ whose reverse-time SDE depends on the score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$,

$$d\mathbf{x} = \big[f(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\big]\, dt + g(t)\, d\bar{\mathbf{w}}, \tag{4}$$

with an equivalent probability flow ordinary differential equation (PFODE) enabling deterministic sampling and likelihoods,

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, t) - \tfrac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}). \tag{5}$$

Practical refinements—e.g., learned reverse variances and cosine noise schedules—further improve sample quality and reduce the number of steps [23].

Discrete state space diffusion models [2] extend DDPMs to categorical variables $\mathbf{x}_t \in \{1, \dots, K\}^D$. The forward process is a Markov chain with transition matrices $Q_t \in [0, 1]^{K \times K}$

[6] Other parameterizations exist, this one is the most frequently used.

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathrm{Cat}(x_t; p = x_{t-1}Q_t). \tag{6}$$

The marginal distribution at step t can be expressed as:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathrm{Cat}(x_t; p = x_0\bar{Q}_t), \qquad \bar{Q}_t = Q_tQ_{t-1}\dots Q_1, \tag{7}$$

where $\bar{Q}_t$ is the cumulative transition matrix. The reverse process predicts categorical distributions with the following parametrization, where a neural network (with weights denoted θ) is trained to predict $\tilde{p_\theta}(\tilde{x_0} \mid x_t)$:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \propto \sum_{\tilde{x}_0} q(x_{t-1}, x_t \mid \tilde{x}_0)\tilde{p_\theta}(\tilde{x_0} \mid x_t). \tag{8}$$

This model supports structured transitions (e.g., absorbing states, lattice kernels) and achieves strong likelihoods on text and image benchmarks.

DIFUSCO introduced diffusion models on graphs to solve CO problems. The solution to the COP can be seen as the adjacency matrix of a graph. This (discrete) adjacency matrix A is relaxed to a real $[0, 1]^{N\times N}$ matrix (N being the number of nodes) with a probabilistic interpretation: the component a_{ij} of A is the probability that the edge between nodes i and j is present in the solution. This matrix is commonly referred to as a heatmap. DIFUSCO casts TSP and MIS as an instance conditioned heatmap generation problem. For the TSP, the predicted heatmap is then projected onto a Hamiltonian tour that is further optimized by a local search algorithm, such as iterating 2-opt.

Training-to-Testing: T2TCO. Following DIFUSCO, T2TCO proposed the training-to-testing paradigm: learn an instance-conditioned distribution (via diffusion) and then conduct gradient-based search at inference time to exploit objective gradients for instance-specific improvement [18].

Structure-Aware Training: IDEQ. Following DIFUSCO and T2TCO, IDEQ significantly improves diffusion-based TSP solvers by leveraging the constrained TSP solution manifold and taking into account the local search in the training objective. It does so by projecting the candidate solution at each time step of the backward process to the manifold of locally optimal Hamiltonian tours. IDEQ also sets a new training objective: instead of one optimal solution, IDEQ is searching for the set of tours that maps to the same local optimum. So, instead of searching for a single point, IDEQ searches for a basin of attraction: this introduces redundancy in the training objective and increases the entropy of the training set. This significantly enhances the performance of IDEQ with respect to DIFUSCO and T2TCO on TSPslib and synthetic TSPs [3].

2.3 Consistency Models for CO

Consistency models (CM) aim to reduce the computational cost of the multistep sampling done in diffusion models by learning a mapping that is invariant across noise levels. The core idea of a Consistency Model is to learn a function $\boldsymbol{f}_\theta : (\mathbf{x}_t, t) \mapsto \mathbf{x}_\epsilon$ that maps any point $\mathbf{x}_t$ on the PFODE trajectory (5) to the starting point of the trajectory $\mathbf{x}_\epsilon$ (which approximates the data $\mathbf{x}_0$). That is, for any trajectory $\{\mathbf{x}_t\}_{t\in[\epsilon,T]}$ that satisfies the PFODE, the consistency function must satisfy the **self-consistency property**:

$$\boldsymbol{f}_\theta(\mathbf{x}_t, t) = \boldsymbol{f}_\theta(\mathbf{x}_{t'}, t') \quad \forall t, t' \in [\epsilon, T]. \tag{9}$$

Essentially, the output of the model is invariant to the time t along the integration path. To ensure that the output is valid, the model must satisfy the boundary condition at the minimum time step ϵ:

$$\boldsymbol{f}_\theta(\mathbf{x}_\epsilon, \epsilon) = \mathbf{x}_\epsilon. \tag{10}$$

In practice, this is usually enforced via skip-connection parameterization using a deep neural network N_θ:

$$\boldsymbol{f}_\theta(\mathbf{x}, t) = c_{\text{skip}}(t)\mathbf{x} + c_{\text{out}}(t)N_\theta(\mathbf{x}, t), \tag{11}$$

where $c_{\text{skip}}(\epsilon) = 1$ and $c_{\text{out}}(\epsilon) = 0$.

Two main training strategies are used: consistency distillation from a teacher diffusion model, or direct consistency training. These approaches allow CM to achieve competitive sample quality with drastically fewer steps than diffusion models, hence for a much reduced computational cost.

FT2T accelerates the T2TCO pipeline by replacing iterative diffusion sampling with a consistency predictor that maps noisy states directly to high-quality data, optionally in a few steps. It uses an inference time consistency-based gradient search to further explore the learned solution space [19]. In our experiments, we find that such a gradient refinement adds computational overhead while providing limited quality gains, motivating a structure-aware, retraining-free alternative.

Our work adopts the structural insights from IDEQ at inference time, composing them with FT2T one step (or a few steps) consistency to obtain PCI, without any retraining needed.

2.4 Other Neural CO Approaches

To avoid the costly generation of the supervised training dataset, other approaches leverage unsupervised learning (like UTSP [22]) or reinforcement learning (RL). RL-based CO solvers are either autoregressive (incremental construction of the solution), or generating the full solution in one-shot. The former category includes BQ-NCO [5], POMO [17] and SymNCO [14] while the latter category includes DIMES [24]. DIMES employs a meta-learning framework

and a continuous parameterization of the solution space which enables a stable REINFORCE-based training. In BQ-NCO the state space of the Markov Decision Problem associated with the problem is simplified by bisimulation quotienting. POMO builds on the attention-based model of [16] but leverages symmetries by data augmentation and stabilizes REINFORCE training by using multiple initiations. SymNCO also leverages symmetries by introducing regularization in REINFORCE and by learning invariant representation for pre-identified symmetries. SymNCO and POMO do not scale well beyond 200 cities for the TSP. ICAM [30] builds upon POMO, modifying the attention mechanism and the reinforcement learning training scheme to significantly improve performance on larger TSP instances (up to 10^3 cities).

Another line of work aims at building solutions by splitting the problem into simpler and smaller sub-problems and merging together the partial solutions. To the best of our knowledge GLOP [29] is the current SOTA for such methods.

Aside pure constructive solvers other encouraging research areas have focused at improving generalization via knowledge distillation [4], meta-learning [31], ensemble methods [7] or local encoding [6].

Finally, ensemble methods leverage the diversity of existing solvers to improve solution generation [8, 9]

3 Projected Consistency Inference

3.1 Projection Operators

Build on earlier works, we propose PCI which composes two projections:

1. **Hamiltonian Tour Reconstruction (P_{Feas}).** Given edge probabilities from f_θ (a heatmap), PCI maps it to a Hamiltonian tour. This is the same operator as applied at the end of the DIFUSCO inference pipeline to reconstruct a valid tour from the heatmap (iterative greedy edge insertion procedure based on the heatmap probabilities which excludes edges that do not produce a valid tour) [26].
2. **Local Search Operator (P_{Local}).** On the Hamiltonian tour resulting from P_{Feas}, PCI applies a local search routine (e.g., 2-opt) to obtain a locally optimal tour.

This procedure is illustrated in figure 1 on a small instance of the TSPliB.

3.2 PCI Algorithm

We interleave consistency prediction (same as original consistency model), projection, and re-noising. The re-noising is the same as used in the orginal consistency model, it is denoted here as $NoiseInject$[7]. This is illustrated in figure 2 and the full algorithm is given in Algorithm 1.

[7] Using the same notations as above, $NoiseInject : x_0, t \mapsto x_t \sim Cat(x_t, p = x_0\bar{Q}_t)$

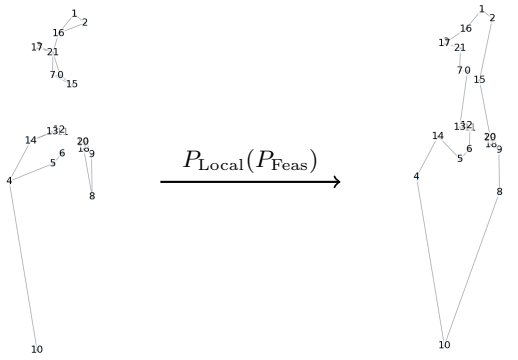

Fig. 1. PCI projection operations illustrated on the TSPlib instance Ulysses22. We use a very small instance for legibility purposes. The left graph is the diffusion-predicted $\hat{x}_0$. Before applying P_{Feas}, the graph may be made of more than one component. Then, the graph on the right results from the application of the 2 projections, P_{Feas} and then P_{Local}: it is a 2-opt locally optimal Hamiltonian tour.

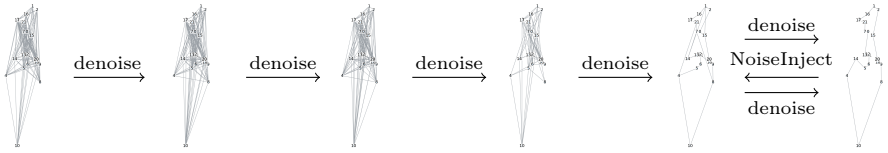

Fig. 2. PCI denoising and re-noising illustrated on Ulysses22 TSPlib instance. The left graph is a random graph with edge probability $p = 1/2$ and the right one is the predicted solution. The denoise operation is the joint consistency prediction and projections: denoise $\sim P_{\text{Local}}(P_{\text{Feas}}(f_\theta))$. We also illustrate the noise injection process. For conciseness we only depict once the partially noised graph (second to the right) and the re-denoised graph but on larger instances these usually differ leading to a solution improvement. There may also be more than one noise injection.

PCI eliminates gradient computations and uses deterministic projections with time-limited local optimization, reducing wall clock by approximately 40% with respect to the FT2T gradient search under matched computational budgets.

Crucially, PCI is *ready out-of-the-box*: it requires no retraining, no hyperparameter tuning, and integrates seamlessly with existing FT2T pre-trained models.

4 Experimental Study

4.1 Experimental Methodology

Experiments were conducted on samples drawn from the FT2T training distributions that is made of uniformly distributed 2D Euclidean TSP with sizes $N = 500$

Algorithm 1 Projected Consistency Inference (PCI)

Require: Trained consistency model f_θ, instance G, number of denoising steps K, noise schedule $\{t^{(k)}\}$, number of renoising steps J, renoise time τ

Initialize $x_t^{(1)} \sim q(\cdot)$ // Backward process
for $k = 1$ **to** K **do**
  $\hat{x}_0 \leftarrow f_\theta(x_t^{(k)}, t^{(k)}, G)$
  $x_0^{(k)} \leftarrow P_{\text{Local}}(P_{\text{Feas}}(\hat{x}_0))$
  $x_t^{(k+1)} \leftarrow \text{NoiseInject}(x_0^{(j)}, t^{k+1})$
end for
Initialize $x_0^{(0)} \leftarrow x_0^{(K)}$ // Renoise steps
for $j = 0$ **to** $J - 1$ **do**
  $x_\tau \leftarrow \text{NoiseInject}(x_0^{(j)}, \tau)$
  $\hat{x}_0 \leftarrow f_\theta(x_\tau, \tau, G)$
  $x_0^{(j+1)} \leftarrow P_{\text{Local}}(P_{\text{Feas}}(\hat{x}_0))$
end for
return $\{x_0^{(j=0)}, ..., x_0^{(j=J)}\}$

and $N = 1000$. The corresponding pre-trained neural models from FT2T were used; these models are available on the Internet on the FT2T GitHub repository [27]. We also conducted experiments on samples from the TSPlib [25], a highly renown benchmark in the TSP community. We used the instances of the TSPlib with size ranging from 10^2 to 10^4 cities with the model pretrained on 1000 cities instances. This tests both the ability to generalize beyond the training distribution and the training size. To be consistent with previous publications [3, 18, 19, 26], the running times being reported are times to solve 128 instances. However, due to the stochastic nature of the generated solution, we averaged the optimality gap and the time to solve 128 instances over $128^2 = 16384$ instances to get more accurate performance mean and variability estimates. Statistical test of superiority to compare optimality gaps were performed using Welch test. For the TSPlib we measured the mean and variance of the optimality gap over 48 replicas of the dataset, with different random seeds, to ensure enough statistical power.

On Sampling Size. The official FT2T implementation [27] does not perform a pure gradient refinement loop; instead, at each step it concatenates the gradient-guided output with the naive (non gradient-guided) one, effectively doubling the candidate pool per iteration. This design introduces a parallel sampling effect beyond the theoretical gradient only update. In our definition the sampling size is the effective number of parallel candidate solutions being processed simultaneously. So a sampling size S from FT2T (for which we keep the notation S to be consistent with the published paper and released code) has to be compared to PCI with sampling size $2S$. Such comparison matches the effective candidate count under similar compute budgets. This reflects the true inference time diversity leveraged by FT2T and PCI.

Hardware Experiments involving GPU were all carried on the same cluster of 8 GPU A100 40GB. A single GPU was used for each experiment except for the TSP-1000 with sampling size 4 and above where the 8 GPUs were used in parallel. Execution times were linear in the number of GPUs and are reported in the table as GPU-min (i.e. 1 min on 8 GPUs = 8 GPU minutes).

LKH3 running time was measured on a Ryzen 9 5900X CPU with 24 cores. All running times are measured based on the sequential processing of the set of instances to make comparison easier.

4.2 Experimental Results

First, we compared FT2T and PCI on random instances generated from Euclidean 2D TSP with 500 cities (results in table 1) and 1000 cities (results in table 2). We see that PCI outperforms FT2T both in terms of compute speed and of performance. PCI also exhibits lower variance. The average optimality gap differences were all significant with $p < .0001$ (Welch test).

Table 1. Comparison of optimality gaps (OG) -reported as mean (standard deviation)- and running times, averaged over 128 instances, on large scale 2D uniformly distributed Euclidean TSP instances. xS denotes a parallel sampling of size S. Note that as explained in the main text Fast-T2T effectively doubles the sampling size so fast-T2T x2S compares with PCI xS and there is no comparison for PCI x1.

	TSP-500			
	1 step, 1 renoise		5 steps, 5 renoises	
Method	OG.	time	OG.	time
PCI x1	0.90% (0.47)	0.20mn	0.49% (0.28)	0.78mn
PCI x2	0.73% (0.38)	0.32mn	0.34% (0.21)	1.3mn
Fast-T2T x1	0.74% (0.49)	0.55mn	0.40 %(0.59)	1.9mn
PCI x8	0.48% (0.27)	1.0mn	0.17% (0.13)	4.0mn
Fast-T2T x4	0.54% (0.73)	1.8mn	0.22% (0.46)	5.8mn

Then we compared FT2T and PCI on instances of the TSPlib. We used the model trained on TSP-1000 instances to solve instances of size ranging from 100 to 10^4 cities. Results are displayed in table 3. PCI outperformed FT2T both in terms of performance and running time. The average optimality gap differences were all significant with $p < .0001$ (Welch test). We also noticed the impact of the gradient search on the memory: GPU RAM requirement is significantly lower with PCI. This results in the ability of PCI to solve larger instances on the same hardware.

These results show that PCI is able to generalize to instances with sizes and/or distributions differing from the ones used for the training of the underlying model. Outperforming FT2T, PCI establishes the new state-of-the-art performance of neural CO methods.

Table 2. Comparison of optimality gaps (OG) -reported as mean (standard deviation)- and running times, averaged over 128 instances, on large scale 2D uniformly distributed Euclidean TSP instances. xS denotes a parallel sampling of size S. Note that as explained in the main text Fast-T2T effectively doubles the sampling size so fast-T2T x2S compares with PCI xS and there is no comparison for PCI x1.

	TSP-1000			
	1 step, 1 renoise		5 steps, 5 renoises	
Method	OG.	time	OG.	time
PCI x1	1.13% (0.37)	0.69mn	0.70% (0.26)	2.4mn
PCI x2	0.94% (0.31)	1.1mn	0.53% (0.21)	4.3mn
Fast-T2T x1	1.0% (0.34)	2.1mn	0.57% (0.23)	7.0mn
PCI x8	0.67% (0.23)	4.0mn	0.31% (0.14)	16.5mn
Fast-T2T x4	0.75% (0.26)	7.8mn	0.36% (0.16)	25.5mn

Table 3. Comparison of optimality gaps (OG) -reported as mean (standard deviation)- and running times on TSPlib instances whose size ranges from 10^2 to 10^4 cities. xS denotes a parallel sampling of size S. Note that as explained in the main text Fast-T2T effectively doubles the sampling size hence the absence of data to compare with PCIx1. The running time is reported to solve once the entire dataset (76 instances).

	TSPlib 100-10,000			
	1 step, 1 renoise		5 steps, 5 renoises	
Method	OG.	time	OG.	time
PCI x1	1.95 %(1.62)	1.4mn	1.37% (1.39)	4.4mn
PCI x2	1.58% (1.43)	2.4mn	1.15 %(1.36)	8.4mn
Fast-T2T x1	1.71 %(1.53)	4.1mn	1.25% (1.43)	13.8 mn
PCI x8	1.13 %(1.35)	8.3 mn	0.83% (1.25)	32.2 mn
Fast-T2T x4	Out of memory		Out of memory	

4.3 Comparison With LKH-3

We compared the time/performance tradeoffs of PCI and LKH3. The time/performance tradeoff for PCI is based on the various settings given in table 2 i.e. sampling size and number of steps. For LKH3 this tradeoff is based on changing the number of runs $(10^2, 10^3, 10^4)$ and the max number of trials per run $(5, 10, 20)$. For both parameters the central value is the default one. Fig. 3 reports the results for instances drawn from the training distribution (Euclidean 2D uniform TSP-1000). Fig. 4 reports the same figures for the instances of the TSPlib.

On these two plots, we note the remarkable performance of PCI on random instances: on these instances, PCI matches LKH3 time/performance trade-off. On TSPlib instances, for the same time budget, we note that PCI is generating tours that are not as good as those returned by LKH3, about 1% off. However, for a specific instance (fl3795), PCI consistently outperformed LKH3. It is well-known that the performance of various algorithms is not the same on random instances and on real-world instances: the latter are more structured, a feature

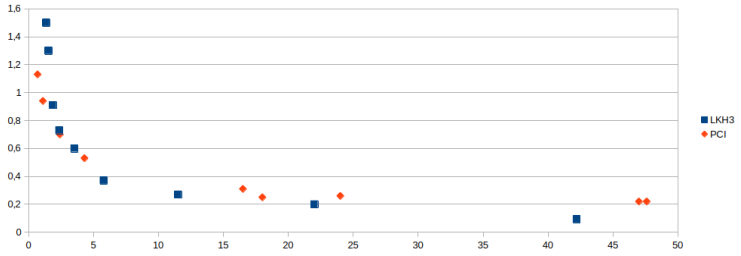

Fig. 3. Total time (x-axis, in minutes) versus mean optimality gap (y-axis, in percentage) performance of PCI and LKH3 on 128 TSP-1000 random instances.

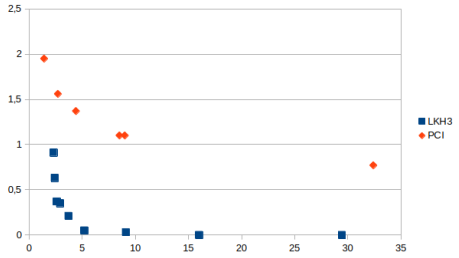

Fig. 4. Total time (x-axis, in minutes) versus mean optimality gap (y-axis, in percentage) performance of PCI and LKH3 on the 76 TSPlib instances with size ranging from10^2 to 10^4 cities.

that LKH3 is designed to exploit, whereas PCI is not able to do alike. This is crucially related to the training of underlying neural models.

Overall, PCI is the current state-of-the-art method in diffusion-based neural CO, but also comes very close to the performances of the state-of-the-art methods on the TSP when evaluated on distributions that match the training ones.

5 Discussion

We have demonstrated experimentally that PCI is an efficient and fast alternative to gradient search to improve inference time solution generation. It is worth noting that these benefits come out of the box without any retraining or fine-tuning the neural model: using publicly released, already pre-trained, models (in this work: [27]); hence at no further training cost.

The gain in inference time is perfectly understood as PCI requires less computational steps; for the gain in solution quality we can hypothesize the following: (1) FT2T motivates gradient-based refinement by arguing that following the objective gradient in a relaxed latent space should improve solution quality. However, the gradient is computed in a continuous relaxation of a discrete combinatorial space. This relaxation is highly non-convex and disconnected; small steps in the relaxed space do not guarantee any improvement in the discrete

objective after decoding. (2) The gradient search uses an energy-based model of the tour length as a surrogate objective for the unknown ground truth optimal solution. Associated gradient refinement only explores a narrow neighborhood around each initial sample. This does not guarantee any improvement after local search. This contrasts with projection-based strategies such as the ones used in PCI which are not local.

Our empirical results confirm this: PCI, which enforces feasibility and leverages structure-aware local moves, outperforms gradient refinement under lower compute budgets.

6 Conclusion and future work

In this paper, we have presented PCI, our new neural CO solver, and demonstrated it on the TSP. We compared the experimental performance of PCI with the state-of-the-art diffusion-based neural network based method FT2T, as well as the LKH3 TSP dedicated heuristic. This comparison shows that PCI very significantly improves the state-of-the-art of diffusion-based neural methods, and though not outperforming LKH3, PCI comes close to its performance. This research is applicable to all types of diffusion application where structural constraints can be formalized and enforced e.g. through projections. This includes other combinatorial optimization problems, such as Vehicle Routing Problems. Compared to LKH-3, our results show that while this neural method can match or even outperform (on short time scale and training distribution) decade-long handcrafted heuristics, it suffers from out-of-training distribution performance drop as shown on the TSPlib dataset. Improving generalizability is one of our future lines of work. Our models are trained using uniformly random instances, and it is well-known that algorithms do not behave on random TSP instances as they do on real-world instances. This prompts the investigation of particular structured random instances to train such models (either to re-train them from scratch, or fine-tune existing models). Similar to IDEQ, to constrain the diffusion to predict a solution on the space of locally optimal Hamiltonian tours, we used projections. Noticeably, the Hamiltonian tour projection uses a greedy insertion procedure. There are other projections of increasing complexity like beam-search or Monte-Carlo Tree Search that may lead to better results. The projection itself may not be the most effective approach to ensure constraint satisfaction; exploring alternative strategies is another line of future research.

Acknowledgments

Mickaël Basson is also a full-time employee of Lilly France detached to work on this project. We acknowledge the Région Hauts-de-France CPER project CornelIA. This work was granted access to the HPC resources of IDRIS under the allocation AD011015857R1 made by GENCI on the Jean Zay supercomputer. Both authors acknowledge the Scool research group for an outstanding working environment.

Bibliography

[1] Applegate, D., Bixby, R., Chvatal, V., Cook, W.: Concorde tsp solver. https://www.math.uwaterloo.ca/tsp/concorde (2003), accessed: 2024-05-06

[2] Austin, J., Johnson, D.D., Ho, J., Tarlow, D., van den Berg, R.: Structured denoising diffusion models in discrete state-spaces. In: Proc. NeurIPS (2021), neurIPS 2021

[3] Basson, M., Preux, P.: Ideq: an improved diffusion model for the tsp. In: Proc. LOD. LNCS, Springer (2025)

[4] Bi, J., Ma, Y., Wang, J., Cao, Z., Chen, J., Sun, Y., Chee, Y.M.: Learning generalizable models for vehicle routing problems via knowledge distillation. In: Advances in Neural Information Processing Systems (NIPS) (2022)

[5] Drakulic, D., Michel, S., Mai, F., Sors, A., Andreoli, J.M.: Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. In: Proc. NeurIPS (2023)

[6] Fang, H., Song, Z., Weng, P., Ban, Y.: Invit: A generalizable routing problem solver with invariant nested view transformer. In: ICML'24: Proceedings of the 41st International Conference on Machine Learning (2024)

[7] Gao, C., Shang, H., Xue, K., Li, D., Qian, C.: Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. In: Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI) (2024)

[8] Gao, C., Shang, H., Xue, K., Qian, C.: Neural solver selection for combinatorial optimization. In: Proceedings of the 42nd International Conference on Machine Learning (ICML). Vancouver, Canada (2025)

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

[10] Heek, J., Hoogeboom, E., Salimans, T.: Multistep consistency models. arXiv preprint arXiv:2403.06807 (2024)

[11] Helsgaun, K.: An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. Tech. rep., Roskilde University (2017)

[12] Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. In: Proc. NeurIPS. pp. 6840–6851 (2020)

[13] Johnson, D., McGeoch, L.: The traveling salesman problem: A case study in local optimization. In: Local Search in Combinatorial Optimisation, pp. 215—-310. John Wiley and Sons Ltd (1997)

[14] Kim, M., Park, J., Park, J.: Sym-nco: Leveraging symmetricity for neural combinatorial optimization. In: Proc. NeurIPS. pp. 1936–1949 (2022)

[15] Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: Proc. ICLR (2019)

[16] Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: Proc. ICLR (2019)

[17] Kwon, Y., Choo, J., Kim, B., Yoon, I., Min, S., Gwon, Y.: POMO: policy optimization with multiple optima for reinforcement learning. In: Proc. NeurIPS. pp. 21188–21198 (2020)
[18] Li, Y., Guo, J., Wang, R., Yan, J.: T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. In: Advances in Neural Information Processing Systems (2023)
[19] Li, Y., Guo, J., Wang, R., Zha, H., Yan, J.: Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. In: Proc. NeurIPS (2024)
[20] Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. Operation Research **21**, 498–516 (1973)
[21] Ma, N., Tong, S., Jia, H., Hu, H., Su, Y.C., Zhang, M., Yang, X., Li, Y., Jaakkola, T., Jia, X., Xie, S.: Scaling inference time compute for diffusion models. In: Proc. CVPR (2025)
[22] Min, Y., Bai, Y., Gomes, C.P.: Unsupervised learning for solving the travelling salesman problem. In: Proc. NeurIPS. pp. 47264–47278 (2023)
[23] Nichol, A.Q., Dhariwal, P.: Improved denoising diffusion probabilistic models. In: Meila, M., Zhang, T. (eds.) Proc. ICML. vol. 139, pp. 8162–8171 (2021)
[24] Qiu, R., Sun, Z., Yang, Y.: Dimes: A differentiable meta solver for combinatorial optimization problems. In: Proc. NeurIPS. pp. 25531–25546 (2022)
[25] Reinelt, G.: Tsplib: A traveling salesman problem library. ORSA Journal on Computing **3**, 376–384 (1991), http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/
[26] Sun, Z., Yang, Y.: DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In: Proc. NeurIPS (2023)
[27] Thinklab-SJTU: Fast-T2T. GitHub repository (2025), https://github.com/Thinklab-SJTU/Fast-T2T
[28] Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Proc. NeurIPS (2015)
[29] Ye, H., Wang, J., Liang, H., Cao, Z., Li, Y., Li, F.: Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In: Proc. AAAI (2024)
[30] Zhou, C., Lin, X., Wang, Z., Tong, X., Yuan, M., Zhang, Q.: Instance-conditioned adaptation for large-scale generalization of neural combinatorial optimization (2024), arXiv:2405.01906
[31] Zhou, J., Wu, Y., Song, W., Cao, Z., Zhang, J.: Towards omni-generalizable neural methods for vehicle routing problems. In: Proceedings of the 40 th International Conference on Machine Learning (PMLR) (2023)